MCC'2017 – The Seventh Model Checking Contest*

Fabrice Kordon[1], Hubert Garavel[2,3], Lom Messan Hillah[4,1], Emmanuel Paviot-Adet[5,1], Loïg Jezequel[6], Francis Hulin-Hubard[7], Elvio Amparore[8], Marco Beccuti[8], Bernard Berthomieu[9], Hugues Evrard[10], Peter G. Jensen[11], Didier Le Botlan[9], Torsten Liebke[12], Jeroen Meijer[13], Jiří Srba[11,14], Yann Thierry-Mieg[1], Jaco van de Pol[13], and Karsten Wolf[12]

[1] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
[2] Univ. Grenoble Alpes, Inria, CNRS, LIG, F-38000 Grenoble, France
[3] Universität des Saarlandes, Saarbrücken, Germany
[4] Université Paris Nanterre, F-92001, Nanterre, France
[5] Université Paris Descartes, F-75005 Paris, France
[6] Université de Nantes, LS2N, UMR CNRS 6597, F-44321 Nantes, France
[7] CNRS, LSV, Ecole Normale Supérieure Paris-Saclay, F-94235 Cachan, France
[8] Università degli Studi di Torino, Italy
[9] LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France
[10] Imperial College London, London, UK
[11] Dept. of Computer Science, Aalborg University, Denmark
[12] Institut für Informatik, Universität Rostock, 18051 Rostock, Germany
[13] Department of Computer Science, University of Twente, Enschede, The Netherlands
[14] FI MU, Brno, Czech Republic

Abstract. Created in 2011, the Model Checking Contest (MCC) is an annual competition dedicated to provide a fair evaluation of software tools that verify concurrent systems using state-space exploration techniques and model checking. This article presents the principles and results of the 2017 edition of the MCC, which took place along with the Petri Net and ACSD joint conferences in Zaragoza, Spain.

1 Goals and Scope of the Model Checking Contest

The Model Checking Contest (MCC) is part of the growing trend of scientific contests, among which one can also mention: the SAT[1] and the SMT[2] competitions, the Hardware Model Checking Competition[3], the Rigorous Examination of Reactive Systems challenge[4], the Timing Analysis Contest[5], and the Competition on Software Verification[6]. The overall goal of these contests is to identify the theoretical approaches that

* Contact author: Fabrice.Kordon@lip6.fr
[1] http://www.satcompetition.org
[2] http://www.smtcomp.org
[3] http://fmv.jku.at/hwmcc/
[4] http://rers-challenge.org
[5] http://www.tauworkshop.com/
[6] https://sv-comp.sosy-lab.org/

are the most fruitful in practice when applied to a variety of examples, and figure out which types of systems are best handled by each approach. These contests also favor the emergence of systematic, rigorous, and reproducible ways to assess the capabilities of verification tools on complex (realistic and synthesized) benchmarks.

The primary goal of the MCC is to evaluate model-checking tools that analyze formal description of concurrent systems, i.e., systems in which several processes run simultaneously, communicating and synchronizing together. Examples of such systems include hardware, software, communication protocols, and biological models. The Model Checking Contest has been actively growing since its first edition in 2011, attracting key people sharing a formal methods background, but with diverse knowledge.

The community gathered around the MCC is actively involved in key activities that contribute to its growth year after year: contributing models to the benchmark, submitting tools to the competition, improving the automated generation of temporal logic formulas, maintaining the repository of models, contributing the infrastructure for running and evaluating the competing tools, improving the performance measurement and assessment tools, and publishing the results.

So far, all editions of the MCC have been using Petri nets to describe the analyzed systems. However, the contest is also open to tools not primarily based on Petri nets. Indeed, we have observed, over several editions, participating tools interfacing a native generic engine with the input format of the MCC.

The present paper reports about the seventh edition, which was organized in Zaragoza as a satellite event of the 38$^{\text{th}}$ International Conference on Application and Theory of Petri Nets and Concurrency and the 17$^{\text{th}}$ International Conference on Application of Concurrency to System Design. The goals of the MCC were first depicted in [31]; for this first edition of the MCC, experiments were conducted on a small number of models, and tools were merely asked to build the reachability graph, perform deadlock detection, and evaluate simple reachability formulas. Over the years, new models and new classes of problems have been added, and enhancements of the evaluation procedure have been introduced, which have been described in [32], which reports about the fifth edition of the MCC held in 2015. Compared to this latter presentation, the present paper:

- highlights the new benchmarks used for the recent editions of the contest, highlighting the variety of the contributed models;
- reports about improvements in the formulas generation workflow and updates in restructuring and simplifying the categories of verification tasks in the competition;
- discusses the impact of the growing number of runs (as a result of the growing number of models and submitted tools) on the evolution of the execution infrastructure and in-house assessment tool *B*ench*K*it;
- provides an in-depth analysis and discussion of the results and involved techniques used in the competing tools, with a comparison between symbolic approaches (i.e., those based on decision diagrams) and explicit ones (i.e., those based on the enumeration of explicit states); and
- presents all the tools that won a gold or silver medal during the 2017 edition of the MCC; for each winning tool, the underlying algorithmic techniques are explained, and the lessons learned from the contest are discussed.

The remainder of this paper is organized as follows. Section 2 presents the models submitted to the 2017 edition of the MCC, highlighting an interesting collection of models (originating from novel distributed algorithms) that have been contributed to the MCC, and describes the way temporal-logic formulas have been produced by the MCC team. Then, the monitoring environment and the experimental conditions are sketched in Section 3. Section 4 focuses on the results of the 2017 edition and the presentation of the tools that won a gold or silver medal in at least one examination. Finally, Section 5 reflects on the overall experience gained in organizing the MCC over the first seven editions, and discusses future work.

2 Models and Formulas

All the tools participating in a given edition of the MCC are evaluated on the same benchmark suite, which is incrementally updated every year. The yearly edition of the MCC starts with a *call for models* inviting the scientific community at large (i.e., beyond the developers of the participating tools) to propose novel benchmarks that will be used for the MCC. These benchmarks consist of *models* and *formulas*.

2.1 Models

Each *model* corresponds to a particular academic or industrial problem, e.g., a distributed algorithm, a hardware protocol in a circuit, a biological process, etc. A model may be parameterized by one or more parameters representing quantities, such as the number of agents in a concurrent system, the number of messages exchanged and the like. To each parameterized model are associated as many *instances* (typically, between 2 and 25) as there are different combinations of parameter values; each non-parameterized model has a single associated instance.

Each instance is a Petri net encoded in the PNML [26] file format. Each model, its instances, and their structural and behavioural properties are described in a synthetic PDF document called *model form* – see [32, Sect. 2] for details about model forms and their preparation. There are two kinds of instances: colored Petri nets and place-transition nets (noted P/T nets). Among the latter, one identifies the particular class of NUPNs (*Nested-Unit Petri Nets*) [17] [32, Sect. 2], a structured form of Petri nets that preserve locality and hierarchy information by recursively expressing a net in terms of parallel and sequential compositions. It is worth mentioning that, beyond the Model Checking Contest, the NUPN model has also been adopted for the parallel problems of the RERS (*Rigorous Examination of Reactive Systems*) competition[7].

In 2017, following the MCC call for models, ten new models (totalling 153 instances) have been proposed, namely: *BART* (a sample speed controller, by Fabrice Kordon), *ClientsAndServers* (an architecture with clients, servers, managers, and resources, by Claude Girault), *CloudReconfiguration* (a dynamic reconfiguration protocol for cloud applications, by Rim Abid, Gwen Salaün, and Noël de Palma), *DLCround* (various distributed implementations of the musical chairs game, by Hugues Evrard),

[7] See `http://www.rers-challenge.org`

Year	2011	2012	2013	2014	2015	2016	2017
New models	7	12	9	15	13	11	10
All models	7	19	28	43	56	67	77
New instances, among which:	95	101	70	138	121	139	153
– *new colored nets*	43	37	24	33	27	9	16
– *new P/T nets*	52	64	46	105	94	130	137
– *new NUPNs (among P/T nets)*	0	0	1	5	15	62	64
All instances	95	196	266	404	525	664	817

Table 1. Accumulation of models and instances over the years

FlexibleBarrier (a novel barrier algorithm for multitasking on GPUs, by H. Evrard), *HexagonalGrid* (a packet-switching network whose ports are situated on the sides of an hexagon, by Tatiana Shmeleva), *JoinFreeModules* (a model of a schedulability problem, by Thomas Hujsa), *NeighborGrid* (a canvas of cellular automata, by Dmitry Zaitsev), *Referendum* (a simple referendum system, by F. Kordon), and *RobotManipulation* (concurrent processes that handle robots, by F. Kordon).

The new models submitted each year are called *surprise models* because they are not known in advance by the tool developers participating in the MCC, contrary to the models submitted during the former years, which are thus called *known models*. The surprise models are merged with the known ones to form a growing collection[8] (continuously expanded since 2011), which gathers systems from diverse academic and industrial fields: software, hardware, networking, biology, etc. Table 1 gives an account of this collection, which currently has 77 models and 817 instances; colored Petri nets, P/T nets, and NUPNs represent respectively 23%, 77%, and 18% of this collection.

2.2 Featured Model Contribution

The collection of MCC benchmarks is a perennial result, which will remain available to the scientific community after the MCC contests have stopped. The usefulness of this collection is already witnessed by nearly fifty scientific publications[9].

The contributors who submit new models play an essential role in every MCC edition. However, these contributors receive much less visibility than the tool developers participating in the MCC, as the latter can be rewarded by podiums and medals. To address this bias, it was decided to put a special focus on the contributor who submitted the most models since the first MCC edition. So far, the most active contributors are Fabrice Kordon (15 models), Lom Messan Hillah (6 models), Hugues Evrard (5 models), Monika Heiner (5 models), and Niels Lohmann (5 models). Given that many of these contributors are already authors of the present article, either as MCC organizers or tool developers, we chose to put the focus on Hugues Evrard by inviting him to present the five models he contributed to the MCC.

These five models are: *MultiwaySync* (2014, prepared with Frédéric Lang), *Raft* (2015), *DLCshifumi* (2016, also with F. Lang), *DLCround* (2017), and *FlexibleBarrier*

[8] The collection of benchmarks is available from `http://mcc.lip6.fr/models.php`

[9] The current list of publications is available from `http://mcc.lip6.fr/bibliography.php`

(2017). These models share two common points: (i) they all describe involved aspects of the implementation of synchronisation among concurrent processes in modern distributed systems, and (ii) these models have not been expressed directly using Petri nets, but rather generated automatically from formal specifications written in a higher-level concurrent language. Precisely, these models have been written in LNT [19], a concurrent language designed as a modern replacement for LOTOS (ISO/IEC international standard 8807). The LNT specifications are first translated to LOTOS using the LNT2LOTOS translator, which is part of the CADP toolbox [18], and then to Petri nets (actually, NUPNs) using the CÆSAR compiler, also available as part of CADP.

MultiwaySync [15] is a distributed synchronisation protocol that implements multiway rendezvous, the generalization of Hoare's rendezvous [25] to more than two concurrent processes. Multiway rendezvous (see [20] for an overview) is used by most process calculi to perform synchronisation and communication between an arbitrary number of concurrent processes. It is more complex than ordinary synchronisation barriers, as each process can be willing to participate in several synchronisations at the same time, but can only engage in a single one. *MultiwaySync* implements multiway synchronisation on top of the lower-level, asynchronous message-passing primitives provided by usual networks; the design of *MultiwaySync* revealed subtle bugs in former protocols implementing multiway synchronisation.

Raft was obtained by formally specifying in LNT the Raft algorithm [37] that enables concurrent processes to reach consensus, even in the presence of failures; this is crucial for implementing fault-tolerant services replicated on many servers, some of which may crash or be unreliable. Raft was designed to replace Paxos, the standard consensus algorithm [33], which is notoriously complex and difficult to implement correctly; the rapid adoption of Raft by several companies (Facebook, Hashicorp, CoreOS) illustrates the need for a "simpler Paxos".

DLCshifumi and *DLCround* model distributed implementations of two well-known games: shifumi (rock-paper-scissors) and musical chairs. These two models, contrary to the three other ones, have not been written by hand in LNT, but produced automatically using the DLC tool [14,16]. DLC (Distributed LNT Compiler) generates, from an LNT specification, a distributed implementation running on a set of machines communicating through TCP sockets and synchronizing using the aforementioned *MultiwaySync* protocol. For verification purpose, DLC can also generate a formal model of such a distributed implementation, where the multiway synchronisation protocol used in the runtime is made explicit. This formal model is itself expressed in LNT, and this is the way *DLCshifumi* and *DLCround* have been produced, before being translated to NUPNs.

Finally, *FlexibleBarrier* describes a special barrier used in the context of cooperative kernels, so as to enable multitasking on GPUs [39]. In this programming model, processes can offer to be killed or forked by the scheduler at precise points of their execution; hence, the number of alive processes varies dynamically and the flexible barrier must interact with the scheduler to know how many processes have to be synchronised.

2.3 Formulas

Tools competing in the MCC are evaluated over five categories of verification tasks: state-space generation, upper-bounds computation (this category was introduced in 2016), reachability analysis, CTL analysis, and LTL analysis (see their description in Sect. 4.1). To maximize tool participation, we further divided the four latter categories into subcategories containing only formulas with a restricted syntax. Each tool developer may choose in which categories/subcategories the tool participates.

In 2015, we consolidated the formula language and provided simplified XML metamodel for each (sub)category, while preserving backward compatibility with previous MCC editions. Since then, the only change to the general metamodel for formulas has been a redefinition of one atomic proposition (called *place-bound* and used only in the new upper-bounds computation category) because tool developers had reported it would be more convenient.

In 2016, we reduced the number of categories. Previously, subcases were simpler versions (with restricted grammar) of larger cases — for example we had *LTLFireability* and its simpler counterpart *LTLFireabilitySimple*. Since every tool participating in the simple subcategories was also participating in its more general counterpart (with similar results in both categories), they were not interesting any more.

For each model instance and each subcategory, 16 formulas are automatically generated and stored into a single XML file (of which a textual version is also provided for the convenience of tool developers). Each tool participating in the corresponding subcategory is requested to evaluate, on the corresponding instance, all or part of the formulas contained in the XML file.

To obtain formulas of good quality we apply the following process for reachability and CTL formulas (see Figure 1). Using the grammar of each category, we generate 320 random formulas of up to a certain depth (7 operators) for each examination on each model instance. Then, we filter these generated formulas, in two steps to keep 16 of them:

- First, we use SAT solving to filter out formulas that are equivalent to true or false independently of the model.
- Then, we pass formulas to SMC, an ad-hoc CTL bounded model checker specifically developed for the competition. If SMC is able to decide the satisfiability of a formula by examining only the first 1000 reachable states (using BFS exploration), then we discard the formula.

If too many formulas are filtered out, we stop filtering and just wait until we reach 16 formulas by random formulas (they thus may be easy to solve) — this happens in particular for models with less than 1000 reachable states.

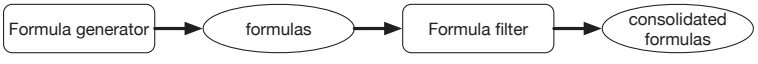


Fig. 1. Process to produce formulas in the MCC.

Let us illustrate this process on the CTL and reachability formulas produced for surprise models in 2017. 175 360 formulas were produced by the formula generator. Then, only 8 768 consolidated formulas remained after filtering (which corresponds to 5% of the original formulas). Among these consolidated formulas, 8 548 are considered to be difficult[10] (97.5%).

For LTL, the second filter is not active yet and we only discard tautologic formulas by considering that atomic propositions are not trivial; this is less efficient than the process we have set-up for other types of formulas. So, our current focus is on providing a better filtering of LTL formulas (SMC, used to check for the complexity of the computation is specialized for CTL). We also aim at generating more realistic formulas (i.e., that would be as close as possible to human-written formulas).

3 Monitoring Environment and Experimental Conditions

Operating the MCC requires to run a tool many times (once per examination per model instance). There were 54 293 runs in 2013, 83 308 in 2014, 169 078 in 2015, 128 682 in 2016, and 91 710 in 2017. To control the increasing need for CPU, we decided to process tools on a subset of the model instances of the models: no model was discarded from the benchmark but most of the instances that could be processed by more than 70% of the tools in 2016 were discarded. Each run can last up to one hour of CPU.

Such a number of executions thus requires a dedicated software environment that can take benefits from recent multi-core machines and powerful clusters. Moreover, we need to measure key aspects of computation, such as CPU or peak memory consumption, in the least intrusive way. To achieve this in an automated way, we developed *B*ench*K*it [30], a software technology (based on QEMU) for measuring time and memory during tool execution. First used during the 2013 edition, *B*ench*K*it was regularly enhanced with new capabilities: dramatic reduction of the execution overhead (we need to boot a virtual machine for every run), the possibility to specify precisely the type of emulated processor (to avoid execution problems when several families of intel-compatible processors were used), and the support of multicore virtual machine to allow concurrent executions of tools.

In 2016 and 2017, the number of runs was reduced. In 2016, most tools only submitted one variant and, in 2017, only a subset of the benchmark was selected. On the one hand, this reduction only allowed CPU needs to stay stable despite the growth of the benchmark: 1549 days in 2015, 1481 in 2016 and 1547 in 2017. The main reason is that formulas are getting smarter (especially in the reachability and CTL examinations, extra work being required for LTL), and tools gradually support more examinations. To cope with this need for CPU, we used more machines kindly lent by their owners, namely:

- bluewhale03 and bluewhale07 (respectively 40 cores @ 2.8GHz and 512GB of memory and 32 cores @3.2GHz and 1024 GB of memory) from the University of Geneva, Switzerland,

[10] Our ad-hoc CTL bounded model checker could not resolve it by exploring 1000 states.

– Caserta (96 cores @ 2.2GHz and 1024GB of memory) from the University of Twente, The Netherlands,
– Ebro (64 cores @ 2.7GHz and 1024GB of memory) from the University of Rostock, Germany,
– quadhexa-2 (24 cores @ 2.66GHz and 128 GB of memory) from the University of Paris Nanterre, France,
– small, 12 nodes (24 cores @ 2.4GHz and 64 GB of memory each) out of the 23 of a cluster of machines at Sorbonne University, France.

Of course, to preserve a sound comparison between tools, runs were divided into several consistent subsets. All runs concerning a given model (i.e., all its instances) and for all the examinations were processed on the same computer.

Post-analysis scripts aggregate data, generate summary HTML pages as well as associated charts (there are 53 118), and compute scores for the contest. They are implemented using 15 kLOC of Ada and a bit of bash. *B*ench*K*it itself consists of approximately 1 kLOC of bash.

4 Participating Tools and Experimental Results

Nine tools participated in the 2017 edition of the Model Checking Contest: GreatSPN-meddly (Univ. Torino, Italy), ITS-Tools (Sorbonne Univ., Paris, France), LoLA (Univ. Rostock, Germany), LTSmin (Univ. Twente, The Netherlands), Marcie (Univ. Cottbus, Germany), smart (Iowa State Univ., USA), Spot (Epita, Le Kremlin Bicêtre, France), TAPAAL (Univ. Aalborg, Denmark) and Tina (LAAS/CNRS and Univ. Toulouse, France). Tina submitted two variants of the tool (in that case, the rules state that only the best variant is considered for a podium).

In this section, we first summarize the global results of the contest, together with our feedback (from Sect. 4.1 to Sect. 4.3). Then, each tool getting a gold or silver medal is briefly presented (Sect. 4.4 onwards).

4.1 Examinations in the contest

Tools are confronted to several examinations: StateSpace, UpperBounds, Reachability, CTL and LTL. *StateSpace* requires the tool to compute the full state space of a specification and then provide information about it. Mandatory information concerns the number of states but tools may also provide additional information like the number of transitions, the maximum number of tokens per marking in the net and the maximum number of tokens that can be found in a place.

UpperBounds requires the tool to compute as a integer value, the exact upper bound of a list of places designated in a formula (there are 16 formulas per model instance).

Reachability, *CTL*, and *LTL* require the tool to evaluate whether formulas are satisfied or not. For each formula, we consider either state-based atomic propositions or transition-based atomic propositions (16 formulas of each type are provided per model instance). In the Reachability examination some formulas check for the existence of deadlocks.

4.2 Results – Podiums and Confidence Rate

Figure 2 shows the ranking of the three top tools in the various examinations proposed by the contest[11]. 100% represents the tool having scored the maximum points in the contest and followers' scores are expressed as percentages of this score. In the StateSpace examination, Tina.tedd was ranked second but the other variant was ranked 7^{th} out of 8 participating tools. We can also note that, for the UpperBound and CTL examinations, the distance between the third tool and its followers was rather small. In the case of CTL, these tools rely on similar symbolic CTL technology. For UpperBounds, different technologies are used but clearly provide similar performances in the case of our benchmark.

One interesting issue of the contest is to evaluate the confidence improvement over the years. To do so, we introduced in 2015 the notion of confidence rate. This value is computed for each tool on the subset of results that are considered to be sound (a majority of at least three tools providing the same value). Then, among these values, the confidence rate is the ratio between the number of correct values and the number of computed values; this ratio is then converted into a percentage.

Table 2 shows the confidence rate of the participating tools for 2017. This year, we computed this rate both separately for each examination and globally for all the examinations together. The absence of value means the tool did not participate in the examination.

Detailed results are provided at `https://mcc.lip6.fr/2017`. Let us highlight two interesting aspects: First, most errors were reported to be either in the model importation or formula importation (thus not in the verification algorithms). Some are also due to a divergence in the interpretation of some semantical points (it was the case in 2016 for some tools but no such evidence was detected in 2017). Second, over the years, tools are dramatically improving: from 89.65% in 2015, the average global confidence rate of participating tools moved to 94.20% in 2016 and then 97.34% in 2017. This can be considered as one of the major benefits of the contest for the community.

We note that importation errors are reducing year after year. This year, there was apparently a very few wrong values issued from erroneous implementation of the algorithmic heart of the tools. The lower confidence rate of smart this year is apparently due to some major changes in the architecture handled by master's students.

[11] Full results can be found at `http://mcc.lip6.fr/2017/results.php`

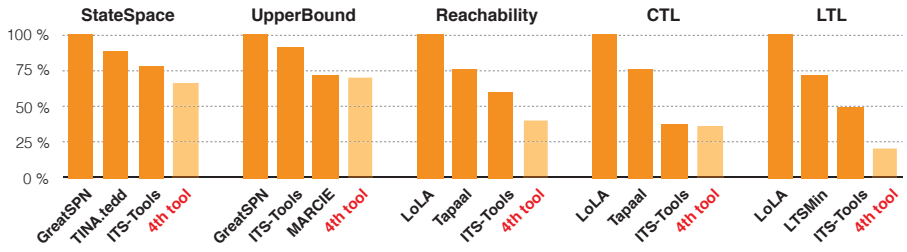


Fig. 2. Tools in the podium for each examination.

Exam.	Global	StateSpace	UpperBounds	Reachability	CTL	LTL
GreatSPN-meddly	99.13%	100%	98.89%	99.18%	99.07%	—
ITS-Tools	96.91%	100%	100%	94.68%	100%	96.33%
LoLA	99.92%	—	100%	100%	99.62%	99.97%
LTSmin	100%	100%	100%	100%	100%	100%
Marcie	100%	100%	100%	100%	100%	—
smart	79.59%	79.59%	—	—	—	—
Spot	100%	—	—	—	—	100%
TAPAAL	100%	100%	100%	100%	100%	—
Tina.sift	97.84%	97.84%	—	—	—	—
Tina.tedd	100%	100%	—	—	—	—

Table 2. Confidence rate of the participating tools in 2017.

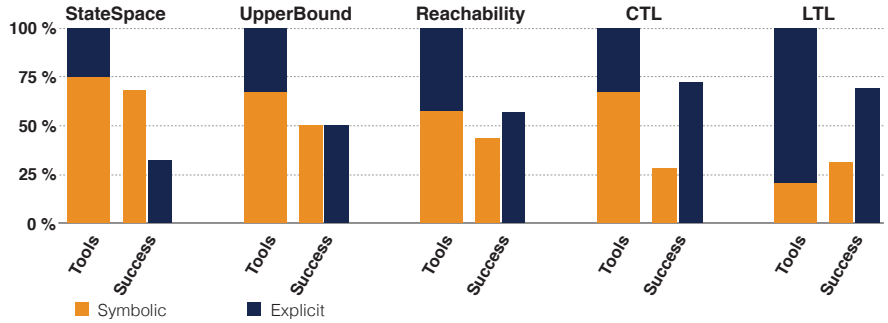
4.3 Involved Techniques

Tools developers were asked to report the techniques used by their tool. The result of this feedback is summarized in Table 3 for the three best tools in each category. Columns refer to techniques, except for the last three ones which show the type of execution: sequential, parallel in a portfolio mode (e.g., several techniques applied in parallel) or parallel (e.g., dedicated parallel algorithms).

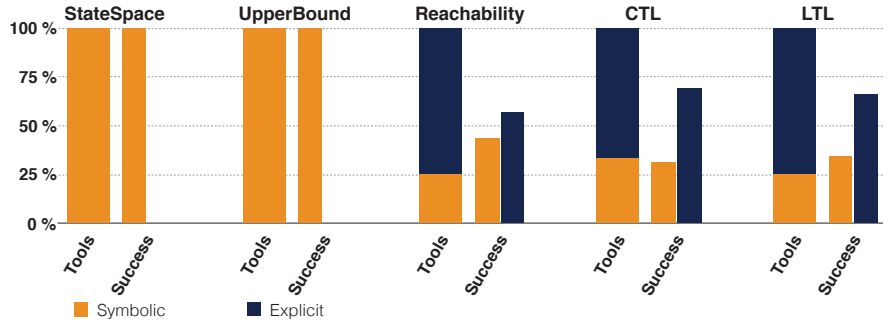
Let us note that, while the winners for StateSpace and UpperBounds examinations are based on symbolic techniques, this is not true for the other examinations where explicit approaches (enriched with optimizations like state compressions, structural reductions or partial orders) are ranked before symbolic tools. This shows that explicit approaches, together with appropriate optimizations, still compete with symbolic tools (we will show more evidences later in this section).

	tool	Dec. Diagrams	Explicit	SAT/SMT	State Compression	Struct. Reduction	Stubborn Sets	Symmetries	Topological	Unfolding to P/T	Use of NUPNs	Sequential	Portfolio	Parallel
State Space	**GreatSPN-meddly**	✓							✓	✓	✓	✓		
	Tina.tedd	✓			✓	✓			✓	✓		✓		
	ITS-Tools	✓							✓	✓	✓	✓		
Upper Bound	**GreatSPN-meddly**	✓							✓	✓	✓	✓		
	ITS-Tools	✓							✓		✓	✓		
	Marcie	✓								✓		✓		
Reach-ability	**LoLA**		✓	✓	✓		✓	✓	✓		✓		✓	
	TAPAAL		✓		✓	✓	✓						✓	
	ITS-Tools	✓	✓	✓			✓		✓		✓		✓	
CTL	**LoLA**		✓		✓		✓		✓		✓		✓	
	TAPAAL		✓		✓								✓	
	ITS-Tools	✓							✓		✓	✓		
LTL	**LoLA**		✓		✓		✓		✓		✓		✓	
	LTSmin		✓		✓						✓			✓
	ITS-Tools	✓	✓		✓		✓		✓		✓		✓	

Table 3. Techniques activated by winning tools in 2017.



(a) All participating tools



(b) Tools in the podium only

Fig. 3. Pondered analysis of symbolic and explicit tools. The first (large) bar in the charts shows the ratio between explicit and symbolic and the two (thin) ones the pondered success rate of each technique when answering examinations. 100% represents all tools or all the computed formulas.

We also note that winning tools often combine several techniques, especially in portfolio mode. This shows a complementarity between the techniques and methods. So, depending on the models and/or the formulas, the best technique varies.

We also observe that the clustering of places that is provided for some models in the NUPN format is used by many tools (it is a way to extract information about the system's structure, for example to compute a better variable order to encode the state space with decision diagrams). A deeper analysis would be of interest to evaluate its impact on tool performances (unfortunately difficult with the currently collected data).

It is globally true that tools participating for several years regularly improve by getting fine-grained inputs from the increasing benchmark of the MCC. Nevertheless, Tina.tedd was globally well ranked for a first participation. At this stage, it is difficult to get more information since most tools so far declare the technique they use on the basis of the examination they compute, and not on the result they are processing. The explicit declaration of the techniques activated result per result will be required in the next edition of the contest so that more accurate information could be gathered for analysis.

	for all tools			for podium tools		
	Reach.	CTL	LTL	Reach.	CTL	LTL
Satisfied formulas	43 599	29 946	15 251	28 921	19 512	13 521
Unsatisfied formulas	44 446	36 445	52 801	28 879	23 355	49 054
Total	88 045	66 391	68 052	57 800	42 867	62 575
Total formulas	222 488			163 242		

Table 4. Dispatching of computed formulas in 2017 by all the tools in the left part, by tools in the podium only on the right part.

The study of the 2017 results can be completed by a pondered comparison between the use of symbolic techniques compared to the use of explicit model checking declared by all tools, and those declared by the tools in the podium (see Figure 3). Since, depending on the examinations, there are sometimes more explicit tools than symbolic ones (and vice-versa), we pondered the techniques declared for correct answers according to the number of symbolic and explicit tools. In these charts, we consider only correct answers.

Figure 3(b) shows that, for StateSpace and UpperBound, tools in the podium are all symbolic. On the contrary, more tools in the podium rely on explicit techniques (in particular, ITS-Tools uses both, since it is a portfolio approach concurrently operating several techniques). However, an analysis of the results for computed formulas is necessary to refine this impression. Table 4 summarizes, for all examinations and for every categories of formulas, the number of satisfied and unsatisfied formulas. It also separates the results for all tools (including those in the podium) from those of the tools in the podium only.

We can extract several lessons concerning formulas. First, it appears that 73% of the total computed formulas are produced by the tools in the podium. Second, it appears that most of the computed LTL formulas are unsatisfied (78%). This second fact is of interest because, it is a situation where explicit tools may be advantaged. This is probably related to the fact that, so far, the quality of LTL formulas is poor compared to the one of reachability and CTL ones. We are expecting progress in tackling this issue in the 2018 edition since we are working on a smarter LTL formula generator.

A last fact, not visible in the charts and tables, should be mentioned. Symbolic techniques are often associated with some sort of structural analysis (e.g., the use of NUPN information) to determine an efficient ordering of variables in the encoding of states. Similarly, explicit tools also operate compression techniques (e.g., partial orders).

4.4 GreatSPN-meddly

GreatSPN-meddly is a symbolic model checker that is part of the GreatSPN framework [3]. The main purpose of this tool consists in building the state space of a Petri net model using Decision Diagrams (DD), in order to verify reachability and CTL properties. All DD algorithms are implemented in the Meddly library[12].

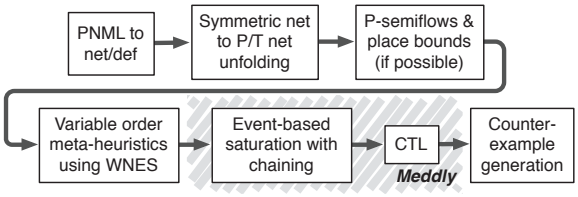
[12] Meddly library: `https://sourceforge.net/projects/meddly`.

GreatSPN-meddly operates on P/T nets with priorities, inhibitor arcs and marking dependent multiplicities. It also supports colored Petri nets through previous unfolding. A key strategy of the tool is the analysis of structural properties. P-semiflows are built before starting the state space, for the purpose of deriving place bounds and for some variable order heuristics. Unfortunately, P-semiflows cannot be built for all models (either because the model does not have any, or because it has an exponential number of them). Therefore, the availability of P-semiflows is optional. If place bounds are not known a priori, a heuristic strategy is used to guess them, with the possibility of restarting the saturation algorithm when the guessed bound is wrong. This is necessary because GreatSPN-meddly uses explicit encoding for transitions (MxD).

The tool includes a small collection of variable ordering heuristics [5], such as Force [2], Noack [23], bandwidth-reduction methods, and some methods explicitly written for GreatSPN-meddly (P-chaining and Gradient-P [4]). A meta-heuristic is also available, which computes multiple variable orders, scores them using a metric function, and selects the one with the smallest score. The tool uses a modified version of NES [5], called *Weighted* NES (WNES). The WNES value of each method is obtained from the NES score (sum of transition spans) multiplied by a method-dependent weight W. Figure 4 shows the meta-heuristics methods table, with the WNES weight W and if the algorithm requires P-semiflows. From the results of the MCC'2017 context, it appears that the new heuristic Gradient-P is very effective. Algorithm weights have been devised empirically.

The model state space is built using the Meddly library, which implements Saturation with chaining using events split by levels. Meddly employs fully-reduced DD for the state space, and identity-reduced DD for the transition relations. Once the state space is built, the tool may evaluate CTL properties on it. CTL formulas are read and evaluated one by one, in sequence. CTL evaluation may optionally produce a tree-like trace for ECTL formulas, in order to present counter-examples (or witnesses) of the evaluated formulas. In order to ensure correctness, the tool was run on thousands of formulas and models provided by the previous MCC editions on a cluster [1]. The availability of a large set of models from previous MCC editions has been used to fix various bugs in model handling (degenerate models, different interpretation of the CTL semantics for dead states, etc.).

Reported Strengths for 2017. The success of GreatSPN-meddly in the MCC'2017 edition is due to several factors. First, various improvements in the Meddly library made the Saturation algorithm performance significantly better. This improved the scalability



Method	W	P
Sloan 1/2	1.00	-
Sloan 1/16	1.15	-
Tovchigrecko/Noack	1.35	-
Cuthill-Mckee	1.01	-
P-chaining	1.00	y
Gradient-P	1.00	y
Force-P	0.50	y

Fig. 4. GreatSPN-meddly pipeline and WNES meta-heuristic table.

of the tool on many large models that could not be computed previously. Secondly, the heuristic variable order strategy is more solid, allowing to find reasonably good variable orders for many models. In particular, some algorithms (Sloan [35], Noack and Gradient-P) proved to be very effective in handling the variety of models of the MCC benchmark [5]. Experience shows that no single heuristic is good enough to treat all models, and a good meta-heuristic is needed. While reasonably good, the current meta-heuristic fails on about 25% of the models. More research on good variable ordering metrics is still needed.

Handling the 2017 "Surprise" Models. GreatSPN-meddly does not exploit precomputed data for models, therefore known and surprise models are treated alike. The rational is that the current strategy, statistically evaluated on the known models set, should be "good enough" for arbitrary models. Of course, this is not always true, and in fact the tool failed, for instance, on the BART model.

Lessons Learned from the Contest. GreatSPN-meddly was prepared for the context by running a large benchmark [5] to test the correctness of the implementation (comparing results with the previous edition values) and the effectiveness of the various heuristics implemented by the tool. This benchmark proved to be of significant value, both to understand the actual performance of many known variable order heuristics, and to test new ones (Gradient-P). The availability of the model set and the previous editions' results is therefore of great value for tool development and testing, and its value goes beyond the MCC competition.

4.5 ITS-Tools

ITS-tools [40] is a model-checker supporting both multiple solution engines and multiple formalisms using an intermediate pivot called the Guarded Action Language (GAL). Both colored models and P/T models are translated to GAL. Properties are translated conjointly with the model to properly trace atomic propositions to their target expressions in GAL.

For colored models, ITS-tools rely on PNMLFW (`http://pnml.lip6.fr/`) a Java reference implementation of the PNML standard to parse the model, then translates the model to a Parametric GAL model (which is roughly the same size as the colored model). During this process, a decomposition of the system is inferred from the color domains, trying to maximize existence of similar sub components. The parametric GAL model is then instantiated to a plain GAL model, but this process exploits binding symmetry as well as sequences of alternatives that provide compact transition representations. As a result, the GAL model obtained is often orders of magnitude smaller than an equivalent P/T (but is semantically equivalent).

For P/T models, ITS-tools embed a simple parser in SAX, then translate to a plain GAL specifications (roughly the same size as the P/T). If NUPN information is available, it also build a corresponding decomposition of the system into an *Instantiable Transition System* (ITS, which gives the tool its name).

After this translation step, ITS-tools obtain a GAL model embedding a set of properties, and possibly a decomposition of the system. At this stage, ITS-Tools operate a number of basic structural reductions before going to state-space exploration. This

is effective at removing constant marking places, detecting test arcs, redundant transitions... ITS-tools also test and simplify properties that can be decided immediately, either by looking at the initial state or when simplification yields a tautology for true or false.

Symbolic Engine. The resulting model is analyzed with our core symbolic solution engine based on Hierarchical Set Decision Diagrams (SDD) [9]. This engine exploits all the features of GAL, including hierarchy and the sequential composition semantics, so that small GAL models (e.g., as produced by our colored translation path) are often checked very efficiently. The engine benefits greatly from the model decomposition into ITS (colored or NUPN models). ITS-tools use a single variable ordering heuristic, based on Force [2]. All examinations are supported on this symbolic engine, and ITS-Tools did reach 100% confidence for answers from this solution engine.

For place bounds and reachability queries, one can simply navigate the SDD representing the full reachable state space, computed using advanced symbolic algorithms including variants of *saturation*.

For CTL, ITS-tools operate a translation to a forward CTL formula where possible, and use variants of constrained saturation to deal with EU and EG operators. ITS-Tools use a general yet precise symbolic invert to deal with predecessor relationships when translation to forward form is not feasible. Some early detection of emptiness was implemented, that helps reduce the workload for simpler formulas (there are yet many in the contest).

For LTL, ITS-tools rely on Spot [13] to translate the properties to Büchi variants, then use our SLAP hybrid algorithm [12] to perform the emptiness check. This algorithm leverages both the desirable on-the-fly properties of explicit techniques and the support for very large Kripke structures (state spaces) thanks to the symbolic SDD backend. All symbolic operations benefit from state-of-the-art saturation variants where feasible.

SAT Modulo Theory Engine. ITS-tools implements translations of the semantic bricks of GAL to SMT predicates enabling the use of solvers such as Z3 (Microsoft) or Yices2 to answer a variety of questions on the model. The encoding of GAL semantics assumes that the model is deterministic however, so the tool needs to determinize the GAL model (an operation that can be explosive for models coming from colored nets) before using the SMT solver. ITS-tools encode the constraints that reflect transition steps, as well as a basis of P-invariants computed using a classic algorithm[13]. If the net is one-bounded (presence of NUPN information) that is also encoded.

With these bricks, ITS-tools implements a bounded-model checking (BMC) decision procedure able to assert that an invariant is contradicted at some depth of exploration. This task is run in parallel with a K-Induction decision procedure able to assert that an invariant holds. These decision procedures are currently rarely competitive with the SDD engine, but they are very good at detecting structurally unfeasible behavior (i.e., K-Induction at step 1 answers many queries), and BMC works well when short counter-example traces exist. These strategies also deal with unbounded nets, though there are very few of these in the contest.

[13] Adapted from the APT tool `https://github.com/CvO-Theory/apt`.

Explicit and Partial Order Reduction Engine. ITS-tools implement a translation from GAL to PINS [29], the format used by the LTSmin tool set. This enables the use of their many verification strategies on GAL models. ITS-tools uses this translation to participate in the reachability and LTL categories only; the only engine for CTL in LTSmin is symbolic which seems redundant with respect to the SDD solution. Mostly the authors wanted to complement their symbolic methods with an explicit solution, that might deal well with models where symbolic techniques fail (e.g., due to not finding a good variable order). To be fair (it is also a competitor), LTSmin is severely limited in its resource usage. ITS-tools focused this year on enabling partial order reduction in LTSmin (POR) if possible (reachability and stuttering invariant LTL properties). To this end the SMT translation bricks were reused to compute event dependency information using the SMT solver.

Further information on the tool is available from `http://ddd.lip6.fr`.

Reported Strengths for 2017. ITS-tools are a mature toolset, able to compete in every single category of the contest including colored nets, and place on the podium in each of them. When possible, we ran the various solution engines in portfolio mode, dedicating up one thread and most memory to SDD, two threads and negligible memory for SMT, and (only) one thread with bounded memory for LTSmin.

ITS-Tools main strength remains its symbolic solution engine based on SDD, which historically held the gold in StateSpace for a long time. The SMT solution does answer some queries very fast in many cases, but these are often relatively trivial properties *in fine*. The explicit engine suitably complements the other solutions, answering many complex queries easily when POR can be activated.

Handling the 2017 "Surprise" Models. ITS-tools treat surprise models exactly like known models, except for Philosophers and SharedMemory models that were manually rebuilt to take advantage of ITS characteristics. The surprise models were handled relatively well by the SDD engine, yielding overall quite good results on this category this year.

Lessons Learned from the Contest. Unfortunately the integration of LTSmin within ITS-tools was buggy in certain cases (bad import/export) so that reliability in both Reachability and LTL was negatively impacted. These problems have been patched. In the post-analysis of the results, the authors detected some unexpected performance bottlenecks that could be solved (LTL translation, time to compute dependency matrices for LTSmin).

The competition drives the development of better and more efficient tools, certainly the authors were inspired by discussing with the other competitors and will integrate some of their ideas by next year, starting with GreatSPN's new meta-order heuristics [5].

4.6 LoLA

LoLA 2.0 is a model checker based on explicit state space exploration and structural Petri net methods. In the reachability competition, LoLA employs a portfolio consisting of state space generation (with stubborn set reduction), the state equation technique [41], a SAT based siphon/trap check [36], and a random walk procedure. In all other

categories, LoLA used standard explicit model checking algorithms. In 2017, stubborn sets were added to the LTL and CTL categories. In the deadlock category, LoLA used the symmetry reduction as well [38].

Reported Strengths for 2017. In 2017, LoLA's developers enabled all the procedures to directly cope with FIREABLE predicates (instead of translating them to place-based predicates). In consequence, LoLA won all fireability subcategories in the MCC 2017. In the CTL and reachability competitions (counting only P/T nets), the advance in the fireability subcategory was large enough to compensate a backlog in the cardinality subcategory.

Many results for LoLA (especially in the reachability category) were produced by the non-standard techniques (symmetry, state equation, siphon/trap, and random walk). This way, LoLA earned many points for solving problems. In several cases, state space search would have solved the problems, too. However, the quick answers by the non-standard procedures gained a lot of bonus points, and saved time that could be transferred to the remaining problems in an examination. LoLA used an enhanced time management for scheduling the available run time to the 16 queries of an examination.

In CTL, LoLA benefitted from its formula preprocessing. Several problems were found to be equivalent to a reachability query, so LoLA could use the much more advanced reachability portfolio. Points gained this way were just enough to stay ahead of TAPAAL.

In LTL, the model checker was completely re-implemented, for the purpose of removing the semantic discrepancies to other tools. The new model checker uses the acceptance condition proposed in [21] which permits much earlier termination in the FALSE case. In fact, about 80% of answers given by LoLA on LTL queries were FALSE answers.

Moreover, LoLA changed the translation of LTL formulas to Büchi automata. In particular, large Boolean combinations of atomic propositions are now treated as single atomic propositions. This way, a lot of translation time is saved. The resulting Büchi automata (hence, the resulting state space) becomes much smaller. These changes propelled LoLA from 70% success (in 2016) to more than 90% in 2017.

The state equation is responsible for more than 50% of the UNREACHABLE answers and quite some REACHABLE answers. Compared to 2016, the transformation of the reachability problem to a disjunctive normal form (DNF) required for the state equation was re-implemented. Instead of using the slow term rewriting system mentioned above, LoLA processes the transformation directly. In this course, LoLA added a simple static analysis that permits the detection of duplicates in the subformulas. This way, the resulting DNF is much smaller and LoLA can handle several problem instances that ran out of memory before.

Handling the 2017 "Surprise" Models. LoLA reads Petri net models in its own file format. Hence, translating the PNML [24] files of the MCC has always been an issue. Thanks to a translator provided by Silvano Dal Zilio (Toulouse), LoLA was able for the first time to process colored Petri nets. Furthermore, LoLA replaced a Python script for translating PNML to the LoLA format with a parser based on flex and bison. This way, translation time of some large net input files could be reduced from more than an hour to a few seconds. Unfortunately, the script did not anticipate all extensions of PNML, so

a translation error caused the loss of all the scores for one of the surprise nets. Finally, LoLA has a new preprocessor for formulas. Huge conjunctions and disjunctions are no longer handled by a (too slow) term rewriting system. Instead, LoLA processed these subformulas with dedicated routines. In consequence, LoLA had more time in 2017 to work on the actual verification tasks.

Lessons learned from the contest. To win a category, it is necessary to have strong verification techniques. Under the conditions of the contest, explicit techniques appear to be ahead of symbolic methods. Thanks to the on-the-fly principle, it seems to be easier for explicit model checkers to earn the low hanging fruits. LoLA ranked only few points ahead of its strongest competitors. Hence, it becomes more and more important to look into data structures and algorithms and to remove all the unnecessary computations. The contest is an excellent motivation for investing time into LoLA. LoLA benefits a lot from the increased confidence (the large benchmark identified remaining bugs) and the efforts spent on better performance. At the same time, success in the contest considerably increases the visibility of LoLA.

4.7 LTSmin

LTSmin[14] [29] has competed in the MCC since 2015. Already in the first editions, LTSmin participated in several subcategories, while since 2017 LTSmin competes in all subcategories, except for colored Petri nets, and reporting the number of fireable transitions in the marking graph.

LTSmin has been designed as a language independent model checker. This allows developers to reuse algorithms that are already implemented for other languages, such as Promela and mCRL2. For the MCC, the developers only needed to implement a PNML front-end and translate the MCC formula syntax. Improvements to the model checking algorithms, like handling integers in atomic formulas, can now in principle also be used in other languages.

LTSmin's main interface is called the Partitioned Interface to the Next-State function (PINS) [29]. Each PINS language front-end needs to implement the next-state function. It must also provide the initial state, and a dependency matrix (see below). The multi-core explicit-state and multi-core symbolic model checking back-ends of LTSmin use this information to compute the state space on the fly, i.e., new states and atomic predicates are only computed when necessary for the algorithm.

A key part of LTSmin are its dependency matrices. Dependency matrices must be precomputed statically by the front-end, and are extensively used during reachability analysis and model checking. An example Petri net, with its dependency matrix, is given in Figure 5. Here transition t_1 does not depend on p_3 or p_1 in any way. Also for properties, a dependency matrix (computed by LTSmin) indicates on which variables each atomic predicate depends. For instance, the dependency matrix of some invariant is shown in Figure 6. This invariant demonstrates LTSmin's new native property syntax. A finer analysis that distinguishes read- and write-dependencies [34] pays off, in particular for 1-safe Petri nets.

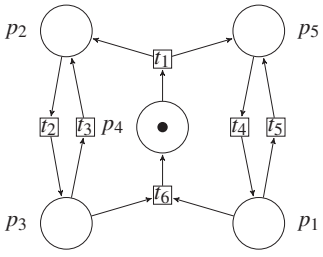
[14] http://ltsmin.utwente.nl

The MCC's reachability and CTL categories are tackled with the multi-core symbolic back-end of LTSmin, which relies on the multi-core Decision Diagram framework Sylvan [11]. Consequently, in these categories, typically billions of states in the marking graph can be explored. The LTL category is handled by the multi-core explicit-state back-end, relying on efficient parallel SCC decomposition [8].

Reported Strengths for 2017. In the reachability analysis categories, LTSmin competes using the symbolic back-end `pnml2lts-sym`, handling enormous state spaces by employing decision diagrams. However, good variable orders are essential. LTSmin provides several algorithms to compute good variable orders, which operate on the transition dependency matrix, for instance Sloan's algorithm [35] for profile and wavefront reduction. LTSmin computes the marking graph symbolically and outputs its size. To compete in the UpperBounds category, LTSmin maintains the maximum sum of all tokens in all places over the marking graph. This can be restricted to a given set of places (using, e.g., `--maxsum`=$p_1 + p_2 + p_3$). For the ReachabilityDeadlock category, the symbolic tool performs deadlock checking on the fly. Also invariant checking (the approach for ReachabilityFireability, and ReachabilityCardinality) is performed on the fly. Invariants can be specified through the `--invariant` option.

LTSmin is unique in the application of multi-core algorithms for symbolic model checking. In particular, both high-level algorithms (exploring the marking graph, and traversing the parse tree of the invariant), as well as low-level algorithms (decision diagram operations) are parallelized. This form of true concurrency allows LTSmin to benefit from the four CPU cores made available in the MCC, instead of a portfolio approach.

The approach LTSmin uses to compete in the CTL model checking categories builds upon symbolic state space generation. After the marking graph is constructed symbolically, the CTL model checking starts. LTSmin's symbolic back-end employs a straightforward μ-calculus model checker with both high-level and low-level parallelism. Similarly to invariant checking the parse tree of the μ-calculus formula is traversed in parallel, as well as the low-level decision diagram operations. LTSmin translates CTL* (a strict superset of CTL) to μ-calculus using tableaux, and evaluates the translated formula. CTL model checking in LTSmin is triggered by the `--ctl` option to `pnml2lts-sym`. For LTL model checking, LTSmin uses explicit-state model checking. So here, each reachable marking of the Petri net is considered individually. However, advanced search techniques are employed to find counter-examples as quickly as possible. In particular, the parallel SCC decomposition using a concurrent Union-Find struc-



$$
\begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \end{array}
\begin{array}{c}
\begin{array}{ccccc} p_1 & p_2 & p_3 & p_4 & p_5 \end{array} \\
\left(\begin{array}{ccccc}
0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} \\
0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\
0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & 0
\end{array} \right)
\end{array}
$$

Fig. 5. Example Model: Petri Net (left) and Dependency Matrix (right)

$$AG(1 \leq p_2 + p_3 \wedge 1 \leq p_5 + p_1) \qquad \begin{array}{c} \\ 1 \leq p_2 + p_3 \\ 1 \leq p_5 + p_1 \end{array} \begin{array}{c} \begin{array}{ccccc} p_1 & p_2 & p_3 & p_4 & p_5 \end{array} \\ \begin{pmatrix} 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \end{pmatrix} \end{array}$$

Fig. 6. Example Invariant Property and the dependency matrix on its atomic predicates

ture [8]. The multi-core explicit back-end of LTSmin constructs the cross product of the marking graph and Büchi automaton on the fly. This approach brought us at the first place in 2016, and second in 2017. LTL model checking is invoked as `pnml2lts-mc` with the `--ltl` option.

Handling the 2017 "Surprise" Models. LTSmin takes no special measures for handling the 2017 "Surprise" models. In fact, LTSmin handles both "Known", and "Stripped" as "Surprise" models. Handling models as if they are never seen before is in line with the philosophy that LTSmin should be a language independent model checker. If (precomputed) structural information of models is to be exploited, LTSmin should be able to do so for every specification language. So far structural information in Petri nets is not generalized to cover other languages too, except for the dependency matrix.

Lessons Learned from the Contest. The CTL back-end can still be improved by specializing the μ-calculus model checker. The surprise models revealed that the parallel algorithms of LTSmin are still lacking full saturation. This is expected to be improved before the 2018 edition of the MCC.

4.8 TAPAAL

TAPAAL [10] is a platform-independent tool suite for modeling, simulation and verification of Petri nets, and their timed extension called timed-arc Petri nets. The tool offers a graphical user interface for compositional design of Petri net models (the different components communicate via shared places or shared transitions), a powerful simulation and trace visualization mode, and a graphical query creation dialog that allows to call three different engines distributed together with TAPAAL: a continuous time engine `verifytapn`, a discrete time engine `verifydtapn`, and the untimed verification engine `verifypn` that participated in the model checking contests in the years 2014–2017. The tool also offers the option to translate timed-arc Petri net models into networks of timed automata and automatically call the UPPAAL engine as its back-end. It is possible to import Petri net models in the PNML format, together with the XML queries used in the model checking competition, as well as to export created nets and formulas in these exchangeable formats. TAPAAL can be downloaded at `www.tapaal.net`.

The untimed engine `verifypn` of TAPAAL performs an explicit-state model checking of weighted P/T nets with inhibitor arcs and it currently supports deadlock detection, reachability analysis of cardinality and fireability propositions, and more recently also verification of CTL formulas. TAPAAL moreover participates in the computation of upper-bounds and state-space exploration, even though these types of analysis are not in the main focus of the developers. Colored nets are not supported at the moment.

Reported Strengths for 2017. Compared to the 2016 version of the `verifypn` engine of TAPAAL (described in detail in [27]), the 2017 version comes with six main novelties: (i) a brand new successor-generator, (ii) a new data-structure for compressing and storing the state-space, (iii) advanced formula preprocessing using linear programming,

(iv) siphon-trap technique for detecting deadlock freedom, (v) improved structural reductions, and (vi) newly implemented partial order reduction for the reachability analysis.

The novelty (i) can be seen as a pure refactoring, with a focus on creating a computationally lightweight and cache-friendly way of generating successors of markings. The faster successor generator also means that larger portions of the state-space can be explored within the given time limits. Hence it reduces the higher memory consumption by compressing the encodings of markings and storing them in a novel tree-like data-structure (ii) called PTrie [28], allowing the tool to efficiently share prefixes of the encoded markings. Early experiments with PTrie were done already in 2016 where the tool developers submitted the experimental version TAPAAL-exp that significantly improved the scores of the tool. Recent improvements of PTrie made the data-structure comparable in performance with state-of-the-art hashmaps, while having a significantly lower memory footprint [28]. In 2017 both the new successor generator as well as PTrie were used also in the CTL model checking.

In 2017, some of the methods originally introduced in [27] were revisited: the heuristic search-strategy has been marginally improved but most importantly, the developers refined and generalized the over-approximation technique from [27] based on state equations and linear programming. In novelty (iii), the tool uses these principles to recursively simplify reachability and CTL formulas, often resulting in significantly smaller or even trivially true/false formulas. The linear programming approach is also used to encode the (iv) siphon-trap property [36] that allows in some cases to show that a net is deadlock free without exploring its state-space. Regarding the novelty (v), the developers generalized and extended the structural reduction rules, taking greater care of weighted arcs and inhibitor arcs. Finally, for the novelty (vi), there is a new implementation of the classical partial order reduction technique based on stubborn sets.

Handling the 2017 "Surprise" Models. As in previous years, the developers took no special measures towards the surprise models. Both known and surprise models were attempted to be solved using formula preprocessing and different search-strategies run in parallel. Timeouts for formula preprocessing and for structural reduction were introduced, limiting their execution time to about one minute. The queries that were not solved in the first parallel phase where then sequentially verified one by one until the one hour time limit was reached.

Lessons Learned from the Contest. In 2017 as well as in 2016, TAPAAL received the second place after LoLA in both the reachability and CTL categories. Given the improvements listed above, there was a hope to challenge LoLA's first place in 2017, however, due to the fact that LoLA started in 2017 to support colored nets that are equally counted in the reachability and CTL categories as the P/T nets, the margin between TAPAAL and LoLA remained similar as in 2016.

It was also realized that the improvement by introducing a stubborn set reduction was not as significant as expected because the input nets in TAPAAL are already preprocessed by structural reduction techniques. Still there is a reasonable gain in combining both techniques. Currently, TAPAAL applies stubborn set reduction only for the reachability analysis but not for CTL model checking, while LoLA introduced in 2017 stubborn sets also for CTL formulas.

Post-contest analysis revealed that a number of small bugs in the newly introduced features of the tool had a significant impact on the tool performance. Most notably a wrong ordering of places in the successor generator and too "loose" over-approximation in the formula preprocessing led to a non-negligible loss of points. As the discovered errors had no impact on the correctness of the tool, these bugs were not noticed when preparing TAPAAL for the 2017 competition.

4.9 Tina

Tina (TIme Petri Net Analyzer) [7] is a toolbox for the editing and analysis of various extensions of Petri nets and Time Petri nets, developed at LAAS-CNRS. It provides a wide range of tools for state space generation, structural analysis, model checking, editing or simulation. Except for the graphic editor, all tools of the toolbox are developed in Standard ML (SML), a high-level and modular functional programming language.

For Tina's first participation to the MCC, two tools were proposed, `sift` and `tedd`, both to compete in the *State Space* category. The `sift` tool has been part of Tina for several years; it implements state-of-the-art enumerative techniques for state space analysis of Petri nets and Time Petri nets. `tedd` is a new symbolic (logic-based) analysis tool for Petri nets, soon to be enriched to handle Time Petri nets and included in Tina.

Reported Strengths for 2017. `tedd` results from the integration of four different components: a library handling Hierarchical Set Decision Diagrams (SDD) [9], a module providing a large choice of variable order heuristics, a preprocessor (shared with `sift`) providing structural reductions, and a tool unfolding high-level colored nets into equivalent P/T nets.

Although still in development, the SDD component has shown competitive performances, on par with most of the symbolic tools present in the contest. The variable order module provides a rich choice of variable orders based on net traversals, semi-flows, flows or names. The Force [2] heuristics can be used to improve any basic order. Hierarchical orders are available but have been seldom used so far.

The single component having most contributed to Tina's results is certainly the structural reduction preprocessor. The preprocessor applies a set of rules (in the style of [6]) aimed at reducing the number of places and transitions of the net while preserving its marking count. The transformations performed consist of removing redundant places, duplicated or statically dead transitions, or transforming start places. Simultaneously, a trace information is recorded so that the number of transitions and token counts of the original net can be reconstructed from those of the reduced net. Though a number of them cannot be reduced at all, many models of the contest can be significantly simplified this way, some drastically with up to 90% of the places removed. This benefits computation of the state space by the SDD module since it has less variables to handle.

The unfolding tool for high-level nets proved convenient in most cases, but unfolding some models yielded a tremendous number of transitions that makes the approach inapplicable. Those high-level models clearly require native enumeration methods.

Handling the 2017 "Surprise" Models. After preprocessing, "Known" models are analyzed using a variable order determined experimentally; no single variable order

fits all models. For other types of models, a small number of variable orders likely to work are chosen from the experiments on "known" models. Analysis proceeds by trying to compute the state space using each of these orders for some amount of time, one after the other. These attempts could have been performed in parallel, with statically partitioned storage, but it was feared that some runs could be short of storage; instead a sequential strategy was chosen. The rationale is that if a variable order is adequate, then computation of the state space is fast and should complete in the amount of time allocated provided sufficient storage is available.

After computation of markings, it is proceeded with computation of the number of transitions. Since SDD do not require to precompute the transition relation, this does not simply amount here to compute the number of paths of some single decision diagram. Instead, an iterative algorithm is used, handling one transition at a time, with some weighting to take duplicated transitions into account. The method proved effective but, on some models, counting transitions has been measured orders of magnitude slower than building their state space. Another ad hoc algorithm has been devised to compute token counts in presence of reductions. The tokens found in redundant places are reconstructed from those found in the remaining places using the trace information of reductions. This could be done efficiently.

Lessons Learned from the Contest. Tina first participation to the contest was found costly in time. Notably, the tools had to be enriched to handle queries specific to the contest like token counts, and we had to develop from scratch a tool for handling high-level colored nets. Yet, thanks to the large number of considered models and their variety, the contest is a unique occasion to test thoroughly and strengthen Tina.

It is nice to notice that Tina, developed in a mostly functional language (which is atypical for model-checking tools), reached a competitive performance level. From the results, development of an efficient structural reduction tool revealed a wise choice. Reductions proved quite useful associated with both `tedd`, a symbolic tool, and `sift`, an enumerative one.

Tina developers conclude with some observations about execution on the virtual machine (VM) in which all tools of the contest are run. The VM does not seem to alter significantly processing times when compared with native executions. However Tina appear to be doubly penalized on storage consumption. The code compiled for Tina embeds a runtime providing automatic memory allocation with garbage collection. A first penalty is that, from the way it operates, the runtime system always requests more memory than needed by the applications. A second is that garbage collection heavily stresses the virtual memory management of the underlying machine, and that virtual memory management by the VM seems to be significantly slower than on the native machine. It was observed on some models a time overhead of over 100%, mostly due to memory management.

Authors. The main architect of the Tina toolbox is Bernard Berthomieu, with numerous insights and contributions by past and present members of the VERTICS team at LAAS, including Silvano Dal Zilio, Didier Le Botlan, François Vernadat, Alexandre Hamez and Pierre-Alain Bourdil. The SDD component of `tedd` has been written by Alexandre Hamez, also the author of `pnmc` [22]. `tedd` and `pnmc` do not share any code but rely on the same SDD technology.

5 Conclusion

This paper presented the outcomes of the 2017 edition of the Model Checking Contest, the detailed results of which can be found at http://mcc.lip6.fr/2017. The positive impact of the Model Checking Contest on the scientific community is demonstrated in at least two ways: the collection of models accumulated during the MCC is being used and cited in a growing number of publications[15], and the confidence level of the participating tools has been constantly increasing since 2015.

For the next editions of the Model Checking Contest, we plan to clarify or evolve certain rules of the contest. Such changes will be mostly based on the feedback received from tool developers. To this aim, a poll has been organized, which gathered answers from 80% of the developers of the tools that participated in the 2017 edition of the MCC. Among the most debated changes, one can mention: *(i)* a clarification of the meaning of transition counts in the StateSpace examination; *(ii)* the possibility to add new, larger instances for those models all the instances of which are simple enough to be solved by all the tools; *(iii)* the introduction, in the PNML files, of structural information about the models (e.g., structurally 1-bounded, simple free choice, strongly connected, etc.), so that tools can rely on such properties to increase efficiency by using dedicated algorithms; *(iv)* the removal of the notion of "stripped" (or "scrambled") models that was used in former editions of the MCC; *(v)* the status of colored nets with respect to P/T nets, and the way temporal-logic formulas are generated for P/T nets that are unfolded from a colored net; *(vi)* the decision whether to disclose in advance or not, to the tool developers, the temporal-logic formulas generated every year for the known models; *(vii)* the preventive measures that could be taken to avoid any bias in performance assessment that might arise from the use of virtual machines; and *(viii)* the requirement that tools should generate, in a common format to be agreed upon, counterexamples (e.g., execution traces) when a temporal formula evaluates to false — such counterexamples may be useful in the rare but definite situations where tools provide diverging answers for a given model-checking problem.

Other points of the discussion more specifically deal with the MCC scoring rules themselves. One can mention: *(ix)* provisions to ensure that parameterized models with many instances do not take excessive weight in the competition; *(x)* determination of the respective weights of known vs surprise models for the next MCC editions; *(xi)* assessment of the bonus points granted so far to the participating tools that compute fast or use less memory; and *(xii)* proposals that would provide incentives for "newcomers", i.e., participating tools that would enter the MCC competition for the first time. We expect that these various measures will noticeably improve the next editions of the MCC and will, perhaps, lead to significant changes in the future podiums.

References

1. Aldinucci, M., Bagnasco, S., Lusso, S., Pasteris, P., Vallero, S., Rabellino, S.: The Open Computing Cluster for Advanced data Manipulation (OCCAM). In: 22nd Int. Conf. on Computing in High Energy and Nuclear Physics. San Francisco (2016)

[15] See http://mcc.lip6.fr/bibliography.php

2. Aloul, F.A., Markov, I.L., Sakallah, K.A.: FORCE: a fast and easy-to-implement variable-ordering heuristic. In: ACM Great Lakes Symposium on VLSI. pp. 116–119. ACM (2003)
3. Amparore, E.G.: A new GreatSPN GUI for GSPN editing and CSLTA model checking. In: Int. Conf. on Quantitative Evaluation of Systems. pp. 170–173. Springer (2014)
4. Amparore, E.G., Beccuti, M., Donatelli, S.: Gradient-based variable ordering of decision diagrams for systems with structural units. In: 15th Int. Symp. on Automated Technology for Verification and Analysis (ATVA). LNCS, vol. 10482, pp. 184–200. Springer (2017)
5. Amparore, E.G., Donatelli, S., Beccuti, M., Garbi, G., Miner, A.: Decision diagrams for Petri nets: which variable ordering? In: Petri Net Performance Engineering conference (PNSE). pp. 31–50. CEUR-WS (2017)
6. Berthelot, G.: Transformations and decompositions of nets. In: Advanced Course on Petri Nets. pp. 359–376. Springer (1986)
7. 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)
8. Bloemen, V., van de Pol, J.: Multi-core SCC-based LTL model checking. In: Hardware and Software: Verification and Testing - 12th International Haifa Verification Conference, HVC. pp. 18–33 (2016)
9. Couvreur, J., Thierry-Mieg, Y.: Hierarchical decision diagrams to exploit model structure. In: FORTE. Lecture Notes in Computer Science, vol. 3731, pp. 443–457. Springer (2005)
10. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K., Møller, M., Srba, J.: TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In: 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12). LNCS, vol. 7214, pp. 492–497. Springer-Verlag (2012)
11. van Dijk, T., van de Pol, J.: Sylvan: Multi-core decision diagrams. In: Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS. pp. 677–691 (2015)
12. Duret-Lutz, A., Klai, K., Poitrenaud, D., Thierry-Mieg, Y.: Self-loop aggregation product - A new hybrid approach to on-the-fly LTL model checking. In: ATVA. Lecture Notes in Computer Science, vol. 6996, pp. 336–350. Springer (2011)
13. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 - A framework for LTL and \omega -automata manipulation. In: ATVA. LNCS, vol. 9938, pp. 122–129 (2016)
14. Evrard, H.: DLC: compiling a concurrent system formal specification to a distributed implementation. In: Chechik, M., Raskin, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS. Lecture Notes in Computer Science, vol. 9636, pp. 553–559. Springer (2016)
15. Evrard, H., Lang, F.: Formal verification of distributed branching multiway synchronization protocols. In: Beyer, D., Boreale, M. (eds.) Formal Techniques for Distributed Systems - Joint IFIP WG 6.1 International Conference, FMOODS/FORTE. Lecture Notes in Computer Science, vol. 7892, pp. 146–160. Springer (2013)
16. Evrard, H., Lang, F.: Automatic distributed code generation from formal models of asynchronous processes interacting by multiway rendezvous. Journal of Logical and Algebraic Methods in Programming 88, 121–153 (2017)
17. 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 (Jun 2015)
18. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: A toolbox for the construction and analysis of distributed processes. Springer International Journal on Software Tools for Technology Transfer (STTT) 15(2), 89–107 (Apr 2013)

19. Garavel, H., Lang, F., Serwe, W.: From LOTOS to LNT. In: Katoen, J.P., Langerak, R., Rensink, A. (eds.) ModelEd, TestEd, TrustEd – Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday. LNCS, vol. 10500, pp. 3–26. Springer (Oct 2017)
20. Garavel, H., Serwe, W.: The unheralded value of the multiway rendezvous: Illustration with the production cell benchmark. In: Hermanns, H., Höfner, P. (eds.) 2nd Workshop on Models for Formal Analysis of Real Systems (MARS). Electronic Proceedings in Theoretical Computer Science, vol. 244, pp. 230–270 (Apr 2017)
21. Geldenhuys, J., Valmari, A.: More efficient on-the-fly LTL verification with Tarjan's algorithm. Theor. Comput. Sci. 345(1), 60–82 (2005)
22. Hamez, A.: A symbolic model checker for petri nets: pnmc. In: Transactions on Petri Nets and Other Models of Concurrency XI, pp. 297–306. Springer (2016)
23. Heiner, M., Schwarick, M., Tovchigrechko, A.: DSSZ-MC - A Tool for Symbolic Analysis of Extended Petri Nets. In: Applications and Theory of Petri Nets. LNCS, vol. 5606, pp. 323–332. Springer Verlag (2009)
24. 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 (Oct 2009)
25. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM 21(8), 666–677 (Aug 1978)
26. ISO/IEC: High-level Petri Nets – Part 2: Transfer Format. International Standard 15909-2:2011, International Organization for Standardization – Information Technology – Systems and Software Engineering, Geneva (2011)
27. Jensen, J., Nielsen, T., Oestergaard, L., Srba, J.: TAPAAL and reachability analysis of P/T nets. LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) 9930, 307–318 (2016)
28. Jensen, P., Larsen, K., Srba, J.: PTrie: Data structure for compressing and storing sets via prefix sharing. In: 14th International Colloquium on Theoretical Aspects of Computing (ICTAC'17). LNCS, vol. 10580, pp. 248–265. Springer (2017)
29. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: LTSmin: High-performance language-independent model checking. In: Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS. pp. 692–707 (2015)
30. Kordon, F., Hulin-Hubard, F.: BenchKit, a Tool for Massive Concurrent Benchmarking. In: 14th International Conference on Application of Concurrency to System Design (ACSD'14), Tunis, Tunisia. pp. 159–165. IEEE Computer Society (Jun 2014)
31. Kordon, F., Linard, A., Buchs, D., Colange, M., Evangelista, S., Lampka, K., Lohmann, N., Paviot-Adet, E., Thierry-Mieg, Y., Wimmel, H.: Report on the Model Checking Contest at Petri Nets 2011. Transactions on Petri Nets and Other Models of Concurrency (ToPNoC) VI, 169–196 (march 2012)
32. Kordon, F., Garavel, H., Hillah, L.M., Paviot-Adet, E., Jezequel, L., Rodríguez, C., Hulin-Hubard, F.: MCC'2015 – The Fifth Model Checking Contest. Transactions on Petri Nets and Other Models of Concurrency 11, 262–273 (2016)
33. Lamport, L.: The part-time parliament. ACM Transactions on Computer Systems 16(2), 133–169 (1998)
34. Meijer, J., Kant, G., Blom, S., van de Pol, J.: Read, write and copy dependencies for symbolic model checking. In: Hardware and Software: Verification and Testing - 10th International Haifa Verification Conference, HVC. pp. 204–219 (2014)
35. Meijer, J., van de Pol, J.: Bandwidth and wavefront reduction for static variable ordering in symbolic reachability analysis. In: NASA Formal Methods - 8th International Symposium, NFM. pp. 255–271 (2016)

36. Oanea, O., Wimmel, H., Wolf, K.: New algorithms for deciding the siphon-trap property. In: 31st International Conference on Applications and Theory of Petri Nets. LNCS, vol. 6128, pp. 267–286. Springer (2010)
37. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: USENIX Annual Technical Conference (USENIX ATC). pp. 305–319. USENIX Association (2014)
38. Schmidt, K.: How to calculate symmetries of Petri nets. Acta Informaticae 36(7), 545–590 (2000)
39. Sorensen, T., Evrard, H., Donaldson, A.F.: Cooperative kernels: GPU multitasking for blocking algorithms. In: Bodden, E., Schäfer, W., van Deursen, A., Zisman, A. (eds.) 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE. pp. 431–441. ACM (2017)
40. Thierry-Mieg, Y.: Symbolic model-checking using ITS-Tools. In: TACAS. Lecture Notes in Computer Science, vol. 9035, pp. 231–237. Springer (2015)
41. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri net state equation. Logical Methods in Computer Science 8(3) (2012)