Study of the efficiency of model checking techniques using results of the MCC from 2015 To 2019

Fabrice Kordon, Lom Messan Hillah, Francis Hulin-Hubard, Loïg Jezequel, Emmanuel Paviot-Adet

International Journal of Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Study of the Efficiency of Model Checking Techniques Using Results of the MCC from 2015 To 2019

Fabrice Kordon · Lom Messan Hillah · Francis Hulin-Hubard
Loïg Jezequel · Emmanuel Paviot-Adet ·

Received: date / Accepted: date

Abstract In various scientific communities dealing with formal analysis, software competitions have emerged and contributed to fostering progress in state of the art and providing insight into the evolution of the involved technologies. The Model Checking Contest (MCC) is one of them; it focuses on asynchronous concurrent systems.

This paper reports what the organizers have observed over five editions of the MCC between 2015 and 2019. It shows the evolution of state-of-the-art model checking tools in performing large and difficult verification tasks by improving existing techniques or designing new and innovative (combinations of) techniques. This paper also shows the impact of such an event on the corresponding scientific community.

The authors thank the following organizations for having lent powerful computers to process all the experiments of the Model Checking Contest over the years: Rostock University, Sorbonne Université, Université de Genève, Université Paris Nanterre, University of Twente.

Fabrice Kordon
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Lom Messan Hillah
Université Paris Nanterre, F-92001, Nanterre, France, and
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Francis Hulin-Hubard
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Loïg Jezequel
Université de Nantes, LS2N, UMR CNRS 6004, F-44321 Nantes, France

Emmanuel Paviot-Adet
Université de Paris, F-75005 Paris, France, and
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Keywords Model Checking · Software Competition · Formal Verification · Concurrent Systems

1 Introduction

Concurrent asynchronous systems, in which several processes run simultaneously, are now widely spread. They range from smartphone applications to connected (soon autonomous) vehicles and affect our daily life. However, model checking the safety and security of such complex and interacting asynchronous systems remains a challenge. The scientific community is currently tackling this issue, both from theoretical and practical points of view – by developing and testing tools. Thus, from the theoretical and practical perspectives, community-driven contests allow the assessment of tools' readiness by evaluating them against substantial and acknowledged benchmarks.

The overall goal is to identify the theoretical approaches that are the most fruitful in practice when applied to various examples and to figure out approaches that best handle specific types of systems. Community-driven contests also favor the emergence of systematic, rigorous, and reproducible ways to assess the capabilities of verification tools on complex (realistic and synthesized) benchmarks.

Since 2011, the Model Checking Contest [51] (MCC) is such an event. It focuses on the model checking of asynchronous and concurrent systems. To model such systems, we use Petri Nets [36], and their standardized ISO/IEC 15909-2 XML-based representation [44] as the input format.

Other communities in the area of formal verification self-organized to establish similar events. For example, the Hardware Model Checking competition [16]

focuses on synchronous systems, while the MCC essentially considers asynchronous ones. The Rigorous Examination of Reactive Systems (RERS) [46] competition evaluates researchers' ability to harness the synergy of different tools and methodologies to tackle verification problems. RERS does not address tool performance as the MCC does. However, RERS also addresses asynchronous systems as well as the MCC. VerifyThis [30] focuses on verification problems encoded by experts in formal or programming languages of their choice, then checked with the help of a verification tool, generally over one day.

Similar to competitions like SAT [8,2], SMT [9], QBF [62] or CASC [69], the MCC emphasizes automatic tools. It aims at evaluating how tools' strategies can automatically cope with complex verification problems. These competitions have a rather similar process: researchers submit their tools that the competition organizers evaluate against a common benchmark on a set of powerful computers.

All these contests are themselves becoming studied objects. As an example, in February 2019, a Lorentz Workshop [55] was dedicated to this topic. Then, in April 2019, 16 verification competitions gathered at TOOLympics [13], a side event of the 25^{th} TACAS conference; this conference mainly focusses on verification tools and theories.

This paper reports the MCC organizers' empirical observations of the MCC's five editions from 2015 to 2019. Our objective is to analyze the data collected over these five years to perceive which techniques turn out to be better than others or, more interestingly, if some new techniques (or combination of techniques) have emerged.

The 2015 edition is the first studied edition because, in that year, we developed a standard way to represent, as a table (stored in a CSV file), the results of all the executions of the participating tools. These five editions gather about half a million tool executions on various specifications of hardware or software systems, communication protocols, biological processes, and many other systems.

According to acknowledged practices of empirical studies, our approach is similar to an exploratory case study [66]. This paper covers the main steps related to i) setting up objectives and research questions, ii) selecting the cases, iii) preparing the data, iv) collecting the data, v) analyzing the data, and vi) reporting our observations.

We are aware that, most often, competing tools have been tuned for the contest. Moreover, we also continuously tune the benchmark and verification tasks. Therefore, this paper does not make definitive claims. It instead reports observed trends and seeks new insights into the evolution of model checking techniques and tools. Thus, from the perspective of highlighting the progress of state-of-the-art techniques in model checking, we believe that our analysis reports valuable insights.

This paper is structured as follows. Section 2 defines the Model Checking Contest's main concepts. Section 3 presents the techniques tools have reported and that are discussed in later sections. Then, Section 4 shows how the reported techniques have evolved over the years, also focusing on the most efficient tools (i.e., those on the podiums) compared to all participating tools. Section 5 presents some key measures about the evolution of reported techniques over the years by considering the verification tasks' hardness. Section 6 evaluates the observed benefits when exploiting structural information (mainly hierarchy) about the processed specifications. Section 7 outlines the impact of the MCC on the design of tools and their techniques and on generating hard verification formulas, while Section 8 briefly outlines the impact of the MCC on its targeted community. Finally, Section 9 ends this paper with concluding remarks.

2 Concepts used in the MCC

The MCC is an annual competition of software tools for model checking concurrent systems. Tools process a benchmark of increasing size – with new models gathered from the whole community added each year – and deal with various examinations, *i.e.*, different sets of problems that they are requested to solve on a given specification.

This section briefly provides general background about how this event operates. We describe the examinations and the benchmarks, composed of models and formulas. More details are provided in [50,51].

2.1 Examinations, runs, and values

An examination is a set of one or several verification tasks required from competing tools. A verification task is a query requiring a single answer such as, for example, "what is the maximum number of tokens in a place?", or "does the model have a deadlock?", or a single CTL formula. A query may thus be a generic question, or a formula.

Tools carry out examinations through *runs* where they compute and return *values*. A run is a single execution of a tool on a model for a given examination. A

value is the result of a query which is a generic question or a formula.

In 2019, the MCC was proposing six examinations:

- *State space generation*, in which tools must generate the state space of the system and provide four numeric values: the number of states, the number of transitions, the maximum number of tokens per marking, and the maximum number of tokens in a place;
- *Deadlock detection*[1], where tools are requested to provide a Boolean value stating if the system has a deadlock or not;
- *Upper bounds computation*[2], where tools are requested to provide 16 numeric values, each one stating the maximum number of tokens in a place or a set of places determined by a formula;
- *Reachability*, in which tools have to provide 16 Boolean values, each corresponding to a reachability formula or invariant;
- *CTL*, in which tools have to provide 16 Boolean values, each corresponding to a computation tree logic formula;
- *LTL*, in which tools have to provide 16 Boolean values, each corresponding to a linear-time temporal logic formula.

In 2019, a total of 91 619 runs (6 126 runs per tool) were performed on 5 clusters or multi-core machines to compute all the results, accumulating about 4 years, 11 months, and 17 days of CPU.

2.2 Benchmark

All the tools participating in a given edition of the MCC are evaluated on the same benchmark suite, which the community incrementally updates every year. The benchmark suite consists of *models* and *formulas*.

The models. Each *model* corresponds to a particular academic or industrial problem. Models come from various authors using different modeling strategies, and dealing with very different problems such as distributed algorithms, hardware protocols, mutual exclusion algorithms, biological processes, and many other types of systems. We receive between 4 and 15 new models each year, which gives an average of 10 models per year over the period from 2015 to 2019.

The models may be parameterized by one or more scaling parameters representing quantities, such as the number of agents in a concurrent system, the number of messages exchanged, etc. The parameterized models yield as many *instances* (so far, between 2 and 25) as there are different combinations of parameter values. The non-parameterized models have only a single associated instance.

For the 2019 edition of the MCC, 93 models were generating 1018 instances. There were 193 colored Petri nets containing high-level information (types in tokens), and 825 Place/Transition (P/T) nets. There is actually one run per instance of a model for a given examination.

Each model author reports a set of fifteen structural (e.g., simple free choice, nested units) and four behavioral properties (e.g., safe, reversible) in the model submission form. These properties take their value from the set $\{true, false, unknown\}$. We double-check these properties for each model instance using caesar.bdd[3], with a time-out of a few minutes. In case of disagreement with the model author over the value of a property, or uncertainty on both sides, the property is set to unknown. We then extract these properties and their value in an XML file that competing tools may (carefully) reuse to their advantage.

Each year, the models from the previous years become the "known" ones, along with the "new" models introduced by the community for the current year. The new models award a higher score per instances processed than known models.

The formulas. While state space generation and deadlock detection are generic questions, we need to produce formulas for the other examinations. So, every year, we propose newly generated formulas for every model in the benchmark. We produce "fresh" formulas for known models every year, thus leading to different results. In 2019, we produced 65 344 formulas in total for upper bounds, reachability, CTL, and LTL.

We try to produce the most "intelligent" possible formulas using random generators and a filtering system that selects complex ones. Section 7.3 discusses the main aspects of this challenge. A more precise description of the formula generation process is provided in [50].

[1] From 2015 to 2018, *Deadlock detection* was part of the *reachability* examination. It moved into a dedicated category that requested only deadlock detection in 2019, but groups several generic properties in 2020. To be consistent with the 2019 presentation scheme, we only refer in this paper to "deadlock", while detailed results in [51] are provided differently or as a part of other examinations.

[2] In 2015, *Upper bounds computation* was part of the *reachability* examination, but became a distinct one in 2016. To be consistent with the 2019 presentation scheme, we only refer in this paper to "Upper bounds", while detailed results in [51] are provided as a part of another examination in 2015.

[3] `https://cadp.inria.fr/man/caesar.bdd.html`

Type	Tools with this type of execution
Sequential	Cunf, enPAC, GreatSPN-Meddly, LoLA (2015-2016), MARCIE, MCC4MCC, PeCAN, pnmc, PNXDD, smart, Spot, TINA, ydd-pt
Parallel	LTSMin
Portfolio	ITS-Tools, LoLA (2017-2019), MCC4MCC, StrataGEM, TAPAAL

Table 1: Classification of tools according to their type of execution.

3 Reported techniques

This section lists the techniques that the participating tools report after carrying out each examination during the MCC. To enhance clarity, we classify the techniques into four categories, described in the remainder of this section:

- Type of execution,
- Support of high-level information,
- Type of model checking,
- Additional techniques.

3.1 Type of execution

This category distinguishes three types of model checkers: (1) *sequential* ones using only one core, (2) *concurrent* ones that implement parallel algorithms, and (3) *portfolio* ones that may operate several techniques simultaneously and in parallel.

Table 1 reports the type of execution for participating tools in the five editions of the MCC from 2015 to 2019. Note that LoLA has changed its execution model in 2017 (from sequential to portfolio). We note there are few concurrent algorithms face to portfolio ones, mostly relying on diversification.

3.2 Support of high-level information

This category distinguishes the way tools support the high-level information provided in some models. In the MCC, there are two types of high-level information:

- data management thanks to typed tokens in colored Petri nets (the semantics of Symmetric nets [22,43] has been chosen for the contest);
- structural information on models thanks to Nested-Unit Petri Nets [34], (NUPN) that encode hierarchically nested units.

Data management with colored Petri nets ("Unf. to. P/T"). So far, only StrataGEM and ITS-Tools natively support typed tokens of colored nets. Some other tools capture this information thanks to a transformation into an equivalent P/T net, also called "unfolding" in the literature [39]. This unfolding is a preprocessing operation that generates an equivalent P/T net, potentially much larger than the original one, and that is semantically equivalent to the original colored one. The model checking tool then uses the equivalent P/T net instead of the colored one.

Nested-Unit Petri Nets ("NUPN"). This structural information (see Section 6.1) was introduced in some models in 2015: not all models contain NUPN information but numerous ones coming from high-level and structured specifications carry traces of the structure of the original specification.

3.3 Type of model checking

This category classifies model checkers by the type of model checking they support:

Explicit model checking ("Explicit"). This technique enumerates states explicitly. It is often associated to other optimization techniques such as partial order reduction, or various compression mechanisms.

Model checking based on decision diagrams ("Dec. Diag."). In this case, decision diagrams encode the configurations of the system, and the operations act on sets of states. The most emblematic example of decision diagrams is BDD [19,21]. The tools participating in the MCC often use more sophisticated variants such as MDD [6], SDD [27] or LDD [28].

Exploiting symmetries of the system ("Symmetries"). The MCC focuses on the model checking of asynchronous systems. Such systems often express symmetries. For example, systems built on peer-to-peer architectures, where processes are interchangeable because they have similar behaviors, often exhibit symmetries. Systems with client/server or master/agent architectures where clients or agents are easily permutable, also exhibit symmetries. Some techniques [22,40,26] capture and use symmetries to increase the performance of the model checking, which can yield gains of orders of magnitude in favorable cases.

Technique	Short name	Short description
Abstractions	Abstractions	Perform abstract reasoning on the model so that a decision about the expected result can be taken rapidly (e.g., Bounded model checking, K-induction).
Compression	Compression	Replace a state space S with another (simplified) state space S' such that S and S' are related in a way (e.g., simulation, bisimulation).
Linear Programming	LP-approx.	Use linear programming to answer or reduce some queries without exploring the state space.
Unfolding into a prefix graph	Unfolding	Provide a very efficient representation of the state space generated by a Petri net model using a variant on the same notation.
Partial order reduction	Partial Order	Reduce the size of the state space, by exploiting the commutativity of concurrently executed transitions.
Query reduction	Query Red.	Reduce the size of a query without the need of explicitly searching through the state-space, for example using state equations.
Satisfiability	SAT/SMT	Use satisfiability for the optimization of the transition relations or of the formula to be checked in order to speed up the model checking.
Structural reductions	Struct. reduc.	Perform structural reductions on the input specification before performing the model checking itself. Preserve the properties to be checked.
Topological analysis	Topological. anl.	Extract structural information from the model (e.g., traps/deadlocks, state equation) and use that information to speed up the analysis or over-approximate the state space.

Table 2: Overview of the "Additional techniques". Appendix A fully describes them (with bibliographic references). Short names of techniques are those used in the diagrams of the following sections.

3.4 Additional techniques

Competing model checkers usually combine the above approaches with other techniques (such as heuristics or optimizations) in order to reduce CPU usage and memory footprint. This category further classifies model checkers according to these other techniques.

Table 2 summarizes the reported techniques in this category. Column **Short name** provides the name by which each technique is refered to in the diagrams in the remainder of this paper. Appendix A provides a detailed description of each technique, along with their appropriate bibliographic references.

Examination	Relevant runs	Runs with results
state space	51 834	29 273 (56.5%)
deadlocks	40 540	26 063 (64.3%)
upper bounds	48 837	30 383 (62.2%)
reachability	92 228	60 243 (65.3%)
CTL	70 541	37 058 (52.5%)
LTL	44 946	27 401 (61.0%)

Table 3: Distribution of the 348 926 relevant runs over the examinations from 2015 to 2019. The second column shows the total number of runs dedicated to a given examination and the last column only those that provided at least one value.

4 Reported techniques over the years, a tool perspective

This section reports on how tools uses the techniques presented in Section 3 between 2015 and 2019. We also bring a focus on the tools that were on the podium one or several years. This is a way to identify the techniques they activate and which probably gave them an advantage.

All the data in Sections 4 to 6 are provided by tools, together with their results. Indeed, they are expected to report the techniques used for solving the considered examination when providing the results.

4.1 General information on the raw data

Between 2015 and 2019, there were a total of 561 837 runs. However some tools were not participating at all to some examinations, thus always returning "do not compete". After having discarded these runs, there remain 348 926 relevant ones. Table 3 shows how these runs are spread over the various examinations.

Four examinations request tools to return Boolean values: only one for deadlock detection, and 16 for each of reachability, CTL, and LTL (i.e., 16 formulas must be processed for each of them). Table 5 shows how the values are distributed in $\mathbb{B}$.

These values are computed by tools and represent only a subset of the total number of evaluated formulas between 2015 and 2019. For example, some of the 65.32% runs dedicated to reachability formulas may

	2015		2016		2017		2018		2019	
State space generation										
	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%
Gold	MARCIE	83.11%	ITS-Tools	73.96%	GreatSPN	87.40%	GreatSPN	90.95%	TINA	93.22%
Silver	pnmc	69.29%	MARCIE	72.47%	TINA	78.28%	TINA	85.04%	GreatSPN	87.22%
Bronze	ITS-Tools	59.48%	pnmc	56.36%	ITS-Tools	67.87%	ITS-Tools	69.25%	ITS-Tools	62.02%
Deadlock detection										
	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%
Gold	TAPAAL	74.96%	TAPAAL	88.25%	LoLA	93.74%	TAPAAL	89.62%	TAPAAL	86.16%
Silver	LoLA	74.47%	LoLA	84.58%	TAPAAL	77.79%	ITS-Tools	84.01%	ITS-Tools	82.51%
Bronze	MARCIE	62.16%	MARCIE	56.17%	ITS-Tools	43.35%	LoLA	82.17%	LoLA	71.26%
Upper bounds										
	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%
Gold	LoLA	68.23%	MARCIE	74.12%	GreatSPN	80.02%	LoLA	96.70%	TAPAAL	96.56%
Silver	MARCIE	64.71%	LoLA	51.87%	ITS-Tools	71.70%	TAPAAL	67.45%	LoLA	88.67%
Bronze	TAPAAL	45.43%	ITS-Tools	42.61%	MARCIE	57.79%	GreatSPN	66.41%	ITS-Tools	61.86%
Reachability formulas										
	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%
Gold	LoLA	91.10%	LoLA	93.53%	LoLA	99.49%	TAPAAL	97.77%	TAPAAL	97.28%
Silver	TAPAAL	75.29%	TAPAAL	71.19%	TAPAAL	75.10%	LoLA	91.93%	LoLA	90.50%
Bronze	MARCIE	43.03%	ITS-Tools	53.95%	ITS-Tools	55.50%	ITS-Tools	69.29%	ITS-Tools	68.44%
CTL formulas										
	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%
Gold	MARCIE	100.00%	LoLA	65.93%	LoLA	81.37%	TAPAAL	90.99%	TAPAAL	91.41%
Silver	—		TAPAAL	62.97%	TAPAAL	65.70%	LoLA	77.60%	LoLA	81.32%
Bronze	—		MARCIE	46.77%	ITS-Tools	34.29%	ITS-Tools	42.94%	ITS-Tools	46.70%
LTL formulas										
	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%	BVT	100.00%
Silver	—		LTSMin	95.35%	LoLA	100.00%	LoLA	98.45%	LoLA	94.04%
Silver	—		LoLA	79.84%	LTSMin	71.87%	LTSMin	83.08%	ITS-Tools	76.55%
Bronze	—		ITS-Tools	34.56%	ITS-Tools	50.46%	ITS-Tools	71.91%	enPAC	3.50%

Table 4: Tools on the podium between 2015 and 2019 (details are available in [51]). BVT corresponds to a "best virtual tool" returning the union of all the values computed by all tools for a given examination in a given edition [75]. We use it to estimate the number of values that some tools on the podium could not compute while at least one other tool did. As an example, the winner of the state space generation examination is getting closer to the total number of computed values every year. In 2015, only one tool was ranked for CTL and no tool participated in LTL.

have only computed some of the 16 proposed formulas in the examination.

The next subsections report an analysis of this large amount of data. First, we have a look at the way competing tools reported the techniques presented in Section 3 between 2015 and 2019. Then, we deal with the evolution of such techniques as reported by podium tools over the same period in order to understand if some techniques have emerged or disappeared.

Examination	Computed values	True	False
deadlocks	26 063	11 016 42.3%	15 047 57.7%
reachability	683 887	339 589 49.7%	344 298 50.3%
CTL	380 012	169 361 44.6%	210 651 55.4%
LTL	341 531	76 332 22.4%	265 199 77.7%

Table 5: Distribution of values computed by tools in $\mathbb{B}$ for deadlocks, reachability, CTL, and LTL, from 2015 to 2019. The second column shows all the computed values for the corresponding examination while the third and fourth columns display the number of satisfied and unsatisfied formulas. State space generation and upper bound computation are not reported because expected values are integers.

4.2 The podium tools

Every year, there is a podium for every examination, highlighting the best three tools. Such tools are considered to be the most efficient since the higher their score is, the more values they have computed. This is a way to identify the most efficient techniques that were operated for the benchmark. So, we will regularly focus on these tools in the next sections.

Table 4 reports the nine tools (out of 17) that were on a podium between 2015 and 2019[4]. A tool may participate with several variants in one edition of the MCC. However, we only consider the best of these variants for

[4] Results are globally expressed in the 2019 way. For instance, deadlock detection was part of the reachability examination before 2019 so the corresponding data are not explicitly displayed on the MCC web site. Similarly, in 2015, computation of upper bounds was also part of the reachability examination and thus not noted explicitly on the MCC web site.

the podium, to avoid any bias when comparing the techniques podium tools have activated, with respect to the others.

Let us introduce an interesting notion inspired from the SAT competition in 2012 [7] : the *best virtual solver* [75]. For the MCC, we call it *best virtual tool* (`BVT`). It is a model checker capable of computing the union of the values computed by all the participating model checkers. `BVT`'s score can be computed as for other tools and the gold medal's score can be compared to the one of `BVT`.

In Table 4, the score distance to `BVT` has been normalized to a percentage: every year, both grading and the benchmark change so numeric values are not relevant. Consequently, we cannot draw conclusions in terms of tools' progression: regularly, we introduce new techniques to generate harder formulas and the community proposes more complex models. This explains why the progression of percentages is not monotonous.

However, the "distance" between the gold medal and `BVT` indicates the complementarity between tools by highlighting the ratio of values the gold medal could not compute. We can observe from Table 4 that, for the later editions of the MCC, the gold (and silver too) medal tools cover a larger part of all the values computed by all tools together (*i.e.,* the "distance" to `BVT` becomes smaller).

4.3 Multi-year observations from 2015 to 2019

Figure 1 aggregates all the data gathered between 2015 and 2019. For each selected technique, we consider how it was reported by all tools (black bars in the charts), and also by tools on the podium (red – or gray in B&W – bars in the charts). The goal is to find out if some techniques are reported more often by the "winning tools" rather than the others, thus providing a clue about their efficiency.

We normalize the appearance of techniques to a percentage. We then measure how frequently such techniques appear. Note that the sum of all techniques in some of the categories presented in Section 3 is not 100%:

- *type of execution*: it is unfortunately not always reported by tools precisely and it cannot be automatically deduced from the number of cores allocated to tools; when 4 cores are required, it can be a portfolio or a concurrent algorithm (this may vary according to the examination);
- *support of high-level information*: it only concerns a subset of the participating tools;
- *additional techniques*: usually, several techniques are operated simultaneously. Indeed, since the beginning of the MCC, we have observed that tools activate more and more techniques simultaneously in the same run.

We have defined a 30% threshold in Figure 1 to outline the most reported techniques. Although this value is arbitrary, we have chosen it as an indicator of the "popularity" of a given technique. We note in Figure 1 that, for a given technique, the black bar (all tools) and the red/gray bar (podium tools) are frequently in the same part of the threshold. We observe that SAT/SMT techniques bring a more significant advantage for Reachability (Fig. 1d), CTL (Fig. 1e) and LTL (Fig. 1f) because the black bar is below the threshold while the red one is above.

Let us now discuss what we observe from the charts of Figure 1.

Type of execution. For state space generation, concurrent or portfolio approaches do not bring any advantage (Fig. 1a). This is probably related to the intensive use of decision diagrams and topological analysis (to find an appropriate encoding) which are more difficult to operate in parallel.

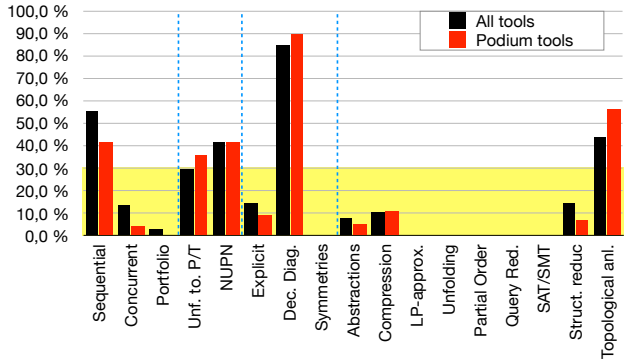
It is the opposite for LTL where almost no sequential tool participates (Fig. 1f). We note that LTSMin, which proposes concurrent algorithms is on the podium from 2016 to 2018 (it was not participating in 2015 and 2019). Since it is the only fully concurrent tool, it is worth noticing because the corresponding bars in the diagram are underestimated and the ratio between portfolio and concurrent is the lowest of all examinations.

Figures 1b to 1e show a trend: sequential is less represented than portfolio in podium tools w.r.t. all tools (remember that tools do not always report it appropriately).

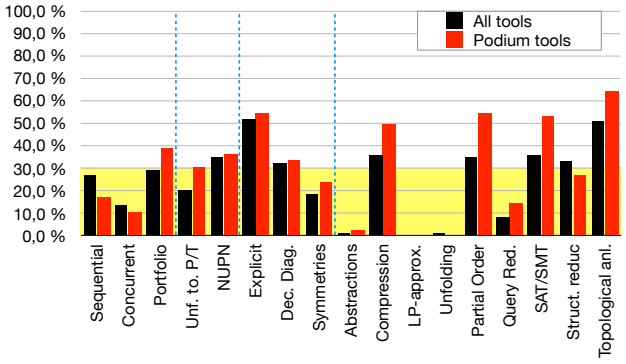
Support of high-level information. Unfolding into P/T nets increases a bit for all examinations, especially for podium tools, mainly because in 2015 and 2016, these tools have increased their support of colored nets, thanks to a library shared between several developers.

The use of NUPN information only gives a marginal advantage in some situations. However, since this notation was gradually adopted by tools, a more precise study focusing on the exploitation of the model hierarchical structure is presented in Section 6.

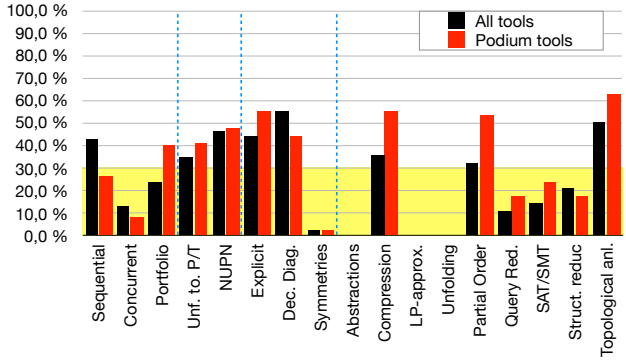
Type of model checking. We consider here explicit model checking, compression techniques based on decision diagrams, and the use of symmetries.



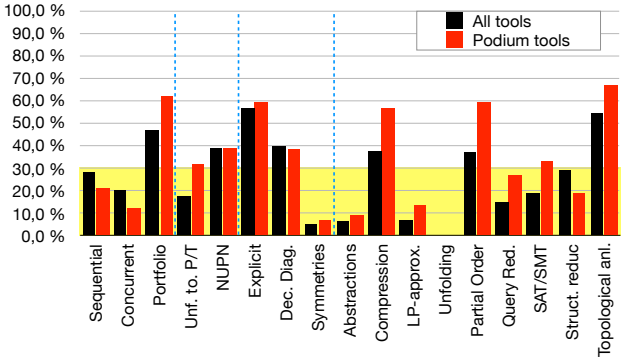
(a) State Space generation



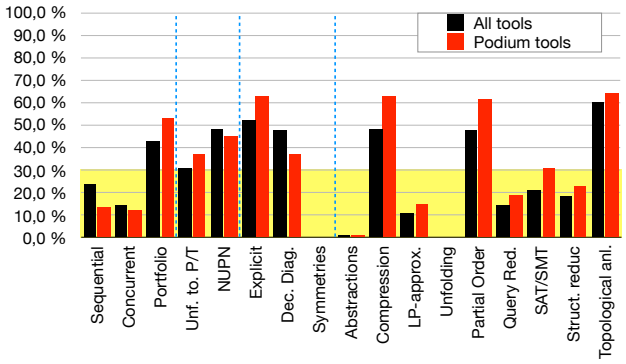
(b) Deadlock detection



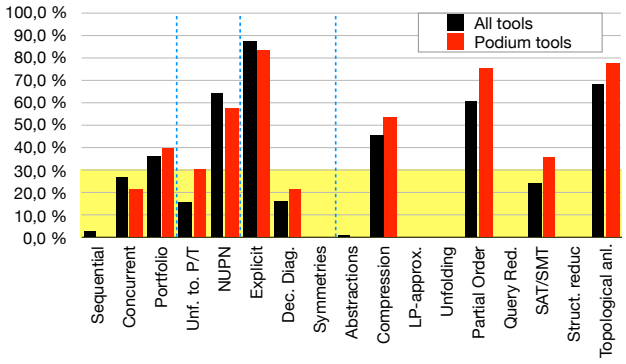
(c) Upper bounds



(d) Reachability



(e) CTL



(f) LTL

Fig. 1: Techniques as expressed by the tools between 2015 and 2019. Black bars denote the percentage of techniques claimed by all tools while red (or gray in B&W) bars correspond to those claimed by the tools being on the podium. A threshold of 30% is also outlined in these charts. Charts are split in four areas (dotted vertical lines): type of execution, handling of high-level information, the type of model checking, and additional properties.

We clearly observe that the use of symmetries is marginal in all examinations and not even mentioned in CTL or LTL (Fig. 1e and 1f). We note that deadlock detection is probably the examination which benefits the most from their use (Fig. 1b).

Except for state space generation where the use of decision diagrams is intensively reported (Fig. 1a), there is no big difference in their use with respect to explicit techniques. In some cases explicit model checking appears to bring an advantage to podium tools (upper bounds, Fig. 1c, reachability, Fig. 1d, CTL, Fig. 1c) while in some others, decision diagrams do (LTL, Fig. 1f). In all situations, the advantage is noticeable but marginal.

Sometimes, ITS-Tools is able to mix decision diagrams and symmetry management, in order to mutu-

ally increase the potential gain provided by exploiting each of these approaches.

Additional techniques. The threshold of 30% is particularly interesting for reporting about these techniques:

- state space reports only one popular technique in this category: topological analysis (Fig. 1a);
- deadlock detection can be computed thanks to many techniques since 5 are reported to be "popular": compression, partial order, SAT/SMT, structural reduction – black bar – and topological analysis (Fig. 1b);
- for other examinations, the three same techniques are "popular": compression, partial order, and topological analysis. SAT/SMT is also popular when formulas are involved but for podium tools only.

Of course, other techniques are reported more marginally in many examinations (except for state space which mentions only four). Among the "less popular" reported techniques, structural reductions do not bring a significant advantage since the red/gray bar is lower than the black one. In all other cases, the activation of the reported techniques brings an advantage to podium tools. We also observe that the use of a prefix graph is extremely rare in the participating tools. In fact, only Cunf [65] implements this technique and was only participating in 2015 in the deadlock detection examination.

These charts clearly show that tools operate several techniques simultaneously:

- when computing deadlocks, TAPAAL [47] is sometimes able to operate simultaneously in a run the four main techniques reported in the chart: topological analysis, partial order, SAT/SMT and compression;
- ITS-Tools [70] and LoLA [74] are able to operate at least two of these techniques simultaneously in a run. However, they do not report structural reductions with another technique;
- when computing upper bounds, TAPAAL is able to simultaneously activate query reduction, SAT/SMT, structural reductions, partial order and compression. LoLA also reports using together SAT/SMT, partial order and compression, or SAT/SMT and topological analysis;
- when computing LTL formulas, ITS-Tools is able to operate three techniques simultaneously in a run : topological analysis, partial order and SAT/SMT.

Examination	Runs with results	Runs involving podium tools
State space	29 273	12 337 (42.14%)
Deadlocks	26 063	14 736 (48.50%)
Upper bounds	30 383	17 012 (65.27%)
Reachability	60 243	38 059 (48.50%)
CTL	37 058	23 414 (63.18%)
LTL	27 401	21 425 (78.19%)
Total	*210 394*	*126 983 (60.35%)*

Table 6: Distribution of runs executed by podium tools among the various examinations from 2015 to 2019. The second column shows the total number of runs providing at least one result for the given examination (third column of Table 3) and the last one those corresponding to podium tools.

4.4 Evolution of techniques for podium tools from 2015 to 2019

The data presented in the previous section does not capture how tools have progressed from 2015 to 2019. The objective of this section is to have a closer look at the evolution of reported techniques over the years.

We focus on the techniques podium tools reported from 2015 to 2019. Since these tools seem to have benefited from using these techniques, we study how their use has evolved over the years. Moreover, the core of 9 podium tools during the period 2015-2019 is relatively stable (see Table 4), thus increasing relevancy of these observations.

To simplify the right part of each chart in Fig. 2, we focus on the "additional techniques" which passed the 30% threshold presented in Section 4.3. For discarded techniques, the visibility is smaller and yearly variations are thus less significant.

Because we consider podium tools only, we used a subset of the 210 421 runs studied in Section 4.3. Table 6 shows the distribution of the 126 983 selected runs with results provided by podium tools. We note that 60.35% of runs with results are computed together by podium tools globally (details per examination are in Table 6). One could think that podium tools are better than others. The data in Table 4 correlate this observation: the gold medal is often very close to the best virtual tool (i.e., the virtual one that computes the union of the values computed by all tools), and we note a significant gap between the gold medal, on the one hand, silver and bronze on the other hand. The other tools being behind.

The analysis performed in this section considers these runs only.

Similarly to Table 5, Table 7 shows how Boolean values are distributed over these 126 983 selected runs.

This distribution is relatively similar to the one for values computed by all tools. So, apparently, results should be consistent.

Let us now discuss what we observe from the charts of Figure 2.

Type of execution. Except for state space generation (Fig. 2a), the use of parallelism increases over the years. For parallel tools, portfollio remains the most popular approach.

Support of high-level information. The use of high-level information increases over the years. Indeed, more tools are able to support colored nets, and the exploitation of hierarchical information (via NUPN) also increases.

Type of model checking. The use of decision diagrams decreases, compared to explicit techniques (strongly associated to some of the "additional techniques").

Additional techniques. It is more difficult to conclude but the use of compression and partial order reduction globally increases strongly for almost all examinations. The use of SAT/SMT solvers varies but increases dramatically in some cases (Fig. 2b to 2d).

4.5 Observed trends

The tracking of techniques over these five editions of the model checking contest shows some trends.

For state space generation, the set of used techniques remains rather stable and reduced. It shows that decision diagrams-based methods are prevalent over other techniques. Therefore, additional techniques mainly aims at providing better orders for the encoding of the system.

For examinations involving formulas, we observe that the variety of techniques used aims at two goals:

- they perform a preliminary analysis whose outputs can either feed another optimization of the state space exploration, or even solve the query;
- they help optimize the state space traversal to compute whether a property is true or not.

These observations confirm that alongside the traditional state space exploration, emerging techniques as such as SAT/SMT, structural analysis, Linear Programming, etc. prove to be useful, and even successful.

Examination	Computed values	True	False
Deadlocks	14 736	7 027 47.7%	7 709 52.3%
Reachability	444 055	224 461 50.6%	219 594 49.5%
CTL	248 832	111 906 45.0%	136 926 55.0%
LTL	260 238	59 092 22.7%	201 146 77.3%

Table 7: Distribution of values computed by podium tools in $\mathbb{B}$ for Deadlocks, Reachability, CTL, LTL, from 2015 to 2019. State space generation and upper bound computation are not reported because expected values are integers. The second column shows all the computed values while the third and fourth columns display the number of satisfied and unsatisfied formulas.

5 Reported techniques over the years, a formula perspective

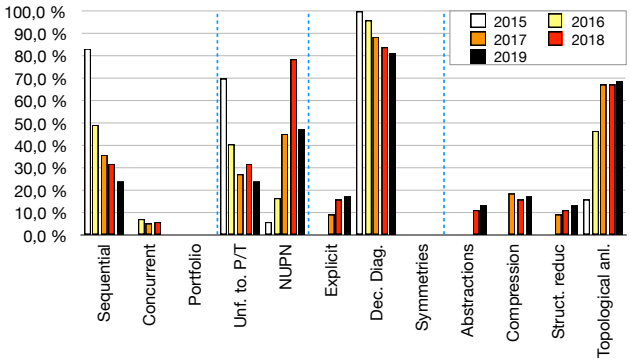
This section provides another perspective to the results presented in Section 4. The approach is to consider the techniques with respect to the difficulty of the formulas, *i.e.,* the difficulty to give a correct truth value for the formula). For that, we focus on the examinations for which the tools are evaluated on several formulas (in practice 16 formulas per examination): Reachability, CTL, and LTL.

Our general methodology consists in comparing the techniques used by the tools that correctly gave the truth value of formulas that we consider of medium difficulty (we call them Type 1 formulas) to the techniques used by the tools that correctly gave the truth value of formulas that we consider of high difficulty (Type 2 formulas). Type 2 formulas are a subset of Type 1 formulas.

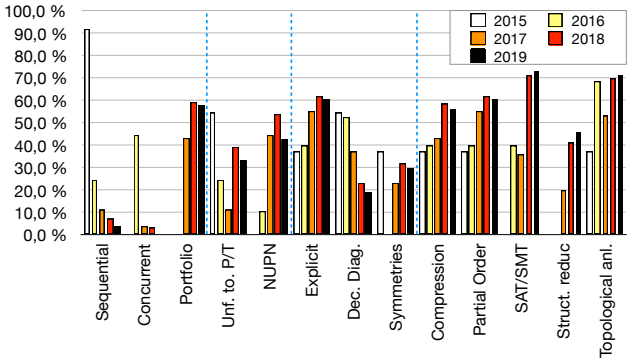
More precisely, we call formulas of Type 1 those formulas for which at least one tool gave a correct truth value and at least one tool did not. All tools together, on such formulas, 155 182 correct answers were given for Reachability, 95 067 for CTL, and 48 085 for LTL.

We call formulas of Type 2 those formulas for which at least one tool gave a correct truth value and at least half of the participating tools did not. This corresponds to 101 306 correct answers for Reachability, 70 988 for CTL, and 34 444 for LTL.

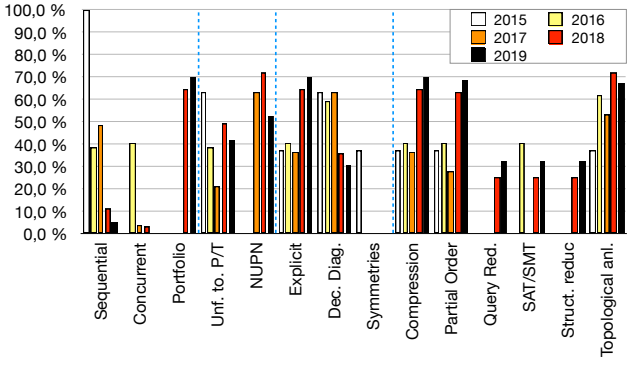
Figure 3 presents the results obtained for formulas of Type 1 (left) and of Type 2 (right). For each examination and each technique, the reported percentage gives the proportion of computed values obtained using this technique among all computed values that were successfully obtained in the considered examination. In principle, we consider that, if a technique is more represented in Type 2 formulas (right) than in Type 1 formulas (left), then it means that this technique has a



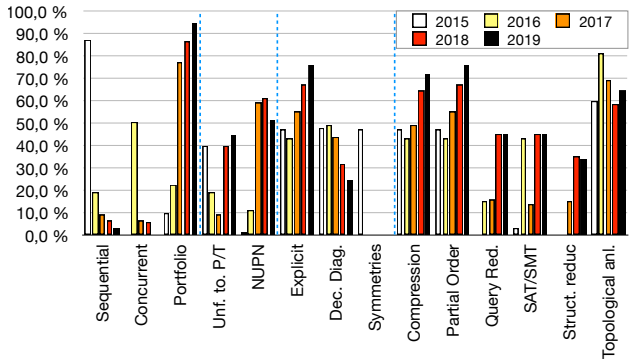
(a) State Space generation



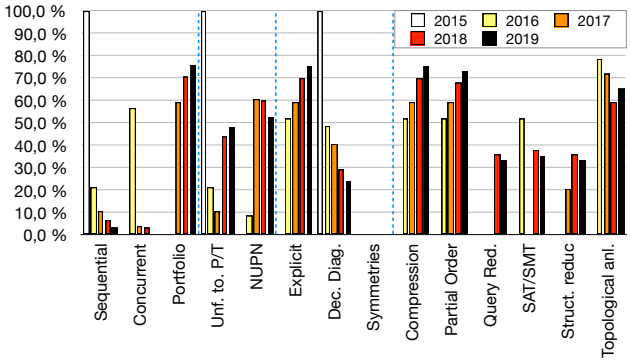
(b) Deadlock detection



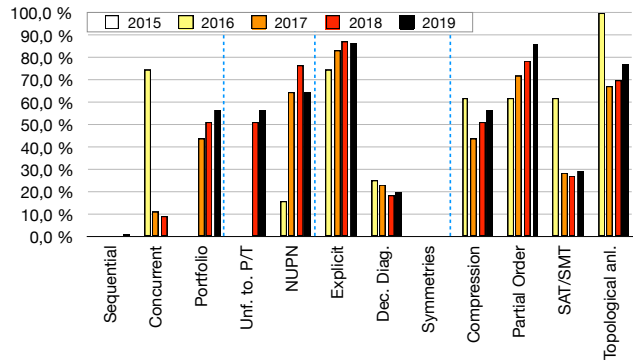
(c) Upper bounds



(d) Reachability



(e) CTL (only one participant in 2015)



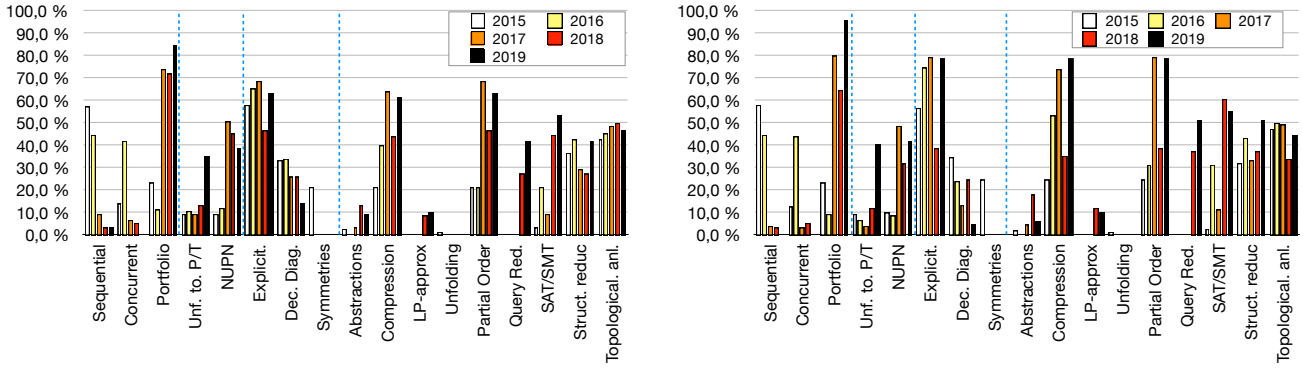
(f) LTL (no participant in 2015)

Fig. 2: Yearly evolution of the main techniques reported for state space generation over 2015-2019. For LTL formulas (figure 2f), in 2015 no tool did participate in this examination so no bars are reported this year.

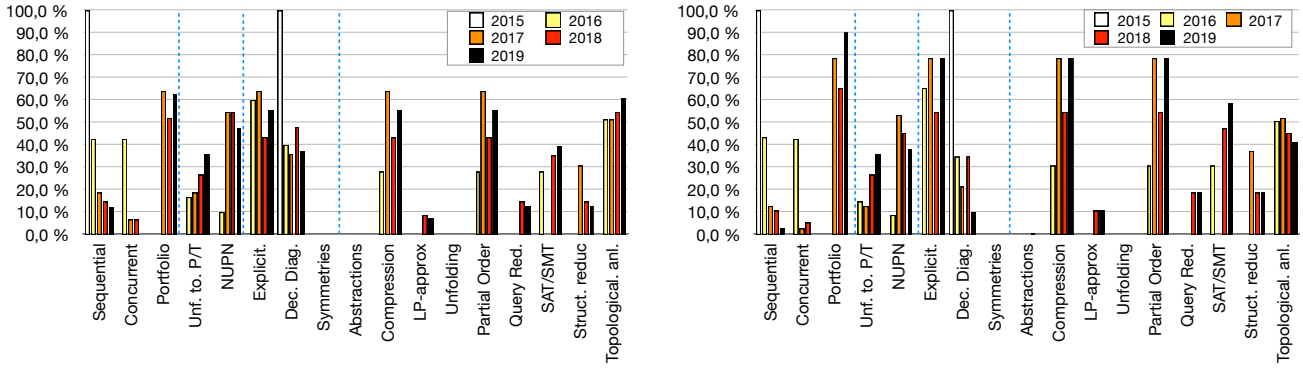
positive impact on the capacity of tools to solve complex verification tasks. With that in mind, we can state the following observations.

Portfolio tools. Section 4 reported an increase in the use of portfolio tools. The results obtained here seem to confirm that these tools are of interest. This observation is particularly true for CTL verification where portfolio tools are much more represented among correct results on Type 2 formulas than Type 1 formulas (for example in 2019, 90% against less than 65%). It is however also true for LTL verification, and, to a lesser extent for Rechability verification, with the only significant difference between Type 1 and Type 2 being in 2019.

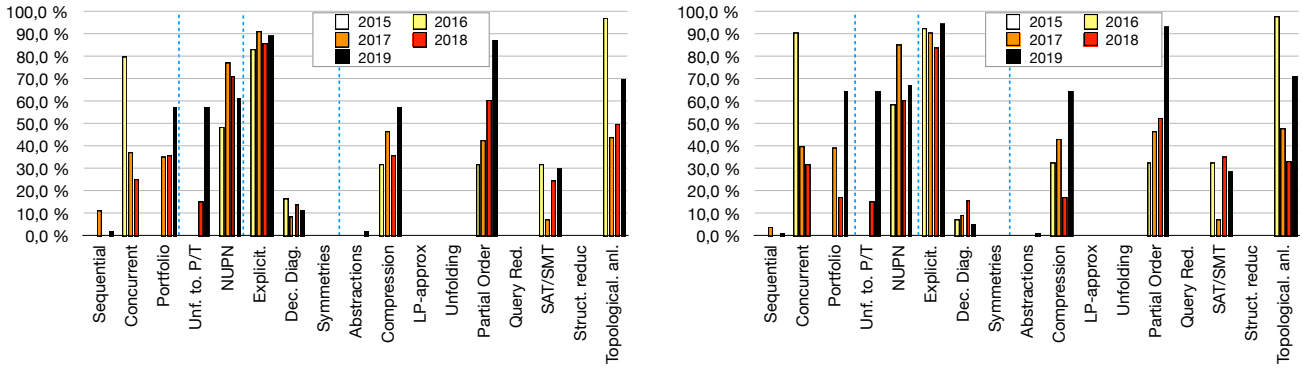
NUPN. We expected that the use of the more accurate information on the structure of the nets provided by NUPN would help tools to deal with verification tasks more easily. However, at the moment, this is not re-



(a) Reachability: Type 1 on the left and Type 2 on the right



(b) CTL: Type 1 on the left and Type 2 on the right



(c) LTL: Type 1 on the left and Type 2 on the right

Fig. 3: Techniques used by tools for solving moderately difficult (Type 1) and difficult formulas (Type 2).

ally something that we can observe from our results. It seems to be a reasonable choice for dealing with Reachability (no significant difference in the proportions of tools using it between Type 1 and Type 2) and LTL (tools using it more represented for formulas of Type 2 than Type 1, with the exception of the 2018 edition). For CTL verification, the use of NUPN can be questioned: the proportion of tools using it for solving Type 1 formulas is generally greater or equal than the proportion of tools using it for solving Type 2 formulas.

Compression and partial orders. The use of these techniques looks very efficient for reachability and CTL verification: it is reported to be used by a significantly larger proportion of tools solving Type 2 formulas than tools solving Type 1 formulas. For LTL verification, no clear impact can be observed as the results really differ depending on the considered year.

Additional techniques. Unfolding to P/T nets seems to bring a significant impact for reachability analysis. One can also notice that topological analysis, while being used by many tools, does not appear to be that effi-

cient for CTL and LTL verification. Finally, the results obtained for the other techniques have a lot of variability across the years, so it is difficult to get a clear insight about them.

6 Using structural information on models

This section reports two interesting findings regarding the use of structural information in model checking. Here, we focus on two types of information: hierarchical structure of the original specification, and a maximum number of tokens in subsets of places. These are carried out thanks to Nested-Unit Petri Nets in the MCC.

6.1 Background

When derived from real problems (*e.g.,* industrial case studies) specifications are usually modular and hierarchically structured. However, standard (low-level) Petri nets do not provide satisfactory means to express such a modular structure. This problem has fueled research for decades on: *(i)* the decomposition of Petri nets into state machines that execute concurrently [54], and *(ii)* the design of many high-level extensions of Petri nets, which enables one to capture the modularity of systems [33, 72, 40].

Exploiting the specification's structure is a quite active research topic, and there have been recent contributions to high-level extensions. As an example, [27] proposes a way to hierarchically encode hierarchically structured models using decision diagrams, which later proved practical efficiency on regular models in [71]. Another approach, the Nested-Unit Petri Nets (NUPN) proposed in [34, 35], introduces "recursively nested units" to describe modular and hierarchical systems using Petri nets.

NUPN also brings additional information for 1-safe nets: each nested unit is 1-safe. This allows tools to use dedicated compression mechanisms. Tools like ITS-Tools [70] (dedicated 1-safe reductions), LoLA [74] (one bit per nested unit special encoding), or LTSMin [49] (dedicated mechanisms for the encoding into decision diagrams) exploit this characteristic.

However, despite clear successes, the benefits of using such information remains a question since it could be related to a given tool or to a few problems only.

6.2 Dedicated experimentation

In the setting of the MCC, NUPN provides structural information. A significant part of the models has been produced from complex case studies and their modular, hierarchical structure is provided as a NUPN "tool-specific" section in the PNML of the models. This is referred to as "high-level information" in Section 3. Given that many tools competing in the MCC'2019 claim to exploit the NUPN information, this section evaluates the benefits, in terms of performance, of using such information. The variety of our benchmark on the one hand, the number of tools and their diverse techniques on the other hand, let us assume that no implementation or benchmark bias should be expected.

To study the impact of structural information on model checking, we used the subset of 239 instances of models from the MCC benchmark (23.4% of the total) that contain the NUPN information. We considered the tools able to use such information in 2019 (using their latest implementations): GreatSPN-Meddly [4], ITS-Tools [70], LoLA [74], LTSMin [49] and TINA [12].

We have reprocessed these tools twice, for all examinations in the conditions of the MCC in 2019, for the benchmark subset: once with models containing the NUPN information, and once without. This dedicated analysis resulted in 21 510 runs for a total of almost 417 days of CPU. These extra experiments were processed on a machine lent to us by the University of Twente (it is also one of the machines used for the MCC): a 96 physical cores @ 2.2GHz with 2048 GB of memory.

6.3 Impact of NUPN on computed values

Table 8 reports the number of values computed when the NUPN information is present, compared to when it is not (namely, on the same MCC models obtained after erasing all NUPN information). The average increase is of about 5.5% more values computed with NUPN than without. However, this increase really depends on tools. For one tool (LoLA), there is no significant difference (a few values less out of almost 29 000). For the other tools, up to 20% extra values are computed when the NUPN information is used.

Tool	# values no NUPN	# values NUPN	more values (in %)
GreatSPN	7 312	7 807	+6.8%
ITS-Tools	14 832	16 483	+11%
LoLA	23 879	23 874	-0.2%
LTSMin	2 224	2 660	+20%
TINA	584	680	+16%
Total	48 831	51 504	+5.5%

Table 8: Total number of computed values with and without the use of NUPN. This was computed on the 21 510 runs processed to evaluate the interest of NUPN.

Examination	#runs	#values (no NUPN)	#values (NUPN)	time (mn) (no NUPN)	time (mn) (NUPN)	faster comput.	#NUPN faster	#NUPN slower
				GreatSPN				
State space	157	623	623	278	245	+14%	28	129
Deadlock	92	92	92	283	240	+18%	17	75
Upper bounds	140	2240	2240	294	237	+24%	27	113
Reachability	140	2 192	2 192	514	444	+16%	29	111
CTL	138	2 165	2 165	603	555	+9%	32	106
Globally	667	7 312	7 312	1 972	1 721	**+14%**	133	534
				ITS-Tools				
State space	115	345	345	224	68	+231%	75	40
Deadlock	184	184	184	107	103	+4%	104	80
Upper bounds	112	1 792	1 792	221	86	+156%	71	41
Reachability	178	2 864	2 864	751	350	+114%	108	70
CTL	159	2 065	2 087	1 377	735	+87%	120	39
LTL	250	3 889	3 934	2 149	1 461	+47%	184	66
Globally	998	11 139	11 206	4 829	2 803	**+72%**	662	336
				LoLA				
Deadlock	176	176	176	80	58	+34%	93	83
Upper bounds	237	3 528	3 528	2 802	2 803	0%	128	109
Reachability	473	6 957	6 962	9 605	9 607	0%	242	231
CTL	472	6 027	6 020	21 849	21 789	0%	236	236
LTL	475	7 170	7 169	9 063	9 120	-1%	239	236
Globally	1 833	23 858	23 855	43 399	43 377	**0%**	938	895
				LTSMin				
State space	103	412	412	507	293	+73%	74	29
Upper bounds	239	1 808	2 192	7 973	6 636	+20%	141	98
Globally	342	2 220	2 604	8 480	6 929	**+22%**	215	127
				TINA				
State space	145	584	584	1 298	485	+167%	119	26
Globally	145	584	584	1 298	485	**+167%**	119	26

Table 9: Evaluation of the variation (without and with NUPN) for cases where no time-out is reported, neither for instances including NUPN, nor those without. Remember that some tools do not participate in all examinations.

Examination	#runs	#values (no NUPN)	#values (NUPN)	time (mn) (no NUPN)	time (mn) (NUPN)	faster comput.	#NUPN faster	#NUPN slower
State space	375	1 380	1 380	2 307	1 091	+111%	177	198
Deadlock	452	452	452	470	403	+17%	214	238
Upper bounds	873	9 952	10 336	11 291	9 763	+16%	486	387
Reachability	791	12 013	12 018	10 870	10 403	+4%	379	412
CTL	769	10 257	10 272	23 828	23 078	+3%	388	381
LTL	725	11 059	11 103	11 212	10 582	+6%	423	302
Globally	3 985	45 113	45 561	59 978	55 387	**+8%**	2 067	1 918

Table 10: Average benefit of the use of NUPN information for each examination category when considering GreatSPN, ITS-Tools, LoLA, LTSMin and TINA on a subset of execution were no time-out is observed neither for models with NUPN information nor models without NUPN information.

The tools that benefit from the use of hierarchical structural information are those that rely on decision diagrams, even if this technique is not always used at each execution (another technique may find a solution earlier).

6.4 Impact of NUPN on execution performances

The analysis of Section 6.3 must be completed by an evaluation of the execution time required to compute such values. To do so, we selected a subset of 3 985 runs from the initial 21 510 processed for this experiment where, for a given model instance and examination, no time-out occurred, neither for instances including NUPN nor for those without. By doing so, our comparison is free from biases arising from such time-outs that would hinder a precise measure of the speed-up. Section 6.5 reports how NUPN also reduces the number of time-outs, thus allowing the computation of more values.

Table 9 gives the detailed results for the considered tools. For each examination we report: the number of considered runs, the number of values computed (with and without NUPN), the total execution time (with and without NUPN), and how much faster the execution

was (expressed as a percentage) deduced from:

$$\frac{time\ without\ NUPN}{time\ with\ NUPN}$$

between these runs. We also report how many times the runs with NUPN were faster (second to last column) or slower (last column) than the corresponding runs without NUPN.

Table 10 shows a summary of the progress for each MCC examination (state space generation, deadlock detection, upper bounds, reachability, CTL, LTL). It shows the same columns as Table 9.

Tables 9 and 10 clearly exhibit a faster computation when the NUPN information is present.

Tools' perspective. Table 9 outlines a great variety of the performance increase. Unsurprisingly, tools (like LoLA) that do not use decision diagrams exhibit lower performances globally (except for deadlock detection). For tools like ITS-Tool, LTSMin or GreatSPN-meddly, which strongly rely on decision diagrams, the NUPN information clearly brings an advantage by helping them deduce better encodings of the input specification.

TINA also relies on decision diagrams, and a newly introduced technique to count states in [11] clearly benefits from structural information.

GreatSPN is usually slower with NUPN, but however overall faster with that additional information. Only 133 runs are sped up (out of 667), however significantly enough to make an impact. So, for GreatSPN, exploiting NUPN information on a single specification is probably a bet.

Let us finally note that in only two cases (LoLA for CTL and LTL), the number of computed formula decreases. For other tools, the use of the NUPN information only increases the number of processed values.

Examinations' perspective. Unsurprisingly, we note a great variance, depending on tools and examinations.

State space examination is the one which benefits the most of the use of structural information with an execution performance of +111% (and even up to +231% for ITS-Tools).

In some situations, deadlock detection also benefits from structural information: we observe a computation performance of +15% for GreatSPN and +25% for LoLA. In fact, this is the only examination where the NUPN information seems to make a difference for LoLA, keeping in mind that LoLA globally computes more results than the other tools.

For other examinations, it is difficult to conclude because only a very few tools are involved:

- for upper bounds, some gain is also observed (up to 156% for ITS-Tools); in that case, the 1-safety of NUPN is the characteristic that helps;
- the average execution performance for reachability analysis and CTL is quite low, even if the implementation provided by ITS-Tools again takes significant advantage of the NUPN information (respectively +61% and +57%).

We note that 448 more values are computed with NUPN in Table 10, while Table 8 reports 2 673 more values. It means that if NUPN brings an advantage to the processing time, it also brings an advantage in the number of computed values. Exploiting the NUPN information probably helps reducing the situation where time-out occurs, which cannot be observed in Tables 9 and 10 that discard runs with time-out.

A surprising result when comparing how frequently NUPN is reported between moderately difficult formulas and difficult formulas (Fig. 3) is that it brings a similar benefit to both levels of difficulty. Thus surprisingly, NUPN does not help more to solve difficult formulas than easier ones.

6.5 Summary

The total number of computed values increases by 5.5%, of which a majority of about 4.5% is associated to runs exhibiting time-outs when NUPN was not present (i.e., the difference found between Tables 8 and 10). Second, despite the variety of tools and models, we observe an average gain of time of about 8%, with several peaks far over 100% for some tools on some examinations.

So, our analysis shows a globally positive effect of structural information as provided by NUPN on tools taking advantage of it. This is not a surprise, since the MCC focuses on asynchronous parallel systems where, due to the interleaving of concurrent sequential processes, the notion of locality is important: it helps to reduce the number of firings tools must perform when exploring the state space (thanks to firing caches). NUPN also helps to compress the representation of markings by allowing to reduce the number of bits or variables required to encode markings.

7 Impact of the MCC on tools, techniques and formulas

Since its first edition in 2011, the MCC has become an important regular event in the community of tool developers, as have several other competitions involving research tools. This section reports the impact the

Tool	Country	2015	2016	2017	2018	2019
Cunf [65]	FR					
enPAC [1]	CN					
GreatSPN-Meddly [4]	IT					
ITS-Tools [70]	FR					
LoLA [74]	DE					
LTSMin [49]	NL					
MARCIE [42]	DE					
MCC4MCC [20]	CH					
PeCAN [56]	VN					
pnmc [41]	FR					
PNXDD [45]	FR					
smart [23]	US					
Spot [29]	FR					
StrataGEM [58]	CH					
TAPAAL [47]	DK					
TINA [12]	FR					
ydd-pt [64]	CH					

Table 11: All the 17 tools which participated over the five editions of the Model Checking Contest considered in this study. Colors help distinguish lines.

MCC has had on tools' performances, model checking techniques, and on formulas over the observed period 2015-2019.

7.1 Long-term tool design and improvement

Every year, discussions with tool developers bring new issues and improvements. The MCC had thus fostered successful new research directions, such as the method to count the size of the state space in TINA [11], which won them the gold medal in 2019.

Table 11 shows the participating tools between 2015 and 2019, together with a recent reference to a publication describing them. We note that among those 17 participating tools, eight participated three times or more, and six participated once only. Unfortunately, three tools are discontinued, often due to the move of the leading developer out of academia: MARCIE, pnmc, and PNXDD (these were on podiums). The existence of long-lasting tools, as well as experimental ones (usually related to a thesis) is one evidence of the long term implication in tool design the MCC has helped to sustain.

7.2 Steady confidence rate increase

Developers sometimes present several variants of their tools to evaluate some alternative algorithms or heuristics. They also share some interesting components such as a library commonly developed and debugged to provide the support of colored nets thanks to the unfolding technique.

2015	2016	2017	2018	2019
35.82%	51.91 %	52.91 %	60.40 %	65.50 %

Table 12: yearly percentage of runs in which tools provide at least one value among all the processed runs (all examinations together), after having discarded those where tools reported no participation. Only common models from 2015 to 2019 are considered.

Since 2015, we measure a confidence rate for every participating tool. It relies on a subset of *trusted values*. A trusted value is the one computed by a significant majority of at least 3 tools that agree on it. For this subset, we compute for each tool the following ratio:

$$\frac{\#\ trusted\ values\ computed\ correctly}{\#\ trusted\ values\ computed}$$

Tools below a given ratio (0.98 = 98% confidence in 2019) do not score any point if they are the only ones to compute a value.

Figure 4 shows how the average confidence rate of tools evolves between 2015 and 2019. In 2019, a new tool had a notable low confidence rate, probably due to some bugs in the formula converter, thus decreasing the average. The eight core participating tools have increased their confidence rate over the years, some of them reaching 100% confidence (others being very close to it).

Table 12 shows the yearly percentage of runs in which tools provide at least one value among all the processed runs (all examinations together), after having discarded those where tools reported no participation. We use in this table data coming from the set of common models from 2015 to 2019 only. The difficulty of formulas also increased (for formula-based examinations) between 2015 and 2019. So, we think that Table 12 illustrates the growing performances of participating tools. There are two reasons for this increase: *(i)* tools participate in more examinations, and *(ii)*, within

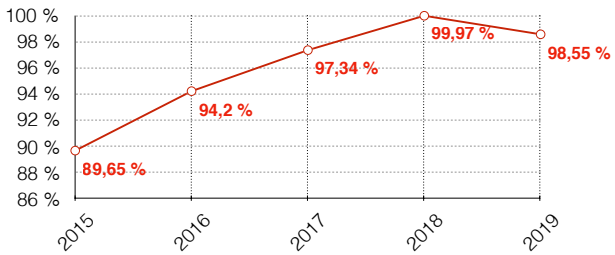


Fig. 4: Average confidence rate of tools between 2015 and 2019. In 2019, a new tools was participating, with some bugs in the formula translation ; this explains the decrease between 2018 and 2019.

an examination, tools compute more values, *i.e.*, solve more problems.

7.3 Improving harder formulas generation

Generating challenging examinations is an issue that is shared by most contests, with different solutions. For instance, in [67] the problem of controlling the hardness of 3-SAT formula is addressed, while [3] deals with the complexity of QSAT formula. In the RERS challenge, a strategy has been elaborated to create complex benchmarks respecting some given properties so that the resulting problem is both difficult to verify and has predictable properties [68].

In the MCC, we of course face this challenge; in our case, it means generating complex formulas. The community produces harder models but we also have to generate harder formulas for examinations requiring such formulas: upper bounds, reachability analysis, CTL, and LTL. The growing complexity of generated formulas can be observed when looking at the percentage of runs returning at least one value, after having discarded those where tools reported no participation. We restricted ourselves to a subset of the largest instances of models that were part of the competition during the five editions (see the list of selected models in Appendix B).

Table 14 shows this rate between 2015 and 2019 for the examination using formulas.

- In 2015 the 20% for CTL can be explained by the fact that only one tool was competing in this examination (and none for LTL the same year, see Table 4);
- In 2016, though, more tools entered the competition to achieve a performance around 60% for CTL and nearly 75% for LTL;
- The performance in CTL for 2019 (nearly 74%) has to be confirmed in the next few years. Reachability, deadlock detection, and CTL are examinations in which tools really improved and perform, now, really well;
- We observe significative improvement of tools from 2015 to 2019, although from one year to another, the set of proposed formulas has changed for upper bounds, reachability, CTL, and LTL examinations.

We have regularly improved the way formulas were generated for the MCC. In 2018 LTL formulas were more evenly dispatched in the supported categories of the Manna&Pnueli LTL hierarchy [59][5]; this may explain the drop of 9% in 2019 compared to 2018 in Table 14. The same year the number of explored states for the bounded model checker, used to assert CTL formula hardness when preparing the contest, doubled; this may explain the drop of 7% in 2018 compared to 2017 in Table 14.

So far, we use bounded model checking for upper bounds, reachability, and CTL, asserting that properties are not "too trivial", while we ensure a minimal number of formulas belonging to each category of the Manna&Pnueli LTL hierarchy. We are currently looking to further improve the process.

8 Impact of the MCC on its community

We present in this section the impact the MCC has had on its community over the observed period 2015-2019.

8.1 A reference benchmark

A large benchmark regularly used and cited in papers[6] is enriched by the community every year. It provides a neutral way to evaluate tools and techniques because models come from various authors and describe different types of systems. This reduces the chances of having some "benchmark effect" that could benefit to some specific techniques only.

8.2 A way to promote collaboration

The MCC promotes collaboration between the participants. As mentioned earlier, several tool developers have shared the unfolder from coloured nets to P/T nets. They have also shared some oracles, to help them correct bugs. Finally, they have also experimented some interesting combinations of tools; for example, the integration and activation of some LTSMin techniques in ITS-Tools [70] in a portfolio.

As for some other competitions having specific needs, we have built a dedicated execution environment: BenchKit [52]. This environment has similar features as BenchExec [14] (used in the SV-COMP) which also measures energy consumption [15]. Another well known execution environment is StarExec [17]; it is used by numerous competitions like Confluence Competition (COCO[7]), the MaxSAT evaluation[8], the QBF evaluation[9] or the SAT competition[10].

[5] Until 2019, the formula generation did not support the categories Reactivity and Obligation in the Manna&Pnueli classification [59].

[6] https://mcc.lip6.fr/bibliography.php#references

[7] http://project-coco.uibk.ac.at

[8] https://maxsat-evaluations.github.io

[9] http://www.qbflib.org/index_eval.php

[10] http://www.satcompetition.org

Examination	2015	2016	2017	2018	2019
Upper bounds	26.9%	28.2 %	39.7 %	48.2 %	63.0 %
Reachability	56.5%	67.6 %	63.7 %	75.8 %	74.8 %
CTL	20.51%	59.6 %	54.4 %	47.6 %	73.6 %
LTL	—	74.3 %	76.4 %	78.6 %	69.8 %

Table 13: yearly percentage of runs in which tools provide at least one value among all the processed runs for examinations requiring a formula on a set of instances that were part of the MCC competition from 2015 to 2019. We consider runs where at least one value was computed after having discarded those where tools reported no participation.

Examination	2015	2016	2017	2018	2019
Upper bounds	26.9%	28.2 %	39.7 %	48.2 %	63.0 %
Reachability	56.5%	67.6 %	63.7 %	75.8 %	74.8 %
CTL	20.51%	59.6 %	54.4 %	47.6 %	73.6 %
LTL	—	74.3 %	76.4 %	78.6 %	69.8 %

Table 14: yearly percentage of runs in which tools provide at least one value among all the processed runs for examinations requiring a formula on a set of instances that were part of the MCC competition from 2015 to 2019. We consider runs where at least one value was computed after having discarded those where tools reported no participation.

9 Conclusion

In this paper, we present a comprehensive report of the data gathered during the five editions of the Model Checking Contest (MCC) from 2015 to 2019. Such analysis can now happen because, since 2015, we publish all results and summaries obtained from running the competition in a standard format. The analysis highlights some trends in model checking-based verification.

Main observations. After an overview of the reported techniques over the considered period, our observations first show that the most efficient tools, *i.e.*, those that reach the podium several times during that period, implement several techniques, sometimes simultaneously activated. They can do so thanks to software architectures inspired from modern compilers: a front end feeds several backends implementing different verification techniques, sometimes in a portfolio mode suitable for concurrent execution [53].

Aside from a concurrent execution in the portfolio approach of activating several algorithms to solve a given verification problem, new concurrent algorithms also emerge. It is mainly the case of LTSMin that was on the LTL verification podium several times between 2015 and 2019.

Decision diagram-based techniques used to be prominent. However, they are now challenged by explicit techniques, combined with additional techniques such as partial order and/or structural analysis. We also discovered that an increasing part of model checking problems is often solved without exploring the state space. It is particularly the case of deadlock detection for tools like LoLA or ITS-Tools. However, this point might be specific to the nature of the specification of problems, Petri nets, for which state equations can be used for structural analysis.

According to the type of verification task (*e.g.*, reachability, CTL, or LTL analysis), a different subset of techniques appears to be more efficient than others. This is also true when considering more difficult formulas, solved by less tools.

Tools benefit from information describing the structure of the specification. Our analysis shows the capability of some tools to successfully exploit such data to speed up the analysis. In the MCC, the NUPN [34] information associated with some models derived from large industrial specifications allows tools to provide more results faster, even if implementation concerns seem to play a non-negligible role in the measured performances.

Note that information about the structure of the specification benefits to symbolic model checking (tools like GreatSPN or ITS-Tools) as well as to explicit model checking (tools like LoLA or TAPAAL).

Validity. We follow the four main concepts to assess the validity of our analysis as defined in [66] for exploratory case studies.

Construct validity: 2015 was already the fifth edition of the MCC, and by then, there was already a common understanding by tool developers of the MCC concepts and model checking techniques presented in sections 2 and 3.

Internal validity: this study does not compare any performance aspects; we only deal with their causal ef-

fect on the ranking, as well as techniques tools have reported. Since each edition is a standalone consistent event, there is no bias in having such a multi-year analysis because only factual and reproducible data is processed.

External validity: the way we run the MCC is similar to several competitions with similar objectives, like SAT [8,2], SMT [9], QBF [62] or CASC [69]. There is a common understanding on the way such studies must be carried out. So, we believe the MCC's findings are relevant to other people investigating model checking techniques and tools.

Reliability: experiments can be reprocessed (not on the exact same machines but in a similar situation) since submitted tools in their virtual machines are still available on the MCC web site [51]. If some local measures could change according to computers' characteristics, we evaluate computed results of examinations (computed values, claimed techniques) that should not change.

Future work. The Model Checking Contest will continue to challenge tools in upcoming editions. It will propose new examinations and more difficult problems to solve, in order to push forward the state of the art of model checking-based verification of asynchronous and concurrent systems.

Acknowledgments. The authors would like to thank the developers of the tools participating in the Model Checking Contest since the beginning to make this event successful. We, more particularly, thank the developers participating between 2015 and 2019 for their kindness while answering our questions about some technical details on their tools.

References

1. enPAC git-hug page (2020). URL `https://github.com/Tj-Cong/enPAC`
2. The international sat competition web page (2020). URL `http://www.satcompetition.org`
3. Amendola, G., Ricca, F., Truszczynski, M.: A generator of hard 2qbf formulas and ASP programs. In: 16th International Conference on Principles of Knowledge Representation and Reasoning, pp. 52–56. AAAI Press (2018)
4. Amparore, E.G., Balbo, G., Beccuti, M., Donatelli, S., Franceschinis, G.: 30 years of GreatSPN. In: Principles of Performance and Reliability Modeling and Evaluation, pp. 227–254. Springer (2016)
5. Baarir, S., Duret-Lutz, A.: SAT-Based Minimization of Deterministic ω-Automata. In: 20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, *Lecture Notes in Computer Science*, vol. 9450, pp. 79–87. Springer (2015)
6. Babar, J., Beccuti, M., Donatelli, S., Miner, A.S.: GreatSPN Enhanced with Decision Diagram Data Structures. In: 31st International Conference on Applications and Theory of Petri Nets, *Lecture Notes in Computer Science*, vol. 6128, pp. 308–317. Springer (2010)
7. Balint, A., Belov, A., Järvisalo, M., Sinz, C.: Sat challenge 2012 (2012). URL `https://baldur.iti.kit.edu/SAT-Challenge-2012`
8. Balint, A., Belov, A., Järvisalo, M., Sinz, C.: Overview and analysis of the SAT challenge 2012 solver competition. Artificual Intelligence **223**, 120–155 (2015)
9. Barrett, C., de Moura, L., Stump, A.: Smt-comp: Satisfiability modulo theories competition. In: 17th International Conference on Computer Aided Verification – CAV, *Lecture Notes in Computer Science*, vol. 3576, pp. 20–23. Springer (2005)
10. Berthelot, G., Roucairol, G.: Reduction of Petri Nets. In: 5th Symposium on Mathematical Foundations of Computer Science 1976, *Lecture Notes in Computer Science*, vol. 45, pp. 202–209. Springer (1976)
11. Berthomieu, B., Botlan, D.L., Dal-Zilio, S.: Petri net reductions for counting markings. In: 25th International Symposium on Model Checking Software (SPIN), *Lecture Notes in Computer Science*, vol. 10869, pp. 65–84. Springer (2018)
12. Berthomieu, B., Ribet, P.O., Vernadat, F.: The tool TINA–construction of abstract state spaces for Petri nets and Time Petri nets. International journal of production research **42**(14), 2741–2756 (2004)
13. Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.): Tools and Algorithms for the Construction and Analysis of Systems – 25 Years of TACAS: TOOLympics, *Lecture Notes in Computer Science*, vol. 11429. Springer (2019)
14. Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: requirements and solutions. Int. J. Softw. Tools Technol. Transf. **21**(1), 1–29 (2019). DOI 10.1007/s10009-017-0469-y
15. Beyer, D., Wendler, P.: CPU Energy Meter: A Tool for Energy-Aware Algorithms Engineering. In: Tools and Algorithms for the Construction and Analysis of Systems, *Lecture Notes in Computer Science*, vol. 12079, pp. 126–133. Springer (2020)
16. Biere, A., van Dijk, T., Heljanko, K.: Hardware model checking competition 2017. In: FMCAD, p. 9. IEEE (2017)
17. Bjørner, N., Benney, E., Gupta, A., Horrocks, I., Iannni, G., Le Berre, D., Wakdmann, J.: Starexec (2020). URL `https://www.starexec.org/starexec/public/about.jsp`
18. Bønneland, F., Dyhr, J., Jensen, P.G., Johannsen, M., Srba, J.: Simplification of CTL formulae for efficient model checking of petri nets. In: 39th International Conference on Application and Theory of Petri Nets and Concurrency, *Lecture Notes in Computer Science*, vol. 10877, pp. 143–163. Springer (2018)
19. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers **35**(8), 677–691 (1986)
20. Buchs, D., Klikovits, S., Linard, A., Mencattini, R., Racordon, D.: A Model Checker Collection for the Model Checking Contest Using Docker and Machine Learning. In: 39th International Conference on Application and Theory of Petri Nets and Concurrency, *Lecture Notes in Computer Science*, vol. 10877, pp. 385–395. Springer (2018)

21. Burch, J., Clarke, E., McMillan, K.: Symbolic model checking: 10^{20} states and beyond. Information and Computation (Special issue for best papers from LICS90) **98**(2), 153–181 (1992)
22. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: A symbolic reachability graph for coloured Petri nets. Theoretical Computer Science **176**(1–2), 39–65 (1997)
23. Ciardo, G., Miner, A.S., Wan, M.: Advanced features in SMART: the stochastic model checking analyzer for reliability and timing. SIGMETRICS Performance Evaluation Review **36**(4), 58–63 (2009)
24. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design **19**(1), 7–34 (2001)
25. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: 12th International Conference on Computer Aided Verification – CAV, *Lecture Notes in Computer Science*, vol. 1855, pp. 154–169. Springer (2000)
26. Colange, M., Hillah, L.M., Kordon, F., Parutto, P.: Extreme Symmetries in Complex Distributed Systems: the Bag-Oriented Approach. In: 17th Monterey Workshop : Development, Operation and Management of Large-Scale Complex IT Systems, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 7539, pp. 330–352. Springer, Oxford, UK (2012)
27. Couvreur, J., Thierry-Mieg, Y.: Hierarchical decision diagrams to exploit model structure. In: 25th International Conference on Formal Techniques for Networked and Distributed Systems – FORTE, *Lecture Notes in Computer Science*, vol. 3731, pp. 443–457. Springer (2005)
28. van Dijk, T., van de Pol, J.: Multi-core decision diagrams. In: Y. Hamadi, L. Sais (eds.) Handbook of Parallel Constraint Reasoning., pp. 509–545 (2018)
29. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 - A framework for LTL and ω-automata manipulation. In: 14th International Symposium on Automated Technology for Verification and Analysis – ATVA, *Lecture Notes in Computer Science*, vol. 9938, pp. 122–129 (2016)
30. Ernst, G., Huisman, M., Mostowski, W., Ulbrich, M.: Verifythis - verification competition with a human factor. In: Tools and Algorithms for the Construction and Analysis of Systems – 25 Years of TACAS: TOOLympics, *Lecture Notes in Computer Science*, vol. 11429, pp. 176–195. Springer (2019)
31. Esparza, J., Hoffmann, P.: Reduction rules for colored workflow nets. In: 19th International Conference on Fundamental Approaches to Software Engineering – FASE (ETAPS), *Lecture Notes in Computer Science*, vol. 9633, pp. 342–358. Springer (2016)
32. Esparza, J., Schröter, C.: Net reductions for LTL model-checking. In: 11th IFIP WG 10.5 conference in Correct Hardware Design and Verification Methods CHARME, *Lecture Notes in Computer Science*, vol. 2144, pp. 310–324. Springer (2001)
33. Fehling, R.: A concept of hierarchical petri nets with building blocks. In: Advances in Petri Nets, *Lecture Notes in Computer Science*, vol. 674, pp. 148–168. Springer (1993)
34. Garavel, H.: Nested-Unit Petri Nets: A structural means to increase efficiency and scalability of verification on elementary nets. In: International Conference on Application and Theory of Petri Nets and Concurrency, *LNCS*, vol. 9115, pp. 179–199. Springer (2015)
35. Garavel, H.: Nested-unit petri nets. J. Log. Algebr. Meth. Program. **104**, 60–85 (2019)
36. Girault, C., Valk, R.: Petri nets for systems engineering - a guide to modeling, verification, and applications. Springer (2003). URL `http://www.springer.com/computer/swe/book/978-3-540-41217-5`
37. Godefroid, P., Peled, D.A., Staskauskas, M.G.: Using partial-order methods in the formal validation of industrial concurrent programs. In: International Symposium on Software Testing and Analysis – ISSTA, pp. 261–269. ACM (1996)
38. Haddad, S.: A reduction theory for coloured nets. In: Advances in Petri Nets 1989, covers the 9th European Workshop on Applications and Theory in Petri Nets, held in Venice, Italy in June 1988, selected papers, *Lecture Notes in Computer Science*, vol. 424, pp. 209–235. Springer (1990)
39. Haddad, S.: Introduction: issues in verification. In: C. Girault, R. Valk (eds.) Petri Nets for Systems Engineering, A Guide to Modeling, Verification, and Applications, pp. 183–200 (2003)
40. Haddad, S., Kordon, F., Petrucci, L., Pradat-Peyre, J.F., Trèves, N.: Efficient State-Based Analysis by Introducing Bags in Petri Net Color Domains. In: 28th American Control Conference – ACC, pp. 5018–5025. Omnipress IEEE, St-Louis, USA (2009)
41. Hamez, A.: A Symbolic Model Checker for Petri Nets: pnmc. Transactions on Petri Nets and Other Models of Concurrency **XI**, 297–306 (2016)
42. Heiner, M., Rohr, C., Schwarick, M., Tovchigrechko, A.A.: Marcie's secrets of efficient model checking. Transactions on Petri Nets and Other Models of Concurrency **XI**, 286–296 (2016)
43. Hillah, L., Kordon, F., Petrucci, L., Trèves, N.: PN standardisation : a survey. In: International Conference on Formal Methods for Networked and Distributed Systems – FORTE, *Lecture Notes in Computer Science*, vol. 4229, pp. 307–322. Springer Verlag, Paris, France (2006)
44. Hillah, L.M., Kindler, E., Kordon, F., Petrucci, L., Trèves, N.: A primer on the Petri Net Markup Language and ISO/IEC 15909-2. Petri Net Newsletter **76**, 9–28 (2009)
45. Hong, S., Kordon, F., Paviot-Adet, E., Evangelista, S.: Computing a hierarchical static order for decision diagram-based representation from P/T nets. Transactions on Petri Nets and Other Models of Concurrency **V**, 121–140 (2012)
46. Jasper, M., Mues, M., Murtovi, A., Schlüter, M., Howar, F., Steffen, B., Schordan, M., Hendriks, D., Schiffelers, R.R.H., Kuppens, H., Vaandrager, F.W.: RERS 2019: Combining synthesis with real-world models. In: Tools and Algorithms for the Construction and Analysis of Systems – 25 Years of TACAS: TOOLympics, *Lecture Notes in Computer Science*, vol. 11429, pp. 101–115. Springer (2019)
47. Jensen, J.F., Nielsen, T., Oestergaard, L.K., Srba, J.: TAPAAL and reachability analysis of P/T nets. Transactions on Petri Nets and Other Models of Concurrency **XI**, 307–318 (2016)
48. Jensen, P.G., Larsen, K.G., Srba, J.: PTrie: Data Structure for Compressing and Storing Sets via Prefix Sharing. In: 14th International Colloquium on Theoretical Aspects of Computing – ICTAC, *Lecture Notes in Computer Science*, vol. 10580, pp. 248–265. Springer (2017)
49. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: Ltsmin: High-performance language-independent model checking. In: 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems – TACAS (ETAPS), *Lecture Notes*

in Computer Science, vol. 9035, pp. 692–707. Springer (2015)
50. Kordon, F., Garavel, H., Hillah, L., Paviot-Adet, E., Jezequel, L., Hulin-Hubard, F., Amparore, E.G., Beccuti, M., Berthomieu, B., Evrard, H., Jensen, P.G., Botlan, D.L., Liebke, T., Meijer, J., Srba, J., Thierry-Mieg, Y., van de Pol, J., Wolf, K.: Mcc'2017 - the seventh model checking contest. Transactions on Petri Nets and Other Models of Concurrency **XIII**, 181–209 (2018)
51. Kordon, F., Garavel, H., Hillah, L.M., Hulin-Hubard, F., Jézéquel, L., Paviot-adet, E.: The Model Checking Contest (2019). URL `https://mcc.lip6.fr`
52. Kordon, F., Hulin-Hubard, F.: BenchKit, a Tool for Massive Concurrent Benchmarking. In: 14th International Conference on Application of Concurrency to System Design – ACSD, pp. 159–165. IEEE Computer Society (2014)
53. Kordon, F., Leuschel, M., van de Pol, J., Thierry-Mieg, Y.: Software Architecture of Modern Model Checkers. In: B. Steffen, G.J. Woeginger (eds.) Computing and Software Science - State of the Art and Perspectives, *Lecture Notes in Computer Science*, vol. 10000, pp. 393–419. Springer (2019)
54. Kordon, F., Peyre, J.F.: Process decomposition for Rapid Prototyping of Parallel systems. In: Proceedings of the 6th International Symposium on Computer and Information Science, Kener, Antalya, Turkey (1991)
55. Kordon, F., van de Pol, J., Seidl, M.: Lorentz workshop, advancing verification competitions as a scientific method (2019)
56. Le, D., Nguyen, H., Nguyen, V., Mai, P., Pham-Duy, B., Quan, T., André, É., Petrucci, L., Liu, Y.: Pecan: Compositional verification of petri nets made easy. In: 12th International Symposium on Automated Technology for Verification and Analysis – ATVA, *Lecture Notes in Computer Science*, vol. 8837, pp. 242–247. Springer (2014)
57. Leino, R.: Dafny: An automatic program verifier for functional correctness. In: 16th International Conference on Logic Programming and Automated Reasoning – LPAR, *Lecture Notes in Artificial Intelligence*, vol. 6355, pp. 348–370. Springer (2010)
58. López Bóbeda, E., Colange, M., Buchs, D.: Stratagem: A generic petri net verification framework. In: 35th International Conference on Application and Theory of Petri Nets and Concurrency – PETRI NETS, *Lecture Notes in Computer Science*, vol. 8489, pp. 364–373. Springer, Tunis, Tunisia (2014)
59. Manna, Z., Pnueli, A.: A hierarchy of temporal properties (invited paper, 1989). In: 9th Annual ACM Symposium on Principles of Distributed Computing – PODC, pp. 377–410. ACM (1990)
60. McMillan, K.L.: A technique of state space search based on unfolding. Formal Methods in System Design **6**(1), 45–65 (1995)
61. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (1989)
62. Narizzano, M., Pulina, L., Tacchella, A.: Ranking and reputation systems in the qbf competition. In: AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, *Lecture Notes in Artificial Intelligence*, vol. 4733, pp. 97–108. Springer (2007)
63. Peled, D.A.: All from one, one for all: on model checking using representatives. In: 5th International Conference on Computer Aided Verification – CAV, *Lecture Notes in Computer Science*, vol. 697, pp. 409–423. Springer (1993)
64. Racordon, D., Buchs, D.: Verifying multi-core schedulability with data decision diagrams. In: 8th International Workshop on Software Engineering for Resilient Systems – SERENE, *Lecture Notes in Computer Science*, vol. 9823, pp. 45–61. Springer (2016)
65. Rodríguez, C., Schwoon, S.: Cunf: A tool for unfolding and verifying Petri Nets with Read Arcss. In: 11th International Symposium on Automated Technology for Verification and Analysis – ATVA, *Lecture Notes in Computer Science*, vol. 8172, pp. 492–495. Springer (2013)
66. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. **14**(2), 131–164 (2009)
67. Selman, B., Mitchell, D.G., Levesque, H.J.: Generating hard satisfiability problems. Artificial Intelligence **81**(1-2), 17–29 (1996)
68. Steffen, B., Jasper, M., Meijer, J., van de Pol, J.: Property-preserving generation of tailored benchmark petri nets. In: 17th International Conference on Application of Concurrency to System Design – ACSD, pp. 1–8. IEEE Computer Society (2017)
69. Sutcliffe, G.: The cade-26 automated theorem proving system competition - casc-26. AI Commun. **30**, 419–432 (2017)
70. Thierry-Mieg, Y.: Symbolic Model-Checking Using ITS-Tools. In: 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems – TACAS (ETAPS), *Lecture Notes in Computer Science*, vol. 9035, pp. 231–237. Springer (2015)
71. Thierry-Mieg, Y., Poitrenaud, D., Hamez, A., Kordon, F.: Hierarchical set decision diagrams and regular models. In: 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems – TACAS (ETAPS), *Lecture Notes in Computer Science*, vol. 5505, pp. 1–15. Springer (2009)
72. Valk, R.: Object petri nets. In: Advances in Petri Nets, *Lecture Notes in Computer Science*, vol. 3098, pp. 819–848. Springer (2004)
73. Valmari, A.: A stubborn attack on state explosion. In: Computer Aided Verification, 2nd International Workshop – CAV, *Lecture Notes in Computer Science*, vol. 531, pp. 156–165. Springer (1991)
74. Wolf, K.: Petri Net Model Checking with LoLA 2. In: 39th International Conference on Application and Theory of Petri Nets and Concurrency – PETRI NETS, *Lecture Notes in Computer Science*, vol. 10877, pp. 351–362. Springer (2018)
75. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Evaluating component solver contributions to portfolio-based algorithm selectors. In: 15th International Conference on Theory and Applications of Satisfiability Testing – SAT 2012, *Lecture Notes in Computer Science*, vol. 7317, pp. 228–241. Springer (2012)

A – Detailed description of additional techniques reported in section 3

Abstractions. This technique denotes a family of methods that perform abstract reasoning on the model so that a decision about the expected result can be taken rapidly. Typical examples of abstractions are: Bounded model checking, K-induction, and counterexample-guided abstraction refinement (CEGAR).

Bounded model checking [24] deals with the encoding of the system and its properties into a satisfiability problem

which is then processed, typically by a SAT or SMT solver. Usually, the transition relation is encoded up to a certain bound with the hope of finding a counterexample. It is a semi-decision algorithm often coupled with K-induction.

K-induction is a SAT-based analysis technique [57] performed in two steps. First, the analyzed property must hold for any sequence of K steps starting from the initial state; then, the SAT solver is used to verify that any path with the property satisfied for K consecutive states, always leads to a K+1 state, that also satisfies the property.

CEGAR [25] consists in checking the property of a system at a coarse (imprecise) abstraction level. If a counterexample is found, the tool checks for its feasibility (e.g., checks if the violation is due to the imprecision of the abstraction or not). If the counterexample is not feasible, then the abstraction must be refined to exclude the false counterexample and then the property must be checked again.

Compression. This technique denotes a family of methods that replace a state space S with another (simplified) state space S' such that S and S' are related in a way (e.g., simulation, bisimulation) so that results obtained for S' imply validity of the results in S [48]. We distinguish decision diagrams and structural reductions from compression.

Linear Programming. This technique denotes the use of linear programming to answer or reduce some queries without exploring the state space.

Unfolding the system into a prefix graph [60]. This technique provides a very efficient representation of the state space generated by a Petri net model using a variant on the same notation. It allows to represent infinite state spaces in a finite way and is also quite useful for very large systems [32, 31].

Partial order reduction. This technique denotes a family of methods that reduce the size of the state space [37], by exploiting the commutativity of concurrently executed transitions. Usually, such commutativity generates a large number of paths from a given state s to another one s' in the state space. Among the partial order techniques, there are stubborn set [73] and ample sets [63].

Query reduction. This technique denotes a family of methods that reduce the size of a query without the need of explicitly searching through the state-space, for example using state equations, detecting tautologies, and formula rewriting [18, 5].

Use of satisfiability. This technique denotes the use of satisfiability for the optimization of the transition relations or of the formula to be verified in order to speed up the model checking. A typical example is the use of SMT solvers in the context of linear-programming. It is also useful to check for the deadlock freeness of a specification.

Structural reductions. Some tools perform structural reductions on the input specification before performing the model checking itself. Berthelot's reductions [10] or Haddad's reductions [38] are typical examples of such transformations. Of course, such reductions must happen in situations where they preserve the properties to be checked.

Topological analysis. Structural information may be extracted from the model (e.g., traps/deadlocks, state equation) and used to speed up the analysis or over-approximate the state space [61].

A good example and application of topological analysis is variable reordering when dealing with the encoding of the model, for example into decision diagrams [45]. It is also mentioned as a way to optimize state compression techniques.

Petri net properties like invariants, siphons or traps, as well as NUPN information or the definitions of types in colored Petri nets are typical topological information considered by tools in the MCC.

B – Instances of models used in the MCC Benchmark to evaluate the hardness of the verification tasks (section 7.3)

The following instances of model have been selected for the analysis presented in section 7.3:

- ARMCacheCoherence-PT-none
- Angiogenesis-PT-50
- BridgeAndVehicles-COL-V80P50N50
- BridgeAndVehicles-PT-V80P50N50
- CSRepetitions-COL-10
- CSRepetitions-PT-10
- CircadianClock-PT-100000
- CircularTrains-PT-768
- DatabaseWithMutex-COL-40
- DatabaseWithMutex-PT-40
- Dekker-PT-200
- Diffusion2D-PT-D50N150
- DrinkVendingMachine-COL-98
- DrinkVendingMachine-PT-10
- ERK-PT-100000
- Echo-PT-d02r19
- Echo-PT-d05r03
- EnergyBus-PT-none
- Eratosthenes-PT-500
- FMS-PT-500
- GlobalResAllocation-COL-11
- GlobalResAllocation-PT-05
- HypercubeGrid-PT-C3K4P4B12
- HypercubeGrid-PT-C5K3P3B15
- IBM319-PT-none
- IBM5964-PT-none
- IBM703-PT-none
- IBMB2S565S3960-PT-none
- IOTPpurchase-PT-C12M10P15D17
- Kanban-PT-1000
- LamportFastMutEx-COL-8
- LamportFastMutEx-PT-8
- MultiwaySync-PT-none
- NeoElection-COL-8
- NeoElection-PT-8
- ParamProductionCell-PT-5
- Parking-PT-864
- PermAdmissibility-COL-50
- PermAdmissibility-PT-50
- Peterson-COL-7
- Peterson-PT-7
- PhaseVariation-PT-D30CS100
- Philosophers-COL-100000
- Philosophers-PT-010000
- PhilosophersDyn-COL-80

- PhilosophersDyn-PT-20
- Planning-PT-none
- PolyORBLF-COL-S04J06T10
- PolyORBLF-COL-S06J06T08
- PolyORBLF-PT-S04J06T10
- PolyORBLF-PT-S06J06T08
- PolyORBNT-COL-S10J80
- PolyORBNT-PT-S10J80
- ProductionCell-PT-none
- QuasiCertifProtocol-COL-32
- QuasiCertifProtocol-PT-32
- Raft-PT-10
- Railroad-PT-100
- ResAllocation-PT-R003C100
- ResAllocation-PT-R100C002
- Ring-PT-none
- RwMutex-PT-r2000w0010
- SafeBus-COL-80
- SafeBus-PT-20
- SharedMemory-COL-100000
- SharedMemory-PT-000200
- SimpleLoadBal-PT-20
- SmallOperatingSystem-PT-MT8192DC4096
- Solitaire-PT-EngCT7x7
- Solitaire-PT-EngNC7x7
- Solitaire-PT-FrnCT7x7
- Solitaire-PT-FrnNC7x7
- Solitaire-PT-SqrCT5x5
- Solitaire-PT-SqrNC5x5
- SquareGrid-PT-130613
- SwimmingPool-PT-10
- TokenRing-COL-500
- TokenRing-PT-050
- UtahNoC-PT-none
- Vasy2003-PT-none