Presentation of the 9th edition of the Model Checking Contest

Elvio Amparore[1], Bernard Berthomieu[2], Gianfranco Ciardo[3], Silvano Dal Zilio[2], Francesco Gallà[1], Lom Messan Hillah[4,5], Francis Hulin-Hubard[4], Peter G. Jensen[6], Loïg Jezequel[7], Fabrice Kordon[4], Didier Le Botlan[2], Torsten Liebke[8], Jeroen Meijer[9], Andrew Miner[3], Emmanuel Paviot-Adet[4,9], Jiří Srba[6,10], Yann Thierry-Mieg[4], Tom van Dijk[9], and Karsten Wolf[8]

[1] Università degli Studi di Torino, Italy
[2] LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France
[3] Department of Computer Science, Iowa State University, USA
[4] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
[5] Université Paris Nanterre, F-92001, Nanterre, France
[6] Dept. of Computer Science, Aalborg University, Denmark
[7] Université de Nantes, LS2N, UMR CNRS 6597, F-44321 Nantes, France
[8] Institut für Informatik, Universität Rostock, 18051 Rostock, Germany
[9] Université Paris Descartes, F-75005 Paris, France
[10] FI MU, Brno, Czech Republic
[11] Formal Methods and Tools, University of Twente

Abstract. The Model Checking Contest (MCC) is an annual competition of software tools for model checking. Tools must process an increasing benchmark gathered from the whole community and may participate in various examinations: state space generation, computation of global properties, computation of some upper bounds in the model, evaluation of reachability formulas, evaluation of CTL formulas, and evaluation of LTL formulas.

For each examination and each model instance, participating tools are provided with up to 3600 seconds and 16 gigabyte of memory. Then, tool answers are analyzed and confronted to the results produced by other competing tools to detect diverging answers (which are quite rare at this stage of the competition, and lead to penalties).

For each examination, golden, silver, and bronze medals are attributed to the three best tools. CPU usage and memory consumption are reported, which is also valuable information for tool developers.

Keyword. Competition, Model Checking, State space, Reachability formulas, CTL formulas, LTL formulas

1 Introduction

The primary goal of the Model Checking Contest (MCC) is to evaluate model-checking tools that are dedicated to the formal analysis of concurrent systems,

Table 1: all the 26 tools which participated over the 9 editions of the Model Checking Contest (the 2019 edition is not yet completed when this paper is written). Years of Involvement are noted with a colored cell.

Tool	**Country**	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**
ACTIVITY-LOCAL	CH	🟨								
Alpina	CH	🟥	🟥	🟥						
Crocodile	FR	🟨	🟨							
Cunf	FR			🟥	🟥	🟥				
enPAC	CN									🟨
GreatSPN-Meddly	IT			🟥	🟥	🟥		🟥	🟥	🟥
Helena	FR	🟨	🟨		🟨					
Irma	CH								🟥	
ITS-Tools	FR	🟨	🟨	🟨		🟨	🟨	🟨	🟨	🟨
LoLA	DE	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
LTSMin	NL					🟨	🟨	🟨	🟨	🟨
M4M	CH								🟥	
MARCIE	DE		🟨	🟨	🟨	🟨	🟨	🟨		
neco	FR		🟥	🟥						
PeCAN	VN						🟨			
pnmc	FR				🟥	🟥	🟥			
PNXDD	FR	🟨	🟨	🟨	🟨	🟨	🟨			
PeTe	DK	🟥								
Sara	DE	🟨	🟨	🟨						
smart	US						🟥	🟥	🟥	🟥
Spot	FR							🟨		
Stratagem	CH				🟥	🟥				
Tapaal	DK					🟨	🟨	🟨	🟨	🟨
TINA	FR							🟥	🟥	🟥
Yaspa	CH	🟨								
ydd-pt	CH						🟥			

in which several processes run simultaneously, communicate and synchronize together. 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 and application areas.

Contributors of models to the benchmarks, tools developers, and the organizers of the MCC are actively involved in meaningful activities that foster the growth of the MCC year after year:

– they contribute to the elaboration of the benchmark by regularly providing specifications to be processed. We currently have 88 models, many having scaling parameters for a total of 951 instances;
– they enrich their tools with new features and strategies, often using inputs from previous editions of the MCC.

So far, all editions of the MCC have been using the standardized description format for Petri nets to describe the analyzed systems: PNML [24]. PNML is an ISO/IEC standard that is suitable to describe concurrent systems. The MCC could appear as being a “Petri net-oriented” competition. However, we observe that several tools coming from other communities were able to read and exploit this input format with non-Petri net-based verification engines in a very efficient way. We observe a regular arriving of new tools together with others that participate for many years. Table 1 summarizes the participating tools over the years.

This year we propose the following examinations: state space generation, computation of global properties, computation of 16 queries with regards to upper bounds in the model, evaluation of 16 reachability formulas, evaluation of 16 CTL formulas, and evaluation of 16 LTL formulas. Since such formulas are randomly generated, having severals of them to be processed reduces the possibility of a bias induced by such a generation (e.g. a single “easy” formula that would change the classification of tools) Details on organizational issues of the recent editions of the MCC are presented in [28].

Results are usually presented during the Petri Net conference, and, for the 25^{th} TACAS anniversary, during the TOOLympics. Developer teams usually submit their tools about 2 months before the presentation of results. Then, several phases are operated by the organizers:

1. the “qualification phase” that is processed on a reduced set of models and with a limited time confinement ; its objective is to check for the interoperability of tools with the execution environment so that no mistake could result in a misuse of the tool or its results;
2. the contest itself where tools are operated on the full benchmark with the real time confinement;
3. once all results are processed, they can be analyzed by the developers for last minute validity checks, this last phase ends when results are officially published.

Usually, we present the results of the MCC[12] alongside the Petri Net conference in June. For the 2019 edition of the MCC, we are joining the TOOLympics to celebrate TACAS’s 25^{th} anniversary. The goal is also to enable discussions between organizers and participants of all the verification competitions involved in this event.

2 Participating Tools

This section presents a short description of all the tools which identified the MCC as a target and were able to provide a short description of their tools and the features they will experiment this year. Note that more participating tools have shown interest in the MCC, but their authors could not provide such a description. This is the case of enPAC[13] for its first participation this year.

[12] See the full history of results on https://mcc.lip6.fr.

[13] https://github.com/tvtaqa/enPAC

2.1 GreatSPN

GreatSPN [3] is an open source framework[14] for modeling and analyzing Petri nets, which includes several tools accessible either through a common graphical interface [2], with a mode of interaction that was recently re-designed to support teaching [5], or through in-line commands for expert users. With GreatSPN the user can draw Petri nets (place/transition nets, colored nets and their stochastic variations) interactively, can compute (and visualize) the RG explicitly or symbolically, can analyze net properties (both qualitative and stochastic), can solve stochastic and Ordinary differential equations/Stochastic differential equations (ODE/SDE) systems, and can model-check CTL logic properties as well as performing stochastic model checking for properties defined using automata (the $\mathrm{CSL}^{\mathrm{TA}}$ logic), and other advanced features. Among the various tools, the framework offers a symbolic model checker called RGMEDD. This tool generates the state space of a Petri net using Multivalued Decision Diagrams (MDD) and implements a CTL model checker with counter-example generation. The implementation of the MDD data structure is based on the highly-optimized Meddly library [7].

Main features of GreatSPN The symbolic model checker of GreatSPN participates in the MCC'2019 competition for StateSpace generation, deadlock search and CTL model checking for both P/T and colored nets. The model evaluation strategy consists of the following main steps:

1. *Translation phase*: a set of support tools convert the PNML model, the NUPN[15] metadata and the CTL formulas into the internal format of GreatSPN. Colored (symmetric) models are unfolded to their corresponding P/T models.
2. *Property computation phase*: several model properties are determined, including the set of minimal P-semiflows, the basis of the P-flows, the place bounds derived from the P-semiflows and the bounds derived using an ILP solver.
3. *Variable order determination*: a set of heuristics try to determine a reasonable set of (static) variable orders for the encoding of the input model. Multiple variable orders can be used at once.
4. *State space generation*: the reachability set (RS) is generated using MDDs, employing a set of different techniques (provided by the Meddly library): event saturation with chaining, on-the-fly saturation and coverability set generation (which allows to detect infinite state spaces for a class of models).
5. *Property assessment*: deadlock and CTL properties are then determined starting from the generated RS. No attempt at model reduction or formula simplification are actually implemented.

[14] https://github.com/greatspn/SOURCES

[15] NUPN [22] stands for Nested Unit Petri Nets, this is a way to carry out structural and hierarchical information about the structure of the modeled system.

The pipeline of GreatSPN is optimized for compact MDD generation (StateSpace examinations). Currently, neither LTL nor CTL* are implemented.

Main strength of GreatSPN We believe that GreatSPN is fairly good in both building the state space (using the highly optimized saturation implementations of Meddly) and in finding a reasonably good variable order for the encoding a given model. In our experience, the variable order has a large impact on the tool performance. GreatSPN implements a large library of heuristics [6] to generate candidate variable orders, with about 30 base algorithms, plus several transformation heuristics (generate a new order given a starting one). This collection of algorithms allows the tool to generate a pool of orders among which a meta-heuristic can choose.

A second strength, again related to variable order, is the availability of a library of several *metrics*, i.e. heuristic functions that tries to evaluate the goodness of a variable order without building the MDD. Metrics are crucial in the design of reliable meta-heuristics.

In the two past years, GreatSPN was ranked gold for the StateSpace examination as wall as gold and bronze for the UpperBound examination.

New features introduced in GreatSPN for the MCC'2019 Over the last years we collected a large experience in variable order evaluation, which ended in the design of a new highly correlating metric [4] called $\mathrm{i_{Rank}}$ that will be used in the MCC'2019.

The $\mathrm{i_{Rank}}$ metric combines the count of the unique and non-productive spans of the incidence matrix (SOUPS), which accounts for places that affects many levels of the MDD, with an estimate of the number of variable dependencies to be recorded by the MDD at each single level (for a given order). This last estimate is extracted from the basis of the P-flows. The $\mathrm{i_{Rank}}$ metric shows very high correlation between its value and the final MDD size: on a test benchmark [4] $\mathrm{i_{Rank}}$ got a correlation of 0.96, while the previously best known metrics (SOUPS and PTS) had a correlation of 0.77 and 0.67, respectively. As a consequence, GreatSPN should have a very high chance of taking a good variable order as a first guess for a very large number of models. For the MCC'2019 the tool also features a new *multiple order evaluation* strategy, that selects and builds the MDD for more than one variable order, further reducing the probability of selecting a bad order.

For now the technique is sequential and single-threaded. We believe that this strategy make the tool very robust in handling general Petri net models, avoiding the risk of sticking to an unlucky (but not impossible) choice for the MDD order.

2.2 ITS-Tools

ITS-tools [34] 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. GAL models are more expressive than Petri nets, and support arithmetic and hierarchical descriptions. ITS-Tools is composed of a user-friendly front-end embedded in Eclipse, and of a command line solution back-end.

Main features of ITS-tools ITS-Tools uses a powerful symbolic solution engine based on Hierarchical Set Decision Diagrams (SDD) [16]. SDD are shared decision diagrams where edges are labeled with a *set* of values. Since decision diagrams compactly represent a set, this allows to introduce hierarchy in the representation, and enables sharing of substructures of the diagram (different edges of the SDD labeled by the same set share their representation).

While this adds a new problem (find a good decomposition of the system) to the classical problem of variable ordering in DD, in many cases there exist SDD representations that are an order of magnitude smaller than the equivalent flat DD. This allows the tool to scale to very large state space sizes. The engine benefits greatly from modular decomposition of the model, either using NUPN [22] information or inferring hierarchy automatically (using Louvain decomposition). All examinations are supported on this symbolic engine.

ITS-tools further leverages two additional solution engines: LTSmin (which is also competing) and features powerful partial order reduction methods, and a SAT modulo theory (SMT) engine currently only used for reachability queries. A new component was introduced in 2018 that performs structural reductions of the input Petri net, using variants of classical pre/post agglomerations rules.

The combination of these complementary engines helps us to solve more problems from diverse categories of models.

Main strength of ITS-Tools The main strength of ITS-tools is its overall robustness and capacity to scale well to very large state spaces. The symbolic engine, developed over the course of a decade includes state of the art data structures and algorithms, specialized and tuned for model-checking.

For place bounds and reachability queries the combination of structural reductions with three solution engines (LTSmin+POR, SDD, SMT) covers a large set of models. For CTL, ITS-tools operate a translation to a forward CTL formula when 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. The symbolic invert computes predecessor relationships, and needs to deal with models that are "lossy" (e.g. assign zero to a variable, what are the predecessor states ?). It starts with an over-approximation computed by supposing that the Cartesian product of variable domains are all reachable states. If necessary (i.e. not for Petri nets) this approximation is refined by intersecting with the reachable set of states computed forward.

For LTL, ITS-tools rely on Spot [21] to translate the properties to Büchi variants, then use the LTSmin engine (with POR) or our SLAP hybrid algorithm [20] 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 back-end. All symbolic operations benefit from state-of-the-art saturation variants when it is possible.

Over the last four editions of the MCC, ITS-Tools was always in the podium for the StateSpace examination (bronze to gold), the UpperBound examination (bronze and silver). It was 3 times in the podium (bronze and silver) for the Reachability formulas and CTL formulas (bronze and silver) as well as in the LTL formulas (bronze and silver).

New features introduced in ITS-Tools for the MCC'2019 Recent development in the tool focused on improving variable order choices, leveraging recent work by the GreatSPN team, and improving automatic decomposition heuristics (using Louvain modularity as a general scalable solution). Further developments concern improvements of the structural reductions, and integration of structural reductions at formula level for CTL leveraging recent work by Tapaal team.

Further information on the tool, as well as sources and installation procedure, is available from http://ddd.lip6.fr.

2.3 LoLA

LoLA (A Low Level Analyser, [36]) is a model checker for Petri nets. It supports place/transition nets as well as high-level Petri nets. Input may be given in a dedicated LoLA format (place/transition nets), or in the markup language PNML (both net classes). Supported properties include the temporal logics LTL and CTL as well as queries for upper bounds for token counts on places.

LoLA is an open source tool written in C++. It is being developed since 1998. It is available at http://service-technology.org/tools. It is purely based on command-line interaction and can be integrated as a backend tool for a modeling platform.

Main features of LoLA LoLA mainly uses standard explicit model checking algorithms for verification. At several stages, however, elements of Petri net theory (the state equation, invariants, siphons and traps, conflict clusters) are employed for acceleration of the algorithms. Theorems of Petri net theory are used in a portfolio approach in addition to the state space based model checking procedures wherever applicable. LoLA can apply several state space reduction methods, including partial order reduction (the stubborn set method), symmetry reduction, the sweep-line method, Karp/Miller graph construction, and bloom filtering. It has several available methods for coding markings (states).

Inputs and results can be exchanged via the UNIX standard streams. In addition, LoLA can provide results in a structured way using the JSON format.

Main strength of LoLA LoLA offers one of the largest collections of stubborn set dialects. For many classes of properties, a dedicated stubborn set method is applied. In addition, several classes of simple queries are solved using dedicated search routines instead of the generic model checking procedure. Assignment of a correct stubborn set dialect is based on a categorisation of the query. If more than one dialect is available, command-line parameters can be used to select one.

For symmetry reduction, LoLA automatically computes a generating set of the automorphism group of the Petri net graph representation. This way, the user does not need to exhibit the symmetries in the model in any way, and the largest possible number of symmetries is used. LoLA computes the symmetries such that the given query is preserved by the reduction.

LoLA uses a large set of rewriting rules for simplifying queries. It can detect more than 100 tautologies of the temporal logics LTL and CTL. Moreover, it uses the Petri net state equation, invariants, and traps for checking whether an atomic proposition is always true or false.

When the sweep-line method is applied, LoLA automatically computes a progress measure that is a pre-requisite for applying that method. The method can thus be applied in push-button style.

The way LoLA offers the reported features includes several original ideas and methods. Many reduction techniques can be combined, making LoLA one of the most competitive model checking tools for Petri nets.

New features introduced in LoLA for the MCC'2019 For the MCC'2019, LoLA will extend its portfolios for several property classes. We add an evaluation of the state equation to properties where reachability or invariance of some state predicate is sufficient or necessary for the original query. Furthermore, we shall run two state space explorations in parallel for queries where more than one stubborn set dialect is available. This way, we aim at exploiting the fact that some properties have a stubborn set dialect that performs extremely well for satisfiable queries, and another dialect that works better for unsatisfiable queries. To avoid competition between the two searches concerning memory, the search speculating for satisfaction (i.e. profiting from the on-the-fly model checking effect) gets a severe memory restriction. It will be killed upon overflow on the assigned memory thus eventually leaving all available memory to the search that speculates for violation (and needs to explore all of the reduced state space).

2.4 LTSmin

LTSmin[16] [27] 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.

For the MCC of this year, LTSmin only competes in the StateSpace and UpperBounds categories, as the tool is now equipped with a fully parallelized symbolic saturation algorithm for computing the state space. Otherwise the tool is relatively unchanged compared to last year, so we restrict the tool to only demonstrate the new techniques.

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

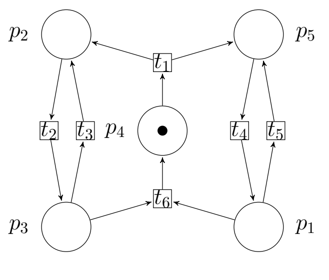
Main features of LTSmin LTSmin has been designed as a language independent model checker. This allowed us to reuse algorithms that were already used for other languages, such as Promela and mCRL2. For the MCC, we only needed to implement a PNML front-end and translate the MCC formula syntax. Improvements 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) [27]. 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 1. 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 2. This invariant demonstrates LTSmin's native property syntax. A finer analysis that distinguishes read- and write-dependencies [30] pays off, in particular for 1-safe Petri nets.

Main srength of LTSmin LTSmin competes using the symbolic back-end http://ltsmin.utwente.nl/assets/man/pnml2lts-sym.html, 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 [31] 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`$=p_1+p_2+p_3$).

LTSmin is unique in the application of multi-core algorithms for symbolic model checking. In particular, both high-level algorithms (exploring the mark-



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

Fig. 1: Example Model: Petri Net (left) and Dependency Matrix (right)

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

Fig. 2: Example Invariant Property and the dependency matrix on its atomic predicates

ing 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.

New features introduced in LTSmin for the MCC 2019 LTSmin is now equipped with a fully multi-core on-the-fly symbolic saturation algorithm as described in [19]. Saturation is an efficient exploration order for computing the set of reachable states symbolically. In the past, attempt to parallelize saturation only resulted in limited speedup. LTSmin now implements on-the-fly symbolic saturation using the Sylvan multi-core decision diagram package. Using the benchmarks of the MCC, we demonstrate in [19] speedups of around 3× with 4 workers, which is the configuration used in the MCC. For some models we obtained superlinear speedups, even scaling to a machine with 48 cores.

2.5 SMART

SMART (Stochastic Model-checking Analyzer for Reliability and Timing) is an open-source[17] software package to study complex discrete-state systems. The SMART input language supports different types of models (non-deterministic or stochastic), described in various ways (including Petri nets, Markov chains, and Kripke structures) and with various types of queries (including temporal logic, probabilistic temporal logic, and performance measures). Models are analyzed with various back-end solution engines, including explicit and symbolic model checking, numerical solution for probabilistic model checking, and simulation.

Main features of SMART SMART supports Petri nets with inhibitor arcs, transition guards, and marking-dependent arc cardinalities. The input language allows users to define arrays of places and transitions, for building large Petri nets with regular structure. A specialized translation tool is used to convert PNML models into the SMART input language.

For symbolic model checking, SMART uses Multivalued Decision Diagrams (MDDs). Each Petri net place (or perhaps a set of places) is mapped to a single MDD variable. A heuristic is used to determine a variable order, as the choice of variable order is often critical to MDD efficiency. On-the-fly saturation [15] is used to generate the state space, with MDDs used to encode sets of states, and an appropriate representation used for the transition relations (either Matrix Diagrams [32] or a specialized implicit representation).

[17] https://smart.cs.iastate.edu.

Main strength of SMART SMART uses MEDDLY: Multivalued and Edge-valued Decision Diagram LibrarY [7], an open-source MDD library[18], as its symbolic model checking engine. The library supports MDDs, EV^+MDDs, and EV^*MDDs natively, along with manipulation algorithms needed for CTL model checking, including saturation. Saturation is often orders of magnitude faster than a traditional breadth-first iteration for building the state space. Constrained saturation [37] can be used for efficient CTL model checking.

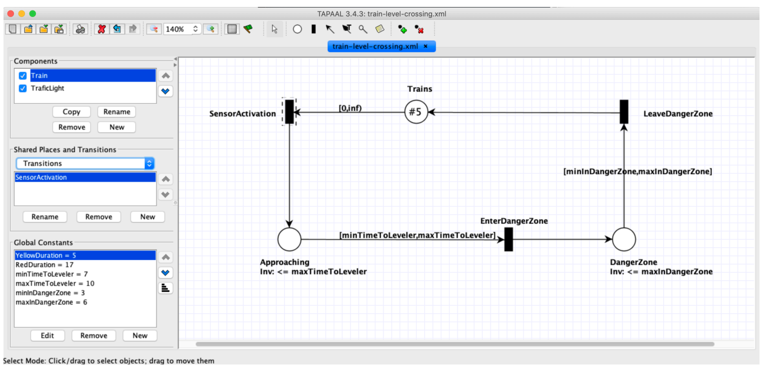
New features introduced in SMART for the MCC 2019 Gradual improvements have been made to SMART and MEDDLY over many years. The recent focus has been on eliminating bottlenecks caused by construction and storage of the Petri net transition relations, by moving to a more efficient representation utilizing implicit nodes that do not need to be updated as the state space grows. Another important area of research has been variable ordering heuristics, including the development of SOUPS (sum of unique productive spans) [33] as an estimate for the cost of transition firings.

2.6 TAPAAL

TAPAAL [18] is a tool suite for verification of Petri nets and their extensions where tokens can be associated with timing features (timed-arc Petri net model) or with colors (colored Petri nets in the style supported by MCC competition). The acronym TAPAAL stands for Timed-Arc Petri nets developed at AALborg university. The tool development started 10 years ago and the current tool release consists of a powerful GUI providing user-friendly, compositional editing of Petri nets (see screenshot at Figure 3) and a number of standalone verification engines supporting CTL verification of colored (untimed) Petri nets, reachability analysis of timed-arc Petri nets with discrete time semantics, including a workflow analysis tool, and a continuous time verification engine. The tool suite also allows to export timed-arc Petri nets as timed automata that can be opened and verified in the tool UPPAAL [8]. TAPAAL supports the import and export of Petri nets in the PNML standard, including the parsing of queries in the XML standard introduced by the MCC competition. The currently released version 3.4.3 of TAPAAL (available at www.tapaal.net) won two gold medals in the reachability and CTL category and one silver medal in upper-bounds at MCC'18.

Main features of TAPAAL The colored (untimed) verification engine of TAPAAL called `verifypn` [25] is the one that participates at MCC and it relies on a preprocessing of both the Petri net model as well as the verified query. The subformulas of CTL queries are recursively analysed [12] with the use of state-equations and linear programming techniques in order to discover easy-to-solve subqueries to reduce the overall query size. This allows us to answer

[18] https://asminer.github.io/meddly/



YellowDuration = 5
RedDuration = 17
minTimeToLeveler = 7
maxTimeToLeveler = 10
minInDangerZone = 3
maxInDangerZone = 6

Fig. 3: Screenshot of TAPAAL GUI

the query on a substantial number of models without even exploring the state-space (in fact 22% of all CTL propositions in MCC'17 can be answered solely by the approximation techniques [12]), or it can reduce a CTL query into a pure reachability question on which a specialized verification method can be used (about 50% of the MCC'17 queries can be reduced to simple reachability [12]). At the same time, the Petri net model is reduced by a continuously updated set of structural reduction rules so that we can eliminate uninteresting concurrency or unimportant parts of the net (relative to the asked query).

As documented in [13], a substantial reduction in the net size can be achieved and this technique often combines well with partial order techniques based on stubborn sets that are as well implemented in TAPAAL. The tool also supports siphon-trap based algorithms for the detection of deadlock freedom. More recently a trace abstraction refinement [14] has been added to the engine and employed at MCC'18. The verification of colored Petri nets is achieved by a self-contained unfolder implemented in the `verifypn` engine in combination with over-approximation by state-equations applied for a given query on the colored net before its unfolding. Furthermore the tool employs a state-of-the-art dependency-graph technique for verification of CTL properties, utilizing the so-called certain zero optimization [17]. Another critical component is an efficient successor-generator, which paired with a compressing, time and memory-efficient state-storage data structure PTrie [26] gives `verifypn` a competitive advantage in MCC.

Main strength of TAPAAL The success of `verifypn` at MCC'18 can to a large degree be attributed to the plethora of techniques with an efficient, internalized implementation. Among the most beneficial techniques are over-approximation of colored Petri nets, structural reductions combined with stubborn set reductions, recursive query simplification algorithms and a symbolic method in the form of trace abstraction refinement. Furthermore, efficiency of the basic components of the successor-generator and the state-compression techniques provide a backbone of the tool.

New features introduced in TAPAAL for the MCC'2019 The main focus for 2019 is the expansion and generalization of net reduction-rules implemented in `verifypn`. Furthermore, discussions with the developers of the tool LoLA provided an inspiration to further optimizations for the upper-bounds category, utilizing linear over-approximation of place bounds.

2.7 TINA.tedd

TINA (TIme Petri Net Analyzer) [11] 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, or simulation. For its third participation to the MCC, we selected a single symbolic tool from TINA, called `tedd`, to compete in the *StateSpace* category.

Main features of TINA.tedd We provide a new symbolic analysis tool for Petri nets that uses a mix between logic-based approaches (decision diagrams); structural reductions; and a new symbolic technique where sets of markings are represented using linear systems of equations. We give more details about the latter below.

At its core, `tedd` is a symbolic state-space generation tool built on top of our own implementation of Hierarchical Set Decision Diagrams (SDD) [16]. In the context of the MCC, we can use state space generation to answer all the questions from the StateSpace examination, that is computing the number of markings of an input net; its number of transitions; and the maximum number of tokens in a marking and in a place. The core capabilities of `tedd`, based on SDD, has shown competitive performances, on par with most of the symbolic tools present in the previous MCC contests.

The tool can accept models in the PNML format and provides a large collection of options for selecting good variable orders. A variable order module provides a rich choice of order computing algorithms based on net traversals and the structural analysis of the net (semi-flows, flows, etc.). In each case, a force [1] heuristics can be used to improve any given order. Hierarchical orders, which are a characteristic of SDD, are also available but have been seldom used so far. An order-grading method allows to choose for each model a variable ordering likely to work. Colored models are also supported using a separate tool that unfolds high-level nets into simpler PT nets. `tedd` also provides some limited support for reachability properties – such as finding dead states and transitions – and can be used with other decision diagrams libraries. At the moment, we provide access to a BDD library for safe nets as well as to the `pnmc` tool [23].

Main strength of TINA.tedd What sets `tedd` apart from other symbolic tools in the competition is the use of a novel state space generation technique based on structural reductions coupled with methods for counting sets of markings. This approach is explained in detail in [10] and was first experimented during MCC'2018, using a limited set of reductions. To the best of our knowledge, `tedd` is the only tool implementing such an approach.

Briefly, we enrich the notion of structural reduction (as found e.g. in [9]) by keeping track of the relationships between the reachable markings of an (initial) Petri net, N_1, and those of its reduced (final) version, N_2, using a system of linear equations with integer coefficients Q. We call Q a set of *reduction equations*.

We provide an automatic system for finding and applying structural reductions on nets. Our reductions are tailored so that the state space of N_1 can be faithfully reconstructed from that of N_2 and the reduction equations, Q. Intuitively, variables in Q include the places found in N_1 and N_2. The reduction equations provide a relation between markings of the two nets in the following sense: when setting variables in Q to values given by a marking of N_2, the set of non-negative, integer solutions to the resulting system all correspond to markings of N_1 (and reciprocally).

Reductions can be applied in succession, giving an irreducible net N_i and a final set of reduction equations Q_i. In particular, when N_i is totally reduced (N_i has an empty set of places), the state space of N_1 corresponds exactly to the solutions of system Q_i. In some sense, Q_i acts as a symbolic, "equational", representation of the reachable markings. This gives a very compact representation of the set of reachable markings that manages to preserve good complexity results for a large class of problems (e.g. finding whether a given marking is reachable, or computing the maximal possible marking for a place). This approach can also be useful when a net is only partially reduced. Then the markings of the residual net is computed symbolically (as an SDD) which, together with the reduction equations, provides a hybrid representation of the reachable markings of N.

New features introduced for the MCC'2019 This is the first edition of the MCC where we will experiment with an extended set of reductions improving upon those presented in [10]. With these new reductions, we are able to totally reduce about 25% of all the instances used in the previous MCC benchmarks. More generally, reductions have a positive impact on about half of the instances. In particular, based on the benchmark provided by the MCC'2018, we are able to compute results for 22 new instances for which no tool was ever able to compute a marking count during the competition.

This year, we will also use more powerful methods for counting markings in the case of totally reducible nets. Indeed, when a net is totally reduced, it is enough to compute the number of non-negative integer solutions to its system of reduction equations. There exist several methods for solving this problem, which amounts to computing the number of integer points in a convex polytope. Actually, this has been a hot topic of research in the last decade and some tools are purposely available for this task, notably `LattE` [29] and `barvinok` [35].

`TINA.tedd` is able to interface with these tools in order to compute the number of reachable markings in a net. With some adjustments, implemented in `tedd`, similar techniques can also be used for computing the number of transitions.

This approach, based on "geometric techniques", is extremely powerful when the number of variables – the number of places in the net – is low. (We restrict its use to nets with less than 50 places in the MCC.) To overcome this limitation, we have recently started experiments with our own counting method, based on a recursion-solving approach, and implemented in a dedicated package called `polycount`. The work on polycount is still preliminary though, since we cannot count transitions in all cases yet.

3 Conclusion

Through the dedication of its related communities, the Model Checking Contest has been achieving the following objectives over the past decade:

– gathering a large set of diverse, complex, and centralized benchmarks composed of concurrent systems formal descriptions;
– providing the environments and frameworks for the emergence of systematic, rigorous and reproducible means to assess model-checking tools;
– fostering the progress of the research and development efforts of model-checking tools to increase their capabilities.

Like the other existing scientific contests, the MCC also aims at identifying the theoretical approaches that are the most fruitful in practice, to possibly enable the research field to figure out which techniques, under specific conditions, best handle particular types of analyses on the systems under consideration.

As the MCC gains maturity, existing organizational challenges shift focus, and new ones appear. Among the former, is how to appropriately increase the benchmark for known models whose current instances are now easily solved, along with the way we generate temporal-logic formulas for them. Another one is creating the provisions to better balance the scoring weights between parameterized models (with regards the number of instances deduced from their scaling parameter) on the one hand, between known and surprise models on the other. Among the new challenges, is what incentives the competition could provide to keep attracting newcomers, i.e., first-time participating tools. Another one is the inclusion of some known verdicts of all previous analyses on each instance of the existing models during the past editions, and allow the competing tools to reliably access this information to speed up and increase efficiency in new analyses.

Finally, we observed a dramatic increase of the tool's confidence (and probably reliability) since this measure was introduced in 2015. Between two editions of the MCC, previous results are used as a testbench for increasing the quality of these tools and developers even exchange their tricks and algorithms. Therefore, we can state that this event benefits to the whole community.

References

1. 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)
2. Amparore, E.G.: A New GreatSPN GUI for GSPN Editing and CSL^{TA} Model Checking. In: QEST. pp. 170–173. Springer (2014)
3. Amparore, E.G., Balbo, G., Beccuti, M., Donatelli, S., Franceschinis, G.: 30 Years of GreatSPN, chap. In: Principles of Performance and Reliability Modeling and Evaluation: Essays in Honor of Kishor Trivedi, pp. 227–254. Springer, Cham (2016)
4. Amparore, E.G., Ciardo, G., Donatelli, S., Miner, A.: i_{Rank}: a variable order metric for DEDS subject to linear invariants. In: TACAS'2019. Lecture Notes in Computer Science, Springer (2019)
5. Amparore, E.G., Donatelli, S.: GreatTeach: A tool for teaching (stochastic) Petri nets. In: ATPN'2018. Lecture Notes in Computer Science, vol. 10877, pp. 416–425. Springer (2018)
6. Amparore, E.G., Donatelli, S., Beccuti, M., Garbi, G., Miner, A.: Decision diagrams for Petri nets: A comparison of variable ordering algorithms. Transactions on Petri Nets and Other Models of Concurrency XIII pp. 73–92 (2018)
7. Babar, J., Miner, A.S.: Meddly: Multi-terminal and Edge-valued Decision Diagram LibrarY. In: Proc. QEST. pp. 195–196. IEEE Comp. Soc. Press (2010)
8. Behrmann, G., David, A., Larsen, K.G., Pettersson, P., Yi, W.: Developing UPPAAL over 15 years. Software - Practice and Experience 41(2), 133–142 (2011)
9. Berthelot, G.: Transformations and decompositions of nets. In: Advanced Course on Petri Nets. pp. 359–376. Springer (1986)
10. Berthomieu, B., Le Botlan, D., Dal Zilio, S.: Petri net reductions for counting markings. In: International Symposium on Model Checking Software. LNCS, vol. 10869, pp. 65–84. Springer (June 2018)
11. 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)
12. Boenneland, F., Dyhr, J., Jensen, P., Johannsen, M., Srba, J.: Simplification of CTL formulae for efficient model checking of Petri nets. In: Proceedings of the 39th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets'18). LNCS, vol. 10877, pp. 143–163. Springer-Verlag (2018)
13. Bønneland, F., Dyhr, J., Jensen, P., Johannsen, M., Srba, J.: Stubborn versus structural reductions for Petri nets. Journal of Logical and Algebraic Methods in Programming 102, 46–63 (2019)
14. Cassez, F., Jensen, P.G., Guldstrand Larsen, K.: Refinement of trace abstraction for real-time programs. In: Hague, M., Potapov, I. (eds.) Reachability Problems. LNCS, vol. 10506, pp. 42–58. Springer International Publishing, Cham (2017)
15. Ciardo, G., Marmorstein, R., Siminiceanu, R.: Saturation unbound. In: Proc. TACAS. pp. 379–393. LNCS 2619 (2003)
16. 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)
17. Dalsgaard, A., Enevoldsen, S., Fogh, P., Jensen, L., Jensen, P., Jepsen, T., Kaufmann, I., Larsen, K., Nielsen, S., Olesen, M., Pastva, S., Srba, J.: A distributed fixed-point algorithm for extended dependency graphs. Fundamenta Informaticae 161(4), 351–381 (2018)

18. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K., Møller, M., Srba, J.: TAPAAL 2.0: Integrated development environment for timed-arc Petri nets. In: Proceedings of the 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)
19. van Dijk, T., Meijer, J., van de Pol, J.: Multi-core On-the-fly Saturation. In: Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS (2019)
20. 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)
21. 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)
22. 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)
23. 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)
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. 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)
26. Jensen, P., Larsen, K., Srba, J.: PTrie: Data structure for compressing and storing sets via prefix sharing. In: Proceedings of the 14th International Colloquium on Theoretical Aspects of Computing (ICTAC'17). LNCS, vol. 10580, pp. 248–265. Springer (2017)
27. 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)
28. Kordon, F., Garavel, H., Hillah, L.M., Paviot-Adet, E., Jezequel, L., Hulin-Hubard, F., Amparore, E., Beccuti, M., Berthomieu, B., Evrard, H., Jensen, P., Le Botlan, D., 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 (ToPNoC) XIII, 181–209 (2018)
29. Loera, J.A.D., Hemmecke, R., Tauzer, J., Yoshida, R.: Effective lattice point counting in rational convex polytopes. Journal of Symbolic Computation 38(4) (2004)
30. 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)
31. 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)
32. Miner, A.S.: Implicit GSPN reachability set generation using decision diagrams. Perf. Eval. 56(1-4), 145–165 (Mar 2004)

33. Smith, B., Ciardo, G.: SOUPS: a variable ordering metric for the saturation algorithm. In: Proc. International Conference on Application of Concurrency to System Design (ACSD). pp. 1–10. IEEE Comp. Soc. Press (Jun 2018)
34. Thierry-Mieg, Y.: Symbolic model-checking using ITS-Tools. In: TACAS. Lecture Notes in Computer Science, vol. 9035, pp. 231–237. Springer (2015)
35. Verdoolaege, S., Seghir, R., Beyls, K., Loechner, V., Bruynooghe, M.: Counting integer points in parametric polytopes using barvinok's rational functions. Algorithmica 48(1), 37–66 (2007)
36. Wolf, K.: Petri net model checking with lola 2. In: Khomenko, V., Roux, O.H. (eds.) Application and Theory of Petri Nets and Concurrency - 39th International Conference, PETRI NETS 2018, Bratislava, Slovakia, June 24-29, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10877, pp. 351–362. Springer (2018), https://doi.org/10.1007/978-3-319-91268-4_18
37. Zhao, Y., Ciardo, G.: Symbolic CTL model checking of asynchronous systems using constrained saturation. In: Proc. ATVA. pp. 368–381. LNCS 5799, springer (2009)