Bisimulation and Model Checking*

Kathi Fisler and Moshe Y. Vardi**

Department of Computer Science, Rice University
Houston, TX 77005-1892
{kfisler,vardi}@cs.rice.edu

Abstract. State space minimization techniques are crucial for combating state explosion. A variety of verification tools use bisimulation minimization to check equivalence between systems, to minimize components before composition, or to reduce a state space prior to model checking. This paper explores the third use in the context of verifying invariant properties. We consider three bisimulation minimization algorithms. From each, we produce an on-the-fly model checker for invariant properties and compare this model checker to a conventional one based on backwards reachability. Our comparisons, both theoretical and experimental, lead us to conclude that bisimulation minimization does not appear to be viable in the context of invariance verification because performing the minimization requires as many, if not more, computational resources as model checking the unminimized system through backwards reachability.

1 Introduction

The state-explosion problem inspires extensive research into state-space reduction techniques. *Bisimulation minimization* [6], a technique that preserves the truth and falsehood of all μ-calculus (and hence all CTL*, CTL, and LTL) properties [4], is particularly attractive in the context of symbolic model checking for two reasons. First, bisimulation can be computed as the fixpoint of a simple boolean expression, so it is easily expressed symbolically. Second, unlike many other reduction techniques, it can be computed automatically, which is consistent with the automated spirit of model checking.

Our earlier work shows that using bisimulation minimization as a preprocessing phase to model-checking reduces the resources needed for model checking [2]. The combined cost of minimization followed by model checking, however, appears to outweigh the costs of simply model checking the unminimized system. This paper explores this problem in the context of verifying safety properties, which is the most fundamental model checking task. We convert three

* The full version of this paper appears as a technical report [3].

** Supported in part by NSF grants CCR-9628400 and CCR-9700061, and by a grant from the Intel Corporation. Part of this work was done when the second author was a Varon Visiting Professor at the Weizmann Institute of Science.

L. Pierre and T. Kropf (Eds.): CHARME'99, LNCS 1703, pp. 338–342, 1999.

bisimulation minimization algorithms into BDD-based, on-the-fly model checkers for safety properties. A combination of theoretical and experimental analyses on these algorithms show that they do not improve resource usage over basic backwards reachability. This strong negative result casts doubt on bisimulation minimization's utility as a state-space reduction technique for global finite-state transition systems in symbolic model checking.

2 Three Bisimulation Minimization Algorithms

Bisimulation minimization algorithms partition a state space into equivalence classes such that states in the same class agree on whether the invariance holds and on their next-state transitions to other classes. These algorithms follow a common outline. First, they partition the states into two blocks: those which satisfy the invariance and those that do not. Next, they repeatedly split existing blocks into new ones until all states in a block agree on their next-state transitions to other blocks. If some states in a block B_1 reach states in B_2 and some do not, B_2 is called a *splitter* of B_1. The minimized system contains one state from each block (now an equivalence class) in the final partition.

The naïve bisimulation minimization algorithm has two shortcomings in the context of symbolic model checking. First, it computes the *relation*, rather than the individual equivalence classes. The BDD for the relation requires twice as many variables as the BDDs for the classes. Second, the naïve algorithm fails to distinguish between reachable and unreachable blocks. Several bisimulation algorithms address one or both of these problems. Our work considers the algorithms by Paige and Tarjan (henceforth PT) [7], Bouajjani, Fernandez, and Halbwachs (henceforth BFH) [1], and Lee and Yannakakis (henceforth LY) [5]. We chose these algorithms for the following reasons:

- **PT:** Has the best provable worst-case running time of bisimulation minimization algorithms that do not distinguish between reachable and unreachable blocks.
- **BFH:** Improves on PT by choosing only reachable blocks to split on each iteration; however, it may split an unreachable block that was split off from the reachable block being split in the current iteration.
- **LY:** Improves on BFH by never splitting an unreachable block.

By splitting few, if any, unreachable blocks, the BFH and LY algorithms are tailored to verification contexts. The PT algorithm, although not so tailored, is interesting because it chooses splitters instead of blocks to split (LY and BFH do the latter). None of the algorithms is fully symbolic. LY and BFH operate on a symbolically-represented transition system and produce an explicit-state minimized transition system. PT is originally expressed for explicit state systems; we converted it to the same hybrid symbolic/explicit style as LY and BFH.

Converting these algorithms to on-the-fly model checkers for safety properties involved adding an extra flag to each block to aid in detecting failed properties. We also restricted the PT-based algorithm to only split reachable blocks. The full paper [3] proves that our new algorithms behave as the originals when the tested properties hold, and always discover failures when they do not.

	State	BR			LY			BFH			PT		
	Vars	Iter	Time	Mem	Iter	Time	Mem	Iter	Time	Mem	Iter	Time	Mem
gigamax	16	6	1.8	5.59	5	2.3	5.60	6	2.1	5.59	6	2.0	5.58
eisenberg*	17	19	1.1	3.80	18	3.3	3.99	19	9.3	4.48	270	9.3	4.28
abp	19	11	0.9	3.81	10	2.3	3.85	11	2.8	3.82	19	2.0	3.86
bakery*	20	58	1.3	3.70	57	6.5	3.87	58	126.9	9.67	212	7.8	4.55
treearb4	23	24	3.5	4.28	23	16.9	5.18	24	99.0	6.14	232	118.3	6.06
elev23	32	1	3.9	8.45	1	4.2	8.54	1	4.4	8.51	1	4.0	8.43
coherence1	37	5	3.6	6.28	4	85.5	22.0	5	33.0	20.0	23	29.5	8.55
coherence2	37	14	9.3	7.81	13	279.3	31.0	14	174.8	21.0	166	567.4	18
coherence3*	37	5	6.5	7.96	4	84.2	20.0	5	24.4	11.0	9	7.9	7.89
coherence4*	37	5	7.3	8.58	4	78.2	18.0	5	34.4	11.0	685	13.8H	68
elev33	45	1	7.0	11.0	1	444.5	17.0	1	443.8	17.0	1	7.2	11
elev43	56	1	11.9	15.0	1	1590.1	42.0	1	1661.0	39.0	1	12.2	15
tcp*	80	1	3.1	7.83	1	3.6	8.06	1	3.0	8.08	1	3.2	7.83

Table 1. Experimental comparison of the algorithms. A $*$ after an experiment name indicates that the tested invariant property fails. The Iter columns indicates how many iterations an algorithm took before locating an error or reaching a fixpoint. The units for the Time and Mem columns are seconds and megabytes, respectively. An H after the time indicates hours, rather than seconds.

3 Theoretical and Experimental Analysis

We want to compare our new minimizing model checkers to backwards reachability (henceforth BR). BR has a similar flavor to bisimulation minimization. Starting from a partition into good and bad states, BR repeatedly identifies good states that reach the bad states and puts them into a new "frontier" set. Our results formally prove that the contents of the successive frontier sets bear close, and often exact, resemblance to the contents of the new blocks computed in each iteration of bisimulation minimization. However, each algorithm computes these similar sets in different ways. We therefore need a way to compare and possibly predict the algorithms' relative performance.

Lower bounds on the numbers of operations that each algorithms performs provide a means of comparison. We have derived the bounds below in terms of several variables: n, the number of iterations BR needs to terminate; M, image operations; I, intersection operations; U, union operations; D, set-difference operations; and E, equality tests.

BR: $n * (M + U + D + 2E + I)$
LY: $(n-1) * (5M + 4I + 3D + 4E)$
BFH: $(M + I + 2E) * \frac{n^2+3n}{2} + n * D$
PT: $n * (2M + D + I + E)$

We have proven that we can bound the number of iterations that each algorithm takes from the number of BR iterations: LY needs one less iteration than BR,

BFH uses the same number of iterations as BR, and PT requires at least as many iterations as BR.

In predicting the behavior of the four algorithms for symbolic model checking with BDDs, the number of image computations (variable M) should be most useful. The bounds indicate that BFH requires fewer image computations than LY on runs requiring fewer than five iterations. However, BFH's performance should degrade as the number of iterations gets larger. Predicting PT's performance from the bounds is more difficult because there is no upper bound on the number of iterations it performs relative to BR. Based on variable M, we expect BR to have the best performance overall.

Table 1 presents time and memory statistics for running these algorithms on a suite of invariant properties, some of which hold and some of which do not. Our experimental framework uses VIS (version 1.2) [8] as a front-end to obtain the BDDs for the transition relation, initial state, and bad states from Verilog designs and CTL invariant properties. Our code for each algorithm uses VIS's routines for performing image computations (using partitioned transition relations). In order to have precise control over the BR experiments, we used our own implementation of BR. All runs were performed on an UltraSparc 140 with 128 megabytes of memory running Solaris 5.5.1.

These results show that BR has better time and memory usage than the three adapted minimization algorithms in all cases except elev23, in which PT achieved a negligible savings in memory over BR. As predicted, BFH does perform worse than LY on the designs that require the largest numbers of iterations (eisenberg, bakery, and treearb4). The difference between BFH and LY is most pronounced for bakery, which required the most iterations. BFH generally requires less memory and time than LY on the smaller examples. The one example that falls outside of our predictions is coherence2: on this example, BFH appears to compute images of much smaller sets than LY, which avoided blowup in the intermediate BDDs. Despite taking many more iterations, PT does surprisingly well in comparison to BFH and LY. With the exception of example coherence4, PT has comparable or better memory performance. Its time performance is comparable or better on all examples but treearb4, coherence2, and coherence4. Thus, under our optimization of not splitting unreachable blocks, choosing unreachable blocks as splitters does not seem to hurt PT's performance.

Combining the theoretical and experimental evidence, using bisimulation minimization either as a part of, or as a pre-processor to, model checking invariant properties of transition systems does not appear to be a viable approach. This does not imply that bisimulation has no role in verification contexts. Minimization in a compositional verification framework may make certain verification problems tractable that would not be so without minimization. Similarly, minimization can be used to collapse infinite-state systems into finite ones for purposes of exhaustive analyses. It remains to be seen what other implications our results have for the use of bisimulation in verification. We plan to continue our investigation in the context of verification of liveness properties.

References

1. Bouajjani, A., J.-C. Fernandez and N. Halbwachs. Minimal model generation. In *Proc. Intl. Conference on Computer-Aided Verification (CAV)*, pages 197–203. Springer-Verlag, 1990. 339
2. Fisler, K. and M. Y. Vardi. Bisimulation minimization in an automata-theoretic verification framework. In *Proc. Intl. Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 115–132. Springer-Verlag, 1998. 338
3. Fisler, K. and M. Y. Vardi. Bisimulation and model checking (extended version). Technical report TR99-339, Rice University, Department of Computer Science, 1999. Available at `http://www.cs.rice.edu/CS/Verification/`. 338, 340
4. Hennessy, M. and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of ACM*, 32:137–161, 1985. 338
5. Lee, D. and M. Yannakakis. Online minimization of transition systems. In *Proc. 24th ACM Symposium on Theory of Computing*, pages 264–274, Victoria, May 1992. 339
6. Milner, R. *A Calculus of Communicating Systems*. Springer Verlag, Berlin, 1980. 338
7. Paige, R. and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16:973–989, 1987. 339
8. The VIS Group. VIS: A system for verification and synthesis. In *Proc. of the 8th Intl. Conference on Computer Aided Verification (CAV)*, pages 428–432. Springer Verlag, July 1996. 341