# Model Classifications and Automated Verification

Radek Pelánek *

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

**Abstract.** Due to the significant progress in automated verification, there are often several techniques for a particular verification problem. In many circumstances different techniques are complementary — each technique works well for different type of input instances. Unfortunately, it is not clear how to choose an appropriate technique for a specific instance of a problem. In this work we argue that this problem, selection of a technique and tuning its parameter values, should be considered as a standalone problem (a verification meta-search). We propose several classifications of models of asynchronous system and discuss applications of these classifications in the context of explicit finite state model checking.

## 1 Introduction

One of the main goals of computer aided formal methods is automated verification of computer systems. In recent years, very good progress has been achieved in automating specific verification problems. However, even automated verification techniques like model checking are far from being a push-button technology. With current verification techniques many realistic systems can be automatically verified, but only if applied to the right level of abstraction of a system and if suitable verification techniques are used and right parameter values are selected.

The first problem is addressed by automated abstraction refinement techniques and received lot of attention recently (e.g., [1, 9]). The second problem, however, did not receive much attention so far and there are only few works in this direction. Ruys and Brinksma [38] describe methodology for model checking 'in the large'. Sahoo et al. [39] use sampling of the state space to decide which BDD based reachability technique is the best for a given model. Mony et al. [29] use expert system for automating proof strategies. Eytani et al. [11] give a high-level proposal to use an 'observation database' for sharing relevant information among different verification techniques.

Automation of the verification process is necessary for practical applicability of formal verification. Any self-respecting verification tool has a large number of options and parameters, which can significantly influence the complexity of verification. In order to verify any reasonable system, it is necessary to set these

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

parameters properly. This can be done only by an expert user and it requires lot of time. We believe that the research focus should not be only on the development of new automated techniques, but also on an automated selection of an existing technique.

### 1.1 Verification Meta-Search

So far most of the research in automated verification has been focused on questions of the verification problem: given a system $S$ and a property (or specification) $\varphi$, determine whether $S$ satisfies $\varphi$. This is a *search* problem — an algorithm searches for an incorrect behaviour or for a proof. Research has been focused on solving the problem for different formalisms and optimizing it for the most useful ones.

We believe that it is worthwhile to consider the following problem as well: given a system and a property, find a technique $T$ and parameter values $p$ such that $T(p)$ can provide answer to the verification problem. This can be viewed as a *verification meta-search problem*. Let an entity responsible for the verification meta-search be called a *verification manager*. The manager has the following tasks:

1. Decide which approach to the verification should be used, e.g., symbolic versus explicit approach, whether to use on-the-fly verification or whether to generate the full state space and then perform verification, etc.
2. Combine relevant information obtained from different techniques, see e.g., Synergy approach [18] for combination of over-approximation and testing.
3. Choose among different techniques (implementations) for a particular verification task and set parameters of a chosen technique.

In this work we focus mainly on the third task of the manager. To give a practical example of this task, we provide two specific cases. Firstly, consider on-the-fly memory reduction techniques — the goal of these techniques is to reduce memory requirements of exhaustive finite state verification. Examples of such techniques are partial order reduction, symmetry reduction, state compression, and caching. Each of these techniques has its merits and disadvantages, none of them is universal (see [32] for an evaluation). Moreover, most of these techniques have parameters which can tune a time/memory trade-off. Secondly, consider algorithms for accepting cycle detection on networks of workstations, which are used for LTL verification of large finite state models. Currently there are at least five different algorithms, each with specific disadvantages and parameters [2].

At the moment the verification manager is usually a human expert. Expert can perform this role rather well, however such 'implementation' of the verification manager is far from automated. There has been attempts to facilitate the human involvement, e.g., by using special purpose scripting languages [25], but such an approach automatizes only stereotypical steps during the verification, not decisions.

The problem can be addressed by an expert system, which perform the meta-search with the use of a set of rules provided by experts. Example of such rules may be:

- If the model is a mutual exclusion protocol then use explicit model checking with partial order reduction.
- If state vector is longer then 30 bytes, then use state compression.
- If the state space is expected to contain a large strongly connected component, then use cycle detection algorithm $X$ else use cycle detection algorithm $Y$.

Another option is to employ an adaptive learning system which remembers characteristics of verification tasks and their results and learns from its own experiences.

### 1.2 The Need for Classifications

At this moment, it is not clear what rules should the verification manager use. But more fundamentally, it is not even clear what criteria should be used in rules. Whatever is the realization of the manager, the manager needs to make decision based on some information about an input model. The information used for this decision should be carefully chosen:

- If the information was too coarse, the manager would not be able to choose among potentially suitable techniques.
- If the information was too detailed, it would be very hard for the manager to apply its expertise and experiences.

We believe that an appropriate approach is to develop several categorical classifications of models and then use these classification for manager's decisions. To be applicable, it must be possible to determine suitable techniques for individual classes of the classification. Moreover, it must be possible to determine a class of a given model without much effort — either automatically by a fast algorithm or easily by user judgement.

In this work we focus on asynchronous concurrent systems, for the evaluation we use models from the BEEM set [35]. For asynchronous concurrent systems one of the most suitable verification techniques is explicit state space exploration. Therefore, we focus not only on analysis of a model structure, but also on the analysis of state spaces. In this work we propose classifications based on a model structure (communication mode, process similarity, application domain) and also classifications based on properties of state spaces (structure of strongly connected components, shape and local structure of a state space). We study relation of these classifications and discuss how they can be useful for the selection of suitable techniques and parameters (i.e., for guiding the meta-search).

The restriction to explicit model checking techniques limits the applicability of our contribution. Note, however, that even for this restricted area, there is a

very large number of techniques and optimizations (there are at least 80 research papers dealing with explicit model checking techniques, see [34] for a list). Moreover, the goal of this work is not to present the ultimate model classification, but rather to pinpoint a direction, which can be fruitful.

## 2 Background

**Used models**. For the evaluation of properties of practically used models we employ models from the benchmark set BEEM [35]. This set contains large number of models of asynchronous systems. The set contains classical models studied in academic literature as well as realistic case studies. Models are provided in a low-level specification language (communicating finite state machines) and in Promela [20]. For our study we have used 115 instances obtained by instantiation of 57 principally different models.

**State spaces**. For each instance we have generated its state space. We view a state space as a simple directed graph $G = (V, E, v_0)$ with a set of vertices $V$, a set of directed edges $E \subseteq V \times V$, and a distinguished initial vertex $v_0$. Vertices are states of the model, edges represent valid transitions between states. For our purposes we ignore any labeling of states or edges. We are concerned only with the reachable part of the state space.

Let us define several parameters of state spaces. We use these parameters for classifications. We have also studied other parameters (particularly those reported in [31]), but these parameters do not lead to interesting classification.

An *average degree* of $G$ is the ratio $|E|/|V|$. A *strongly connected component* (SCC) of $G$ is a maximal set of states $C \subseteq V$ such that for each $u, v \in C$, the vertex $v$ is reachable from $u$ and vice versa. Let us consider the breadth-first search (BFS) from the initial vertex $v_0$. A *level* of the BFS with an index $k$ is a set of states with distance from $v_0$ equal to $k$. The *BFS height* is the largest index of a non-empty level, *BFS width* is the maximal size of a BFS level. An edge $(u, v)$ is a *back level edge* if $v$ belongs to a level with a lower or the same index as $u$. The *length* of a back level edge is the difference between the indices of the two levels.

**Reduction techniques**. In the following, we often mention two semantics based reduction techniques. Under the notion *partial order reduction* (POR) we consider all techniques which aim at reducing the number of explored states by reducing the amount of interleaving in the model, i.e., we denote by this notion not just the classic partial order reduction technique [16], but also other related techniques, e.g., confluence reduction [5], simultaneous reachability analysis [30], transition compression [24]. *Symmetry reduction* techniques aim at reducing the number of explored states by considering symmetric states as equivalent, see e.g. [22].

## 3 State Space Classifications

We consider three classifications based on properties of state spaces. Two of them are based on "global" properties (structure of SCC and shape), one is based on "local" features of state spaces.

### 3.1 Structure of SCC Components

There is an interesting dichotomy with respect to structure of strongly connected components, particularly concerning the size of the largest SCC (see Fig. 1). A state space either contains one large SCC, which includes nearly all states, or there are only small SCCs. Based on this observation, we propose the following classification:

**A type** (acyclic): a state space is acyclic, i.e., it contains only trivial components with one state,

**S type** (small components): a state space is not acyclic, but contains only small components; more precisely we consider a state space to be of this type if the size of the largest component is smaller then 50% states.

**B type** (big component): a state space contain one large component, most states are in this component.

In order to apply the classification for automated verification, we need to be able to detect the class of a model without searching its full state space. This can be done by random walk exploration [36], for example by the following simple method based on detection of cycles by random walk. We run 100 independent random walks through the state space. Each random walk starts at the initial state and is limited to at most 500 steps. During the walk we store visited states, i.e., path through the state space. If a state is revisited then a cycle is detected and its length can be easily computed. At the end, we return the length of the longest detected cycle. Fig. 1 shows results of this method. For the class **A** the longest detected cycle is, of course, always 0. For the class **S** the longest detected cycle is usually between 10 and 35, for the class **B** it is usually above 30. This illustrates that even such a simple method can be used to quickly classify state spaces with a reasonable precision.

What are possible applications of this classification? For the **A** type it is possible to use specialized algorithms, e.g., dynamic partial order reduction [12] or bisimulation based reduction [33, p. 43-47]. The sweep line method [8] deletes from memory states, which will never be visited again. This method is useful only for models with state spaces of the type **A** or **S**.

The performance of cycle detection algorithms[1], which are used for LTL verification, is often dependent on the SCC structure. For example a distributed

[1] Note that cycle detection algorithm are usually executed on the product graph with a formula [42] and not on the state space itself. However, our measurements indicate that the structure of product graphs is very similar to structure of plain state spaces. The measurements were performed on product graphs included in the BEEM [35] set.

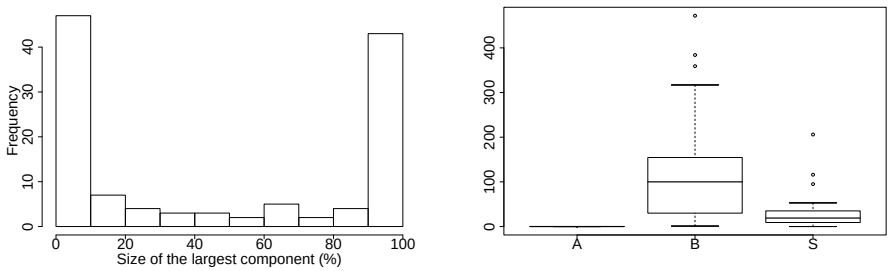


**Fig. 1.** The first graph shows the histogram of sizes of the largest SCC component in a state space. The second graph shows the longest detected cycle using random walk; results are grouped according to class and presented using a boxplot method (lines denote minimum, 25th quartile, median, 75th quartile and maximum, circles are outliers).

algorithm based on localization of cycles is suitable only for **S** type state spaces; depth-first search based algorithm [21] can also be reasonably applied only for **S** type state spaces, because for **B** type state spaces it tends to produce very long counterexamples, which are not practical. On the other hand, (explicit) one-way-catch-them-young algorithm [6] has complexity $O(nh)$, where $h$ is height of the SCC quotient graph, i.e., this algorithm is more suitable for **B** type state spaces. Similarly, the classification can be employed for verification of branching time logics (e.g., the algorithm in [7] does not work well for state spaces consisting of one SCC).

### 3.2 Shape of the State Space

We have found that several global state space parameters are to certain extent related: average degree, BFS height and width, number and length of back level edges. In this case the division into classes is not so clear as in the previous case. Nevertheless, it is possible to identify two main classes with respect to these parameters[2]:

**H type** (high): small average degree, large BFS height, small BFS width, few long back level edges.

**W type** (wide): large average degree, small BFS height, large BFS width, many short back level edges.

This classification can be approximated using an initial sample of the BFS search. The classification can be used in similar way as the previous one. Sweep

[2] Note that this classification is not complete partition of all possible state spaces. The remaining classes, however, do not occur in practice. The same holds for as several other classifications which we introduce later.

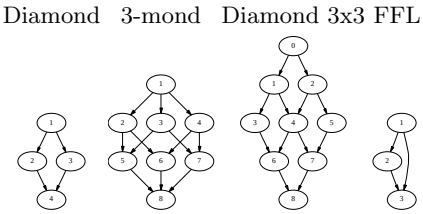


**Fig. 2.** Illustrations of motifs

line [8] and caching based on transition locality [37] work well only for state spaces with short back level edges, i.e., these techniques are suitable only for **W** type state spaces. On the other hand, the complexity of BFS-based distributed cycle detection algorithm [3] is proportional to number of back level edges, i.e., this algorithm works well only on **H** type state spaces.

For many techniques the **H**/**W** classification can be used to set parameters appropriately: algorithms which exploit magnetic disk often work with individual BFS levels [41]; random walk search [36] and bounded search [23] need to estimate the height of the state space; techniques using stratified caching [15] and selective storing of states [4] could also take the shape of the state space into account.

### 3.3 Local Structure

Now we turn to a local structure of state spaces, particularly to typical subgraphs. Recently, so called 'network motifs' [28, 27] were intensively studied in complex networks. Motifs are studied mainly in biological networks and are used to explain functions of network's components (e.g., function of individual proteins) and to study evolution of networks.

We have systematically studied motifs in state spaces. We have found the following motifs to be of specific interest either for abundant presence or for total absence in many state spaces: *diamonds* (we have studied several variations of diamond-like structures, see Fig. 2), which are well known to be present in state spaces of asynchronous concurrent systems due to the interleaving semantics; *chains* of states with just one successor, we have measured occurrences of chains of length 3, 4, 5; *short cycles* of lengths 2, 3, 4, 5, which are not very common in most state spaces; and *feed forward loop* (see Fig. 2), which is a typical motif for networks derived from biological systems [28], in state spaces it is rather rare.

We have measured number of occurrences of these motifs and studied correlations of their occurrences. With respect to motifs we propose the following classes:

**D type** (diamond): a state space contains many diamonds, usually no short cycles and only few chains of feed forward loops,

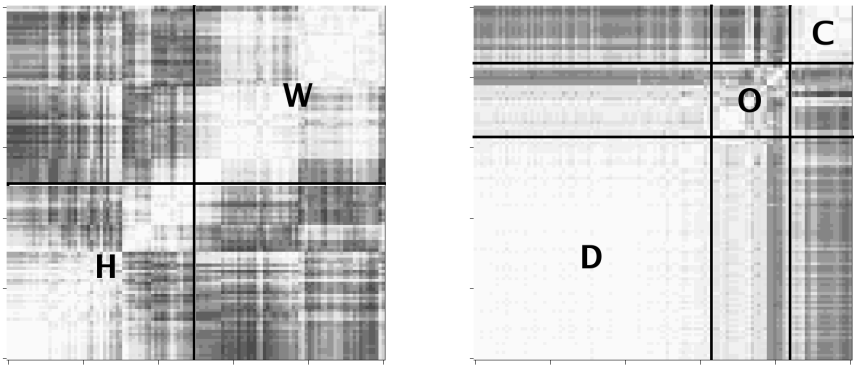


**Fig. 3.** Correlation matrix displaying correlation of 116 state spaces. Light color means positive correlation, dark color means negative correlation. The first matrix shows correlation with respect to average degree, BFS height and width, number and length of back level edges (all parameters are normalized). The second matrix shows correlations with respect to presence of studied motifs.

**C type** (chain): a state space contains many chains, very few diamonds or short cycles,

**O type** (other): a state space either contains short cycles and/or feed forward loops, chains are nearly absent, diamonds may be present, but they are not dominant.

Identification of these classes can be performed by exploration of a small sample of the state space. This classification can be used to choose among memory reduction techniques. For **D** type state spaces it is reasonable to try to employ POR, whereas for **C** type state spaces this reduction is unlikely to yield significant improvement. On the other hand, for **C** type state spaces good memory reduction can be obtained by selective storing of states [4]. The classification can be also used for tuning parameter values, particularly for technique which employ local search, e.g., random walk enhancements [36, 40], sibling caching and children lookahead in distributed computation [26], heuristic search.

### 3.4 Relation among State Space Classifications

Table 1. presents number of models in different combinations of classes. Specific numbers presented in the table are influenced by the selection of used models. Nevertheless, it is clear that presented classifications are rather orthogonal, there is just slight relation between the shape and the local structure.

# 4 Model Classifications

Now we turn to classifications based directly on a model. At first, we study classifications according to model structure, which are relevant particularly with respect to reduction techniques based on semantics (e.g., partial order reduction, symmetry reduction). Secondly, we study models from different application domains and show that each application domain has its characteristics with respect to presented classifications.

## 4.1 Model Structure

Classifications based on structure of a model are to some extent dependent on a specific syntax of the specification language. There are many specification languages and individual specification languages significantly differ on syntactical level. However, if we restrict our attention to models of asynchronous systems, we find that most specification languages share the following features:

- a model is comprised of a set of processes,
- a process can be viewed as a finite state machine extended with variables,
- processes communicate either via channels or via globally shared variables.

We discuss several possible classifications based on these basic features. Categorization of a model according to these classifications can be determined automatically by static analysis of a model. This issue is dependent on a particular specification language and it is rather straightforward, therefore, we do not discuss it in detail.

**Communication Mode** With respect to communication we can study the predominant mean of communication (shared variables or channels) and the communication structure (ring, line, clique, star). It turns out that these two features are coupled, i.e., with respect to communication we can consider the following main classes:

**DV type** (dense, variable): processes communicate via shared variables, the communication structure is dense, i.e., every process can communicate with (nearly) every other process,

**SC type** (sparse, channel): processes communicate via (buffered) channels, the communication structure is rather sparse, e.g., ring, star, or tree.

**N type** (none): no communication, i.e., the model is comprised of just one process.

This classification is related particularly to partial order reduction techniques. The classification is completely orthogonal to state space classifications (see Table 1.).

**Table 1.** Relations among classifications. For each combination of classes we state the number of models in the combination. In total there are 115 classified models, all models are from the BEEM set [35]. Reported state space classifications are based on traversal of the full state space.

State space classifications

| | all | **H** | **W** | **D** | **O** | **C** |
|---|---|---|---|---|---|---|
| all | 115 | 58 | 57 | 75 | 23 | 17 |
| **A** | 24 | 10 | 14 | 19 | 3 | 2 |
| **S** | 37 | 18 | 19 | 21 | 9 | 7 |
| **B** | 54 | 30 | 24 | 35 | 11 | 8 |
| **H** | 58 | | | 43 | 3 | 12 |
| **W** | 57 | | | 32 | 20 | 5 |

Model classifications

| | all | **S2** | **S1** | **S0** |
|---|---|---|---|---|
| all | 115 | 41 | 39 | 35 |
| **DV** | 44 | 31 | 9 | 4 |
| **SC** | 61 | 10 | 30 | 21 |
| **N** | 10 | 0 | 0 | 10 |

State space versus model classifications

| | all | **A** | **S** | **B** | **H** | **W** | **D** | **O** | **C** |
|---|---|---|---|---|---|---|---|---|---|
| all | 115 | 24 | 37 | 54 | 58 | 57 | 75 | 23 | 17 |
| **S2** | 41 | 10 | 9 | 22 | 20 | 21 | 37 | 3 | 1 |
| **S1** | 39 | 8 | 17 | 14 | 20 | 19 | 20 | 10 | 9 |
| **S0** | 35 | 6 | 11 | 18 | 18 | 17 | 18 | 10 | 7 |
| **DV** | 44 | 8 | 12 | 24 | 20 | 24 | 33 | 6 | 5 |
| **SC** | 61 | 12 | 22 | 27 | 36 | 25 | 40 | 10 | 11 |
| **N** | 10 | 4 | 3 | 3 | 2 | 8 | 2 | 7 | 1 |

**Process Similarity** A common feature in models of asynchronous systems is the occurrence of several similar processes (e.g., several participants in a mutual exclusion protocol, several users of an elevator, several identical nodes in a communication protocol). By 'similarity' we mean that processes are generated from one template by different instantiations of some parameters, i.e., we do not consider symmetry in any formal sense (cf. [22]). With respect to similarity, a reasonable classification is the following:

**S2 type** All processes are similar.
**S1 type** There exists some similar processes, but not all of them.
**S0 type** There is no similarity among processes.

This classification is clearly related to symmetry reduction. It can also be employed for state compression [19]. This classification is again orthogonal to state space classifications and only slightly correlated with the communication mode classification (**S1** is related to **SC**, **S2** is related to **DV**), for details see Table 1.

**Other** We briefly mention several other possible classifications and their applications:

**Table 2.** Relation of state space classification and model type

| | | SCC struct. | | | shape | | local struct | | | comm. | | | proc. sim. | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | all | **A** | **S** | **B** | **H** | **W** | **C** | **D** | **O** | **SC** | **DV** | **N** | **S2** | **S1** | **S0** |
| all | 115 | 24 | 37 | 57 | 58 | 57 | 17 | 75 | 23 | 61 | 44 | 10 | 41 | 39 | 35 |
| com. protocol | 24 | 0 | 10 | 14 | 15 | 9 | 5 | 18 | 1 | 24 | 0 | 0 | 0 | 7 | 17 |
| controller | 17 | 1 | 7 | 9 | 15 | 2 | 3 | 12 | 2 | 12 | 5 | 0 | 0 | 13 | 4 |
| leader el. | 12 | 12 | 0 | 0 | 6 | 6 | 0 | 12 | 0 | 8 | 4 | 0 | 9 | 3 | 0 |
| mutex | 28 | 0 | 8 | 20 | 13 | 15 | 0 | 25 | 3 | 2 | 26 | 0 | 28 | 0 | 0 |
| sched. | 18 | 9 | 3 | 6 | 4 | 14 | 2 | 5 | 11 | 2 | 6 | 10 | 1 | 5 | 12 |
| other | 16 | 2 | 9 | 5 | 5 | 11 | 7 | 3 | 6 | 13 | 3 | 0 | 3 | 11 | 2 |

- Data/Control intensity of a model (Is a model concerned with data manipulation and arithmetic?); related to abstraction techniques [17, 1, 9], which focus on reducing the data part of the model.
- Tightly/Loosely coupled processes (What is the proportion of interprocess and intraprocess computation?); important for thread-modular techniques [13].
- Length of a state vector; relevant particularly for state compression techniques [19].

### 4.2 Application Domain

Finally, we discuss application domains of asynchronous concurrent systems. Table 2. presents relations among application domains and previously discussed classifications. The table demonstrates that models from each application domain have specific characteristics. Knowledge of these characteristics can be helpful for the development of (commercial) verification tools specialized for a particular application domain. Beside that, characteristics of models can be used to develop templates and design patterns [14, 10], which can facilitate the modeling process.

**Mutual exclusion algorithms** The goal of a mutual exclusion algorithm is to ensure an exclusive access to a shared resource. Models of these algorithms usually consist of several nearly identical processes which communicate via shared variables; communication structure is either clique or ring; individual processes are usually rather simple. State vectors are relatively short; state space usually contains one big strongly connected component, with many diamonds. POR and symmetry reduction may be useful, but careful modeling may be necessary in order to make them applicable.

**Communication protocols** The goal of communication protocols is to ensure communication over an unreliable medium. The core of a model is a sender process, a receiver process, and a bus/medium; the communication structure is therefore usually linear (or simple tree). Processes communicate by handshake; shared variables are not used. Processes are not similar, sender/receiver processes can be rather complicated. State vectors are rather long; state space is not acyclic, it is rather high, often with many diamonds. POR is usually applicable.

**Leader election algorithms** The goal of leader election algorithms is to choose a unique leader from a set of nodes. Models consist of a set of (nearly) identical processes, which are rather simple. Processes are connected in a ring, tree, or arbitrary graph; communication is via (buffered) channels. State spaces are acyclic with diamonds. POR, symmetry reduction, and specialized techniques for acyclic state spaces [12] may be applicable.

**Controllers** Models of controllers usually have centralized architecture: a controller process communicates with processes representing individual parts of the system. The controller process is rather complex, other processes may be simple. The communication can be both by shared variables and handshake. State vectors are rather long; state spaces are high, usually with diamonds. Due to the centralized architecture semantics-based reduction techniques are hard to apply.

**Scheduling, planning, puzzles** Planning and scheduling problems and puzzles are not the main application domain of explicit model checkers. Nevertheless, there are good reasons to consider them together with asynchronous systems (similar modeling formalism, research in combinations of model checking and artificial intelligence techniques). Models often consist of just one process. Planning, scheduling problems have wide state space without prevalence of diamonds or chains. State spaces are often acyclic.

**Other application domains** Similar characterizations can be provided for many other application domains. Examples of other often studied application domains are cache coherence protocols, device drivers or data containers.

## 5 Conclusions and Future Work

We argue that it is important not just to develop (narrowly focused) techniques for automated verification, but also to automatize the verification *process*, which is currently usually performed by an expert user. To this end, it is desirable to have classifications of models. We propose such classifications for asynchronous systems; these classifications are based on properties of state spaces and on structure of models. We also discuss examples of applications of these classifications; particularly the following two types of application:

- indication of suitable techniques to use for verification of the given model,
- setting suitable parameter values for the verification.

In the paper we provide several specific examples of such application; we note that these are just examples, not a full list of possible applications. The presented classifications are also not meant to be the final classifications of asynchronous systems. We suppose that further research will expose the need for other classification or for the refinement of presented classification. Moreover, for other application domains and verification techniques (e.g., synchronous systems, symbolic techniques, bounded model checking) it will be probably necessary to develop completely new classifications. Nevertheless, we believe that our approach can provide valuable inspiration even for this direction.

This work is a part of a long term endeavour. We are continuously developing the benchmark set BEEM [35]. Using the presented classification, we are working on experimental evaluation of the relation of classes and performance of different techniques. We are also developing techniques for estimation of state space parameters from samples of a state spaces, such estimations (e.g., the size of a state space), can be useful for guiding the verification meta-search. Finally, the long term goal is to develop an automated 'verification manager', which would be able to learn from experience.

## References


1. 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.
2. J. Barnat, L. Brim, and I Cerná. Cluster-based ltl model checking of large systems. In *Proc. of Formal Methods for Components and Objects (FMCO'05), Revised Lectures*, volume 4111 of *LNCS*, pages 259–279. Springer, 2006.
3. J. Barnat, L. Brim, and J. Chaloupka. Parallel breadth-first search LTL model-checking. In *Proc. Automated Software Engineering (ASE 2003)*, pages 106–115. IEEE Computer Society, 2003.
4. G. Behrmann, K. G. Larsen, and R. Pelánek. To store or not to store. In *Proc. of Computer Aided Verification (CAV'03)*, volume 2725 of *LNCS*. Springer, 2003.
5. S. Blom and J. van de Pol. State space reduction by proving confluence. In *Proc. of Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 596–609, 2002.
6. I. Černá and R. Pelánek. Distributed explicit fair cycle detection. In *Proc. SPIN workshop*, volume 2648 of *LNCS*, pages 49–73. Springer, 2003.
7. A. Cheng, S. Christensen, and K. Mortensen. Model checking coloured petri nets exploiting strongly connected components. Technical Report DAIMI PB – 519, Computer Science Department, University of Aarhus, 1997.
8. 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.
9. 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.
10. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In *Proc. Workshop on Formal Methods in Software Practice*, pages 7–15. ACM Press, 1998.
11. 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.
12. C. Flanagan and P. Godefroid. Dynamic partial-order reduction for model checking software. In *Proc. of Principles of programming languages (POPL'05)*, pages 110–121. ACM Press, 2005.
13. C. Flanagan and S. Qadeer. Thread-modular model checking. In *Proc. of SPIN Workshop*, volume 2648 of *LNCS*, pages 213–224, 2003.
14. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

15. J. Geldenhuys. State caching reconsidered. In *Proc. of SPIN Workshop*, volume 2989 of *LNCS*, pages 23–39. Springer, 2004.
16. P. Godefroid. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032 of *LNCS*. Springer-Verlag, 1996.
17. S. Graf and H. Saidi. Construction of abstract state graphs with pvs. In *Proc. of Computer Aided Verification (CAV '97)*, pages 72–83, London, UK, 1997. Springer-Verlag.
18. B. S. Gulavani, T. A. Henzinger, Y. Kannan, A. V. Nori, and S. K. Rajamani. Synergy: a new algorithm for property checking. In *Proc. of Foundations of software engineering*, pages 117–127. ACM Press, 2006.
19. G. J. Holzmann. State compression in SPIN: Recursive indexing and compression training runs. In *Proc. of SPIN Workshop*, 1997.
20. G. J. Holzmann. *The Spin Model Checker, Primer and Reference Manual*. Addison-Wesley, Reading, Massachusetts, 2003.
21. G. J. Holzmann, D. Peled, and M. Yannakakis. On nested depth first search. In *Proc. SPIN Workshop*, pages 23–32. American Mathematical Society, 1996.
22. C. N. Ip and D. L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1–2):41–75, 1996.
23. P. Krčál. Distributed explicit bounded ltl model checking. In *Proc. of Parallel and Distributed Methods in verifiCation (PDMC'03)*, volume 89 of *ENTCS*. Elsevier, 2003.
24. R. P. Kurshan, V. Levin, and Hüsnü Yenigün. Compressing transitions for model checking. In *Proc. of Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 569–581, 2002.
25. F. Lang. Compositional verification using svl scripts. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 465–469. Springer, 2002.
26. 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.
27. R. Milo, S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, and U. Alon. Superfamilies of evolved and designed networks. *Science*, 303(5663):1538–1542, March 2004.
28. R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, October 2002.
29. 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.
30. K. Ozdemir and H. Ural. Protocol validation by simultaneous reachability analysis. *Computer Communications*, 20:772–788, 1997.
31. R. Pelánek. Typical structural properties of state spaces. In *Proc. of SPIN Workshop*, volume 2989 of *LNCS*, pages 5–22. Springer, 2004.
32. 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.
33. R. Pelánek. *Reduction and Abstraction Techniques for Model Checking*. PhD thesis, Faculty of Informatics, Masaryk University, Brno, 2006.

34. R. Pelánek. Web portal for benchmarking explicit model checkers. Technical Report FIMU-RS-2006-03, Masaryk University Brno, 2006. Available at `http://anna.fi.muni.cz/models/`.
35. R. Pelánek. Beem: Benchmarks for explicit model checkers. In *Proc. of SPIN Workshop*, volume 4595 of *LNCS*, pages 263–267. Springer, 2007.
36. 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.
37. 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.
38. T. C. Ruys and E. Brinksma. Managing the verification trajectory. *International Journal on Software Tools for Technology Transfer (STTT)*, 4(2):246–259, 2003.
39. 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.
40. H. Sivaraj and G. Gopalakrishnan. Random walk based heuristic algorithms for distributed memory model checking. In *Proc. of Parallel and Distributed Model Checking (PDMC'03)*, volume 89 of *ENTCS*, 2003.
41. 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.
42. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In D. Kozen, editor, *Proc. of Logic in Computer Science (LICS '86)*, pages 332–344. IEEE Computer Society Press, 1986.