FI MU

Faculty of Informatics
Masaryk University Brno

Verification Manager: Automating the Verification Process

by

Radek Pelánek
Václav Rosecký

FI MU Report Series FIMU-RS-2009-02

March 2009

Verification Manager: Automating the Verification Process[*]

Radek Pelánek   Václav Rosecký
Faculty of Informatics
Masaryk University Brno, Czech Republic

March 4, 2009

Abstract

Although model checking is usually described as an automatic technique, the verification process with the use of model checker is far from being fully automatic. With the aim of automating the verification process, we elaborate on a concept of a verification manager. The manager automates some step of the verification process and enables efficient parallel combination of different verification techniques. We describe a realization of this concept for explicit model checking and discuss practical experience. Particularly, we discuss the problem of selection of input problems for evaluation of this kind of tool.

1 Introduction

Model checking consists of three phases: modeling, specification, and algorithmic verification. The first two are acknowledged to involve manual effort and user expertise, although researchers have proposed several techniques to automate these steps [2, 4, 7]. The third step is standardly considered to be fully automatic. We argue that in practice even the third step requires significant manual effort and user expertise and that it is important to focus on automating even this step.

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

1.1 Motivation for Automating the Verification Process

Given a model and a specification, model checking algorithmically checks all possible behaviours of the model and gives us 'yes' or 'no' answer. In practice, however, model checking techniques often reach a limit (on time or memory consumption) and do not give any clear answer. To obtain an answer, it is sometimes necessary to restore to a more abstract model, but in many cases it is sufficient to suitably tune parameters of the model checker. Hence the process of using a model checker can be quite elaborate and far from automatic.

In order to successfully verify a model, it is often necessary to select appropriate techniques and parameter values. The selection is difficult, because there is a very large number of different heuristics and optimization techniques – our review of techniques [17] found more than 100 papers just in the area of explicit model checking. These techniques are often complementary and there are non-trivial trade-offs which are hard to understand. In general, there is no best technique. Some techniques are more suited for verification, other techniques are better for detection of errors. Some techniques bring good improvement in a narrow domain of applicability, whereas in other cases they can worsen the performance [17]. The user needs a significant experience to choose good techniques.

Moreover, models are usually parametrized and there are several properties to be checked. Thus the process of verification requires not just experience, but also a laborious effort, which is itself error prone.

Another motivation for automating the verification process comes from trends in the development of hardware. Until recently, the performance of model checkers was continually improved by increasing processor speed. In last years, however, the improvement in processors speed has slowed down and processors designers have shifted their efforts towards parallelism [9]. This trend poses a challenge for further improvement of model checkers. A classic approach to application of parallelism in model checking is based on distribution of a state space among several workstations (processors) [8, 11, 25]. This approach, however, involves large communication overhead. Given the large number of techniques and hard-to-understand trade-offs, there is another way to employ parallelism: to run independent verification runs on individual workstations (processors) [9, 17, 20]. This approach, however, cannot be efficiently performed manually. We need to automate the verification process.

1.2 Our Proposal: Verification Manager

In our overview study of techniques for fighting state space explosion, we reached the following conclusion [17]: "Simple techniques are often sufficient. Rather then optimizing the performance of sophisticated techniques, we should use simple techniques, and study how to combine these simple techniques, and how to run them effectively in parallel." In this work we try to realize this idea.

We propose a concept of a verification manager. Verification manager is a tool which automates the verification process. As an input it takes a (parametrized) model and a list of properties. Then it employs available resources (hardware, verification techniques) to perform verification – the manager distributes the work among individual workstations, it collects results, and informs the user about progress and final results. Decisions of the manager (e.g., which technique should be started) are governed by a 'verification strategy'. The verification strategy needs to be written by an expert user, but since it is generic, it can be used on many different models. In this way even a layman user can exploit experiences of expert users. The general concept of verification manager is further developed in Section 2.

As a proof of concept we introduce a prototype of the verification manager for an area of explicit model checking – Explicit Model checking MAnager (EMMA). Realization of EMMA is described in Section 3. We also describe experiences with EMMA over models from the benchmark set BEEM [16]. Our experiences with evaluation raise some methodological issues – it turns out that it is quite difficult to perform fair evaluation of this kind of tool. These experiences are described in Section 4.

1.3 Related work

This paper is part of our long term effort. We have developed the BEEM benchmark set [16] and using this benchmark set we have performed several evaluation studies [20, 21]. With the use of these studies we have provided overview of techniques for fighting state space explosion [17]. In one of our publications [18] we have already suggested the basic concept of a verification manager. In this work we integrate all this previous progress and provide realization of the manager, which is based on insight gained from previous studies.

The most related work by other researchers is by Holzmann et al. Holzman and Smith [10] describe a tool for automated execution of verification runs for several model parameters and correctness properties; they use one fixed verification technique. Recently, Holzmann et al. [9] proposed 'swarm verification', which is based on parallel execution of many different techniques. Their approach, however, does not allow any communication among techniques and they do not discuss the selection of techniques that are used for the verification (verification strategy).

Owen et al. [13, 14] discuss complementarity issues in verification and propose to combine different tools. They illustrate the principles on large case studies. The approach is, however, not automated. Garavel and Lang [6] propose a scripting language for description of verification strategies for automating the process of compositional verification. The script calls techniques sequentially and has to be written specifically for each model (as opposed to our approach in which the strategy can be used universally and which executes techniques in parallel).

There are several other works, which employ similar ideas in different context or on a more abstract level. Sahoo et al. [23] use sampling of the state space to decide which BDD based reachability technique is the best for a given model. Mony et al. [12] use expert system for automating proof strategies. Eytani et al. [5] give a high-level proposal to use an 'observation database' for sharing relevant information among different verification techniques.

2 Verification Manager

In this section we discuss the general proposal for the verification manager and verification strategy. In the next section we introduce a concrete realization of these concepts.

2.1 Verification Meta-Search

Most of the research in automated verification is focused on the verification search problem: given a model M and a property φ, determine whether M satisfies φ, i.e., the aim is to search for an incorrect behaviour or for a proof. We believe that it is worthwhile to consider the verification meta-search problem [18] as well: given a model and a property, find a technique T and values p of its parameters such that $T(p)$ can provide an answer to the verification problem. This can be viewed as a stand-alone search problem which uses algorithms for the verification problem as black-boxes. Our aim is to study methods for performing this meta-search and heuristics to guide the search.

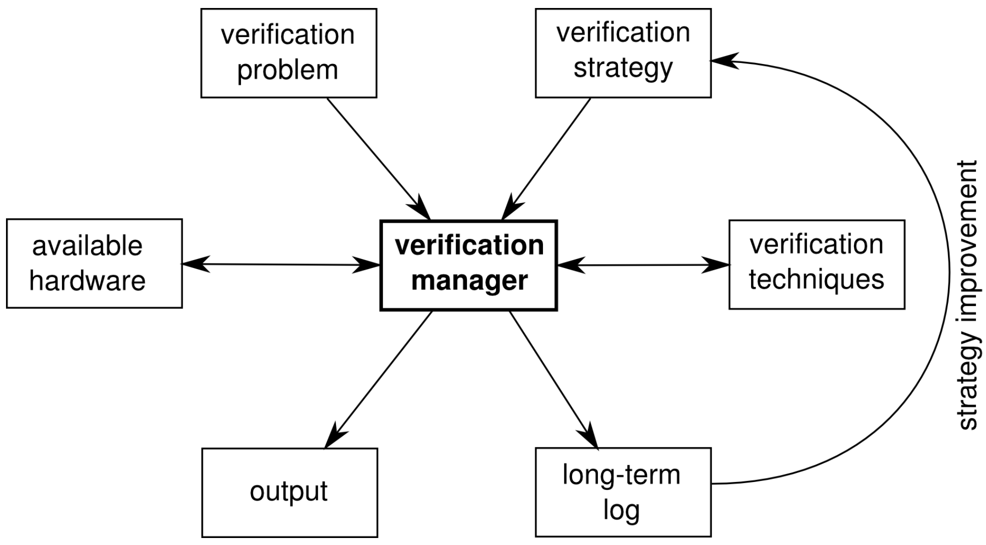

Figure 1: Verification manager — context.

We call an entity responsible for the verification meta-search a *verification manager* and a heuristic for the meta-search a *verification strategy*.

2.2 Functionality of the Manager

Let us be more specific about the functionality of the verification manager. Figure 1 gives the context, in which the manager operates:

Verification problem A model and a list of properties. The model may be parametrized, in such a case the input also contains list of instances (described by values of model parameters).

Verification strategy A heuristic for the meta-search problem (see below).

Verification techniques Available techniques which can be used for verification.

Available hardware Hardware available for verification (e.g., a network of workstations).

Output Running report about progress and final report about results.

Long-term log Stored data about performance of techniques. It can be used for improvement of strategy and for collecting benchmarking data (see discussion in Section 4).

Basic functionality of the verification manager consists of starting and stopping verification tasks. When starting a task, manager decides which technique with what parameter values should be run on what computer. This decision is based on verification strategy, currently available hardware, and results obtained so far. There are several ways in which a technique can terminate:

- The technique simply finishes.
- Timeout specified by verification strategy is reached and manager forces the technique to terminate.
- Based on intermediate results manager concludes that all results for a given model instance are already known, and therefore it terminates the technique.

Manager also collects all results and presents them to the user.

Note that this is just a basic functionality, that we are able to realize at this moment (see Section 3). We believe that in future it is feasible and meaningful to further expand the functionality of the manager, e.g., by using static analysis to analyze a model, or by employing several tools and incorporating translation among specification languages into the manager (see future work in Section 5).

2.3 Verification Strategy

The verification manager performs the verification meta-search by starting and terminating verification tasks. But how should it proceed with the meta-search? Which techniques should be called? The "meta state space" cannot be searched exhaustively – taking into account parameters of techniques, the meta state space is infinite. Experimental results reported in research papers and overall experience of the community suggest that there is no optimal deterministic method to search the meta state space. We therefore need some heuristic for the meta-search – a verification strategy. This strategy should be optimized for an application domain of interest using a feedback from a long-term log.

There are two extreme approaches to realization of a verification strategy. The first one is to use "hard-wired" strategy, i.e., to include the strategy as an integral part of the manager and to encode it in a high-level programming language. The advantage of this approach is that we can encode arbitrary strategy in this way. The disadvantage is that the basic functionality of the manager is not separated from the heuristics and therefore it is harder to develop the strategy and to improve it with the use of feedback.

The second extreme approach is to let the manager construct the strategy on its own. With this approach we specify just the list of available techniques and some learning algorithm (e.g., classifier system or genetic algorithm) and we let the manager to construct a strategy by learning. This approach would require very large amount of data to work well. At this moment, we are not convinced that this approach would lead to better strategy than strategy constructed by human expert. Nevertheless, this approach may be useful in the long-term application of manager (learning with the use of feedback from long-term log).

We believe that a reasonable way is between these two approaches. We fix a basic skeleton of the strategy (e.g., priority based scheme) and implement support for this skeleton into the manager. Specifics of the strategy (e.g., order of techniques, values of parameters) are specified separately in a simple format – this specification of strategy can be easily and quickly (re)written by an expert user.

3 Implementation: EMMA

In previous section we discussed the general concept of verification manager. Now we introduce a concrete implementation of the concept. Since this is the first step in this direction, we restrict our attention to explicit model checking and detection of safety errors. Our implementation is called EMMA (Explicit Model checking MAnager) and is available at:

`http://anna.fi.muni.cz/~xrosecky/emma_web`

3.1 Architecture

EMMA is based on the Distributed Verification Environment (DiVinE) [3]. All used verification techniques are implemented in C++ with the use of DiVinE environment. At the moment, we use the following techniques: breadth-first search, depth-first search, random walk, directed search, bitstate hashing (with refinement), and under-approximation based on partial order reduction. All techniques are publicly available either as a part of the DiVinE library or as a part of another published study [20, 21]. Other techniques can be easily incorporated.

The manager itself is implemented in Java. At the moment manager supports as the underlying hardware a network of workstations connected by Ethernet. Communication is based on SSH and stream socket, it consists of the following messages (messages are encoded in XML):

- manager $\rightarrow$ workstation:
  - initialization of a particular verification technique,
  - forced termination of a verification technique (e.g., timeout),
- workstation $\rightarrow$ manager:
  - intermediate result when an error is detected,
  - final results after termination of a technique.

The verification manager EMMA takes the following inputs (in correspondence with Figure 1): strategy (see below), list of available techniques, list of available workstations, and description of a parametrized model. Model description consists of model source code (in DVE format [15]) and XML file describing parameter values and properties (compatible with the BEEM project [16]).

During the verification the manager outputs intermediate results and information about the progress of verification. At the end of the verification the manager provides all results and summary information.

3.2 Strategy

Strategy description is given in the XML format. Specifically, we use two files to describe the strategy (see Fig. 2):

1. specification of verification techniques – this file contains for each technique:
   - name of a technique,
   - way to call the technique (i.e., name of executable file which should be called and its options),
   - list of parameters and their default values.
2. specification of a strategy – the verification strategy itself, i.e., specification of which techniques should be called, in what order.

```
<strategy>
  <total-timeout>1800</total-timeout>
  <run>
    <algorithm>random_walk</algorithm>
    <timeout>50</timeout>
    <params>
      <param>
        <name>max_depth</name>
        <value>500</value>
      </param>
    </params>
  </run>
  <run>
    <algorithm>random_dfs</algorithm>
    <timeout>10</timeout>
  </run>
  <run>
    <algorithm>bfs_reach</algorithm>
    <timeout>60</timeout>
  </run>
</strategy>
```

Figure 2: Example of a strategy.

For the first evaluation we use a simple priority-based strategies. For each technique we specify priority, timeout, and parameter values; techniques are executed in order according to their priorities.

The architecture of EMMA is built in such a way, that it can easily cope with more sophisticated strategies, particularly 'verification in phases'. With this approach, techniques are divided into several phases; in each phase techniques are called in parallel (e.g., in priority-based order); phases are executed sequentially. This approach provides the following possibilities:

- Start with a quick 'error detection phase' (many diverse error detection techniques) and then continue with a long 'verification phase' (few optimized technique for traversing the whole state space).
- We can have two consecutive phases with same techniques, based on result of the first phase we can modify priorities, timeouts, and parameter values for the second phase – employing the observation that different instances of a same parametrized model have similar state space properties and similar techniques work for them [19].

4 Experiences

In this section we report our (qualitative) experiences from using EMMA and we discuss why it is difficult to provide fair quantitative evaluation of the tool and different strategies.

4.1 General Experiences

We have done experiments with models from BEEM [16], using 4 workstations connected with fast Ethernet. The experiments were done with parametrized models with usually about 5 models and several properties to be checked. We set a overall timeout for the verification to 30 minutes. The general experiences are the following:

- Most of the results are obtained quickly (usually within the first minute). This stresses the usefulness of running report about results.
- The manager significantly simplifies the use of model checker for parametrized models even for an experienced user – this contribution is not easily measurable, but is very important for practical applications of model checkers.

4.2 Examples of Executions

EMMA distribution contains a script for visualization of executions. These visualizations can be used for better understanding of functionality of EMMA and for development and improvement of verification strategies.

Figure 3 shows a diagrams of a executions of two strategies over two models (more examples of visualizations are given on the tool web page). Firewire link [24] is a model of a communication protocol with many reachable properties[1], Szymanski protocol [1] is a mutual exclusion protocol with several versions, most of which are correct, i.e., specified properties are not reachable.

Strategy A consists of a large number of 'error detection' techniques with a short timeout. Strategy B consists of just one technique – simple depth first search with long timeout. On a Firewire model, which has many reachable goals, strategy A is much more successful: from 60 verification problems it can provide answer to 58 of them, whereas strategy B can answer only 45 of them; strategy A is moreover faster. On the other hand, on a Szymanski protocol, for which it is necessary to traverse the whole state space, the simple strategy B is more successful: from 24 verification problems it can answer 23 of them, whereas strategy A can answer only 21 of them.

These examples illustrate a more general issue with evaluation of a verification manager. How should we select input problems for evaluation?

4.3 Selection of Input Problems

Selection of input problems is an important, but often neglected issue in experimental evaluation. This issue is important even for evaluation of several techniques of the same type – see for example [20] for discussion of the influence of model selection (toy versus complex models) in the evaluation of error detection techniques. Selection of input problems becomes even more crucial issue when evaluating verification meta-search.

By the selection of input data we determine to large extend the results that we obtain from experiments. When we use mainly models without errors, strategies which focus on verification are more successful than strategies tuned for finding errors. When we use models with many easy-to-find errors, there are negligible differences among strategies and we can be tempted to conclude that the choice of strategy does not matter. When we use models with just few hard-to-find errors, there are significant differences among strategies; the success of individual strategies is, however, dependent very much on a choice of particular models and errors. The preceding statements are not just speculations, they are observations based on our experiences with EMMA. By suitable selection of input problems we could "demonstrate" (even using quite large set of inputs) both that "verification manager brings significant improvement" and "verification manager is rather useless".

[1]Most of these properties are not errors, but just protocol configurations for which we want to check reachability.

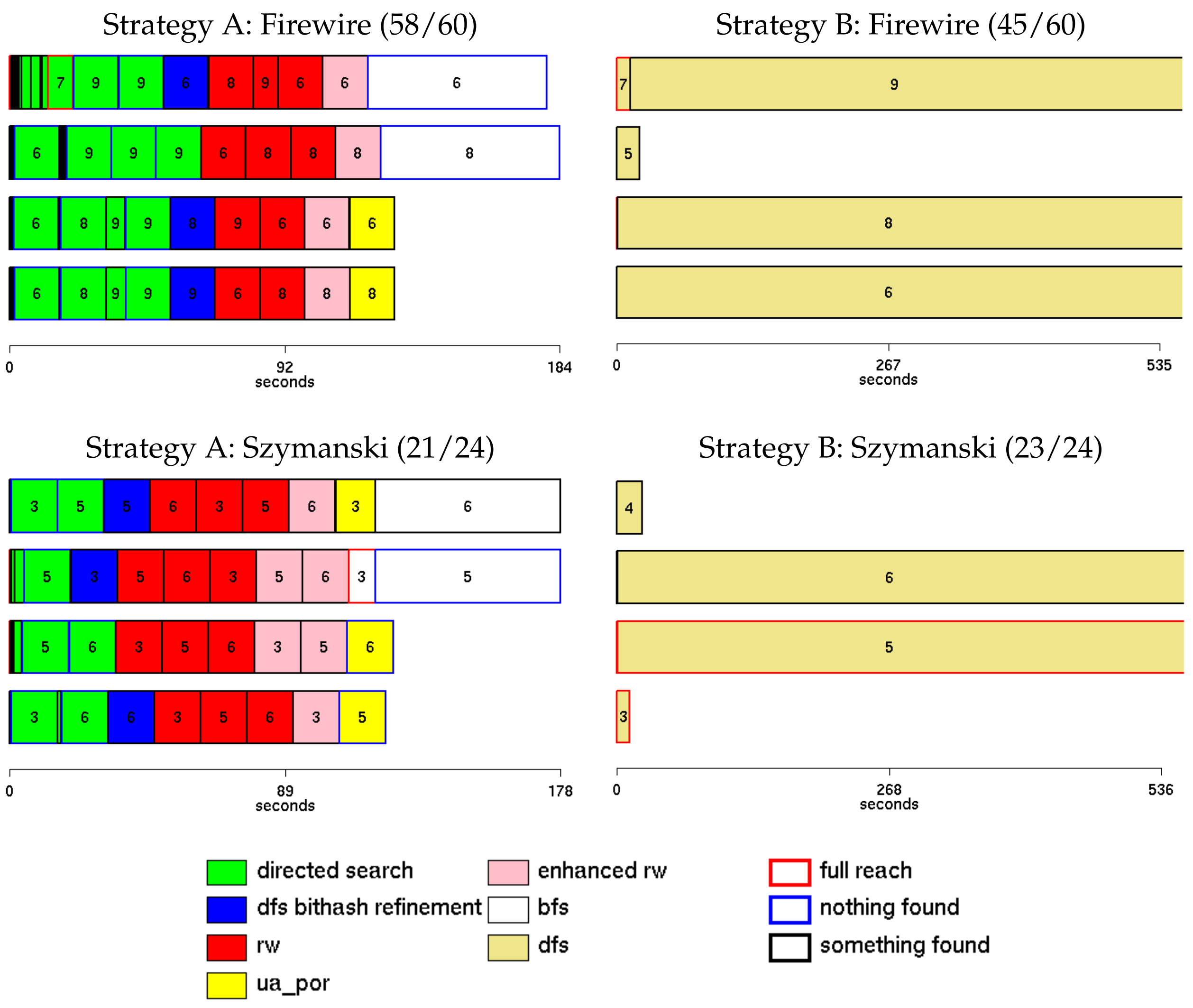

Figure 3: Illustration of EMMA executions on 4 workstations for two models and two strategies. Each line corresponds to one workstation; numbers in boxes are identifications of model instances.

So what are the ‘correct’ input problems? The ideal case, in our opinion, is to use a large number of realistic case studies from an application domain of interest; moreover, these case studies should be used not just in their final correct versions, but also in developmental version with errors. However, this ideal is not realizable at this moment – although there is already a large number of available case studies in the domain of explicit model checking, developmental versions of these case studies are not publicly available.

The employment of verification manager could help to overcome this problem. The long-term log can be used to archive all models and properties for which verification was performed (of course with user's content). Data collected in this way can be latter used for evaluation.

4.4 Comparison of Strategies

We have performed comparison of different strategies by running EMMA over (different selections of) models from BEEM [16] (probably the largest collection of models for explicit model checkers). Due to the above described bias caused by selection of models, we do not provide numerical evaluation, but only general observations:

- For models with many errors, it is better to use strategy which employs several different (incomplete) techniques.
- For models, which satisfy given property, it is better to use strategy which calls just one simple state space traversal technique with a large timeout.
- If two strategies are comprised of same techniques (with just different priorities, timeouts), there can be a noticeable difference among them, but this difference is usually less than order of magnitude. Note that differences among individual verification techniques are often larger than order of magnitude [20].

Suitable verification strategy depends on the application domain and also on the "phase of verification" – different strategies are suitable for early debugging of a model and for final verification. Thus the expected usage of tool like EMMA is the following:

- Expert user does a bit of experimenting over particular application domain and writes few strategies and hints on how to use them.
- These strategies can be then easily used by other users.

5 Conclusions and Future Work

We argue that although model checking technique is automatic, the verification process with the use of model checker is rather complicated and involves significant human expertise. In order to automatize this process, we propose a concept of a verification manager. We also introduce a realization of this concept for explicit model checking (EMMA tool). Experience show that:

- The manager significantly enhances the usability of a model checker, even for an experienced user (particularly for parametrized models with several properties).
- The performance of the different verification strategies depends on selected input problems. It is difficult to perform fair evaluation of this kind of tool.
- We should not expect to find a universal strategy, rather we should look for different strategies for different application domains and verification phases. It should, however, be sufficient to have just few strategies, so that it is easy to select a suitable one even for an inexperienced user.

In order to evaluate a verification manager in a more quantitative way, it is necessary to collect a benchmark of developmental versions of models with realistic and hard-to-find errors (current benchmarks contain mainly correct final version of models). A proposed 'long-term log' of the verification manager can be an appropriate way to collect such a benchmark.

This is just first step towards automatized execution of verification techniques. There are several important directions for future research:

- For the first prototype we restricted our attention only to detection of safety errors. However, the approach can be directly used also for verification of properties expressed in temporal logic. The approach should be particularly useful for verification of LTL properties, since there are many different algorithms and each is suitable for a different purpose (error detection, verification). We also plan to add other techniques such as state caching, state compression, and partial order reduction.
- The architecture of EMMA is designed in such a way, that it should be possible to employ several model checking tools (with the use automatic translator between specification languages). This extension should improve performance and allow further expansion of applicability (e.g., symbolic techniques).
- The manager could use static analysis and estimators [22] to guide the verification meta-search.
- Using the data from the long-term log, it could be interesting to study ways for automatic construction (improvement) of verification strategy by machine learning.

- By analysis of result, manager should be able to provide additional information for the user, e.g., which errors are simple (hard) to find.

References

[1] J. H. Anderson, Y.-J. Kim, and T. Herman. Shared-memory mutual exclusion: major research trends since 1986. *Distrib. Comput.*, 16(2-3):75–110, 2003.

[2] T. Ball, R. Majumdar, T. D. Millstein, and S. K. Rajamani. Automatic predicate abstraction of c programs. In *Proc. of Programming Language Design and Implementation (PLDI 2001)*, pages 203–213. ACM Press, 2001.

[3] 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`.

[4] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. of Computer Aided Verification (CAV 2000)*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.

[5] Y. Eytani, K. Havelund, S. D. Stoller, and S. Ur. Toward a Framework and Benchmark for Testing Tools for Multi-Threaded Programs. *Concurrency and Computation: Practice and Experience*, 19(3):267–279, 2007.

[6] H. Garavel and F. Lang. Svl: A scripting language for compositional verification. In *Proc. of Formal Techniques for Networked and Distributed Systems (FORTE '01)*, pages 377–394. Kluwer, B.V., 2001.

[7] G. J. Holzmann and M. H. Smith. Software model checking: extracting verification models from source code. *Softw. Test., Verif. Reliab.*, 11(2):65–79, 2001.

[8] 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.

[9] G.J. Holzmann, R. Joshi, and A. Groce. Tackling large verification problems with the swarm tool. In *Proc. of Model Checking Software: The SPIN Workshop*, volume 5156 of *LNCS*, pages 134–143. Springer, 2008.

[10] G.J. Holzmann and M.H. Smith. Automating software feature verification. *Bell Labs Technical Journal*, 5(2):72–87, 2000.

[11] F. Lerda and R. Sisto. Distributed-memory model checking with SPIN. In *Proc. of SPIN workshop*, volume 1680 of *LNCS*. Springer, 1999.

[12] H. Mony, J. Baumgartner, V. Paruthi, R. Kanzelman, and A. Kuehlmann. Scalable automated verification via expert-system guided transformations. In *Proc. of Formal Methods in Computer-Aided Design (FMCAD'04)*, volume 3312 of *LNCS*, pages 159–173. Springer, 2004.

[13] D. Owen, D. Desovski, and B. Cukic. Effectively combining software verification strategies: Understanding different assumptions. In *Proc. of International Symposium on Software Reliability Engineering (ISSRE '06)*, pages 321–330. IEEE Computer Society, 2006.

[14] D.R. Owen. *Combining Complementary Formal Verification Strategies to Improve Performance and Accuracy*. PhD thesis, West Virginia University, 2007.

[15] R. Pelánek. Web portal for benchmarking explicit model checkers. Technical Report FIMU-RS-2006-03, Masaryk University Brno, 2006.

[16] R. Pelánek. BEEM: Benchmarks for explicit model checkers. In *Proc. of SPIN Workshop*, volume 4595 of *LNCS*, pages 263–267. Springer, 2007.

[17] R. Pelánek. Fighting state space explosion: Review and evaluation. In *Proc. of Formal Methods for Industrial Critical Systems (FMICS'08)*, 2008. To appear.

[18] R. Pelánek. Model classifications and automated verification. In *Proc. of Formal Methods for Industrial Critical Systems (FMICS'07)*, volume 4916 of *LNCS*, pages 149–163. Springer, 2008.

[19] R. Pelánek. Properties of state spaces and their applications. *International Journal on Software Tools for Technology Transfer (STTT)*, 10(5):443–454, 2008.

[20] R. Pelánek, V. Rosecký, and P. Moravec. Complementarity of error detection techniques. In *Proc. of Parallel and Distributed Methods in verifiCation (PDMC)*, 2008.

[21] 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.

[22] R. Pelánek and P. Šimeček. Estimating state space parameters. Technical Report FIMU-RS-2008-01, Masaryk University Brno, 2008.

[23] D. Sahoo, J. Jain, S. K. Iyer, D. Dill, and E. A. Emerson. Predictive reachability using a sample-based approach. In *Proc. of Correct Hardware Design and Verification Methods (CHARME'05)*, volume 3725 of *LNCS*, pages 388–392. Springer, 2005.

[24] M. Sighireanu and R. Mateescu. Verification of the link layer protocol of the ieee-1394 serial bus (firewire). *Software Tools for Technology Transfer (STTT)*, 2(1):68–88, November 1998.

[25] U. Stern and D. L. Dill. Parallelizing the Murφ verifier. In *Proc. of Computer Aided Verification (CAV 1997)*, volume 1254 of *LNCS*, pages 256–267. Springer, 1997.