Report on the Model Checking Contest at Petri Nets 2011

Fabrice Kordon[1], Alban Linard[2], Didier Buchs[2], Maximilien Colange[1], Sami Evangelista[3], Kai Lampka[4], Niels Lohmann[5], Emmanuel Paviot-Adet[1], Yann Thierry-Mieg[1], and Harro Wimmel[5]

[1] LIP6, CNRS UMR 7606, Université P. & M. Curie – Paris 6
4, place Jussieu, F-75252 Paris Cedex 05, France
{Fabrice.Kordon,Maximilien.Colange,
Emmanuel.Paviot-Adet,Yann.Thierry-Mieg}@lip6.fr

[2] Centre Universitaire d'Informatique, Université de Genève
7, route de Drize, CH-1227 Carouge, Switzerland
{Alban.Linard,Didier.Buchs}@unige.ch

[3] LIPN, CNRS UMR 7030, Université Paris 13
99, av. J-B Clément, 93430 Villetaneuse, France
sami.evangelista@lipn.univ-paris13.fr

[4] Department of Information Technology, Uppsala University, Sweden
kai.lampka@it.uu.se

[5] Universität Rostock, 18051 Rostock, Germany
niels.lohmann@uni-rostock.de, Wimmel@Informatik.Uni-Oldenburg.de

Abstract. This article presents the results of the Model Checking Contest held within the SUMo 2011 workshop, a satellite event of Petri Nets 2011. This contest aimed at a fair and experimental evaluation of the performances of model checking techniques applied to Petri nets.

The participating tools were compared on several examinations (state space generation, deadlock detection and evaluation of reachability formulæ) run on a set of common models (Place/Transition and Symmetric Petri nets). The collected data gave some hints about the way techniques can scale up depending on both examinations and the characteristics of the models.

This paper also presents the lessons learned from the organizer's point of view. It discusses the enhancements required for future editions of the Model Checking Contest event at the Petri Nets conference.

Keywords: Petri Nets, Model Checking, Contest.

1 Introduction

When verifying a system with formal methods, such as Petri nets, one may have several questions such as:

> "When creating the model of a system, should we use structural analysis or an explicit model checker to debug the model?"

> "When verifying the final model of a highly concurrent system, should we use a symmetry-based or a partial order reduction-based model checker?"
>
> "When updating a model with large variable domains, should we use a decision diagram-based or an abstraction-based model checker?"

Results that help to answer these questions are spread among numerous papers in numerous conferences. The choice of the models and tools used in benchmarks is rarely sufficient to answer these questions. Benchmark results are available a long time after their publication, even if the computer architecture has changed a lot. Moreover, as they are executed over several platforms and composed of different models, conclusions are not easy.

The objective of the Model Checking Contest @ Petri nets is to compare the efficiency of verification techniques according to the characteristics of the models. To do so, the Model Checking Contest compares tools on several classes of models with scaling capabilities, *e.g.*, values that set up the "size" of the associated state space.

Through a benchmark, our goal is to identify the techniques that can tackle a given type of problem identified in a "typical model", for a given class of problem (*e.g.*, state space generation, deadlock detection, reachability or causal analysis, etc.).

The first edition of the Model Checking Contest @ Petri nets took place within the context of the SUMo workshop (International Workshop on Scalable and Usable Model Checking for Petri Nets and other models of concurrency), co-located with the Petri Nets and ACSD 2011 conferences, in Newcastle, UK. The original submission procedure was published early March 2011 and submissions gathered by mid-May 2011. After some tuning of the execution environment, the evaluation procedure was operated on a cluster early June. Results were presented during the SUMo workshop, on June 21st, 2011[1].

Let us mention similar events we are aware of. The *Hardware Model Checking Contest*[2] started in 2007 focuses on circuit verification by means of model checking. It is now associated with the CAV (Computer Aided Verification) and FLOC (Federated Logic Conference) conferences. This event ranks the three best tools according to a selected benchmark. It is an almost yearly event (2007, 2008, 2010 and 2011).

The *Timing Analysis Contest*[3] within PATMOS 2011 (International Workshop on Power and Timing Modeling, Optimization and Simulation) also considers the verification of electronic designs with a focus on timing analysis.

The *Verified Software Competition*[4] hold within the Verified Software: Theories, Tools and Experiments (VSTTE) conference [24], in August 2010. This

[1] This presentation can be found at `http://sumo.lip6.fr/MCC-2011-report/MCC-2011-report.pdf`, and raw data of the benchmarks at `http://sumo.lip6.fr/MCC-2011-report/MCC-results-data.zip`

[2] `http://fmv.jku.at/hwmcc11/index.html`

[3] `http://patmos-tac.inesc-id.pt`

[4] `http://www.vscomp.org`

competition was held as a forum where researchers could demonstrate the strengths of their tools through the resolution of five problems. The main objective of this event was to evaluate the efficiency of theorem proving tools against SAT-solving.

The *Satisfiability Modulo Theories Competition*[5] takes place within the context of the CAV conference. Since 2005, its objective is to evaluates the decision procedures for checking satisfiability of logical formulas.

Finally, the *SAT Competitions*[6] proposes to evaluate the performance of SAT solvers. This event occurs yearly since 2002 and identifies new challenging benchmarks each years.

With respect to these existing events, the Model Checking Contest at Petri Nets puts emphasis on the specification of parallel and distributed systems and their qualitative analysis. So far, we consider Petri Nets as input specification (later editions might also consider other formalisms suitable for concurrency).

The goal of this paper is to report the experience from this first Model Checking Contest. It reflects the vision of the MCC'2011 organizers, as it was first presented aside the conferences in Newcastle, revised and augmented by feedback from the tool developers who participated in this event. All tool developers are listed in Section 9.

The article is structured as follows. Section 2 presents the evaluation methodology, before a brief presentation of the models in Section 3 and the participating tools in Section 4. Then, Sections 5 to 7, detail some observations we made about the efficiency of techniques with regards to their implementation in the participating tools. Finally, Section 8 discusses some issues risen by this first edition of the Model Checking Contest as well as some clues for the organization of next editions.

All over the paper, we outline in this way the lessons learned from the first edition. These lessons constitute changes to be applied in the next edition of this event.

2 Evaluation Methodology

The Model Checking Contest is based on one major assumption: there is no silver bullet in model checking. Thus, the choice of the techniques should depend on:

- the characteristics of the model, for instance its formalism (Place/Transition net or Symmetric net), the marking bounds (safe or unsafe nets), or the number of synchronizations in a model;
- the action to be performed by the model checker, for instance state space generation, deadlock detection, or evaluation of reachability, CTL or LTL formulæ;
- the possible interaction between techniques, for instance abstractions with partial orders, or decision diagrams with symmetries;
- the position in the development process, for instance when the model is being designed, during its debugging, or during its final checking.

[5] `http://www.smtcomp.org`
[6] `http://www.satcompetition.org`

There is already plenty of publications on theoretical complexity of model checking algorithms. Theoretical complexity is sometimes misleading, as it can hide huge constants, that make algorithms unusable in practice. Moreover, the efficiency of tools varies a lot, even when they use the same techniques.

An experimental evaluation of tools efficiency is thus required in model checking. Articles about tools also provide benchmarks that have numerous drawbacks [22]:

- They usually cover only some selected tools on some selected models. Thus, benefits of involved algorithms for some model characteristics cannot be evaluated.
- The choice of the tools used for comparison is sometimes biased, because the authors may not know other good competing tools, or because they could not convert their models to process them.
- There is no guarantee that the comparison is fair, because the authors of the article may not know other tools as well as their own tool. For instance, some settings can require some expertise to be set appropriately.
- As benchmarks are performed on several architectures, they also cannot be compared between articles.

2.1 The Overall Procedure

For the first edition of the Model Checking Contest, only a subset of all actions provided by model checkers were requested. They are called "examinations":

- computation of the state space with a report on its size;
- computation of the deadlocks with a report on their number;
- evaluation of reachability formulæ to detect wether a state, depicted by a logic formula, can be reached from the initial state or not (10 satisfiable ones and 10 unsatisfiable ones) with a report on the computed results (true or false).

Examinations were run on several parameterized models. Each model has a "scaling value", used to increase its complexity, for instance the number of philosophers in the Dining Philosophers problem. For each model, each scaling parameter value defines a *model instance*. The Model Checking Contest provided all model instances in Petri Net Markup Language format. As tools were allowed to give their own version of the models, as long as examinations return the same results, we fixed all scaling values before the contest, and provided them in advance to the participants.

Model checking has two "enemies": memory consumption when it comes to store large state spaces (or portion of state space) and computation time, when the number of states grows. There is usually a trade-off between lower memory footprint and lower computation time. Thus, our objective was to measure both memory and CPU usage.

The "examinations" requested for the contest were performed thanks to an invocation script that iterated invocation of each tool over models instances. This invocation script is presented in Algorithm 1.

```
Input: $M$, a set of scalable models to be processed
foreach $m \in M$ do
    Operate a prologue for $m$
    foreach $v$, scaling values for $m$ do
        Operate state space generation on $m$ for scale value $v$
        Operate deadlock computation on $m$ for scale value $v$
        Operate check of satisfiable reachability formulæ on $m$ for scale value $v$
        Operate check of unsatisfiable reachability formulæ on $m$ for scale value $v$
    Operate an epilogue for $m$
```

Algorithm 1. Actions performed for each tool by the invocation script

When a tool fails on a model for a particular scaling value, we still try the tool for higher values. This ensures that a tool can fail for any reason on a scaling parameter, and still compete for other values. It has a cost, as we cannot exit the loop and thus save overall computation time for the contest. As we can expect an increase in the number of participating tools, the number of models and the number of scaling values, we might consider this optimization in the next Model Checking Contest.

Prologue and epilogue. A prologue is executed prior to any execution performed on a model. This prologue enables one to prepare data for a given model if necessary. For example, LoLA used this prologue to compile itself on the executing machine, thus avoiding library compatibility problems.

When all examinations have been processed for all the scaling values of a given model, an epilogue is executed, typically to enable tools to delete temporary files.

There is no time or memory usage measures for the prologue and the epilogue actions. These actions are considered as "preparations" for the contest. Tool developers were free to put any preprocessing of the model into the prologue. Some tools like LoLA compile the tool during the first execution of the prologue. Some others include their preprocessing inside the examinations: for instance PNXDD unfolds the model and computes a DD variable order during examinations.

To measure the whole computation time for all tools, we should, in the next edition, measure time and memory spent in the prologue and epilogue, making the computation time more comparable between tools

Examination. Other actions required by the model checkers are executed in a confined environment, to restrict the total execution time of the full evaluation. Both time and memory are reported in a log file. The way confinement and measures were performed is presented in Section 2.2.

Operating a command is performed through a wrapper script customized by the tool submitter. This script must report results of the examination in a standardized and structured way. The results are the number of states in state space generation, the number of deadlocks, or the evaluation of a formula. Moreover, the tool must list the techniques used to work on the examination.

These techniques may be specific to a given examination, the value of a scaling parameter or the processed model.

2.2 Confinement and Measure of CPU and Memory Usage

Evaluation of tools was performed on a cluster of 24 hyper-threaded 2.4 GHz bi-Pentium Xeon 32 bits with 2 GBytes of memory and running Linux Mandriva 2010-2. Monitored actions were confined to the following constraints: 1 800 seconds of CPU and 1.75 GBytes of maximum memory[7].

Prior to the submission deadline, we were using for our tools a solution that appeared to be efficient: `memtime`[8] from the `UPPAAL` community. This program works similarly to the `time` Unix command, but reports both CPU time and memory allocation up to a maximum that can be easily configured.

Once submissions were collected, we discovered that reported memory usage was only concerning the top process (a shell script due to the wrapper script encapsulation technique), even if memory confinement was working well. We thus only used `memtime` for its confinement capabilities and CPU measures associated with the `memusage` command that provides a log of all allocations in the system. We evaluated memory consumption by parsing these log files. The idea is to show the memory use peak, that corresponds to the "user feeling".

The solution we elaborated allowed us to perform measures in a satisfactory way. However, it was too intrusive because it is based on an interposition library that overrides memory allocation and has an impact on performances.

Also, the computation of a diagnostic of a failure (time or memory exhaustion) could not be fully automated, some case having to be checked manually from the log themselves like, in `LoLA` or `PNXDD`, where some memory overflow were initially detected as stack overflow.

For the next edition, executions will be run on virtual machines (e.g. QMU) that could be monitored from "outside", thus allowing more flexibility and safety in measures, as well as the support of other operating systems (such as Windows).

3 The Selected Models

For this first edition, seven models were selected: three Place/Transition nets, and four colored nets modeled in Symmetric nets. These models were selected from known and reused benchmarks. We provide a brief description of the models here. Due to lack of space, we cannot provide a picture of the models in this article, but it can be found in the Model Checking Contest web site[9].

Note that for this first edition, the scaling value of all P/T models increased the number of tokens in places, but did not change the structure of the net. On the contrary, the scaling value of all Colored models increased the number of places and transitions in the Equivalence P/T net, but not the number of tokens.

[7] To leave 250 MBytes for the operating system.

[8] `http://www.uppaal.org/`

[9] `http://mcc.lip6.fr/2011`

The Models. Let us first provide a brief description of the proposed models.

FMS belongs to the `GreatSPN` and `SMART` [10] benchmarks. It models a Flexible Manufacturing System [9]. The scaling parameter corresponds to the number of initial tokens held in three places. The following values were used: $2, 5, 10, 20, 50, 100, 200, 500$.

Kanban [8] models a Kanban system. The scaling parameter corresponds to the number of initial tokens held in four places. The following values were used: $5, 10, 20, 50, 100, 200, 500, 1\,000$.

MAPK models a biological system: the Mitogen-Activated Protein Kinase Kascade [20]. The scaling parameter changes the initial number of tokens held in seven places. The following values were used: $8, 20, 40, 80, 160, 320$.

Peterson models Peterson's mutual exclusion algorithm [35] in its generalized version for N processes. This algorithm is based on shared memory communication and uses a loop with $N-1$ iterations, each iteration is in charge of stopping one of the competing processes. The scaling factor is the number of involved processes. The following values were used: $2, 3, 4, 5, 6$.

Philosophers models the famous Dining Philosophers problem introduced by E.W. Dijkstra in 1965 [41] to illustrate an inappropriate use of shared resources, thus generating deadlocks or starvation. The scaling parameter is the number of philosophers. The following values were used: 5, 10, 20, 50, 100, 500, 1 000, 5 000, 10 000, 50 000, 100 000.

SharedMemory is a model taken from the `GreatSPN` benchmarks [7]. It models a system composed of P processors competing for the access to a shared memory (built with their local memory) using a unique shared bus. The scaling parameter is the number of processors. The following values were used: $5, 10, 20, 50, 100, 200, 500, 1\,000, 2\,000, 5\,000, 10\,000, 20\,000, 50\,000$.

TokenRing is another problem proposed by E.W. Dijkstra [14]. It models a system where a set of machines is placed in a ring, numbered 0 to $N-1$. Each machine i only knows its own state and the state of its left neighbor, i.e., machine $(i-1) \mod (N)$. Machine number 0 plays a special role, and it is called the "bottom machine". A protocol ensuring non-starvation determines which machine has a "privilege" (e.g. the right to access a resource). The scaling parameter is the number of machines. The following values were used: $5, 10, 15, 20, 30, 40, 50, 100, 200, 300, 400, 500$.

Characteristics of the Models. All the selected models are bounded. Their main characteristics are summarized in Table 1. None of the Place/Transition nets is safe (or 1-bounded) because the scaling parameter affects the initial marking. On the contrary, all colored models are safe (in the colored sense where each color cannot appear more than once in a marking) because the scaling parameter only changes the color types.

We also note some characteristics of our colored models. First, color types are either a range of integers, or cartesian products of them. There are two types of

Table 1. Summary of model's characteristics

		Safe	Cartesian product of color types	Non equal guards	Broadcast function	Succ & pred functions
P/T	FMS					
	Kanban					
	MAPK					
Colored	Peterson	✓	✓	✓	✓	✓
	Philosophers	✓			✓	✓
	SharedMemory	✓	✓	✓	✓	
	TokenRing	✓	✓	✓		✓

guards: the ones using equality only (= or binding with the same input variable) and others ($\neq, <, >, ...$) that are interesting because they generate asymmetries in the state space. Arc labels can be a single constant or variable, or use the "broadcast" that is the set containing all values in a color type. Arcs and guards may also use incrementation ($+n$) or decrementation ($-n$) operators. Finally, let us note that the "broadcast" can be used to define the initial marking (it is then a dense one, e.g. all values of the color domain are represented).

When analyzing results of the Model Checking Contest, we observed there was no safe Place/Transition nets and no unsafe Colored nets. In the 2012 Model Checking Contest, we will scale the Petri nets also by their structure, by providing the Place/Transition nets equivalents for all Colored nets. For this first edition we provided two kinds of scaling parameters: one based on the number of tokens in places, the other based on color domains. For the next edition, we should also provide a mix between them, and various models with no scaling parameters, such as industrial cases.

4 Participating Tools

Ten tools where submitted. They are summarized in Table 2.

Table 2. Summary of data on participating tools

Tool Name	Team	Institution	Country	Contact Name
`ACTIVITY-LOCAL`	TIK	ETHZ	Switzerland	K. Lampka
`AlPiNA`	CUI/SMV	Univ. Geneva	Switzerland	D. Buchs
`Crocodile`	LIP6/MoVe	UPMC	France	M. Colange
`ITS-Tools`	LIP6/MoVe	UPMC	France	Y. Thierry-Mieg
`LoLA`	Team Rostock	Univ. Rostock	Germany	N. Lohmann & K. Wolf
`PNXDD`	LIP6/MoVe	UPMC	France	E. Paviot-Adet
`PeTe`	Stud. Group d402b	Univ. Aalborg	Denmark	J. Finnemann Jensen
`Sara`	Team Rostock	Univ. Rostock	Germany	H. Wimmel & K. Wolf
`YASPA`	TIK	ETHZ	Switzerland	K. Lampka
`helena`	LIPN/LCR	Univ. Paris 13	France	S. Evangelista

Tool Description. We provide here a brief description of the participating tools.

ACTIVITY−LOCAL[10] [29] works on any type of Place/Transition nets with inhibitor arcs and weighted arcs. It combines decision diagram-based state space encoding with explicit state space exploration. To avoid the Peak problem observed for decision diagrams with incremental generation, this tool composes them in an original way. The transition relation induced by the same transition of the P/T net is encapsulated in its own "submodel". ACTIVITY−LOCAL executes an explicit state space traversal for each of these submodels and inserts the detected state-to-state transitions into the corresponding DD (one per submodel).

To cope with dependencies among the transitions of the P/T net, ACTIVITY−LOCAL structures the explicit exploration as a *selective* breadth-first scheme. It only explores the transitions that are in a dependency set of the analyzed transition.

When a local fixed point is reached, ACTIVITY−LOCAL performs a symbolic reachability analysis used to elaborate the complete state space. This second step is implemented as a partitioned symbolic reachability analysis [5], using greedy chaining [34] and a new DD operator [28].

AlPiNA[11] [23] stands for Algebraic Petri nets Analyzer and is a symbolic model checker for Algebraic Petri nets. It can verify various state properties expressed in a first order logic property language.

Algebraic Petri nets (APNs) (Petri nets + Abstract Algebraic Data Types) is a powerful formalism to model concurrent systems in a compact way. Usually, concurrent systems have very large sets of states, that grow very fast in relation to the system size. Symbolic Model Checking (DD-based one) is a proven technique to handle the State Space Explosion for simpler formalisms such as Place/Transition nets. AlPiNA extend these concepts to handle algebraic values that can be located in net places.

For this purpose AlPiNA uses enhanced DDs such as Data Decision Diagrams and Set Decision Diagrams for representing the place vectors and Σ Decision Diagrams [3] for representing the values of the APN. It also allows to specify both algebraic and topological clusters to group states together and thus to reduce the memory footprint. Particular care has been taken to let users freely model their systems in APNs and in a second independent step to tune the optimization parameters such as unfolding rules, variable order, and algebraic clustering. Compared to Colored net approaches, AlPiNA [4] solves problems related to the unbounded nature of data types and uses the inductive nature of Abstract Algebraic Data Types to generalize the unfolding and clustering techniques to any kind of data structure.

AlPiNA's additional goal is to provide a user friendly suite of tools for checking models based on the Algebraic Petri nets formalism. In order to provide great user experience, it leverage on the powerful eclipse platform.

[10] No official distribution yet.

[11] Tool is available at `http://alpina.unige.ch`.

`Crocodile`[12] [12] was initially designed as a demonstration tool for the so-called symbolic/symbolic approach [39]. It combines two techniques for handling the combinatorial explosion of the state space that are both called "symbolic".

The first "symbolic" technique concerns the reduction of the reachability graph of a system by its symmetries. The method used in `Crocodile` was first introduced in [6] for the Symmetric nets, and was then extended to the Symmetric nets with Bags (SNB) in [18]. A symbolic reachability graph (also called quotient graph) can be built for such types of Petri nets, thus dramatically reducing the size of the state space.

The second "symbolic" technique consists in storing the reachability graph using decision diagrams, leading to a symbolic memory encoding. `Crocodile` relies on Hierarchical Set Decision Diagrams [13]. These present several interesting features, such as hierarchy, and the ability to define inductive operations.

Still under development, `Crocodile` essentially generates the state space of a SNB and then processes reachability properties.

`ITS−Tools`[13] [40] are a set of tools to analyze Instantiable Transition Systems, introduced in [40]. This formalism allows compositional specification using a notion of type and instance inspired by component oriented models. The basic elementary types are labeled automata structures, or labeled Petri nets with some classical extensions (inhibitor arcs, reset arcs…). The instances are composed using event-based label synchronization.

The main strength of `ITS−Tools` is that they rely on Hierarchical Set Decision Diagrams [13] to perform analysis. These decision diagrams support hierarchy, allowing to share representation of states for some subsystems. When the system is very regular or symmetric, recursive encodings [40] may even allow to reach logarithmic overall complexity when performing analysis. Within the contest, the Philosophers and TokenRing examples proved to be tractable using this recursive folding feature.

Set Decision Diagrams also offer support for automatically enabling the "saturation" algorithm for symbolic least fixpoint computations [19], a feature allowing to drastically reduce time and memory consumption. This feature was used in all computations.

`LoLA`[14] [43] is an explicit Petri net state space verification tool. It can verify a variety of properties ranging from questions regarding single Petri net nodes (*e.g.*, boundedness of a place or quasiliveness of a transition), reachability of a given state or a state predicate, typical questions related to a whole Petri net (*e.g.*, deadlock freedom, reversibility, or boundedness), and the validity of temporal logical formulae such as CTL. It has been successfully used in case studies from various domains, including asynchronous circuits, biochemical reaction chains, services, business processes, and parameterized Boolean programs.

[12] Tool is available at `http://move.lip6.fr/software/Crocodile`.
[13] Tool is available at `http://ddd.lip6.fr`.
[14] Tool is available at `http://www.informatik.uni-rostock.de/tpp/lola`.

For each property, LoLA provides tailored versions of state space reduction techniques such as stubborn sets, symmetry reduction, coverability graph generation, or methods involving the Petri net invariant calculus. Depending on the property to be preserved, these techniques can also be used in combination to only generate a small portion of the state space.

For the Model Checking Contest, only one configuration of LoLA was submitted since, in the beginning, the necessary efforts were not predictable. This configuration was tailored for checking the reachability of a state that satisfies a given state predicate. This check was combined with two reduction techniques. First, a dedicated version of the stubborn sets [37] aimed at exploiting concurrency in the model and that allows to prioritize the firing of those transitions that lead to states closer to the goal state. This method is known to perform extremely well on reachable states while other methods [27] also available in LoLA would excel on unreachable states. Second, LoLA calculates place invariants to determine so-called implicit places [38]. The marking of such places does not need to be stored explicitly, but can be deduced from the marking of the other places. Typically, this reduction allows to reduce the memory usage by 20% to 60%.

PNXDD[15] generates the state-space of Place/Transition nets. When Colored nets are used in the Model Checking Contest, equivalent P/Ts are obtained after an "optimized" unfolding [25] (unused places and transitions are detected and suppressed).

State Space storage relies on Hierarchical Set Decision Diagrams [13] (SDDs). These are decision diagrams with any data type associated to arcs (see *e.g.*, [32] for an overview of DD-like structures). If the associated data type is another SDD, hierarchical structures can be constructed.

Since PNXDD exploits hierarchy, a state is seen as a tree, where the leaves corresponds to places marking. This particular structure offers greater sharing opportunities than a, for instance, vector based representation. The conception of such a tree is critical to reach good performances and heuristics are being elaborated for this purpose [21]. The one used for the Model Checking Contest is based on [1]: for colored models that do scale via the size of color types, PNXDD uses a tree-like version of this heuristic, while the original version is kept when colored models only scale via the number of tokens in the initial marking.

PeTe[16] is a graphical Petri net modeling and verification tool written in C++/Qt. PeTe can answer reachability formulæ using two techniques. First, PeTe attempts to disprove reachability of a query using over-approximation. This is done by solving the state equation using integer programming. If a solution is found PeTe attempts to tighten the approximation using trap testing as presented in [15]. Detailed description and variations of this approach can be found in [15].

If over-approximation cannot disprove reachability, PeTe attempts to prove reachability with straightforward Best-First-Search of the state space, using a

[15] Tool is available at `https://srcdev.lip6.fr/trac/research/NEOPPOD/wiki/pnxdd`.
[16] Tool is available at `https://github.com/jopsen/PeTe`.

simple heuristic presented in [17]. So far, PeTe does not support state space generation or deadlock detection. PeTe is maintained and available under GNU GPLv3.

Sara[17] uses the state equation, known to be a necessary criterion for reachability and in a modified way also for other properties like coverability, to avoid enumerating all possible states of a system [36]. A minimal solution of the state equation in form of a transition vector is transformed into a tree of possible firing sequences for this solution. A firing sequence using all the transitions given in the solution (with the correct multiplicity) reaches the goal.

For tree pruning, partial order reduction is used, especially in the form of stubborn sets [37,26]. If the goal cannot be reached using the obtained solution, places that do not get enough tokens are computed. Constraints are built and added to the state equation (in a CEGAR-like fashion [11]). These constraints modify the former solution by adding transition invariants, temporarily allowing for additional tokens on the undermarked places.

Sara detects unreachable goals either from an unsatisfiable state equation or by cutting the solution space to finite size when the repeated addition of transition invariants is known not to move towards the goal. A more involved explanation of the algorithm behind Sara can be found in [42].

YASPA[18] relies on Zero-Suppressed Decision Diagrams (ZDDs) [33,31] together with a partitioned symbolic reachability analysis [5] and greedy chaining [34]. However, contrary to other decision diagram-based tools it neither depends on pre-generated symbolic representations of state-to-state transitions, nor on the use of standard decision diagram operators. Instead, symbolic reachability analysis is carried out by means of customized ZDD-algorithms that are directly synthesized from the Place/Transition net with inhibitor arcs.

As a key feature the synthesized ZDD operators are organized in a strictly local manner. This is achieved by assigning an identity semantics to those variables of the decision diagram which refer to places that are neither pre nor post condition of a given transition. Moreover, the ZDD operators apply decision diagram-related recursion rules which implement the subtraction, addition and testing of tokens.

By executing these synthesized ZDD operators in a fixed point iteration, YASPA delivers the state space and transition relation of the system.

helena[19] [16] is an explicit state model checker for High Level Petri nets. It is a command-line tool available available under the GNU GPL.

helena tackles the state explosion problem mostly through a reduction of parallelism operated at two stages of the verification process. First, static reduction rules are applied on the model in order to produce a smaller net that – provided some structural conditions are verified – is equivalent to the original one but has a smaller reachability graph. Second, during the search, partial order reduction is employed to limit, as much as possible, the exploration of redundant paths in the reachability graph. This reduction is based on the detection of independent

[17] Tool is available at http://www.service-technology.org/tools/download.

[18] Tool is available at http://www.tik.ee.ethz.ch/~klampka.

[19] Tool is available at http://helena-mc.sourceforge.net.

transitions of the net at a given marking. Other reduction techniques are also implemented by helena, e.g., state compression, but were disabled during the contest due to their inadequacy with the proposed models.

Summary of Techniques Used by Participating Tools. Altogether, these tools implement numerous techniques, as summarized in Table 3. We note that several tools stack several techniques such as decision diagrams with (sometimes) symmetries, or abstractions with partial orders.

We also note that numerous types of decision diagrams are used in participating tools. YASPA uses customized Zero-Suppressed Multi-Terminal Decision Diagrams [30]. ITS−Tools, PNXDD, Crocodile and AlPiNA are using Hierarchical Set Decision Diagrams [40]. AlPiNA also uses a variant called Σ Decision Diagrams dedicated to algebraic systems [3].

Table 3. Summary of techniques used by tools

	Reachability Graph	**Deadlock Detection**	**Formula Evaluation**
ACTIVITY−LOCAL	Explicit Decision Diagrams		
AlPiNA	Decision Diagrams	Decision Diagrams	
Crocodile	Symmetries Decision Diagrams		Symmetries Decision Diagrams
ITS−Tools	Decision Diagrams Symmetries (opt)	Decision Diagrams Symmetries (opt)	Decision Diagrams Symmetries (opt)
LoLA			Explicit Partial Orders State Compression
PNXDD	Decision Diagrams		
PeTe			Explicit State Equation
Sara			Abstractions Partial Orders State Equation
YASPA	Decision Diagrams		
helena	Explicit	Explicit Abstractions Partial Orders	

‖In the next edition, more precision will be required to classify techniques.

5 Observations on State Space Generation

This section analyzes the results of the Model Checking Contest for state space generation. It first presents the highest parameter computed by the tools for each model. Then it compares in Section 5.1 the maximum parameter reached, together with the evolution of computation time and memory consumption on

Place/Transition net models, before doing the same analysis on Colored net models in Section 5.2. We have found this distinction to be the most significant for state space generation.

Table 4 summarizes the highest parameter reached by the tools for each model. This table, as well as Tables 5 and 6, should be interpreted using the legend below:

⬜	The tool does not participate.
░	The tool participates, but cannot compute for any scaling value.
▒	The tool participates.
▓	The tool participates and reaches the best parameter among tools.
■	The tool participates and reaches the maximum parameter.
n ?	The tool fails for an unknown reason, after reaching parameter n.
n ▥	The tool fails because of memory exhaustion, after reaching parameter n.
n ⏲	The tool fails because of maximum time is elapsed, after reaching parameter n.

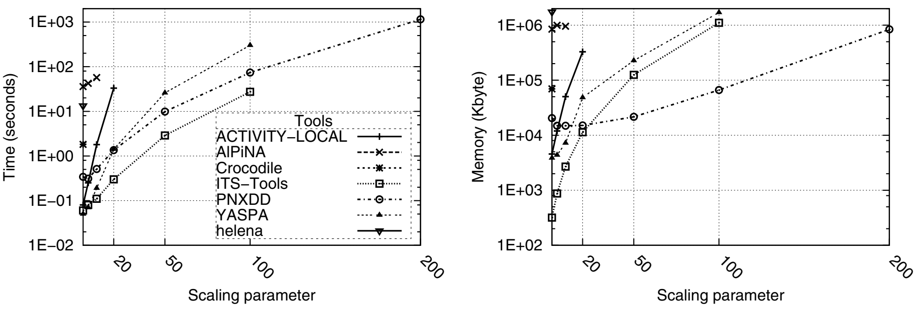
Table 4. Results for the state space generation examination

		ACTIVITY−LOCAL	AlPiNA	Crocodile	ITS−Tools	LoLA	PNXDD	PeTe	Sara	YASPA	helena
P/T	FMS	▒ 20 ⏲	▒ 10 ⏲	▒ 2 ▥	▒ 100 ▥		▓ 200 ⏲			▒ 100 ▥	▒ 2 ▥
	Kanban	▒ 20 ⏲	▒ 10 ⏲	░ ▥	▒ 200 ▥		▓ 500 ▥			▒ 50 ?	░ ▥
	MAPK		▒ 8 ⏲	░ ▥	▓ 160 ▥		▓ 160 ▥			▒ 8 ▥	
Colored	Peterson		▒ 3 ⏲				▓ 5 ?				▒ 2 ⏲
	Philosophers		▒ 500 ⏲	▒ 10 ⏲	■ 100 000		▒ 1 000 ▥				▒ 10 ▥
	SharedMemory		▒ 20 ?	▒ 20 ▥	■ 50 000		▒ 100 ⏲				▒ 10 ⏲
	TokenRing		▒ 10 ?		▓ 50 ▥		▒ 15 ⏲				▒ 10 ⏲

Table 3 shows that almost all the tools competing for state space generation use decision diagrams. So, this technique seems to be the most common choice when doing state space generation. From Table 4, we observe that among DD-based tools, there is a great variation in the maximum scaling parameter reached. The ratio between the value reached by the worst and the best DD-based tools is 1 : 5 for TokenRing and 1 : 10 000 for Philosophers.

Comparing tools for the state space examination is not a trivial task. The Model Checking Contest organizers encountered several problems, all concerning the returned size of the state space, which was initially used to check the answers:

1. For `helena`, both tool developers and the Model Checking Contest organizers agreed to disable all optimizations – structural reductions and state compression – because they lead to the generation of a 1-state reachability graph. It did not seem to make sense in this examination.
2. `Crocodile` on Colored net also returns fewer states, because it is computing the quotient reachability graph.
3. We also noted, for FMS and TokenRing, a variation in the state space size, that was apparently due to some variation in the encoding of the model. Some tools, like `ACTIVITY−LOCAL` and `YASPA` adapted the model taken from `GreatSPN`, for instance by removing instantaneous transitions. These variations did not seem large enough to require the tool developers to check their models and tools.



(a) Time (b) Memory

Fig. 1. Memory and time measure for state space generation on the FMS model

5.1 Place/Transition Net Models

Note that, for Place/Transition net models, the sizes of the state space for the highest parameter reached by the tools are:

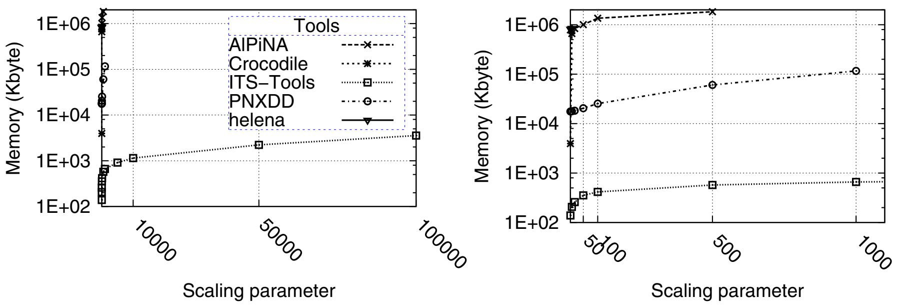
$$\mathrm{FMS}_{200} = 1.95 \times 10^{25} \text{ states}$$
$$\mathrm{Kanban}_{500} = 7.09 \times 10^{26} \text{ states}$$
$$\mathrm{MAPK}_{160} = 1.06 \times 10^{23} \text{ states}$$

Figure 1 shows the memory and time evolution of state space generation for the FMS model. It is typical of what we observe for this examination on Place/Transition nets. It experimentally shows that we can divide DD-based tools into two groups: the first one ("bad results") only reaches FMS_{20} (around 6×10^{12} states) at most, the second one ("good results") reaches FMS_{100} (around 2.7×10^{21} states) at most.

The "bad results" group is composed of `Crocodile`, `AlPiNA` and `ACTIVITY−LOCAL`. All these tools are not dedicated to Place/Transition nets: `Crocodile` is intended for Colored nets with bags in tokens, `AlPiNA` is optimized for Algebraic Petri nets, and `ACTIVITY−LOCAL` works on any type of P/T nets with inhibitor arcs and weighted arcs. These three tools do not get better results on the two other P/T models, Kanban and MAPK. Crocodile has bad performances because it does not know how to exploit symmetries from Place/Transition nets; moreover, it appeared that management of multisets of tokens needed some improvement. The developers of `AlPiNA` discovered that it has bad performance for P/Ts because the tool is implicitly optimized for safe Petri nets.

On the contrary, tools that handle formalisms closer to the Place/Transition nets obtain good results. `YASPA`, `ITS−Tools` and `PNXDD` handle at least FMS_{100}. `YASPA` is a bit less effective on Kanban, and is comparable to the "bad tools" for MAPK. We noted that for Kanban, the results are not consistent with measures made by the author of `YASPA`, that shows similar performance as for FMS (we could not find any explanation for this).

Among the "good results" group, we can see that `PNXDD` has better results than `ITS−Tools` for FMS and Kanban. The explanation is that for these nets, `PNXDD`



(a) Full chart (b) Zoom on smaller scaling parameters

Fig. 2. Memory measure for state space generation (Philosophers)

does not use hierarchical DDs, contrary to `ITS−Tools`. Because the scaling value only increases the number of tokens, and not the size of the net, the cost of using hierarchy is not covered by the gains it provides.

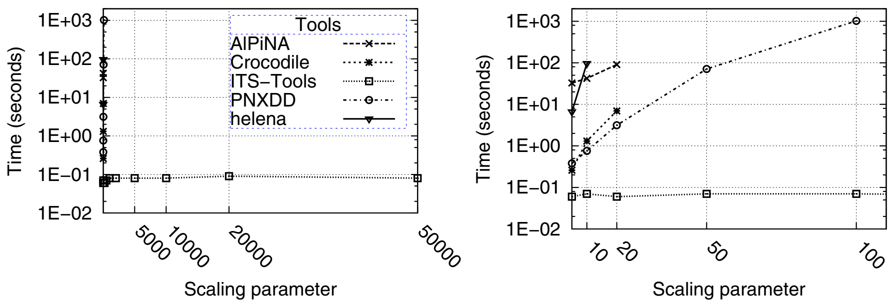
No tool could reach the maximum scaling value for Place/Transition nets (500 for FMS, 1 000 for Kanban, 320 for MAPK). As these numbers have been selected based on known results in papers, this is not surprising. On the contrary `PNXDD` is close to the maximum parameters. For the next Model Checking Contest, scaling parameter of the 2011 models will be increased. This analysis of the examinations should be done each year, in order to increase tools efficiency, as it was observed in the SAT Competition.

5.2 Colored Net Models

We provide in Figure 2 the memory measure for the Philosophers model and in Figure 3 the CPU consumption for SharedMemory. Since one technique is very efficient, the leftmost part of the figures show measures for all the scaling parameter while the rightmost part only focus on the subset of values where all tools provide results. These figures are of interest because they show some extreme performance of some techniques in favorable cases.

Execution of tools on colored models showed interesting points:

– `helena` obtains results comparable to some decision diagram-based tools on SharedMemory and TokenRing (see Figure 4 for TokenRing). As all optimizations of `helena` are disabled for this examination, it shows that these DD-based tools are quite inefficient for these models;
– `Crocodile` has heterogeneous results: it is as good as `AlPiNA` on SharedMemory (see Figure 3b), but reaches only a low parameter on Philosophers (see Figure 2b). This is apparently due to a non optimal exploitation of symmetries; optimization could also be performed on the implementation;
– `ITS−Tools` reaches impressive parameters compared to the other tools on the Colored net models it handled.



(a) Full chart (b) Zoom on smaller scaling parameters

Fig. 3. CPU measure for state space generation (SharedMemory)

Thus, we can divide the tools into two groups: the "quite good" one containing all tools except `ITS−Tools`, and the "excellent" one containing this last tool.

Figures 2a and 3a illustrate the real interest of a technique, used in `ITS−Tools`, associated to hierarchical decision diagrams: "recursive folding" [40]. This technique can be activated for very regular models such as Philosophers and SharedMemory. It consist in splitting recursively the model in subcomponents. First, the system is split in "two halfs", and then each half in "two fourth", *etc.* Associated with the hierarchical decision diagrams used in `ITS−Tools`, the result is impressive: this tool is able to process both models for the maximum provided values. In both cases, the number of states exceeds the floating points representation. For smaller parameters, the state space sizes are given below:

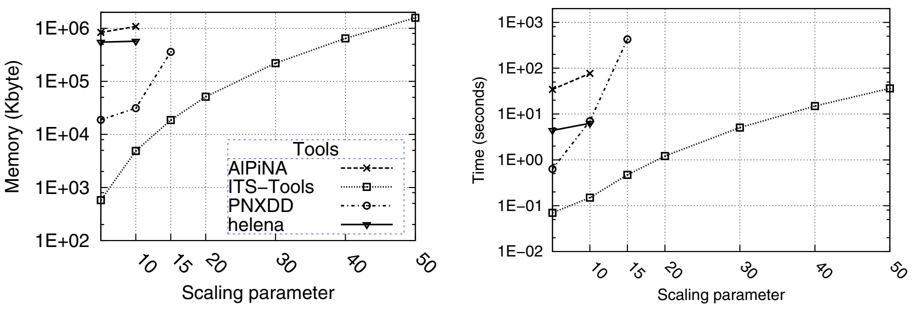
$$\text{Philosophers}_{10\,000} = 1.63 \times 10^{4771} \text{ states}$$
$$\text{SharedMemory}_{10\,000} = 5.43 \times 10^{4778} \text{ states}$$

Figures 2b and 3b are a zoom on the left part of figures 2a and 3a. It shows the performances of "second tools" that correspond to the following state space size:

$$\text{Philosophers}_{1\,000} = 1.13 \times 10^{477} \text{ states}$$
$$\text{SharedMemory}_{100} = 5.15 \times 10^{47} \text{ states}$$

Figure 4 shows measures for the TokenRing model where the recursive folding technique cannot be activated. Decision diagram-based tools are much less performant than previously, the largest computed state space holds 1.98×10^{27} states "only".

Apart the results of `ITS−Tools`, the results of `PNXDD` and `AlPiNA` are useful for another remark. When comparing these two tools on Philosophers, SharedMemory and TokenRing, we see that `AlPiNA` and `PNXDD` have comparable results on Philosophers and TokenRing, whereas `PNXDD` is far better on SharedMemory (due to different hierarchical structures). From this, we can deduce that Hierarchy in decision diagrams offers interesting results.



(a) Memory consumption (b) CPU consumption

Fig. 4. Measure for state space generation (TokenRing)

There is a large difference between the performances of DD-based tools for Philosophers and SharedMemory (many orders of magnitude for the values of both scaling parameters), whereas the difference is lower for TokenRing. But ITS–Tools and PNXDD should get closer results on TokenRing, as there is no recursive folding for this model. Although both tools use the underlying Place/-Transition nets for this colored model, the unfolding of color, as well as the construction of the hierarchical structure, are not the same.

Decision diagram based tools clearly can reach impressive scaling values for state space generation, when using hierarchical decision diagrams together with recursive folding. To do state space generation, we can recommend three tools: PNXDD is very efficient and works both for Place/Transition nets and Colored nets, YASPA is dedicated to Place/Transition nets and has heterogeneous results, and ITS–Tools is extremely efficient on Coloreds but requires to manually transform the model.

Academic models seem easy for the good state space generators. We should provide some industrial models in the next Model Checking Contest, as they are usually not as regular as academic models.

The Peterson model seems reluctant to all the implemented techniques. Only three tools could handle it, and the best processed scaling values are very low: 5 for PNXDD, 3 for AlPiNA and 2 for helena. This corresponds to very small state spaces compared to the ones reached for other models:

$$\text{Peterson}_3 = 2.07 \times 10^4 \text{ states}$$
$$\text{Peterson}_5 = 6.30 \times 10^8 \text{ states}$$

6 Observations on Deadlock Detection

The data collected for deadlock detection is summarized in Table 5. It must be read as Table 4. Let us note that only three tools did participate in this examination.

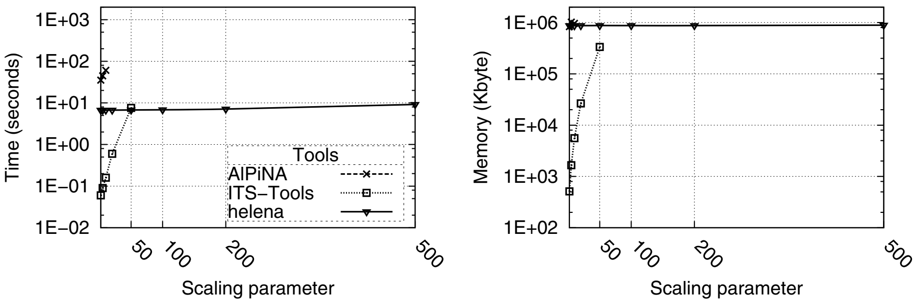
Table 5. Results for the deadlock detection examination

		ACTIVITY–LOCAL	AlPiNA	Crocodile	ITS–Tools	LoLA	PNXDD	PeTe	Sara	YASPA	helena
P/T	FMS		10 ⏲		50 ▦						500
	Kanban		10 ⏲		200 ▦						10 ▦
	MAPK		8 ⏲		160 ▦						
Colored	Peterson		3 ⏲								3 ⏲
	Philosophers		100 ⏲		100 000						10 ⏲
	SharedMemory		20 ?		50 000						10 ⏲
	TokenRing		10 ?		50 ▦						10 ⏲

As this examination required to count the number of deadlocks instead of just discovering one, some tools could not participate. For instance, LoLA stops when the first deadlock is found. We should only ask for the detection of at least one deadlock in the next edition of the Model Checking Contest, to have more competing tools. We will also propose to refer to deadlocks in formula to be evaluated.

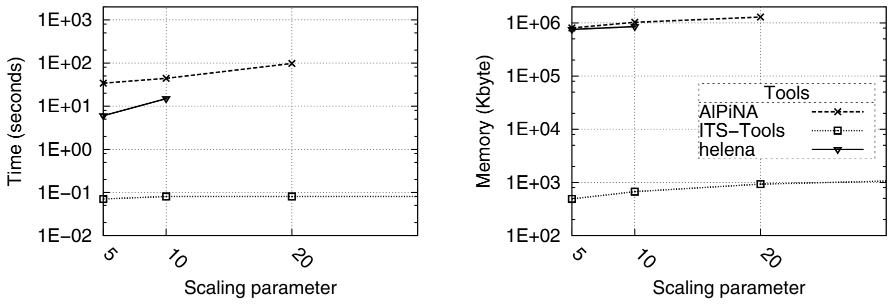
In Table 4, we see that helena has an inconstant behavior. It works very well for FMS, reaching the maximum scaling value in constant time and memory, as shown in Figure 5. On the contrary, this tool handles only small instances for the other models. It shows that abstractions and partial orders provide good results is this case where both CPU and memory usage are almost constant. The abstraction mechanism used by helena is based on Berthelot's structural reductions [2], that remove transitions irrelevant from a concurrency perspective. FMS is a perfect case for that method in that it exhibits a lot of parallelism but little concurrency.

For all models, except FMS and Peterson, and especially Colored ones, the results are close to state space generation: decision diagram based tools obtain



(a) CPU consumption (b) Memory consumption

Fig. 5. Measures for deadlock examination on FMS



(a) CPU consumption (b) Memory consumption

Fig. 6. Measures for deadlock examination on SharedMemory

quite good results, especially `ITS−Tools` when recursive folding can be applied. Figure 6 shows the evolution of CPU time and memory consumption for the SharedMemory model, where recursive folding enables `ITS−Tools` to process all instances. The dominance of decision diagram techniques in this examination is probably due to the fact that, since it was required to report the number of deadlocks instead of just the detection of at least one, tools must investigate the full state space, thus making this examination behave like the state space generation.

7 Observations on Reachability Formulas

The data collected for evaluation of satisfiable and unsatisfiable reachability formulæ is summarized in Table 6. It must be read as Table 4.

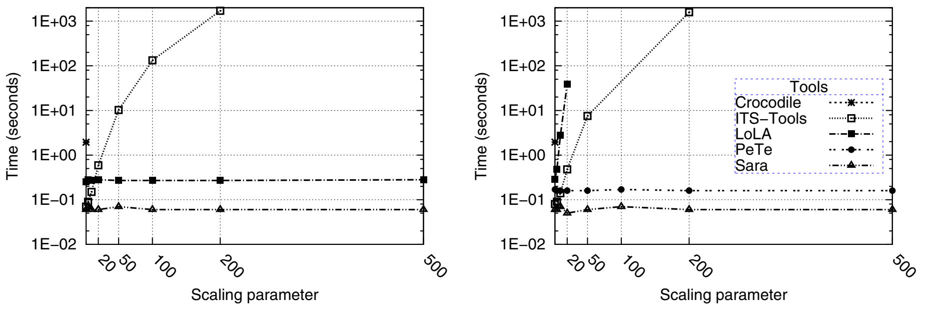
From Table 6, we can clearly state that for Place/Transition nets, there are tools that perform extremely well for satisfiable formulæ (`LoLA` and `Sara`) where some others are much better for unsatisfiable formulæ (`PeTe` and `Sara`). In that context, decision diagram based tools can perform well (see `ITS−Tools`), but do not reach the maximum values. The reason why `LoLA` does not reach a very high parameter on Kanban is still not understood. `Sara` is clearly interesting for Place/Transition nets, as it reaches the maximum scaling value for both satisfiable and unsatisfiable formulæ.

The formulæ to verify were only conjunctions of place markings. The efficiency of tools may depend on the operators used in properties. For instance, formulæ with disjunctions and inequalities can lead to worse results in `Sara`. The 2012 Model Checking Contest should thus provide more different formulæ, and also describe their properties, to get a detailed analysis of the tools' performances. Note also that `Sara` gives an answer very quickly, as seen for FMS in Figure 7.

Tools that perform well on satisfiable formulæ for P/T nets are `LoLA` and `Sara`. These tools explore the state space and stop their execution as soon as they

Table 6. Results for the reachability formulæ examination

		ACTIVITY-LOCAL	AlPiNA	Crocodile	ITS-Tools	LoLA	PNXDD	PeTe	Sara	YASPA	helena
Satisfiable reachability formulæ											
P/T	FMS			2 ▤	200 ◷	500		▤	500		
	Kanban			▤	200 ▤	200 ▤		▤	1 000		
	MAPK			▤	160 ▤	320		▤	80 ▤		
Colored	Peterson										
	Philosophers			5 ◷	5 000 ▤	10 ▤					
	SharedMemory			10 ▤	20 000 ▤						
	TokenRing				40 ▤						
Unsatisfiable reachability formulæ											
P/T	FMS			2 ▤	50 ▤	20 ▤		500	500		
	Kanban			▤	200 ▤	10 ▤		1 000	1 000		
	MAPK			▤	160 ▤	20 ▤		320	320		
Colored	Peterson										
	Philosophers			5 ◷	5 000 ▤	10 ▤					
	SharedMemory			10 ▤	20 000 ▤						
	TokenRing				50 ▤						



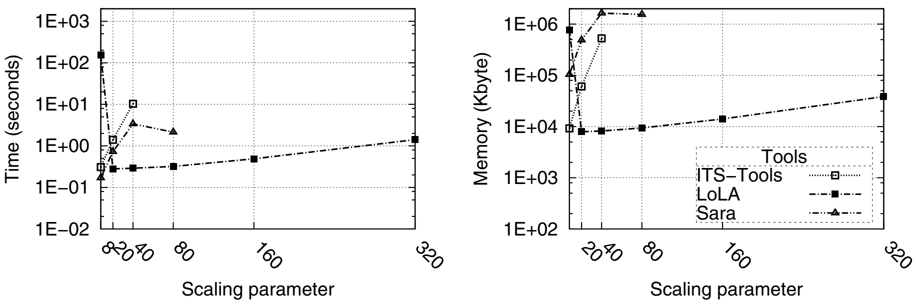
(a) Satisfiable formulæ (b) Unsatisfiable formulæ

Fig. 7. Evolution of CPU consumption for reachability properties on FMS

have found a violation of the property to be verified. Thus, since a satisfiable formula is verified before the full state space is explored, they perform better on satisfiable formulæ than on unsatisfiable ones.

Tools that perform well on unsatisfiable formulæ for P/T nets are `PeTe` and `Sara`. To do so, they first evaluate the state equation of the P/T net against the reachability formula. If the result of such an evaluation shows the formula is structurally unverifiable, the tool does not need to explore, even partially, the state space. Otherwise, exploration to extract a counter-example is necessary.

`Sara` combines the two techniques and is thus quite efficient in both cases. This effect is illustrated in Figure 7 that is representative of the behavior of tools for P/Ts.



(a) CPU (b) Memory

Fig. 8. Evolution of CPU consumption satisfiable formulæ (MAPK)

Figure 8 illustrates an interesting fact on partial order with `LoLA` on the MAPK benchmark (both CPU and memory). For satisfiable formulæ, both memory and CPU performances are much better for MAPK_{320} than for MAPK_8. This is due to the less parallel nature of MAPK for small scaling values, thus degrading performances of partial order techniques. When the scaling parameter grows, parallelism is generated and the technique becomes useful.

The data for Colored net models permit similar observations as for state space generation, because only decision diagram based tools participated, except `LoLA` for the Philosophers model. We can observe that `LoLA` has difficulties to scale up with this model. This may be because when the parameter grows, this model has more places, whereas P/T models only have more tokens. However, if we can state that decision diagrams are a good technique for reachability analysis, the collected data are not sufficient to generalize this assertion to reachability properties.

8 Discussion

This section proposes a global discussion on the Model Checking Contest'2011 results from several points of view. First, we focus on the user (*i.e.*, an engineer willing to verify software) point of view in Section 8.1. Then, Section 8.2 considers the tool developer point of view. Last, Section 8.3 recalls the lessons learned by the Model Checking Contest organizers for the next edition.

8.1 Engineer Point of View: There Is No Silver Bullet

Models, like software, have a lifecycle. It can be roughly decomposed into: its creation, its verification, and its evolution. However, the evolution phase is a sequence of editions and verifications. Thus, the model lifecycle can be simplified in a sequence of edition and of verification. During each phase, properties are checked on the model. But the kind of properties may vary.

We have used two kinds of properties in this Model Checking Contest: deadlock detection and reachability properties. For both, we can also distinguish the case where the property is satisfiable, and the case where it is not.

For Place/Transition Net Models. During the model development, we want to debug the model. To do so, the chosen properties must be checked quickly. Except the case when ITS−Tools can use recursive folding, all tools spend at least 10 seconds to answer (for very small configurations) ; the answering time grows rapidly. Thus, deadlock detection is currently not efficient enough to debug large models. Instead, tools like PeTe, LoLA and Sara can answer in less than 1 second for reachability properties. Sara gives a quick answer whether the property is satisfiable or not, whereas LoLA is more efficient when the property is satisfiable, and PeTe when it is not. A good idea would be to run both LoLA and PeTe in parallel and stop them as soon as one of them answers. Reachability properties could even be checked in background, while editing the model. Then, for users, model creation would be very close to source code edition in modern IDEs, that make use of continuous on-the-fly compilation.

During the model verification phase, all tools can be used, as there is time to do longer checks. Deadlock detection is currently adapted to this phase, as it is a rather long process. For Place/Transition nets, there is no added value in state space generation, as reachability properties can be checked during the edition phase.

For deadlocks, helena can be impressively efficient, for the FMS model, or not as good as decision diagram based tools. There should be some investigations on why. Also, ITS−Tools shows impressive performance when models can be "folded". Hierarchical extensions of Petri nets are clearly interesting.

For Colored Net Models. Decision diagram based tools are very efficient for state space computation. Using this state space they are then able to find deadlocks and check properties. But state space computation is usually quick only for small models. As most tools that do not use decision diagrams did not participate for Colored net models, we cannot conclude yet about which techniques should be used.

We must provide P/T equivalents for all Colored models in the next Model Checking Contest, so that more tools can compete.

8.2 Lessons Learned by Tool Developers

The Model Checking Contest can help tool developers to discover some unexpected behaviors and compare strategies and techniques among the various participating tools in common situations.

As an illustration, AlPiNA developers discovered that it is currently mostly adapted to safe Petri nets. The tool got bad results on all the Place/Transition nets of the contest. Analysis revealed that AlPiNA inefficiency on non-safe nets is due to the particular decision diagrams used in the tool.

The contest is also a good way to test the integration of model checkers in an "alien" environment. This can be a basis to extend cooperation and exchange of data between model checkers and promote further cooperation.

As an illustration, `LoLA` follows several UNIX principles. This made an integration to the contest scripts very smooth. First, for each model, there was a dedicated compiled version of `LoLA` that can exploit any possible optimization the CPU architecture could offer. Second, "UNIX pipelines" made the evaluation of the reachability results very simple (use of `grep` to filter outputs).

8.3 Points Risen by the Discussion at MCC'2011

Several points were raised during the discussion held during the MCC'2011 in Newcastle. We present here the most interesting ones.

A Difficult Model: Peterson. One point was outlined in the Model Checking Contest: the Peterson model seems reluctant to all the implemented techniques. The best processed scaling values are 5 (`PNXDD`) for the state space generation and 3 (`helena`, `AlPiNA`) for deadlock detection. This corresponds to very small state spaces compared to the ones reached for other models. This exhibits an interesting situation to be handled by tools.

Need for a "Push-Button" Examination. As it is organized, the Model Checking Contest is efficient to identify how some model characteristics could be tackled by some model checking techniques. However, this does not cover the use of model checkers by non-specialists. For this kind of users there should be a efficient "push-button" use of such tools. This aspect should be considered in further editions of the Model Checking Contest. An idea should be to find "surprise models" from case studies, that are not known by the competitors when they submit (and only published when results are known).

Doing CPU and Memory Measures Is Tricky. Measuring and confining software executions during this first Model Checking Contest was not trivial. Tools are written in several languages, some of which are based on shell scripts, interpreters or virtual machines. Moreover, tools are allowed to create subprocesses and catch signals. To avoid most problems while not being intrusive, we plan to execute tools within a virtual machine monitored to operate time and memory measures.

9 Conclusion

This paper reported our experience with the first Model Checking Contest @ Petri nets. This event and its results were welcomed by the Petri Net Community, as the discussion held at a special session of SUMo'2011 showed.

From the tool developers' point of view, such an event allows to compare tools on a common benchmark that could become a public repository. Also, some mechanisms established for the contest, such as a language to elaborate the formula to be verified could become, over the years, a common way to provide formulæ to the various tools developed by the community.

Results also provided hints to the tool developers with regards to the optimization of some techniques in their tool. At least, developers of `AlPiNA` and `Crocodile` attest that some development is being currently done to improve the model checking engine from the results of the model checking contest. This will benefit to the entire community.

From the organizer's point of view, numerous lessons were learned on the process, the analysis of results and the selection of benchmark models. Several points will be integrated in further edition of the Model Checking Contest.

As an illustration, the next edition to be held in 2012 comes with a new step in the process: a call for models that will allow us to gather more models, exposing tools to a larger range of situations. Properties will be extended to CTL and LTL formulas, as well as with structural properties, such as bounds or liveness, and their counterpart in temporal logic. Finally, a "blind" set of models will also be proposed to reproduce a situation where tools are used "as is" by non specialists (and thus with default optimization activated only).

Acknowledgements. The Model Checking Contest @ Petri nets organizers would like to thank the following people for the help they provided in setting up this event: Clément Démoulins (infrastructure), Nicolas Gibelin (infrastructure and cluster management), Lom Hillah (production of PNML files), Emmanuel Paviot-Adet and Alexis Marechal (description of selected models), and Steve Hostettler (definition of properties).

The Model Checking Contest organizers would also like to thank the tool developers who made possible such a contest. They are:

– `ACTIVITY−LOCAL` and `YASPA`: Kai Lampka;
– `AlPiNA`: Steve Hostettler, Alexis Marechal, and Edmundo Lopez;
– `Crocodile`: Maximilien Colange;
– `ITS−Tools`: Yann Thierry-Mieg;
– `LoLA`: Karsten Wolf;
– `PeTe`: Jonas Finnemann, Thomas Nielsen and Lars Kærlund;
– `PNXDD`: Silien Hong and Emmanuel Paviot-Adet;
– `Sara`: Harro Wimmel and Karsten Wolf;
– `helena`: Sami Evangelista and Jean-François Pradat-Peyre.

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. Berthelot, G.: Transformations and Decompositions of Nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 254, pp. 359–376. Springer, Heidelberg (1987)

3. Buchs, D., Hostettler, S.: Sigma Decision Diagrams. In: Preliminary Proceedings of the 5th International Workshop on Computing with Terms and Graphs, pp. 18–32. No. TR-09-05 in TERMGRAPH workshops, Università di Pisa (2009)
4. Buchs, D., Hostettler, S., Marechal, A., Risoldi, M.: AlPiNA: An Algebraic Petri Net Analyzer. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 349–352. Springer, Heidelberg (2010)
5. Burch, J., Clarke, E., Long, D.: Symbolic Model Checking with Partitioned Transition Relations. In: Halaas, A., Denyer, P.B. (eds.) International Conference on Very Large Scale Integration, pp. 49–58. North-Holland, Edinburgh (1991)
6. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: On well-formed coloured nets and their symbolic reachability graph. In: Jensen, K., Rozenberg, G. (eds.) Proceedings of the 11th International Conference on Application and Theory of Petri Nets, ICATPN 1990. Reprinted in High-Level Petri Nets, Theory and Application. Springer (1991)
7. Chiola, G., Franceschinis, G.: Colored GSPN Models and Automatic Symmetry Detection. In: The Proceedings of the Third International Workshop on Petri Nets and Performance Models, PNPM 1989, pp. 50–60. IEEE Computer Society (1989)
8. Ciardo, G.: Advances in compositional approaches based on kronecker algebra: Application to the study of manufacturing systems. In: 3rd International Workshop on Performability Modeling of Computer and Communication Systems, pp. 61–65 (1996)
9. Ciardo, G., Trivedi, K.: A decomposition approach for stochastic reward net models. Perf. Eval. 18, 37–59 (1993)
10. Ciardo, G., Lüttgen, G., Siminiceanu, R.: Efficient Symbolic State-Space Construction for Asynchronous Systems. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 103–122. Springer, Heidelberg (2000)
11. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-Guided Abstraction Refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
12. Colange, M., Baarir, S., Kordon, F., Thierry-Mieg, Y.: Crocodile: A Symbolic/Symbolic Tool for the Analysis of Symmetric Nets with Bag. In: Kristensen, L.M., Petrucci, L. (eds.) PETRI NETS 2011. LNCS, vol. 6709, pp. 338–347. Springer, Heidelberg (2011)
13. Couvreur, J.-M., Thierry-Mieg, Y.: Hierarchical Decision Diagrams to Exploit Model Structure. In: Wang, F. (ed.) FORTE 2005. LNCS, vol. 3731, pp. 443–457. Springer, Heidelberg (2005)
14. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM 17(11), 643–644 (1974)
15. Esparza, J., Melzer, S.: Verification of safety properties using integer programming: Beyond the state equation. Formal Methods in System Design 16, 159–189 (2000)
16. Evangelista, S.: High Level Petri Nets Analysis with Helena. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 455–464. Springer, Heidelberg (2005)
17. Finnemann, J., Nielsen, T., Kærlund, L.: Petri nets with discrete variables. Tech. rep. (2011), `http://jopsen.dk/downloads/PetriNetsWithDiscreteVariables.pdf`
18. Haddad, S., Kordon, F., Petrucci, L., Pradat-Peyre, J.F., Trèves, N.: Efficient State-Based Analysis by Introducing Bags in Petri Net Color Domains. In: 28th American Control Conference, ACC 2009, pp. 5018–5025. Omnipress IEEE, St-Louis (2009)

19. Hamez, A., Thierry-Mieg, Y., Kordon, F.: Building efficient model checkers using hierarchical set decision diagrams and automatic saturation. Fundamenta Informaticae 94(3-4), 413–437 (2009)
20. Heiner, M., Gilbert, D., Donaldson, R.: Petri Nets for Systems and Synthetic Biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 215–264. Springer, Heidelberg (2008)
21. Hong, S., Kordon, F., Paviot-Adet, E., Evangelista, S.: Computing a Hierarchical Static Order for Decision Diagram-Based Representation from P/T Nets. In: Jensen, K., Donatelli, S., Kleijn, J. (eds.) ToPNoC V. LNCS, vol. 6900, pp. 121–140. Springer, Heidelberg (2012)
22. Hostettler, S., Linard, A., Marechal, A., Risoldi, M.: Improving the significance of benchmarks for petri nets model checkers. In: 1st International Workshop on Scalable and Usable Model Checking for Petri Nets and Other Models of Concurrency, pp. 97–111 (2010)
23. Hostettler, S., Marechal, A., Linard, A., Risoldi, M., Buchs, D.: High-Level Petri Net Model Checking with AlPiNA. Fundamenta Informaticae 113 (2011)
24. Klebanov, V., Müller, P., Shankar, N., Leavens, G.T., Wüstholz, V., Alkassar, E., Arthan, R., Bronish, D., Chapman, R., Cohen, E., Hillebrand, M., Jacobs, B., Leino, K.R.M., Monahan, R., Piessens, F., Polikarpova, N., Ridge, T., Smans, J., Tobies, S., Tuerk, T., Ulbrich, M., Weiß, B.: The 1st Verified Software Competition: Experience Report. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 154–168. Springer, Heidelberg (2011)
25. Kordon, F., Linard, A., Paviot-Adet, E.: Optimized Colored Nets Unfolding. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 339–355. Springer, Heidelberg (2006)
26. Kristensen, L.M., Schmidt, K., Valmari, A.: Question-guided Stubborn Set Methods for State Properties. Formal Methods in System Design 29(3), 215–251 (2006)
27. Kristensen, L.M., Valmari, A.: Improved Question-Guided Stubborn Set Methods for State Properties. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 282–302. Springer, Heidelberg (2000)
28. Lampka, K.: A new algorithm for partitioned symbolic reachability analysis. In: Workshop on Reachability Problems. ENTCS, vol. 223 (2008)
29. Lampka, K., Siegle, M.: Activity-Local State Graph Generation for High-Level Stochastic Models. In: Measuring, Modelling, and Evaluation of Systems 2006, pp. 245–264 (2006)
30. Lampka, K.: A new algorithm for partitioned symbolic reachability analysis. Electron. Notes Theor. Comput. Sci. 223, 137–151 (2008)
31. Lampka, K., Siegle, M., Ossowski, J., Baier, C.: Partially-shared zero-suppressed multi-terminal bdds: concept, algorithms and applications. Formal Methods in System Design 36, 198–222 (2010)
32. Linard, A., Paviot-Adet, E., Kordon, F., Buchs, D., Charron, S.: polyDD: Towards a Framework Generalizing Decision Diagrams. In: 10th International Conference on Application of Concurrency to System Design, ACSD 2010, pp. 124–133. IEEE Computer Society, Braga (2010)
33. Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In: Proc. of the 30th Design Automation Conference, DAC, pp. 272–277. ACM/IEEE, Dallas (Texas), USA (1993)
34. Pastor, E., Roig, O., Cortadella, J.: Symbolic Petri Net Analysis using Boolean Manipulation, Technical Report of Departament Arquitectura de Computadors (UPC) DAC/UPC Report No. 97/8 (1997)

35. Peterson, G.L.: Myths about the mutual exclusion problem. Inf. Process. Lett. 12(3), 115–116 (1981)
36. Schmidt, K.: Narrowing petri net state spaces using the state equation. Fundamenta Informaticae 47(3-4), 325–335 (2001)
37. Schmidt, K.: Stubborn Sets for Standard Properties. In: Donatelli, S., Kleijn, J. (eds.) ICATPN 1999. LNCS, vol. 1639, pp. 46–65. Springer, Heidelberg (1999)
38. Schmidt, K.: Using Petri Net Invariants in State Space Construction. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 473–488. Springer, Heidelberg (2003)
39. Thierry-Mieg, Y., Ilié, J.-M., Poitrenaud, D.: A Symbolic Symbolic State Space Representation. In: de Frutos-Escrig, D., Núñez, M. (eds.) FORTE 2004. LNCS, vol. 3235, pp. 276–291. Springer, Heidelberg (2004)
40. Thierry-Mieg, Y., Poitrenaud, D., Hamez, A., Kordon, F.: Hierarchical Set Decision Diagrams and Regular Models. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 1–15. Springer, Heidelberg (2009)
41. Wikipedia: Dining philosophers problem (2011), `http://en.wikipedia.org/wiki/Dining_philosophers_problem`
42. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri Net State Equation. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 224–238. Springer, Heidelberg (2011)
43. Wolf, K.: Generating Petri Net State Spaces. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 29–42. Springer, Heidelberg (2007)