Predictive Reachability Using a Sample-based Approach

Debashis Sahoo[1], Jawahar Jain[3], Subramanian Iyer[2], David Dill[1], and E. Allen Emerson[2]

[1] Stanford University, Stanford CA 94305, USA,
[2] University of Texas at Austin, Austin, TX 78712, USA
[3] Fujitsu Lab. of America

Abstract. BDD based reachability methods suffer from lack of robustness in performance, whereby it is difficult to estimate which one should be adopted for a given problem. We present a novel approach that examines a few short *samples* of the computation leading to an automatic, robust and modular way of reconciling the various methods for reachability. Our approach is able to intelligently integrate diverse reachability techniques such that each method can possibly get enhanced in efficiency. The method is in many cases orders of magnitude more efficient and it finishes all the invariant checking properties in VIS-Verilog benchmarks.

1 Introduction

BDD based reachability methods suffer from wild inconsistency in performance, whereby it is difficult to estimate which method should be adopted for a given problem. We analyze four different ways of doing reachability analysis, forward or backward reachability using partitioned [4] or unpartitioned BDDs [1, 2] for state set representation. It is often the case that though one method can compute reachability easily, the others find it very difficult. In this paper, we present a completely automatic strategy to determine the more effective method by running a few short *samples* of the above methods. These samples provide a short initial sampling of the performance of the various methods by observing the initial computations until a predefined cutoff in BDD size is reached. This approach determines the best direction for reachability analysis as well as the effectiveness of performing state space partitioning. Note that each method has its own domain of applicability. We have designed our approach so that it can benefit from the strengths of each method.

Importantly, at the end of the independently run samples, we allow all their computation to be shared. This can significantly enhance the performance of each technique. In many cases the reduction in reachability time for standard OBDD methods can be dramatic when its reached state set is augmented using information from POBDD samples.

2 Prediction using short samples

We use a sample-based algorithm to predict the effective method. A *sample* for an algorithm is a short initial computation using that algorithm.

The algorithm runs one sample each of the backward and forward partitioned reachability followed by forward and backward symbolic monolithic (non-partitioned) reachability. This order is chosen because we find backward reachability and partitioned reachability more suitable for finding bugs. Therefore, if there is a "easy" bug, then it can be found during the sampling process. The samples are run until a predefined size *cutoff* is exceeded. This cutoff is small enough to allow efficient performance of symbolic operations and is set at a fixed multiple of the representation size of the state transition relation.

If the samples themselves do not finish the computation, they are used to predict the most effective approach for the rest of the computation. Firstly, the appropriate direction is determined from the samples of symbolic forward and backward reachability. We use the number of images completed as measure for deciding the most effective method.

After selecting the direction, the algorithm tries to predict whether partitioned reachability is more effective than the monolithic approach, where state sets are represented as single BDDs. This is done by considering the number of states reached by samples run using both approaches in the selected direction. If the total number of reachable states explored by either method is significantly better than that of the other method, then we have a winner. If this number is comparable for both approaches, then a meaningful metric to break the tie seems to be the *rate of coverage* defined as number of states covered vs. corresponding time.

In this manner, the samples are used to pick a method that is likely to be the most effective method.

2.1 Augmenting the state sets

To avoid the repeated overlapping computations, after deciding the effective method the algorithm augments the initial states and the invariant by adding the states reached by all samples. In the forward direction, the reachability analysis starts from the union of the reached states using both forward samples. Likewise, the error set, which is set of states that satisfy the negation of the invariant, is replaced by the union of the sets states reached by the two backward samples. If the direction of computation is backward, then the error states are the start set and the augmented initial states are the target. This allows the computations performed by the samples to be reused.

In the next section, we describe our experiments and analyze the results.

3 Experiments

We compare the methodology proposed in this paper with the forward and backward reachability approaches of VIS and static partitioned reachability analysis. We compute one sample each in forward and backward directions, using partitioned as well as non-partitioned data structures for the state set in reachability. Our current package is not optimized with respect to partitioned exploration of state space. For example, it doesn't implement all the efficient heuristics presented in [3, 5].

All experiments were run on identical dual processor Xeon machines. They were allowed to run for a maximum time of one day, and the memory available to each run was bounded by 500MB.

Benchmarks

For experiments on reachability and invariant checking, we chose the public domain circuits from the VIS-Verilog [7] benchmark suite. In the following, we indicate property number i of circuit named *ckt* as *ckt_i*. Table 1 shows the runtime for checking the invariants of the VIS-verilog benchmark circuits for five methods. The entry "T" and "M" in the table represents a timeout limit of 1 day and memory out limit of 500MB.

4 Conclusion

In this paper, we presented an automatic self-tuning sample-based approach to address the inconsistency in performance of the BDD based reachability techniques. Many of the circuits time-out on one or other direction of reachability and some abort even when using partitioning. However, we find that the circuits aborted by backward are finished by forward and vice-versa in many cases. Note, the samples enable one to automatically select the appropriate method and the performance of the sample-centric approach is very robust and always significantly better than the worst. Such cases are shown in Table 1 (d). The table shows that the completely automatic sample-based approach is able to pick the right method from a set of different methods by using short samples of their initial reachability computation.

In a few cases, the wrong method may be picked, but even so, the sample-based approach is able to complete, due to the information available from the other samples. A more detailed version of this paper can be obtained from *http://verify.stanford.edu/PAPERS/dsahoo-charme05-e.pdf* [6].

5 Acknowledgments

The authors thank Fujitsu Laboratories of America, Inc for their gifts to support the research. Prof. Emerson thanks the NSF for support via grants CCR-009-8141 and CCR-020-5483. Prof. Dill thanks the NSF for support via grants CCR-012-1403. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] R. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

[2] O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines based on symbolic execution. In *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems*, 1989.

[3] S. Iyer, D. Sahoo, C. Stangier, A. Narayan, and J. Jain. Improved symbolic Verification Using Partitioning Techniques. In *Proc. of CHARME 2003*, volume 2860 of *Lecture Notes in Computer Science*, 2003.

[4] A. Narayan. et. al., Reachability Analysis Using Partitioned-ROBDDs. In *ICCAD*, pages 388–393, 1997.

[5] D. Sahoo, S. Iyer, J. Jain, C. Stangier, A. Narayan, David L. Dill, and E. Allen Emerson. A partitioning methodology for bdd-based verification. In Alan J. Hu and Andrew K. Martin, editors, *FMCAD*, volume 3312 of *Lecture Notes in Computer Science*, pages 399–413. Springer, 2004.

[6] D. Sahoo, J. Jain, S. Iyer, David L. Dill, and E. Allen Emerson. Predictive reachability using a sample-based approach. In *http://verify.stanford.edu/PAPERS/dsahoo-charme05-e.pdf*, 2005.

[7] VIS. Verilog Benchmarks http://vlsi.colorado.edu/~ vis/.

ckt_inv	Inv Res : Pass / Fail	Time in sec. Static Pobdd Fwd	Static Pobdd Bwd	Vis Fwd	Vis Bwd	Trace Based
(a) Advantage due to intersection of Forward and Backward						
vsa16a_7	F	2610	808	2146	458	77
vsaR_1	F	M	124	24558	56	54
(b) Advantage due to POBDD State Space Representation						
am2901_1	F	T	67	T	431	68
ball_6	F	175	T	T	T	103
ball_7	F	22	T	3530	T	45
palu_1	F	1.0	684	714	4630	1.8
sp_product_1	P	50	T	740	507	52
(c) Addition of Partitioned Traces Makes Subsequent Unpartitioned Reachability Easier						
FIFOs_1	P	M	M	2986	T	1973
blackjack_1	P	5750	T	2273	T	1234
blackjack_2	P	6268	T	20565	T	979
blackjack_4	P	5795	T	2259	T	1307
ns3_1	P	43569	T	16840	19166	5269
ns3_5	P	T	T	14696	T	6456
ns3_6	P	48721	M	28063	T	4938
ns3_7	P	M	T	22612	T	7220
(d) Robust Predictive Capability: Timeouts Avoided						
am2910_1	F	660	5.3	T	2.0	5.8
b12_1	F	48	9528	48	2561	77
b12_2	F	T	T	T	8019	25535
b12abs_2	F	2977	449	163	536	446
blackjack_3	F	1054	T	3371	T	1337
blackjack_5	P	62752	T	2614	T	13259
crc_1	F	20459	1.5	T	0.9	1.5
eight_1	P	4.5	1194	1.1	173	5.8
eight_2	P	4.6	2466	1.1	344	6.2
mm_product_1	P	600	T	49	352	154
ns3_2	P	M	8895	21602	16454	24903
ns3_3	P	T	85851	T	2050	4751
ns3_4	P	M	24477	24539	3770	6263
ns3_8	P	71494	T	6268	29196	50938
ns3_9	P	81048	3174	18247	479	9373
ns3_10	P	75011	2834	9518	604	12946
ns3_11	P	60490	10.9	51166	8.2	10.9
ns3_12	P	65219	27.3	49968	8.2	25.7
rotate32_1	F	53033	1.5	51078	0.7	1.5
s1269b_1	P	3351	1.3	12994	0.7	1.3
s1269b_5	P	3379	3.5	13677	0.6	3.5
soapLTL4_1	P	254	T	80.1	T	408
soap_1	P	176	T	45.6	T	181
soap_2	P	77.3	T	30.1	T	81.9
soap_3	P	47.8	T	46.4	T	80.9
spinner32_1	F	33356	8.3	43264	1.9	9.5
vsa16a_1	P	M	43.0	T	18.4	42.6
vsa16a_2	P	T	27.6	T	16.8	27.4
vsa16a_4	P	T	41.5	T	19.2	41.3
vsa16a_5	P	M	42.1	T	18.8	41.6
vsa16a_6	F	2499	60.5	1387	25.5	61.0
vsa16a_8	F	2498	61.7	1387	27.4	59.8

"T" is Timeout of 86,400 s; "M" is Memory out of 500 MB.

Table 1. Invariant Checking on ALL Vis-Verilog benchmarks that take more than 10 minutes in at least one of the methods.