COMPARATIVE ANALYSIS OF TOOLS FOR QUANTITATIVE FORMAL MODELS ON JANI, PRISM AND MODEST INPUTS

Mohd Asif Khan Khaishagi
Information Technology
National Institute of Technology, Karnataka
Surathkal, Mangalore, Karnataka, India
asifk.202it013@nitk.edu.in

Praful Kumar
Information Technology
National Institute of Technology, Karnataka
Surathkal, Mangalore, Karnataka, India
praful.202it020@nitk.edu.in

Madhusmita Das
Information Technology
National Institute of Technology, Karnataka
Surathkal, Mangalore, Karnataka, India
madhusmitadas.197it004@nitk.edu.in

Biju R Mohan
Information Technology
National Institute of Technology, Karnataka
Surathkal, Mangalore, Karnataka, India
biju@nitk.edu.in

***Abstract*—Quantitative Formal models are used for the analysis of properties like probabilistic analysis and states of a model. There are several different tools like ePMC, modes, mcsta, STORM, PRISM, UPPAL, DREAM, SPIN, NuSMV which are available for automatic analysis of the models. Different tools have different merits and have preferred use cases. In this paper we have taken ePMC, modes, mcsta, STORM and PRISM tools are compared them on the basis of four formalisms which are based on Markov Chain: CTMC, DTMC, MDP and MA. The input format considered for the different formalisms are JANI and PRISM, which are supported by the Quantitative formalisms considered in the experiments. Data is taken from the QComp2019 competition, which was held for the comparsions of tools and mainly focused on drawing conclusions about the tools. Code for running scripts and generated results and charts are available at https://github.com/asifk1997/QCompToolPerformanceAnalysis**

***Index Terms*—CTMC, DTMC, MDP, MA, ePMC, modes, mcsta, STORM, PRISM**

I. Introduction

The quantitative formalisms are the extension of Formal automata. It is nothing but adding the probabilities of transmissions and certain other parameters depending on the formalism used. Quantitative formalism is required for the probabilistic analysis of models so that one may determine the behaviour of the model in future with the help of given input. Generally, the input here is given in the form of states and their probabilities of transmission from one state to another and in some cases like CTMC, the state residence time is also given which also determines the total time a system may remain in a particular state.

One can easily determine these transitions and states in case of small systems by doing calculations or making simple programs. In case of complex systems where a system may contain many states and a large number of transitions, it becomes an almost impossible task to perform calculations correctly and then there is requirement of certain tools which may do these tasks for the user and give them correct results. However, the availability of large number of tools makes the selection of tools a confusing task and the user might end up selecting the wrong tool or failing to perform the required task.

In this paper we have done the comparative analysis of the different Model checking tools for their performance on different models. Some of the tools were taken from the competition QComp2019[1] which was the competition held for the comparison of these tools. The nature of the competition was such that it focused on the comparison of tools rather than declaring a winner on who constructed the best model. The tools that we have compared are ePMC[4], modes[6], mcsta[7], PRISM[5] and STORM[3]. We have selected these tools on the basis of some common characterisitics like Quantitative formalism amongst them and the comparison is done on the basis of common outputs that are shown by different tools. We have focused mainly on Peak memory usage, Property time and Property values which are shown in the output of these tools. Formalisms that we have considered are CTMC, DTMC, MDP and MA, out of which CTMC, DTMC and MDP support JANI and PRISM inputs and MA support JANI and MODEST inputs.

The paper further has section II which discusses the background concepts in detail, containing the Quantitative Formalisms, Input format and Model checking tools, section III discusses Experimental setup that we have used for the experiments, section IV tells the case studies used for different formalisms, section V discusses the experimental results and in the end in section VIII we have put the conclusions on the basis of experimental results.

II. Literature survey

Verification of Markov Decision Processes Using Learning Algorithms[8] verifies the MDP formalism by taking two different instantiations. The goal was to avoid exhaustive exploration of state space. Also, the main focus of the paper was probabilistic reachability, which is the core property of the formal model verification. The first assumes that full knowledge of the MDP is available, and performs a heuristic-driven partial exploration of the model, yielding precise lower and upper bounds on the required probability. The second tackles the case where we may only sample the MDP, and yields probabilistic guarantees, again in terms of both the lower and upper bounds, which provides efficient stopping criteria for the approximation

A Statistical Model Checker[9] is a formal verification technique for stochastic systems and it can be used for state space exploration and probabilistic rechability of those states. This can even answer questions like "what is the probability that a website working today will be working tomorrow also" or "what is the expected time to complete a given task". This is popular due to it's ability of handling non-Markovian models and complex continuous systems efficiently unlike the other models. SMC is the integration of formal models and the Monte Carlo simulation. They presented MODES as the Statistical model checker as it is efficient in calculating probabilities of the rare events and also statistically approximate the optimum schedulers. MODES is a part of MODEST toolset which also includes MCSTA and MOTEST tools. These tools accept MODEST and xSADF input format. The earliest version of the MODES was introduced in 2012 and focused on partial-order and conluence reduction based techniques and focused on determining that for a non-deterministic model the maximum and minimum values are same and if the implicit randomized scheduler can be safely used. In 2014, MODES was made part of the MODEST toolset and was redesigned. The version 3 of the MODEST toolset provides access to new modeling features and formalisms and even support for the JANI input format.

Quantitative Model and Tool Interaction[10] JANI specification is an input format for model exchange and tool interaction. The goal of JANI is to make model specification process easy and reduce the effort required to develop verification tools for comparison of differnt models. Supporting the JANI input format gives access to a large number of existing models for testing and bechmarking purposes as it focuses on the code that is easily human-writable and easily machine-readable. JANI is supported by many of the popularly used quantitative verification tools. JANI model allows the conversation of several high-level modeling languages to JANI-model and also vice-versa insome cases. Some popular examples are IOSA, MODEST, xSADF, pGCL and PRISM. JANI model is supported by FIF, ISCASMC, the MODEST toolset, STORM and PRISM by converting JANI-model to PRISM with the help of ISCASMC tool.

Fast Debugging of PRISM Models[11] Along with the specificatoins the model checking can also provide feedback on some counterexamples to make the case studies more understandable. The technique in this paper to compute minimal crtical command set provides five times faster results in comparison to the previously defined techniques. Conuterexample generation in the tools can convince the users about the importance of formal verification. Some model checkes like SPIN and NuSVM provide this feature of generating counterexamples which even tell how the change in certain variable or a property will have effect on the model and the final results in the form of property values. Such features of generating counterexamples in not there in many popular probabilistic model checking tools like PRISM and MRMC. In the approach proposed here they generate counterexamples in PRISM by deleting the commands of PRISM probabilistic program resulting in smallest PRISM probabilistic program violating the reachability property at hand. This problem is NP-hard. The idea behind the approach is to deduce necessary conditions for an optimal solution by a static analysis of the probabilistic program. MAXSAT solver is used to compute the smallest set of commands that in in accordance with the given set of constraints.

A Modern Probabilistic Model Checker[12] The STORM tool is capable of checking both the discrete and continuous variations of Markov chain based models and Markov decision process. The purpose of this model checking tool is ease of use by the user for the various formalisms, verification algorithms, probilistic models and various other properties, this is unlike most other open source tools available. The STORM tool is coded in C++ with around 1000,000 lines of code. The use of C++ also makes this model faster as it is faster language as compared to the other languages present for the development of tools and softwares. The STORM tool does not support Statistical model checking, however it relies on numerical and symbolic model checking. The main specilities of STORM tool are that it supports the widely used input foramts like th PRISM input, generalized Petri nets, dunamic fault trees and conditional probabilistic programs. It also supports MA(Markov Automata) along with the MDP. It can do explicit state model checking along with the fully symbolic model cheking. It also provides the python API which helps in the prototyping of the other tools with ease. It can also generate counterexamples along with the permissive schedulers. Also, the performance of STORM is better than other tools on the PRISM input. Along with so many merits, there are certain demerits of the STORM tool like it does not support the LTL model checking unlike the PRISM tool.

Verification of Probabilistic Real-Time Systems[13] the PRISM 4.0 release of the PRISM model checker was added with features like the quantitative verification of probabilistic timed automata showing real-time and non-deterministic properties of the tools also. PRISM is an open source tool with certain features like building, verifying and refining probabilistic models. The PRISM model checking tool also provides statistical model checking. The open source PRISM model checking tool is coded in a mix of Java and C++. The

main functionalities of the PRISM model checking tool are modelling and construction of various probabilistic models along with costs and rewards, a variety of model of model checking engines both symbolic and symmetric state, a GUI with model editor, simulator and graphing along with the command line tool.

III. Background

A. Quantitative formalisms

There are various methods for verifying the correctness or performance of any program or model. Here, we define Quantitative Formalisms as the mathematical foundation for the formal verification of the Quantitative models that will be defined further in the paper. We can also say that these formalisms are mathematicaaly well defined class of objects that form the semantics of any concrete model.

For the formalisms defined below, Markov chains play a role of foundation. Markov chains, named after Andrey MArkov are nothing but a probabilistic model that determine the state changing probabilities at every state. It can simply be defined as the extension of the automata(Deterministic/Non-Deterministic) in which we add the probabilities of all the transactions in each state and also adding the time in each specifying how long the system stays in that particular state. The Markov chains follow markov property, that states - *" The future evolution of the sysytem is independent of its past states"* OR *"The current state of the model contains all the information that influence its's future evolution"*. This property is also called Memorylessness.

The four formalisms based on Markov Chains that we have taken in this paper are as follows:

1) DTMC: The *Discrete Time Markov Chain* is a a sequence of random variables where the steps in the sequence is defined by the set of states and the probabilities associated with each state. Here the probabilities are represented by using a stochastic matrix of size SxS where S is the set of countable states. A matrix is a stochastic matrix if all the probabilities from a state are in the range [0,1] and the sum of all the probabilities from any state is equal to 1.

2) CTMC: The *Continuous Time Markov Chain* is a chain of sequence of random variables just like the above defined DTMC. However, a random variable **X** is added in every state, which determines the state residence time in that state. Now, the process will stay in the state for the expected time **E[X]** and will jump to the nect state with the probabilities according to the stochastic matrix similar to DTMC.

3) MDP: The *Markov Decision process* formalism is based on the action reward feedback i.e. it takes actions according to the actions taken and feedback received on them. Here, the probabilities depend on the four factors: Current state, Next state, Reward and Action. Here also all the probabilities are in the range [0,1] and sum of all the probabilities from a state is equal to 1.

4) MA: The *Markov Automata* is useful when we want to determine the system which are non-deterministic, and depend on probability and time delays of the states.

According to the above Markov chain formalisms we can derive the following conclusions:

- if there is any input after every state in the schedule, we may use the MDP formalism
- if the time is discrete, then we may use DTMC. However, if the time is continuous then we may use CTMC.

B. Input formats

It is easy to model simple systems by using the above mentioned formalisms. But, if we have to model complex systems then it would become a very inconvinient task for us. Hence, there is a requirement of the Input formats which are simply modelling languages that allow us to compactly describe large automata of various complexities. In this paper we have used PRISM, JANI and Modest modelling languages for the tools used further for the experimentation.

C. Model checking tools

Here, we have taken five general purpose model checking tools for the comparative analysis and the final results are produced on the basis of their performances.

1) ePMC: [4] It is written in Java and C and runs on Windows, Linux and MAC Operating systems. It has a small core and plugins provide the additional functionalities like model parsing and model checking. The availability of plugins in ePMC[4] tool make it an excellent choice for the purpose of model parsing, checking and many other purposes. ePMC[4] supports PRISM and JANI inputs and DTMC, CTMC, MDP and stochastic games as formalisms.

2) modes: [6] It is a tool from the Modest toolset and includes nondeterministic choices, continuous system dynamics, probabilistic decisions and timings and even real time behaviour including non deterministic delays.

3) mcsta: [7] It is written in C sharp and works well on Linux and Mac Operating systems. The mcsta[7] is model checker from the Modest toolset built on it's infrastructure, it supports MODEST, JANI and xSADF inputs and also it compiles models to bytecode. This tool can easily analyse MDP and MA formalisms, but will analyse DTMC and CTMC as special case of MDP and MA respectively. Hence, it doesn't get the performance of dedicated algorithms.

4) PRISM: [5] It is mainly written in Java with some parts in C++ and also runs on Windows, Linux and MAc Operating systems. Being an open source tool it can be connected to many other extensions in order to increase it's functionalities. PRISM tool supports PRISM as the input format. This tool also has GUI which enable us to feature model checking, simulation and even graph plotting.

5) STORM: [3] This tool accepts PRISM and JANI inputs along with dynamic fault trees, probabilistic programs and stochastic petri nets. This supports the analysis of DTMC, CTMC, MDP and MA formalisms. It includes C++ API, Python API and and Command line interface to access tool's features. It can be run on Linux and Mac operating systems.

IV. Experimental setup

For performing the experiments we have used the ACER PREDATOR HELIOS 300 laptop computer. It has Intel core i5 8th generation processor, 8GB RAM, 4GB NVIDIA GeForce GTX 1050Ti GPU, 128GB SSD and a 1TB of HDD. This is a high performance personal laptop computer.

The tools were installed on the Linux operating system and we performed the results on the single machine in order to provide same environment for all the experiments. We have used Linux operating system for the experiments because of the ease of work and familiarity with the Linux terminal as compared to the Windows' command prompt. For the purpose of scripting we have used python3 programming language. We used plotly library for plotting graphs. The versions of the tools taken are the latest ones that are available at the time of performing the experiments. Tool versions are as follows.

- STORM VERSION : Storm 1.6.4 (dev) DEBUG BUILD
- EPMC revision df0f89c42d066118f8258ec42a231bdc58bfaa8f
- MODES : Modest Toolset, version v3.1.75-gcc6169502
- MCSTA : Modest Toolset, version v3.1.75-gcc6169502.
- PRISM : Version: 4.6

V. Case studies

We have taken the following case studies from the QComp2019 website[2]: The filenames in the table are exact filenames which are taken from Qcomp2019 website[2]. And we have done run all the properties but for further experiments we have considered only one property from each model. We first run most of the files on storm tool and those who were running successfully were chosen.

1) CTMC
   - Cluster
   - Embedded
   - FMS
   - Kanban
   - Polling
2) DTMC
   - BRP
   - Crowds
   - EGL
   - Herman
   - Leader-sync
3) MDP
   - Consensus
   - CSMA
   - Firewire
   - Pacman
   - Philosophers-MDP
4) MA
   - Bitcoin-attack
   - Breakdown-queues
   - DPM
   - Erlang
   - FTWC

All Case Studies			
Type of Formal Model	Filename	Property	Constants
CTMC	cluster.v1	qos1	N=2, T=2000, t=20
CTMC	embedded.v1	actuators	MAX_COUNT=2, T=12
CTMC	fms.v1	productivity	n=1
CTMC	kanban.v1	throughput	t=2
CTMC	polling.12.v1	s1_before_s2	T=16
DTMC	brp.v1	p1	N=16, MAX=2
DTMC	crowds.v1	positive	TotalRuns=3, CrowdSize=5
DTMC	egl.v1	messagesA	N=5, L=2
DTMC	herman.3.v1	steps	
DTMC	leader_sync.3-2.v1	time	
MA	bitcoin-attack.v1	T_MWinMin	MALICIOUS=20, CD=6
MA	breakdown-queues.v2	Min	K=8
MA	dpm.v2	PminQueuesFull	N=4, C=4, TIME_BOUND=1
MA	erlang.v2	PminReach	K=5000, R=10, TIME_BOUND=5
MA	ftwc.v3	TimeMax	N=4, TIME_BOUND=5
MDP	consensus.2.v1	c2	K=16
MDP	csma.2-2.v1	all_before_max	
MDP	firewire.false.v2	time_max	delay=3, deadline=200
MDP	pacman.v1	crash	MAXSTEPS=5
MDP	philosophers-mdp.3.v1	eat	

VI. Experiments

All the tools were run in their default mode of execution. We have set a timeout of 10 minutes i.e. if some model execution takes more than 10 minutes we halt its execution. We first created a script in python in order to perform the experiments and then the results obtained were collected in a csv file, which were later used for the purpose of plotting graphs.

A. *Peak memory usage*

Peak Memory usage is the amount of maximum memory used when running the given input file and calculating the property values. We have compared models of particular formalism with other. We took different tools and run them on specific formalism model and noted down their peak memory usage and then we made charts to visualize them. Peak memory usage is important property to know if you are working on a low memory system. So your computer should atleast that much free memory or else the model will not run completely or else throw error. To run the program correctly and completely peak memory usage condition should be abided by the system.

In Fig 1,2,3,4 JANI input files were used and the files were run on the MODES[6], MCSTA[7] and STORM[3] tool.Here, those inputs which did not run properly or threw an exception are given the value 0 in the charts above. The value 0 is very rarely present in case of STORM[3] tool, this shows that it is vary robust, but along with that it has the highest peak memory usage among all the tools. The comparison was done on MODES[6], MCSTA[7] and STORM[3] because only they give peak memory usage after the model run. Also MODES[6] and MCSTA[7] support only jani and modest input so we couldn't calculate their prism input counterpart comparison.

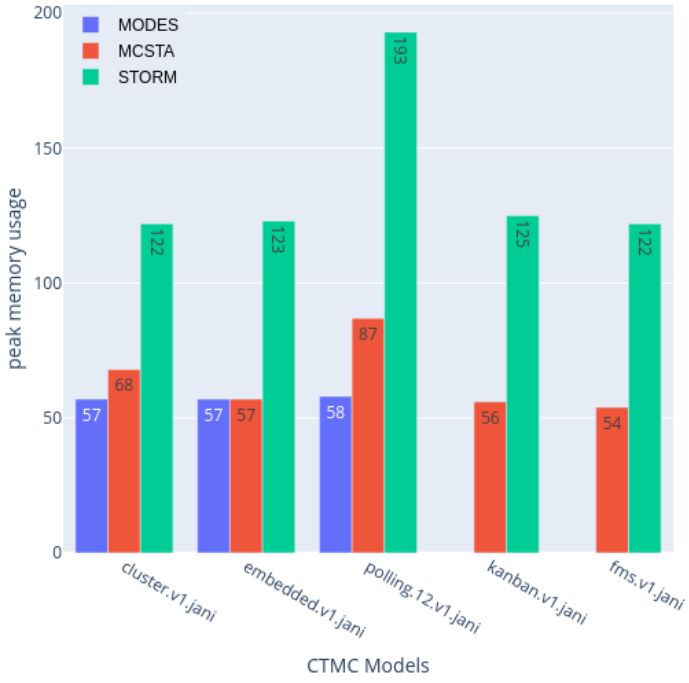

Fig. 1. CTMC Peak memory usage for MODES, MCSTA and STORM tools.

In Fig. 1 we took 5 CTMC models and run them and noted their peak memory usage and created chart based on that. Here the peak memory usage is highest for STORM[3] and MCSTA[7] is almost half of STORM[3]. MCSTA[7] slightly higher than MODES[6].

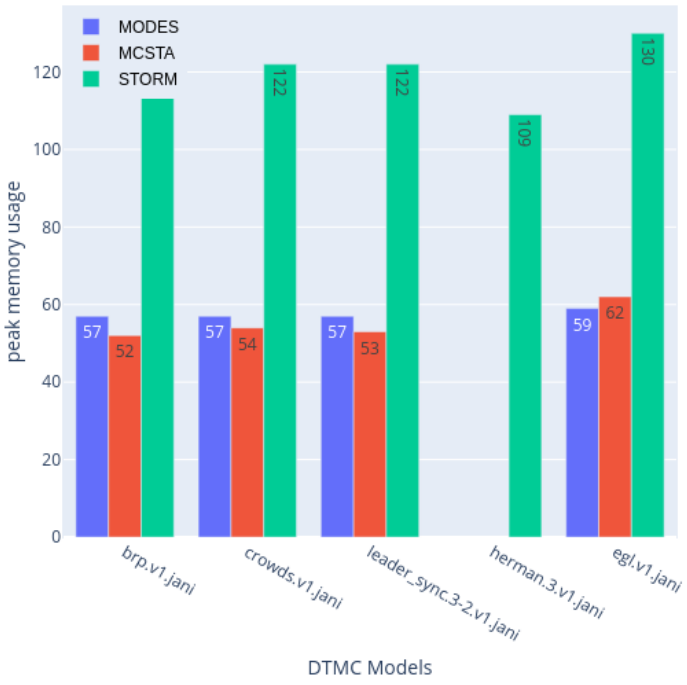

Fig. 2. DTMC Peak memory usage for MODES, MCSTA and STORM tools.

In Fig. 2 we took 5 DTMC models and run them and noted their peak memory usage and created chart based on that. Here the peak memory usage is highest for STORM[3] and MCSTA[7] is almost half of STORM[3]. MCSTA[7] slightly lower than MODES[6] in 3 cases.

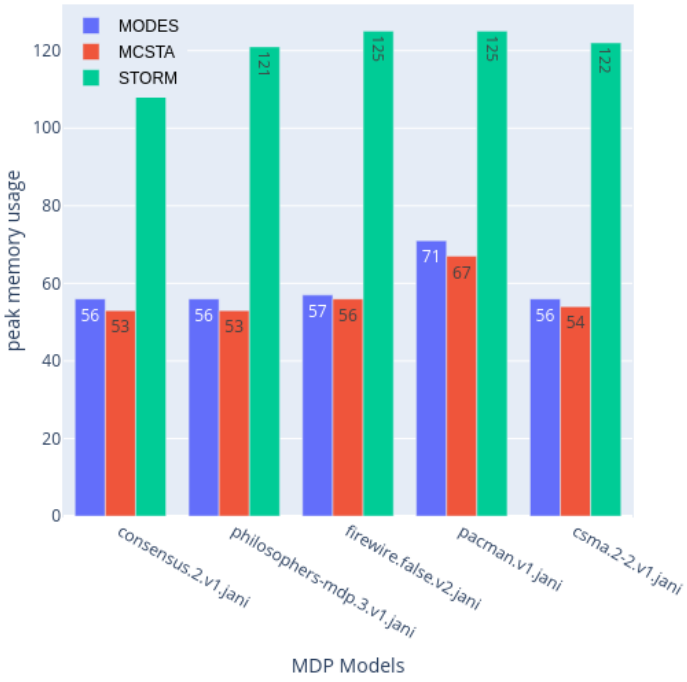

Fig. 3. MDP Peak memory usage for MODES, MCSTA and STORM tools.

In Fig. 3 we took 5 MDP models and run them and noted their peak memory usage and created chart based on that.

Here the peak memory usage is highest for STORM[3] and MCSTA[7] is almost half of STORM[3]. MODES[6] slightly higher than MCSTA[7].

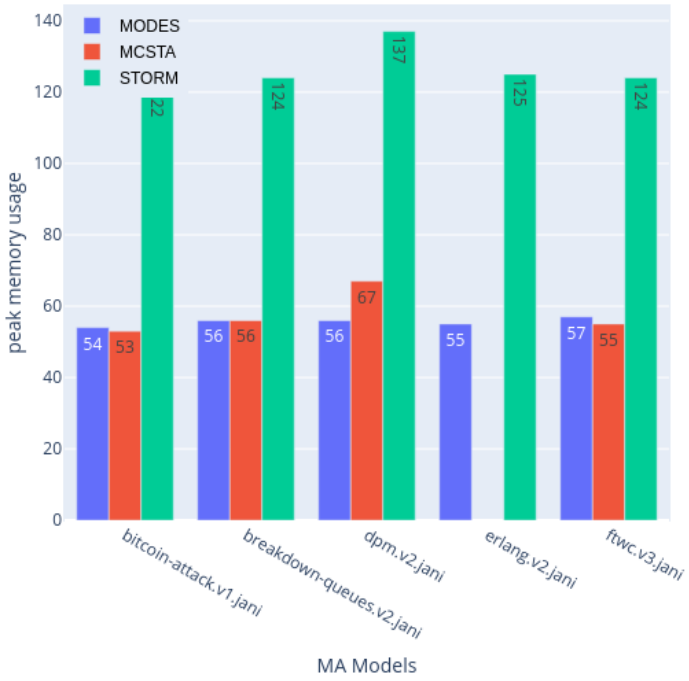

Fig. 4. MA Peak memory usage for MODES, MCSTA and STORM tools.

In Fig. 4 we took 5 MA models and run them and noted their peak memory usage and created chart based on that. Here the peak memory usage is highest for STORM[3] and MCSTA[7] is almost half of STORM[3]. MCSTA[7] is almost equal to MODES[6] on three models.

Based on this charts in Fig 1,2,3,4 we can see that peak memory usage of storm tool is higher than other tools. MODES[6] and MCSTA[7] have almost similar peak memory usage.

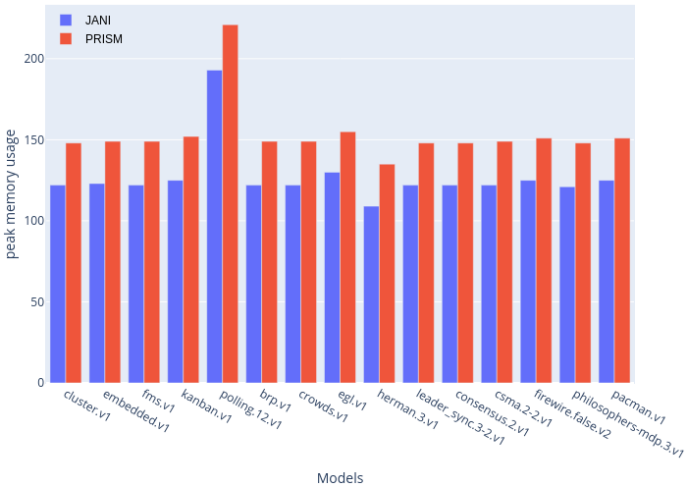

Fig. 5. Peak memory usage for JANI and PRISM inputs on STORM tool.

In Fig. 5 we compared the STORM[3] tool peak memory usage on different inputs namely JANI and PRISM. The JANI files and PRISM files were taken from QComp2019 website [2]. Here the PRISM input files took more peak memory usage than JANI input files.

By observing the above experiments we can conclude that among the tools taken for comparison the STORM[3] tool shows the highest peak memory usage as compared to the other modeling tools and the PRISM input has more peak memory usage when compared to the JANI input.

B. Property Values Comparison

Property values are the values like probability of reaching the state or number of steps in reaching a certain state. These values vary from model to model and have different range in different cases. In some case studies, the tool shows error which means there is an exception in running the model. Incompatible in some cases mean that the tool is not compatible with the given input.

Here we took a sample of values to see if there is any difference between the output of properties on a model with different properties. When running the model most of them have values similar upto 1 decimal places and then there is variation. On this bargraphs we have written what actual output was on the particular bar of model to compare. We have taken some values out of total experiments for plotting the graph.

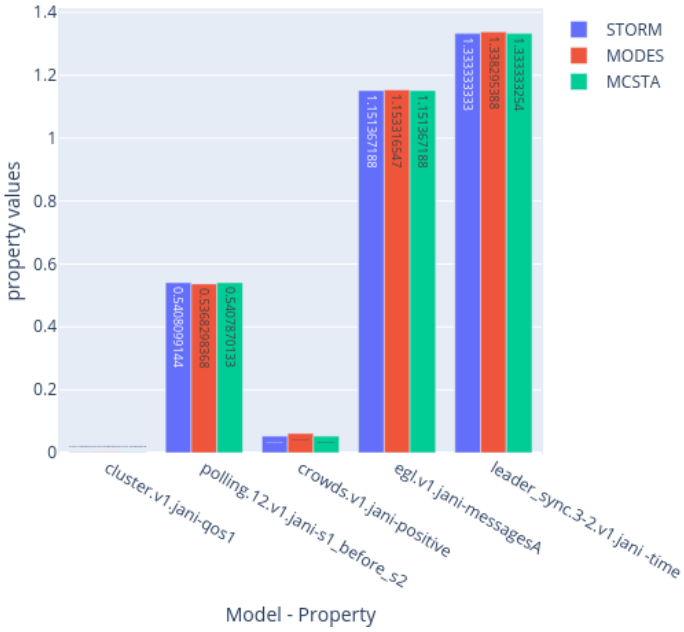

Fig. 6. Property values in STORM, mcsta and MODES tools.

In Fig. 6 we selected few models to check if there is variation in the property values. The values are written on the bar plot. There are slight variations after one decimal place.

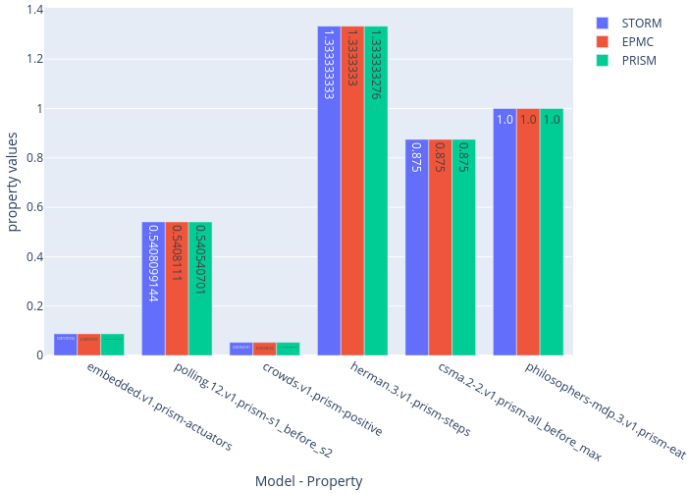

Fig. 7. Property values in PRISM, STORM and ePMC tools.

In Fig. 7 we selected few models to check if there is variation in the property values. The values are written on the bar plot. There are slight variations after two decimal place.

C. Property Time Comparison

Property time is basically the time taken to evaluate a certain property value in a model. This is important to know which tool runs fast or slow to evaluate a model performance based on user requirement. The property time comparisons in Figure 8, Figure 9 and Figure 10 shows us that when comparing STORM[3] and MCSTA[7] tools, in most of the cases the MCSTA[7] tool takes longer time than STORM[3] tool. However, comparing the STORM[3], ePMC[4] and PRISM[5] tools shows that generally ePMC[4] has the highest property times and among storm and prism the comparison is not clear since storm takes longer on some inputs and prism on another. Comparison of JANI and PRISM input formats shows that there is no major difference in the property times shown by them but the prism input is slightly higher on STROM[3] tool.

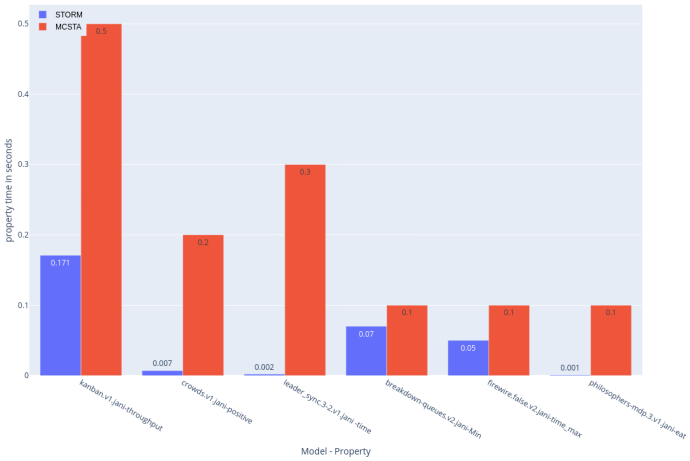

Fig. 8. Property Times in STORM, MCSTA tool on jani input.

In Fig. 8 we took few jani models to check which tool takes highest time. MCSTA[7] takes higher time than STORM[3] in most of the cases.

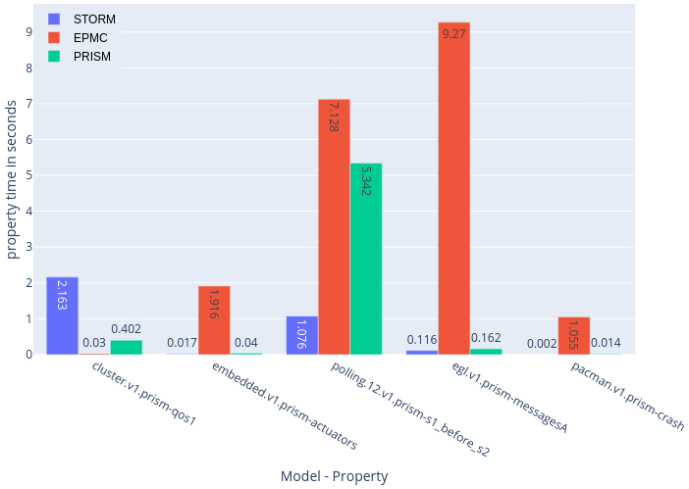

Fig. 9. Property Times in STORM, ePMC and PRISM tools on prism inputs.

In Fig. 9 we took few prism models to check which tool takes highest time. In most of the cases ePMC[4] takes highest time and then PRISM[5] and then STORM[3]

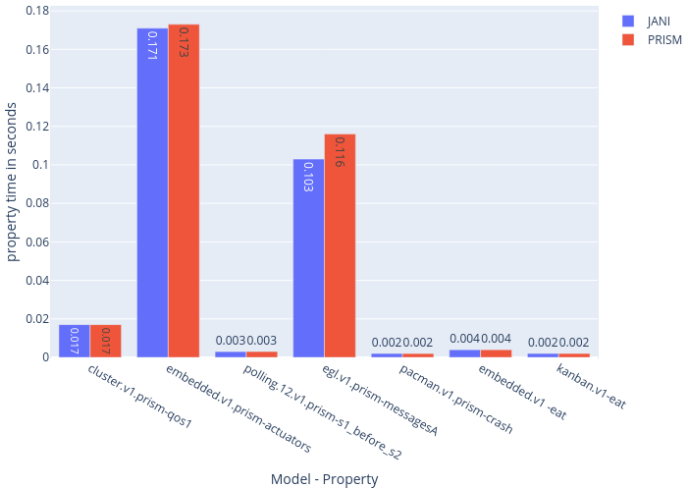

Fig. 10. Property Times in JANI and PRISM input formats on STORM tool.

In Fig. 10 we took some of the models of jani and their prism conterparts and there is slight difference between jani and prism inputs with prism input being slightly higher than jani input.

VII. Observations

By performing the experiments and observing the results we can derive the following conclusions:

- There is considerable difference between same input model having different input types.

- STORM has more peak memory usage than other tools. However, modes and mcsta have similar peak memory usage.
- The peak memory usage is greater when we have PRISM input format as compared to the JANI input format.
- Property time is more in case of ePMC tool.
- Property values are similar in most of the cases for all the tools. However, in some cases STORM and mcsta show slightly lesser values.
- When taking PRISM and JANI inputs on STORM tool, the property time is similar.
- CPU time is generally greater than wall clock time because there are more cores in the system we are doing experiment on.
- ePMC doesn't work on most of the inputs compared to other tool which we have taken.
- Prism tool need to converge in default mode before 10000 iterations else it throws error.

VIII. Conclusion

Comparing tools for specific property is important when you want to use a tool for specific task.Here we have compared the tools based on specific properties namely peak memory usage,their property values and property time. STORM[3] tool works fast compare to other tools but the peak memory usage is also higher, ePMC[4] tool takes much higher time compared to other tools and also throws error on most of inputs. Prism tool works well but falls behind by storm in terms of speed. MODES[6] and MCSTA[7] both perform similarly as they are part of same Modest checker tool, MCSTA[7] takes more time in most of the cases as in Fig. 8. STORM[3] tool is good for comparing different input types as it support PRISM as well as JANI inputs.

References

[1] Ernst Moritz Hahn, Arnd Hartmanns, Christian Hensel, Michaela Klauck, Joachim Klein, Jan Křetínský, David Parker, Tim Quatmann, Enno Ruijters, Marcel Steinmetz [*The 2019 Comparison of Tools for the Analysis of Quantitative Formal Models*]. QComp 2019 Competition Report, 2019.

[2] QComp 2019 competition website, `https://www.qcomp.org/`

[3] The Probabilistic model checker STORM `https://www.stormchecker.org/index.html`

[4] EPMC - (An Extendible Probabilistic Model Checker) `https://github.com/ISCAS-PMC/ePMC`

[5] PRISM MODEL CHECKER `https://www.prismmodelchecker.org/`

[6] MODES : Disk-based explicit-state model checker `https://www.modestchecker.net/`

[7] MCSTA : Statistical model checker `https://www.modestchecker.net/`

[8] Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Kwiatkowska, David Parker, and Mateusz Ujma [*Verification of Markov Decision Processes Using Learning Algorithms*]. Verification of MDP formalism

[9] Carlos E. Budde, Pedro R. D'Argenio, Arnd Hartmanns, Sean Sedwards [*Statistical model checker*]. A Statistical Model Checker for Nondeterminism and Rare Events

[10] Carlos E. Budde, Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges, Andrea Turrini *The JANI model*JANI: Quantitative Model and Tool Interaction

[11] Christian Dehnert, Nils Jansen, Ralf Wimmer, Erika Ábrahám, Joost-Pieter Katoen *PRISM models debugging*. Fast Debugging of PRISM Models

[12] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk *STORM tool paper*. A Storm is Coming:A Modern Probabilistic Model Checker

[13] Marta Kwiatkowska, Gethin Norman, and David Parker *PRISM 4.0 tool*. PRISM 4.0: Verification of Probabilistic Real-Time Systems