A Model Checker for Hierarchical Probabilistic Real-time Systems*

Songzheng Song[1], Jun Sun[2], Yang Liu[3], Jin Song Dong[4]

[1] NUS Graduate School for Integrative Sci and Engineering, National University of Singapore
songsongzheng@nus.edu.sg
[2] Information System Technology and Design, Singapore University of Technology and Design
sunjun@sutd.edu.sg
[3] Temasek Lab, National University of Singapore, tslliuya@nus.edu.sg
[4] School of Computing, National University of Singapore, dongjs@comp.nus.edu.sg

Abstract. Real-life systems are usually hard to control, due to their complicated structures, quantitative time factors and even stochastic behaviors. In this work, we present a model checker to analyze hierarchical probabilistic real-time systems. A modeling language called PRTS is used to specify such systems, and automatic zone-abstraction approach, which is probability preserving, is used to generate finite state MDP. We have implemented PRTS in model checking framework PAT so that friendly user interface can be used to edit, simulate and verify PRTS models. Some experiments are conducted to show our tool's efficiency.

1 Introduction

Real-life systems could be complicated because of hierarchical structures and complex data operations; real-time behaviors are sometimes essential in such systems due to the interaction with the real world; in addition, unreliable environments could result in stochastic behaviors so that probability is necessary. These characteristics present the difficulty in properly designing and developing such systems. Applying model checking techniques in this domain is therefore very challenging, due to the requirements of an expressive enough modeling language as well as efficient model checking algorithms.

In this work, we present a new model checker to analyze complex systems. A modeling language Probabilistic Real-time System (PRTS) [16] is used to cover complicated system structures and data operations, real-time behavior and probability, meanwhile *dynamic zone abstraction* [16] is applied to handle the infinite state space caused by real-time factors. Different from zone abstraction used in other models such as Probabilistic Timed Automata (PTA) [10], our approach guarantees forward analysis after abstraction is precise. PRTS supports several widely used properties such as reachability checking, LTL checking and reward checking, with which users could analyze different aspects of the system. Our tool (public available at [1]) has been developed as a stand alone plug-in module in the verification framework PAT [14, 11] to support the editing, simulation and verification of PRTS models with a friendly user interface.

* This research was partially supported by research grant "SRG ISTD 2010 001" from Singapore University of Technology and Design and "MOE2009-T2-1-072" from MOE of Singapore.

Related Work There are several model checkers exploring probabilistic real-time systems based on PTA. UPPAAL [4] supports real-time, concurrency and recently data operations as well as probability (UPPAAL-PRO), but lacks support for hierarchical control flow and is limited to maximal probabilistic reachability checking. PRISM [9] is popular in verifying systems having concurrency, probability and the combination of real-time and probability. However, it does not support hierarchical systems, but rather networks of flat finite state systems. Another tool mcpta [6] supports the verification of PTA by translating models into PRISM and only supports reachability checking. In addition, these tools only support simple data operations, which could be insufficient in modeling systems which have complicated structures and complex data operations.

2 Modeling with PRTS

In this section, we briefly introduce our modeling language PRTS, which extends Communicating Sequential Processes (CSP) with real-time and probabilistic behaviors.

Syntax A subset of process constructors of PRTS are listed below to present its modeling abilities. Note that process constructors, like (conditional) choice, sequential and parallel compositions adopted from CSP for modeling hierarchical concurrent systems, are skipped due to the space limitation and readers can refer to [16] for details.

$$P = a\{program\} \rightarrow P \mid Wait[d]P \ timeout[d] \ Q \mid P \ interrupt[d] \ Q \\ \mid P \ deadline[d] \mid P \ within[d] \mid pcase\{pr_0 : P_0;\ pr_1 : P_1;\ \cdots;\ pr_k : P_k\}$$

Data Operation PRTS supports shared memory models using global variables, which can be integer, boolean, integer array, and even arbitrary user-defined data structures. A user-defined data structure can be defined externally using programming languages like C#, Java, C and so on, and then imported into the model[5]. Data operations in PRTS are invoked through syntax $a\{program\} \rightarrow P$, which executes event a and $program$ simultaneously, and behaves as P afterwards.

Real-time Several timed process constructors are supported in PRTS to capture the real-time behaviors of the system. Process $Wait[d]$ idles for d time units, where d is an integer constant. In $P \ timeout[d] \ Q$, the first observable event of P shall occur before d time units elapse (since the process is *activated*). Otherwise, Q takes over control after d time units. $P \ interrupt[d] \ Q$ behaves as P until d time units elapse, and then Q takes over control. PRTS extends Timed CSP [13] with additional timed process constructs. $P \ deadline[d]$ constrains P to terminate before d time units. $P \ within[d]$ requires that P must perform an observable event within d time units.

Probability $pcase\{pr_0 : P_0;\ pr_1 : P_1;\ \cdots;\ pr_k : P_k\}$ is used to model the randomized behaviors of a system. Here pr_i is a positive constant to express the probability weight. Intuitively, it means that with $\frac{pr_i}{pr_0+pr_1+\cdots+pr_k}$ probability, the system behaves as P_i. Obviously the sum of all the probabilities in one $pcase$ is 1.

Note that the probabilistic real-time systems modeled in PRTS can be fully hierarchical, since P and Q in the above constructors can be any processes. This is different from PTA based languages which often have the form of a network of flat PTA.

[5] Details can be found in PAT's user manual.

Operational Semantics The semantic model of PRTS is Markov Decision Processes (MDP) because of its mixture of non-deterministic and probabilistic choices. Note here we assume the valuations of variables and the processes reachable from the initial configuration are finite, therefore an MDP can have infinite states only due to its dense time transitions. In [16], we have defined $concrete\ firing\ rules$ and $abstract\ firing\ rules$ respectively. The former describes the operational semantics of PRTS, while the latter captures the execution behaviors of PRTS models after $zone\ abstraction$. This abstraction is necessary since it generates finite state space from a PRTS model so that traditional probabilistic model checking techniques can be used.

Our automatic zone abstraction approach is probability preserving for several properties such as reachability checking and LTL checking. A proof sketch is as follows: given a concrete MDP and one of its scheduler, a discrete-time Markov Chain (DTMC) can be defined; we can always build a corresponding DTMC in the abstract MDP to guarantee these two DTMC are time-abstract bi-similar, and vice versa [18, 16].

We remark that forward reachability of PTA using zone abstraction is not accurate [10]. In PTA, given an abstract DTMC defined by the abstract model and a scheduler, it is possible that there is no corresponding concrete DTMC can be defined from the concrete model. Therefore the maximum (minimum) probability of reachability property in the abstract model is an upper (lower) bound of the accurate result. Some approaches such as [8] are used to solve this problem.

3 System Analysis

In our model checker, PRTS models can be analyzed by the built-in editor, simulator and verifier, through which we could investigate system behaviors of the models. In this section we briefly present how the simulator and verifier work.

3.1 Simulation

Our tool provides a discrete-event simulator which allows users to interactively and visually simulate system behaviors. In simulation, PRTS models follow the abstract operation semantics in order to guarantee that each step reflects a meaningful execution of the system. Users could choose automatic simulation, which means the simulator will randomly execute the model and generate the random states, or manual simulation, which allows users to choose next event from the current enabled events. Through simulation, users could visually check how the model executes step by step, which is very useful in system design and analysis, especially when there are some undesired executions found in verification. Simulation is a good complement to verification since users could have an intuitive observation and it makes debugging more convenient.

3.2 Verification

Compared with simulation, automatic verification plays a more important role in system analysis since it indicates the accurate result of whether a property is satisfied in a system. Two aspects are quite significant in verification with a model checker. One is the

properties it can support, and the other is the efficiency of the verification algorithms. In the following, we review several widely used properties in PRTS, and some techniques in the verification algorithm to speed up the model checking procedure.

Properties Supported PRTS supports multiple kinds of useful properties in system design since they are focusing on different aspects of the system. Because MDP has non-deterministic choices and (infinite) many schedulers, we consider the maximum and minimum probability of a specified property and mainly follow the algorithms in [3].

Reachability Checking The maximum/minimum probability of reaching specific target states could be checked using numerical iterative method.

Reward Checking The maximum/minimum accumulated rewards/costs to reach the target states could be calculated also through the iterative method. In PRTS we just consider the action reward, that is, assigning each visible action a reward which is a rational number.

LTL Checking In PRTS we support LTL-X (LTL without 'next' operator) since in abstract model the semantics of 'next' is hard to define. In our setting, LTL formula can be built from not only atomic state propositions but also events so that it is called SE-LTL [5]. It is very expressive and suitable for PRTS since our language is both event-based and state-based. We adopt the Rabin automata-based approach to calculate the maximum/minimum probability that an SE-LTL is satisfied.

Refinement Checking A desired property could be defined as a non-probability model and we can check a trace refinement relation between this model and the system specification [17].

Efficient Verification Techniques In our implementation, after zone abstraction we adopt mainly two techniques to enhance the efficiency of verification.

Counter Abstraction For some protocols having similar behaviors, we can group those processes together using counter abstraction [12, 15]. Its extension to probabilistic system is still valid, whose proof is similar to work [7]. This approach reduces the state space without affecting the probability of specific properties which are irrelevant with processes identifiers.

Safety Checking via Refinement Checking LTL formulas can be categorized into either safety or liveness [2]. In [17], we have proven that safety property can be verified via refinement checking. Given an SE-LTL property, our tool supports automatic safety detection. The experiment results show that sometimes it reduces verification time significantly compared with Rabin automata approach [17].

4 Implementation and Experiments

PRTS has been integrated into PAT, which is implemented with C# and can run on all widely-used operating systems. To make our tool more practical, we have developed a Visual Studio 2010 plug-in (available at [1]) to edit, simulate and verify PRTS models inside Visual Studio. Next, we demonstrate some experiments[6] to show the efficiency

[6] Due to space constraint, detailed information of the models and properties can be found in [1].

System	Random			Nearest		
	Result(pmax)	States	Time(s)	Result(pmax)	States	Time(s)
lift=2; floor=2; user=2	0.21875	20120	1.47	0.13889	12070	1.33
lift=2; floor=2; user=3	0.47656	173729	15.04	0.34722	83026	6.23
lift=2; floor=2; user=4	0.6792	777923	90.66	0.53781	308602	28.31
lift=2; floor=2; user=5	0.81372	2175271	406.29	0.68403	740997	85.29
lift=2; floor=3; user=2	0.2551	72458	5.13	0.18	38593	2.89
lift=2; floor=3; user=3	0.54009	1172800	150.20	0.427	500897	48.05
lift=2; floor=4; user=2	0.27	170808	13.06	0.19898	86442	6.11
lift=3; floor=2; user=2	0.22917	562309	86.88	0.10938	266621	34.25

Table 1. Experiments: Lift System

of our tool; the testbed is a PC running Windows XP with Intel P8700 CPU@2.53GHz and 2GB memory.

First, we use a multi-lift system to demonstrate the effectiveness of PRTS. Such system contains different components, e.g. lifts and buttons; it usually has timing requirements in service and users may have random behaviors. An interesting phenomena in such system is that a user presses the button outside the lifts, but one lift on the same direction passes by without serving him/her. This is possible since the lift which is assigned to serve this user is occupied by other users for a long time, and other lifts reach that user's floor first and pass by.

The experiments results are listed in Table 1. We analyze two kinds of task assignment mechanisms: assigning to nearest lift and assigning to a random lift. From the table we could conclude that the first mechanism is better, since it has a smaller probability to ignore users' requests and this is consistent with common sense.

Next, we compare our model checker with PRISM on verifying benchmark systems of probabilistic real-time system. Here we use two PTA models described in [8]. One is the $\mathit{firewire\ abstraction}$ (FA) for IEEE 1394 FireWire root contention protocol and the other is $\mathit{zeroconf}$ (ZC) for Zeroconf network configuration protocol. We build PTA models using PRISM and PRTS models using PAT, and verify the desired reachability properties to check the efficiency of these two tools. Here we choose PRISM's default verification technique: stochastic games since it usually has the best performance [8].

The results are listed in Table 2. '-' means that experiment takes more than 1 hour. The parameter of each model is the deadline constrain; for PRISM, $\mathit{Iteration}$ means how many refinements the $\mathit{stochastic\ game}$ approach executes to get the precise result. In these cases we notice PRTS is much faster than PRISM's PTA since our approach just uses zone abstraction and theirs must have additional refinement procedure.

5 Conclusion

In this work, we proposed a model checker for hierarchical probabilistic real-time systems. Its effectiveness and efficiency are demonstrated through several case studies. As for future work, we are exploring more aspects of probabilistic real-time system such as zeno-check and digitization, and various properties such as real-time property.

System	Result	PAT		PRISM		
		States	Time(s)	States	Iterations	Time(s)
FA(10K)	0.94727	1352	0.15	1065	19	1.98
FA(20K)	0.99849	5030	0.13	8663	34	65.08
FA(30K)	0.99994	11023	0.45	34233	45	575.03
FA(300K)	>0.99999	726407	30.74	-	-	-
ZC(100)	0.49934	404	0.15	135	0	0.28
ZC(300)	0.01291	4813	0.65	2129	26	2.73
ZC(500)	0.00027	12840	2.39	10484	44	63.19
ZC(700)	1E-5	24058	5.78	31717	60	427.70

Table 2. Compared with PRISM

References

1. PRTS Model Checker. `http://www.comp.nus.edu.sg/~pat/cav12prts`.
2. B. Alpern and F. B. Schneider. Recognizing Safety and Liveness. *Distributed Computing*, 2(3):117–126, 1987.
3. C. Baier and J. Katoen. *Principles of Model Checking*. The MIT Press, 2008.
4. G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, Wang Yi, and M. Hendriks. UPPAAL 4.0. In *QEST*, pages 125–126. IEEE, 2006.
5. S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/Event-Based Software Model Checking. In *IFM*, volume 2999 of *LNCS*, pages 128–147. Springer, 2004.
6. A. Hartmanns and H. Hermanns. A modest approach to checking probabilistic timed automata. In *QEST*, pages 187–196, September 2009.
7. M. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *CAV'06*, volume 4114 of *LNCS*, pages 234–248. Springer, 2006.
8. M. Kwiatkowska, G. Norman, and D. Parker. Stochastic games for verification of probabilistic timed automata. In *FORMATS*, volume 5813 of *LNCS*, pages 212–227, 2009.
9. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806, pages 585–591, 2011.
10. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic Verification of Real-time Systems with Discrete Probability Distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.
11. Y. Liu, J. Pang, J. Sun, and J. Zhao. Verification of population ring protocols in pat. In *TASE*, pages 81–89. IEEE Computer Society, 2009.
12. A. Pnueli, J. Xu, and L. D. Zuck. Liveness with (0, 1, infty)-counter abstraction. In *CAV*, pages 107–122, 2002.
13. S. Schneider. *Concurrent and Real-time Systems*. John Wiley and Sons, 2000.
14. J. Sun, Y. Liu, J. S. Dong, and J. Pang. PAT: Towards Flexible Verification under Fairness. In *CAV*, volume 5643 of *LNCS*, pages 709–714. Springer, 2009.
15. J. Sun, Y. Liu, A. Roychoudhury, S. Liu, and J. S. Dong. Fair model checking with process counter abstraction. In *FM*, pages 123–139. Springer, 2009.
16. J. Sun, Y. Liu, S. Song, J. S. Dong, and X. Li. Prts: An approach for model checking probabilistic real-time hierarchical systems. In *ICFEM*, pages 147–162, 2011.
17. J. Sun, S. Z. Song, and Y. Liu. Model Checking Hierarchical Probabilistic Systems. In *ICFEM*, volume 6447 of *LNCS*, pages 388–403. Springer, 2010.
18. J. Sun, S. Z. Song, Y. Liu, and J. S. Dong. PRTS: Specification and Model Checking. Technical report, 2011. `http://www.comp.nus.edu.sg/~pat/preport.pdf`.