Fighting State Space Explosion: Review and Evaluation

Radek Pelánek*

Department of Information Technology, Faculty of Informatics
Masaryk University Brno, Czech Republic
xpelanek@fi.muni.cz

Abstract. In order to apply formal methods in practice, the practitioner has to comprehend a vast amount of research literature and realistically evaluate practical merits of different approaches. In this paper we focus on explicit finite state model checking and study this area from practitioner's point of view. We provide a systematic overview of techniques for fighting state space explosion and we analyse trends in the research. We also report on our own experience with practical performance of techniques. Our main conclusion and recommendation for practitioner is the following: be critical to claims of dramatic improvement brought by a single sophisticated technique, rather use many different simple techniques and combine them.

1 Introduction

If you are a practitioner who wants to apply formal methods in industrial critical systems, you have to address following problems: Which of the many approaches should I use? If I want to improve the performance of my tool, which of the techniques described by researchers should I use? Which techniques are worth the implementation effort? These are important questions which are not easy to answer, nevertheless they are seldom addressed in research papers.

Research papers rather propose a steady flow of novel techniques, improvements, and optimizations. However, the experimental work reported in research papers has often poor quality [59] and it is difficult to judge the practical merit of proposed techniques. The goal of this paper is to provide an overview of research and realistic assessment of practical merits of techniques. The paper should serve as a guide for a practitioner who is trying to answer the above given questions.

It is not feasible to realize this goal for the whole field of formal verification at once. Therefore, we focus on one particular area (explicit model checking) and give an overview of research in this area and report on practical experience. Even though our discussion is focused on one specific area, we believe that our main recommendation — that it is better to combine several simple techniques rather than to focus on one sophisticated one — is applicable to many other areas of formal methods.

* Partially supported by GA ČR grant no. 201/07/P035.

Explicit model checking is principally very simple — a brute force traversal of all possible model states. Despite the simplicity of the basic idea, explicit model checking is still the best approach for many practically important areas of application, e.g., verification of communication protocols and software. The popularity of the approach is illustrated by large number of available tools (e.g., Spin, CADP, mCRL2, Uppaal, Divine, Java PathFinder, Helena) and widespread availability of courses and textbooks on the topic (e.g., [10]).

The main obstacle in applying explicit model checking in practice is the state space explosion problem. Hence, the research focuses mainly on techniques for fighting state space explosions — during the last 15 years more than 100 papers have been published on the topic, proposing various techniques for fighting state space explosion. What are these techniques and how can we classify them? What is the real improvement brought by these techniques? Which techniques are practically useful? Which techniques should a practitioner study and use?

We try to answer these questions, particularly we provide the following:

– We overview techniques for fighting state space explosion in explicit model checking and divide them into four main areas (Section 2).
– We review and analyse research on fighting state space explosion, and discuss main trends in this research (Section 3).
– We report on our own practical experience with application and evaluation of techniques for fighting state space explosion (Section 4).
– Based on the review of literature and our experience, we provide specific recommendation for practitioner in industry (Section 5).

The main aim of this paper is to present and support the following message: Rather than optimizing the performance of a single sophisticated technique, we should use many different simple techniques, study how to combine them, and how to run them effectively in parallel.

2 Overview of Techniques for Fighting State Space Explosion

Fig. 1. gives an algorithm Explore which explores the reachable part of the state space. This basic algorithm can be directly used for verification of simple safety properties; for more complex properties, we have to use more sophisticated algorithms (e.g., cycle detection [72]). Nevertheless, the basic ideas of techniques for fighting state space explosion are similar. For clarity, we discuss these techniques mainly with respect to the basic Explore algorithm.

The main problem of explicit state space exploration is state space explosion problem and consequently memory and time requirements of the algorithm Explore. Techniques for fighting state space explosion can be divided into four main groups:

1. Reduce the number of states that need to be explored.

```
proc Explore($M$)
  $Wait = \{s_0\}$;  $Visited = \emptyset$
  while $Wait \neq \emptyset$ do
      get $s$ from $Wait$
      explore state $s$
      foreach $s' \in$ successors of $s$ do
        if $s' \notin Visited$ then
            add $s'$ to $Wait$
            add $s$ to $Visited$ fi od
  od
end
```

Fig. 1. The basic algorithm which explores all reachable states. Data structure *Wait* (also called open list) holds states to be visited, data structure *Visited* (also called visited list, closed list, transposition table, or just hash table) stores already explored states.

2. Reduce the memory requirements needed for storing explored states.
3. Use parallelism or distributed environment.
4. Give up the requirement on completeness and explore only part of the state space.

In the following we discuss these four types of approaches and for each of them we list examples of specific techniques.

2.1 State Space Reductions

When we inspect some simple models and their state spaces, we quickly notice significant redundancy in these state spaces. So the straightforward idea is to try to exploit this redundancy and reduce the number of states visited during the search. In order to exploit this idea in practice, we have to specify which states are omitted from the search and we have to show the correctness of the approach, i.e., prove that the visited part of the state space is equivalent to the whole state space with respect to some equivalence (typically bisimulation or stutter equivalence).

State based reductions State based reductions exploit observation that if two states are bisimilar then it is sufficient to explore successors of only one of them. The reduction can be performed either on-the-fly during the exploration or by a static modification of the model before the exploration. Examples of such reductions are symmetry reduction [12,20,43,44,70,74], live variable reduction [21,69], cone of influence reductions, and slicing [19,35].

Path based reductions Path based reductions exploit observation that sometimes it is sufficient to explore only one of two sequences of actions because they are just different linearizations of "independent" actions and therefore have the same effect. These reductions try to reduce the number of equivalent interleavings. Examples of such reductions are transition merging [18,48], partial order reduction [27,33,40,63,64], τ-confluence [11], and simultaneous reachability analysis [55].

Compositional methods Systems are often specified as a composition of several components. This structure can be exploited in two ways: compositional generation of the state space [46] and assume-guarantee approach [22,32,66].

2.2 Storage Size Reductions

The main bottleneck of model checking are usually memory requirements. Therefore, we can save some memory at the cost of using more time, i.e., by employing some kind of time-memory trade-off. The main source of memory requirements of the algorithm Explore is the structure *Visited* which stores previously visited states. Hence, techniques, which try to lower memory requirements, focus mainly on this structure.

State compression During the search, each state is represented as a byte vector which can be quite large (e.g., 100 bytes). In order to save space, this vector can be compressed [25,26,30,39,49,56,73] or common components can be shared [38]. Instead of compressing individual states, we can also represent the whole structure *Visited* implicitly as a minimized deterministic automaton [41].

Caching and selective storing Instead of storing all states in the structure *Visited*, we can store only some of these states — this approach can lead to revisits of some states and hence can increase runtime, but it saves memory. Techniques of this type are for example:

– caching [24,28,65], which deletes some currently stored states when the memory is full,
– selective storing [9,49], which stores only some states according to given heuristics,
– sweep line method [15,54,68], which uses so called progress function; this function guarantees that some states will not be revisited in the future and hence these states can be deleted from the memory.

Use of magnetic disk Simple use of magnetic disk leads to an extensive swapping and slows down the computation extremely. So the magnetic disk have to be used in a sophisticated way [7,8,71] in order to minimize disk operations.

2.3 Parallel and Distributed Computation

Another approach to manage a large number of states is to use even more brute force — more processors.

Networks of workstations Distributed computation can be realized most easily by network of workstations connected by fast communication medium (i.e., workstations communicate by message passing). In this setting the state space is partitioned among workstations (i.e., each workstation stores part of the data structure *Visited*) and workstations exchange messages about states to be visited (*Wait* structure), see e.g., [23,50,51]. The application of distributed environment for verification of liveness properties is more complicated, because classical algorithms are based on depth-first search, which cannot be easily adapted for distributed environment. Hence, for verification of liveness properties we have to use more sophisticated algorithms, see e.g., [1,2,3,5,13,14].

Multi-core processors Recently, multi-core processors become widely available. Multi-core processors provide parallelism with shared memory, i.e., the possibility to reduce run-time of the verification by parallel exploration of several states, see e.g., [4,42].

2.4 Randomized Techniques and Heuristics

If the memory requirements of the search are too large even after the application of above given techniques, we can use randomized techniques and heuristics. These techniques explore only part of the state space. Therefore, they can help only in the detection of an error; they cannot assist us in proving correctness.

Heuristic search (also called directed or guided search) States are visited in an order given by some heuristics, i.e., *Wait* list is implemented as priority queue [31,47,67]. Different heuristic approach is to use genetic algorithm which tries to 'evolve' a path to a goal state [29].

Random walk and partial search Random walk does not store any information and always visits just one successor of a current state [34,60]. This basic strategy can be extended in several ways, e.g., by visiting a subset of all successors (instead of just one state), storing some states in the *Visited* structure, or combining random walk with local breadth-first search, see e.g., [36,45,52,53,60].

Bitstate hashing The algorithm does not store whole states but only one bit per state in a large hash table [37]. In a case of collision some states are omitted by the search. A more involved version of this technique is based on Bloom filters [16,17].

3 Research Analysis

What are the trends in the research literature about techniques for fighting state space explosion? Is the quality of experimental evidence improving? How significant is the improvement reported in research papers? How is this improvement changing over time?

3.1 Research Papers

In order to answer the above given questions, we have collected and analyzed large set of research papers. More specifically, we collected research papers that

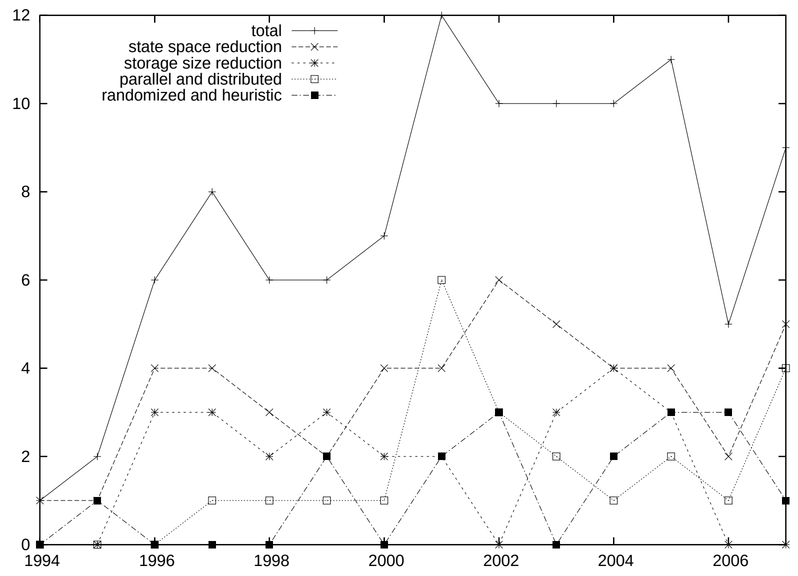

Fig. 2. Numbers of publications; note that some papers can be counted in two categories.

describe techniques for fighting state space explosion in explicit model checking of finite state systems[1].

The collection contains more than 100 papers – these papers were obtained by systematically collecting papers from the most relevant conferences and by citation tracking. The full list of reviewed papers, which includes all papers referenced above, is freely available[2]. The collection is certainly not complete, but we believe that it is a good sample of research in the area.

Fig. 2. shows the number of publications in each year during the last 13 years. Although there are rises and downfalls, the overall flow of publications on the topic is rather steady. The figure also shows that all four areas described in the previous section are pursued concurrently.

3.2 Quality of Experiments

Although some of the considered research is rather theoretically oriented (e.g., partial order reduction), all considered techniques are in fact heuristics which aim at improving performance of model checking tools. So what really matters is the practical improvement brought by each technique. To assess the improvement

[1] In few cases we also include techniques which are not purely explicit, but target the similar application domain (i.e., the experiments are done on same models as for other included papers).

[2] `http://www.fi.muni.cz/~xpelanek/amase/reductions.bib`

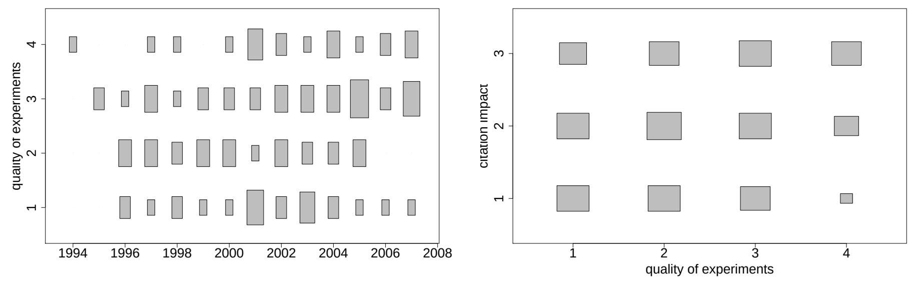

Fig. 3. The first graph shows the quality of experiments reported in model checking papers during time. The size of a box corresponds to a number of published papers in a given year and quality category. The second graph shows the relation between experiment quality and citation impact; citation impact is divided into three categories: less than 10 citations, 10–30 citations, more than 30 citations; only publication before 2004 are used.

it is necessary to perform experimental evaluation. Only good experiments can provide realistic evaluation of practical merits of proposed techniques.

In order to study the quality of experiments, we classify experiments in each paper into one of four classes, depending on the number and type of used models[3]:

1. Random inputs or few toy models.
2. Several toy models (possibly parametrized) or few simple models.
3. Several simple models (possibly parametrized) or one large case study.
4. Exhaustive study of parametrized simple models or several case studies.

Fig 3. presents the quality of experiments in papers from our sample. The figure shows that the quality on average is not very good and, what is even more disappointing, that there is slow progress in time, although many realistic case studies are available (see [59] for more detailed discussion of these issues).

Since there is a large number of techniques, it is important to compare performance of novel techniques with previously studied one. However, analysis of our research sample shows that only about 40% papers contain some comparison with similar techniques; this ratio is improving with time, but only slowly. Moreover, the comparison is usually only shallow.

Our analysis also shows one encouraging trend. Fig. 3. shows that there is a relation between quality of experiments and citation impact of a paper — research with better experiments is more cited.

[3] The classification is clearly slightly subjective. Nevertheless, we believe that the main conclusions of our analysis do not depend on the subjective factor.

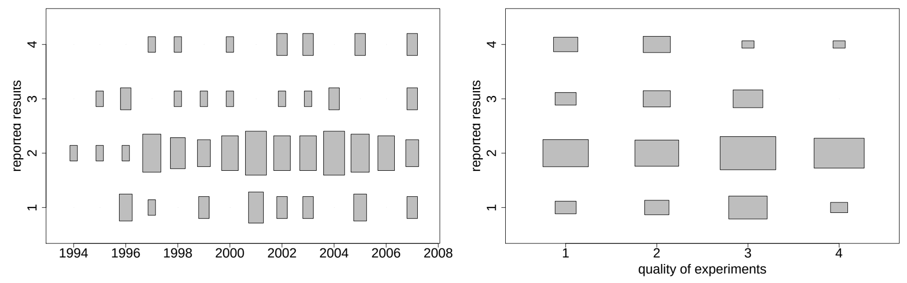

Fig. 4. Reported improvement with respect to time and quality of experiments.

3.3 Reported Improvement

Before the discussion of improvements reported in research papers, we clarify the terminology that we use to measure this improvement. We use the notion 'reduction ratio' to denote the ratio between the memory consumption of the technique for fighting state space explosion and the memory consumption of the standard reachability (exploration of full reachable state space). Some authors report 'reduced by' factor, i.e., if we report 'reduction ratio' 80%, it means that the memory consumption was 'reduced by' 20%. Note that in this section we analyze reduction ratios as reported by authors, not what we consider realistic reduction ratios of techniques.

For clarity of presentation, we again divide the reported reduction ratios into four classes:

1. Reported reduction ratio is 50% or worse (or sometimes good but sometimes worse than 100%).
2. Reported reduction ratio is in most cases 10%-50%.
3. Reported reduction ratio is in most cases 1%-10%.
4. Reported reduction ratio is better than 1% (or exponential improvement is reported or only out-of memory for full search is reported, i.e., reduction ratio is impossible to assess).

Fig. 4. shows that in most cases the reported reduction is in the second category (reduction between 10% and 50%). The relation with the quality of experiments clearly demonstrates that this is also the most realistic evaluation — better results are often caused by poor experiments, not by special features of techniques.

There is no clear trend with respect to time, i.e., it seems that novel techniques do not significantly improve on performance of previous techniques. This does not automatically mean that the recent research is misguided. In some cases novel technique provides principally different way how to obtain the reduction and can be combined with previously proposed techniques in orthogonal way. Novel technique can also extend the application domain of previously studied techniques.

As we already mentioned, the research in this domain is purely practically motivated. However, the amount of research into certain topic is not really related to its practical merit. For example, our experience (described in next section) shows that the dead variable reduction brings similar improvement as partial order reduction. Nevertheless, there are significantly more research papers about partial order reduction than about dead variable reduction. This is probably not due to the practical merits of partial order reduction, but because it can be extensively studied theoretically.

4 Practical Experience

In this section we report on our experience with techniques for fighting state space explosion. Our experience is based on large-scale research studies, which are described in stand-alone publications. Here we provide only a brief description of these studies and present their main conclusions. Technical details can be found in cited papers.

Our experience report obviously does not cover all techniques for fighting state space explosion. However, we cover all four areas described in Section 2 and the main conclusions are in all cases similar, so we believe that it is reasonable to generalize our experience.

4.1 On-the-fly State Space Reductions

We evaluated several techniques for on-the-fly state space reductions. Setting of this study (see [57] for details):

– Implementation: publicly available implementations of explicit model checkers (Spin, Murphy, DiVinE).
– Models: models included in tool distributions plus few more publicly available case studies.
– Techniques: dead variable reduction, partial order reduction, transition merging, symmetry reduction.

When we measured the performance of techniques over realistic models, we found that the reduction ratio is usually worse than what is reported in research papers — research papers often use simple models with artificially high values of model parameters. More specifically, the main results of our evaluation are the following: dead variable reduction works on nearly all models, reduction ratio is usually between 10% and 90%; partial order reduction works only in some cases, reduction ratio is between 5% and 90%; transition merging works in similar cases as partial order reduction, it is weaker but easier to realize, reduction ratio is usually between 50% and 95%; symmetry reduction works only for few models (symmetric ones), reduction ratio is usually between 8% and 50%.

Our main conclusion from this study are the following:

– Each technique is applicable only to some types of models. No technique works really universally; more specialized techniques yield better reduction.

- On real models, no single technique is able to achieve reduction ratio significantly under 5%. Claims about drastic reduction, which occur in some papers, are not really appropriate.
- Since there are many techniques and many of them are orthogonal, most models can be reduced quite significantly.

4.2 Caching and Compression

From the area of 'storage size reduction' techniques we evaluated two techniques for reducing memory consumption of the data structure *Visited*. Setting of this study (see [62] for details):

- Implementation: all techniques are implemented in uniform way using the DiVinE environment [6] (source codes are publicly available).
- Models: 120 models from BEEM (BEnchmarks for Explicit Model checkers) [59].
- Techniques: state caching with 7 different caching strategies, state compression with Huffman coding (two variants: static code and code computed by training runs).

In the study we also reviewed previous research on storage size reduction techniques. We found that using proper parameter values with our simple and easy-to-implement techniques, we were able to achieve very similar results to those reported in other works which use far more sophisticated approaches. Concrete results of the evaluation are the following:

- Caching strategies are to a certain degree complementary. Using an appropriate state caching strategy, the reduction ratio is in most cases 10% to 30%.
- Using state compression, the reduction ratio is usually around 60%.
- The two techniques combine well.

4.3 Distributed Exploration

From the area of parallel and distributed techniques we report on the basic distributed approach to explicit model checking: we have a network of workstations connected by fast Ethernet, workstations communicate via message passing (MPI library), state space is partitioned among workstations. In this setting the reduction ratio is clearly bounded by $1/n$, where n is the number of workstations. In practice the reduction ratio is worse because of communication overhead. Here we report on results of our evaluation, however the results are rather typical in this area.

Setting of this study (see [58] for details):

- Implementation: the DiVinE tool (public version).
- Models: 120 models from BEEM (BEnchmarks for Explicit Model checkers) [59].

– Techniques: distributed reachability on 20 workstations.

In this study the speedup varies from 2 to 12, typical value of the speedup is between 4 and 6 (i.e., reduction ratio around 20%). We also found that the speedup is negatively correlated with the speed of successor generation by the tool.

4.4 Error Detection Techniques

From the area of 'randomized techniques and heuristics' we have chosen 9 techniques and evaluated their performance. In this case we do not study the reduction ratio, because it is not known — the experiments are done on models for which the standard reachability is not feasible. Therefore, we focus on relative performance of techniques and on the issue of complementarity.

Setting of this study (see [61] for details):

– Implementation: all techniques are implemented in uniform way using the DiVinE environment [6] (source codes are publicly available),
– Models: 54 models (with very large state space) from BEEM [59].
– Techniques: breadth-first search, depth-first search, randomized DFS, two variants of random walk, bitstate hashing with repetition, two variants of directed search, and under-approximation refinement based on partial order reduction.

For the evaluation we used several performance measures: number of steps needed to find an error, length of reported counterexample, and coverage metrics. The main results of this study are the following:

– There is no single best technique. Results depend on used performance metrics, even for a given metric, the most successful technique is the best one only over 25% of models.
– It is important to focus on complementarity of techniques, not just on their overall (average) performance. For example, in our study the random walk technique had rather poor overall performance, but it was successful on models where other techniques fall, i.e., it is a useful technique which we should not discard.

5 Conclusions

This paper is concerned with techniques for fighting state space explosion problem in explicit model checking. We review the research in the area during the last 15 years (more than 100 research papers) and report on our practical experience. As a result of our review we identify four main groups of techniques: state space reductions, storage size reductions, parallel and distributed computation, randomization and heuristics. These four groups are rather orthogonal and can be combined; within each group techniques are often based on similar ideas and their combination can be difficult.

The review of research shows that despite a steady flow of publications on the topic, the progress is not very significant — in fact the reduction ratio reported in research papers stays practically the same over the last 15 years. This analysis stresses the need for good practical evaluation. However, realistic evaluation of research progress is complicated by rather poor experimental standards and by unjustified claims by researchers.

Results reported in research papers often make an impression of dramatic improvements. Our practical experience suggests that it is not realistic to get better reduction ratio than 5% with a single technique, in fact in most cases the obtained reduction ratio is between 20% and 80%. Nevertheless, this does not mean that techniques for fighting state space explosion are not useful. Techniques of different types can be combined, and together they might be able to bring a significant improvement.

Our experience also suggest that simple techniques are often sufficient. The performance obtained by sophisticated techniques is often similar to performance of basic techniques from each area. Complicated techniques often achieve better results only for specialized application domains. This observation can be also supported by analysis of techniques implemented in model checking tools. Tools usually implement basic versions of many techniques, sophisticated techniques are often implemented only in a tool used by authors of the technique.

To summarise, we propose following recommendations for those who want to apply model checking in practice:

- Use large number of simple techniques of different types.
- Do not try to find 'the best' technique of a specific type. Try to find a set of simple complementary techniques and run all of them (preferably in parallel).
- Be critical to claims in research papers, particularly if the experimental evidence is poor.
- Use sophisticated techniques only if they are specifically targeted at your domain of application.
- Focus on combination of orthogonal techniques.

Researchers, we believe, should focus not just on the development of novel techniques, but also on issues of techniques combination, selection, and efficient scheduling: How to select right technique for a given model? In what order we should try available techniques? Can information gathered by one technique be used by another techniques?

Acknowledgment

I thank Václav Rosecký, Pavel Moravec, and Jaroslav Šeděnka for cooperation on practical evaluation of techniques.

References

1. J. Barnat, L. Brim, and I. Černá. Property driven distribution of nested DFS. In *Proc. of Workshop on Verification and Computational Logic*, number DSSE-TR-2002-5 in DSSE Technical Report, pages 1–10. University of Southampton, UK, 2002.
2. J. Barnat, L. Brim, and J. Chaloupka. Parallel breadth-first search LTL model-checking. In *Proc. of Automated Software Engineering (ASE'03)*, pages 106–115. IEEE Computer Society, 2003.
3. J. Barnat, L. Brim, and J. Chaloupka. From distributed memory cycle detection to parallel LTL model checking. *ENTCS*, 133(1):21–39, 2005.
4. J. Barnat, L. Brim, and P. Rockai. Scalable multi-core ltl model-checking. In *Proc. of SPIN Workshop*, volume 4595 of *LNCS*, pages 187–203. Springer, 2007.
5. J. Barnat, L. Brim, and J. Stříbrná. Distributed LTL model-checking in SPIN. In *Proc. of SPIN Workshop on Model Checking of Software*, volume 2057 of *LNCS*, pages 200–216. Springer, 2001.
6. J. Barnat, L. Brim, I. Černá, P. Moravec, P. Rockai, and P. Šimeček. DiVinE - a tool for distributed verification. In *Proc. of Computer Aided Verification (CAV'06)*, volume 4144 of *LNCS*, pages 278–281. Springer, 2006. The tool is available at `http://anna.fi.muni.cz/divine`.
7. J. Barnat, L. Brim, and P. Šimeček. I/o efficient accepting cycle detection i/o efficient accepting cycle detection. In *Proc. of Computer Aided Verification (CAV'07)*, volume 4590 of *LNCS*, pages 281–293. Springer, 2007.
8. J. Barnat, L. Brim, P. Šimeček, and M. Weber. Revisiting resistance speeds up i/o-efficient ltl model checking. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, volume 4963 of *LNCS*, pages 48–62. Springer, 2008.
9. G. Behrmann, K. G. Larsen, and R. Pelánek. To store or not to store. In *Proc. of Computer Aided Verification (CAV 2003)*, volume 2725 of *LNCS*. Springer, 2003.
10. M. Ben-Ari. *Principles of the Spin Model Checker.* Springer, 2008.
11. S. Blom and J. van de Pol. State space reduction by proving confluence. In *Proc. of Computer Aided Verification (CAV 2002)*, number 2404 in LNCS, pages 596–609. Springer, 2002.
12. D. Bosnacki. A light-weight algorithm for model checking with symmetry reduction and weak fairness. In *SPIN*, volume 2648 of *LNCS*, pages 89–103. Springer, 2003.
13. L. Brim, I. Černá, P. Krčál, and R. Pelánek. Distributed ltl model checking based on negative cycle detection. In *Proc. Foundations of Software Technology and Theoretical Computer Science (FST TCS 2001)*, volume 2245 of *LNCS*, pages 96–107. Springer, 2001.
14. I. Černá and R. Pelánek. Distributed explicit fair cycle detection. In *Proc. SPIN workshop*, volume 2648 of *LNCS*, pages 49–73. Springer, 2003.
15. S. Christensen, L.M. Kristensen, and T. Mailund. A sweep-line method for state space exploration. In *Proc. of Tools and Algorithms for Construction and Analysis of Systems (TACAS 2001)*, volume 2031 of *LNCS*, pages 450–464. Springer, 2001.
16. P. C. Dillinger and P. Manolios. Bloom filters in probabilistic verification. In *Proc. of Formal Methods in Computer-Aided Design (FMCAD)*, volume 3312 of *LNCS*, pages 367–381. Springer, 2004.
17. P. C. Dillinger and P. Manolios. Fast and accurate bitstate verification for SPIN. In *Proc. of SPIN workshop*, volume 2989 of *LNCS*, pages 57–75. Springer, 2004.

18. Y. Dong and C. R. Ramakrishnan. An optimizing compiler for efficient model checking. In *Proc. of Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX)*, pages 241–256. Kluwer, B.V., 1999.
19. M B. Dwyer, J. Hatcliff, M. Hoosier, V. P. Ranganath, Robby, and T. Wallentine. Evaluating the effectiveness of slicing for model reduction of concurrent object-oriented programs. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *LNCS*, pages 73–89, 2006.
20. E. A. Emerson and T. Wahl. Dynamic symmetry reduction. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 of *LNCS*, pages 382–396. Springer, 2005.
21. J. C. Fernandez, M. Bozga, and L. Ghirvu. State space reduction based on live variables analysis. *Journal of Science of Computer Programming (SCP)*, 47(2-3):203–220, 2003.
22. C. Flanagan and S. Qadeer. Thread-modular model checking. In *Proc. of SPIN Workshop*, volume 2648 of *LNCS*, 2003.
23. H. Garavel, R. Mateescu, and I. Smarandache. Parallel state space construction for model-checking. In *Proc. SPIN Workshop*, volume 2057 of *LNCS*, pages 217–234. Springer, 2001.
24. J. Geldenhuys. State caching reconsidered. In *SPIN*, volume 2989 of *LNCS*, pages 23–38. Springer, 2004.
25. J. Geldenhuys and P. J. A. de Villiers. Runtime efficient state compaction in SPIN. In *Proc. of SPIN Workshop*, volume 1680 of *LNCS*, pages 12–21. Springer, 1999.
26. J. Geldenhuys and A. Valmari. A nearly memory-optimal data structure for sets and mappings. In *Proc. of Model Checking Software (SPIN)*, volume 2648 of *LNCS*, pages 136–150. Springer, 2003.
27. P. Godefroid. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032 of *LNCS*. Springer, 1996.
28. P. Godefroid, G. J. Holzmann, and D. Pirottin. State space caching revisited. In *Proc. of Computer Aided Verification (CAV 1992)*, volume 663 of *LNCS*, pages 178–191. Springer, 1992.
29. P. Godefroid and S. Khurshid. Exploring very large state spaces using genetic algorithms. In *Proc. of Tools and Algorithms for Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *LNCS*, pages 266–280. Springer, 2002.
30. J. Gregoire. State space compression in spin with GETSs. In *Proc. Second SPIN Workshop*. Rutgers University, New Brunswick, New Jersey, 1996.
31. A. Groce and W. Visser. Heuristics for model checking java programs. *Software Tools for Technology Transfer (STTT)*, 6(4):260–276, 2004.
32. O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994.
33. G. Gueta, C. Flanagan, E. Yahav, and M. Sagiv. Cartesian partial-order reduction. In *Proc. of SPIN Workshop*, volume 4595 of *LNCS*, pages 95–112. Springer, 2007.
34. P. Haslum. Model checking by random walk. In *Proc. of ECSEL Workshop*, 1999.
35. J. Hatcliff, M. B. Dwyer, and H. Zheng. Slicing software for model construction. *Higher Order Symbol. Comput.*, 13(4):315–353, 2000.
36. G. J. Holzmann. Algorithms for automated protocol verification. *AT&T Technical Journal*, 69(2):32–44, 1990.
37. G. J. Holzmann. An analysis of bitstate hashing. In *Proc. of Protocol Specification, Testing, and Verification*, pages 301–314. Chapman & Hall, 1995.
38. G. J. Holzmann. State compression in SPIN: Recursive indexing and compression training runs. In *Proc. of SPIN Workshop*, 1997.

39. G. J. Holzmann, P. Godefroid, and D. Pirottin. Coverage preserving reduction strategies for reachability analysis. In *Proc. of Protocol Specification, Testing, and Verification*, 1992.
40. G. J. Holzmann and D. Peled. An improvement in formal verification. In *Proc. of Formal Description Techniques VII*, pages 197–211. Chapman & Hall, Ltd., 1995.
41. G. J. Holzmann and A. Puri. A minimized automaton representation of reachable states. *Software Tools for Technology Transfer (STTT)*, 3(1):270–278, 1998.
42. G.J. Holzmann and D. Bosnacki. The design of a multicore extension of the spin model checker. *IEEE Transactions on Software Engineering*, 33(10):659–674, 2007.
43. R. Iosif. Symmetry reduction criteria for software model checking. In *Proc. of SPIN workshop*, volume 2318 of *LNCS*, pages 22–41. Springer, 2002.
44. C. N. Ip and D. L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1–2):41–75, 1996.
45. M. D. Jones and J. Sorber. Parallel search for LTL violations. *Software Tools for Technology Transfer (STTT)*, 7(1):31–42, 2005.
46. J.P. Krimm and L. Mounier. Compositional state space generation from Lotos programs. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1997)*, volume 1217 of *LNCS*, pages 239–258. Springer, 1997.
47. A. Kuehlmann, K. L. McMillan, and R. K. Brayton. Probabilistic state space search. In *Proc. of Computer-Aided Design (CAD 1999)*, pages 574–579. IEEE Press, 1999.
48. R. P . Kurshan, V. Levin, and H. Yenigün. Compressing transitions for model checking. In *Proc. of Computer Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, pages 569–581. Springer, 2002.
49. K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *Proc. of Real-Time Systems Symposium (RTSS'97)*, pages 14–24. IEEE Computer Society Press, 1997.
50. F. Lerda and R. Sisto. Distributed-memory model checking with SPIN. In *Proc. of SPIN workshop*, volume 1680 of *LNCS*. Springer, 1999.
51. F. Lerda and W. Visser. Addressing dynamic issues of program model checking. In *Proc. of SPIN Workshop*, volume 2057 of *LNCS*, pages 80–102. Springer, 2001.
52. F. Lin, P. Chu, and M. Liu. Protocol verification using reachability analysis: the state space explosion problem and relief strategies. *Computer Communication Review*, 17(5):126–134, 1987.
53. M. Mihail and C. H. Papadimitriou. On the random walk method for protocol testing. In *Proc. Computer Aided Verification (CAV 1994)*, volume 818 of *LNCS*, pages 132–141. Springer, 1994.
54. T. Mailund and W. Westergaard. Obtaining memory-efficient reachability graph representations using the sweep-line method. In *TACAS*, volume 2988 of *LNCS*, pages 177–191. Springer, 2004.
55. K. Ozdemir and H. Ural. Protocol validation by simultaneous reachability analysis. *Computer Communications*, 20:772–788, 1997.
56. B. Parreaux. Difference compression in spin. In *Proc. of Workshop on automata theoric verification with the SPIN model checker (SPIN'98)*, 1998.
57. R. Pelánek. Evaluation of on-the-fly state space reductions. In *Proc. of Mathematical and Engineering Methods in Computer Science (MEMICS'05)*, pages 121–127, 2005.
58. R. Pelánek. Web portal for benchmarking explicit model checkers. Technical Report FIMU-RS-2006-03, Masaryk University Brno, 2006.

59. R. Pelánek. BEEM: Benchmarks for explicit model checkers. In *Proc. of SPIN Workshop*, volume 4595 of *LNCS*, pages 263–267. Springer, 2007.
60. R. Pelánek, T. Hanžl, I. Černá, and L. Brim. Enhancing random walk state space exploration. In *Proc. of Formal Methods for Industrial Critical Systems (FMICS'05)*, pages 98–105. ACM Press, 2005.
61. R. Pelánek, V. Rosecký, and P. Moravec. Complementarity of error detection techniques. In *Proc. of Parallel and Distributed Methods in verifiCation (PDMC)*, 2008.
62. R. Pelánek, V. Rosecký, and J. Šeděnka. Evaluation of state caching and state compression techniques. Technical Report FIMU-RS-2008-02, Masaryk University Brno, 2008.
63. D. Peled. Combining partial order reductions with on-the-fly model-checking. In *Proc. of Computer Aided Verification (CAV 1994)*, volume 818 of *LNCS*, pages 377–390, 1994.
64. W. Penczek, M. Szreter, R. Gerth, and R. Kuiper. Improving partial order reductions for universal branching time properties. *Fundamenta Informaticae*, 43(1-4):245–267, 2000.
65. G. D. Penna, B. Intrigila, I. Melatti, E. Tronci, and M. V. Zilli. Exploiting transition locality in automatic verification of finite state concurrent systems. *Software Tools for Technology Transfer (STTT)*, 6(4):320–341, 2004.
66. A. Pnueli. In transition from global to modular temporal reasoning about programs. *Logics and models of concurrent systems*, pages 123–144, 1985.
67. K. Qian and A. Nymeyer. Guided invariant model checking based on abstraction and symbolic pattern databases. In *Proc. of Tools and Algorithms for Construction and Analysis of Systems (TACAS 2004)*, volume 2988 of *LNCS*, pages 487–511, 2004.
68. K. Schmidt. Automated generation of a progress measure for the sweep-line method. In *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 192–204. Springer, 2004.
69. J. P. Self and E. G. Mercer. On-the-fly dynamic dead variable analysis. In *Proc. of SPIN Workshop*, volume 4595 of *LNCS*, pages 113–130. Springer, 2007.
70. A. P. Sistla and P. Godefroid. Symmetry and reduced symmetry in model checking. In *CAV*, volume 2102 of *LNCS*, pages 91–103. Springer, 2001.
71. U. Stern and D. L. Dill. Using magnetic disk instead of main memory in the murphi verifier. In *Proc. of Computer Aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 172–183. Springer, 1998.
72. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In D. Kozen, editor, *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS '86)*, pages 332–344. IEEE Computer Society Press, 1986.
73. W. Visser. Memory efficient state storage in SPIN. In *Proc. of SPIN Workshop*, pages 21–35, 1996.
74. T. Wahl. Adaptive symmetry reduction. In *Proc. of Computer Aided Verification (CAV'07)*, volume 4590 of *LNCS*, pages 393–405. Springer, 2007.