EMMA: Explicit Model Checking Manager (Tool Presentation)*

Radek Pelánek and Václav Rosecký

Department of Information Technology, Faculty of Informatics
Masaryk University Brno, Czech Republic

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. In this paper we elaborate on concept of a verification manager, which contributes to automation of the verification process by enabling efficient parallel combination of different verification techniques. We introduce a tool EMMA (Explicit Model checking MAnager), which is a practical realization of the concept, and discuss practical experience with the tool.

1 Introduction

Although model checking algorithms are automatic, 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 [5] identified 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 [5]. 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 [2]. 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). This approach, however, involves

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

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) [2,5,8]. This approach, however, cannot be efficiently performed manually. We need to automate the verification process.

With the aim of automating the verification process, we elaborate on a general concept of a verification manager [6] and we provide its concrete realization for the domain of explicit model checking. We also describe experience with the tool and discuss problematic issues concerning fair evaluation.

The most related work is by Holzmann et al.: a tool for automated execution of verification runs for several model parameters and correctness properties using one fixed verification technique [3] and 'swarm verification' based on parallel execution of many different techniques [2]; 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).

This paper describes the main ideas of our approach and our tool EMMA. More details are in the technical report [7] (including a more detailed discussion of related work).

2 Concept and Implementation

Verification manager is a tool which automates the verification process (see Fig. 1). 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. Long-term log is used to store all input problems and verification results. It can be used for evaluation of strategies and for their improvement.

As a proof of concept we introduce a prototype of the verification manager for the domain of explicit model checking – Explicit Model checking MAnager (EMMA). The tool is publicly available on the web page:

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

EMMA is based on the Distributed Verification Environment (DiVinE) [1]. All used verification techniques are implemented in C++ with the use of DiVinE library. 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. Other techniques available in DiVinE can be easily incorporated.

The manager itself is implemented in Java. Currently, the manager supports as the underlying hardware a network of workstations connected by Ethernet. Communication is based on SSH and stream socket.

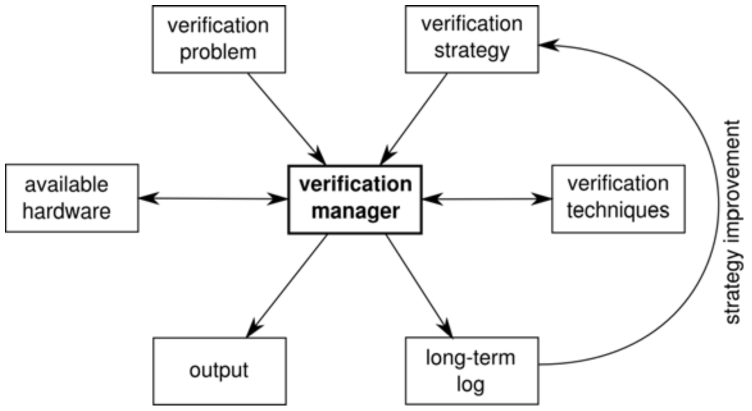

Fig. 1. Verification manager — context

We can view the manager as a tool for performing a search in a 'meta state space' of verification techniques and their parameters [6]. To perform this meta-search we need some heuristic – that is our verification strategy. There are several possible approaches to realization of a strategy (see [7]). We use the following: we fix a basic skeleton of the strategy 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.

In the implementation the strategy description is given in the XML format. For the first evaluation we use a simple priority-based strategies. For each technique we specify priority, timeout, and parameter values; techniques are executed according to their priorities.

EMMA provides visualizations of executions (Fig. 2). These visualizations can be used for better understanding of the tool functionality and for improvement of strategies.

3 Experiences

The first experience is that 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.

We also performed comparison of different strategies by running EMMA over models from BEEM [4] (probably the largest collection of models for explicit model checkers). We found that results depend very much on selection of input problems and that it is very difficult to give a fair evaluation. When we use mainly models without errors, strategies which focus on verification are more successful than strategies tuned for finding errors (Fig. 2, model Szymanski). When we use models with easy-to-find errors, there are negligible differences

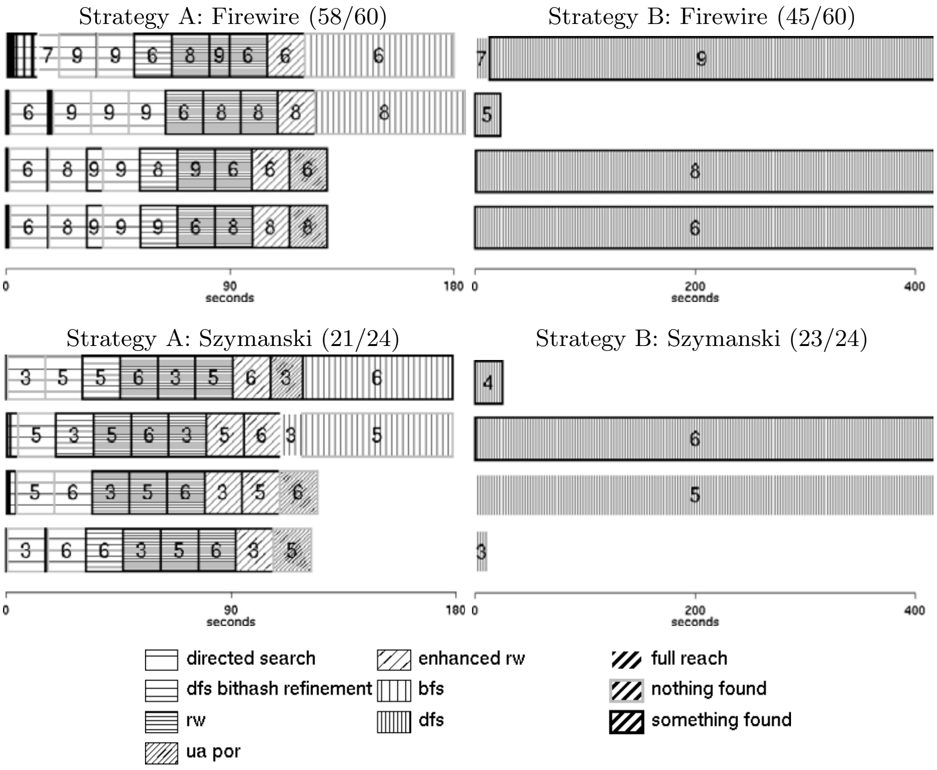

Fig. 2. Illustration of EMMA executions on 4 workstations for two models and two strategies. Each line corresponds to one workstation; numbers in boxes are identifications of model instances. The ratio X/Y means the number of decided properties (X) to number of all properties to be verified (Y). For color version see [7] or tool web page.

among strategies and we can be tempted to conclude that the choice of strategy does not matter. When we use models with hard-to-find errors, there are significant differences among strategies (Fig. 2, model Firewire); the success of individual strategies is, however, dependent very much on a choice of particular models and errors. 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".

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 (with user's content). Data collected in this way can be latter used for evaluation.

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 most of properties, 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 and 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 [8].

Thus even with the use of a manager, we do not have a single universal approach. 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. Nevertheless, the usage of a model checker becomes much more simple, since it suffices to use (and understand) just few strategies, which can be constructed by an expert user specifically for a given application domain of interest.

References

1. Barnat, J., Brim, L., Černá, I., Moravec, P., Rockai, P., Šimeček, P.: DiVinE - a tool for distributed verification. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 278–281. Springer, Heidelberg (2006), `http://anna.fi.muni.cz/divine`
2. Holzmann, G.J., Joshi, R., Groce, A.: Tackling large verification problems with the swarm tool. In: Havelund, K., Majumdar, R., Palsberg, J. (eds.) SPIN 2008. LNCS, vol. 5156, pp. 134–143. Springer, Heidelberg (2008)
3. Holzmann, G.J., Smith, M.H.: Automating software feature verification. Bell Labs Technical Journal 5(2), 72–87 (2000)
4. Pelánek, R.: BEEM: Benchmarks for explicit model checkers. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 263–267. Springer, Heidelberg (2007)
5. Pelánek, R.: Fighting state space explosion: Review and evaluation. In: Proc. of Formal Methods for Industrial Critical Systems, FMICS 2008 (2008) (to appear)
6. Pelánek, R.: Model classifications and automated verification. In: Leue, S., Merino, P. (eds.) FMICS 2007. LNCS, vol. 4916, pp. 149–163. Springer, Heidelberg (2008)
7. Pelánek, R., Rosecký, V.: Verification manager: Automating the verification process. Technical Report FIMU-RS-2009-02, Masaryk University Brno (2009)
8. Pelánek, R., Rosecký, V., Moravec, P.: Complementarity of error detection techniques. In: Proc. of Parallel and Distributed Methods in verifiCation, PDMC (2008)