PRTS: An Approach for Model Checking Probabilistic Real-time Hierarchical Systems

Jun Sun[1], Yang Liu[2], ***Songzheng Song***[2], Jin Song Dong[2], Xiaohong Li[3]

[1]Singapore University of Technology and Design
[2]National University of Singapore
[3]Tianjin University

October 28, 2011

Outline

1. Motivation
2. Language Syntax of PRTS
3. Operational Semantics
   - Concrete Configurations
   - Abstraction
4. Verification
5. Evaluation
6. Conclusion

Outline

1. Motivation
2. Language Syntax of PRTS
3. Operational Semantics
   - Concrete Configurations
   - Abstraction
4. Verification
5. Evaluation
6. Conclusion

Motivation

Model checking real-life systems is usually difficult.

Motivation

Model checking real-life systems is usually difficult.

- quantitative timing factors
- unreliable/random environment
- complex data operations
- hierarchical control flows

Multi-lift System

- Serving time/responding time
- Random user behaviors
- Task assign algorithm
- lifts/users/buttons...

Multi-lift System

Scenario : one user presses the lift button, but one lift traveling on the same direction passes by without serving him/her!

Our Approach

1. Design an expressive modeling language supporting features like real-time, hierarchy, concurrency, data structures as well as probability.
2. Build a model checker for this language in order to analyze the behavior of such systems.
3. Verify widely used properties such as reachability checking and LTL checking.

Our Approach

1. Design an expressive modeling language supporting features like real-time, hierarchy, concurrency, data structures as well as probability.
2. Build a model checker for this language in order to analyze the behavior of such systems.
3. Verify widely used properties such as reachability checking and LTL checking.

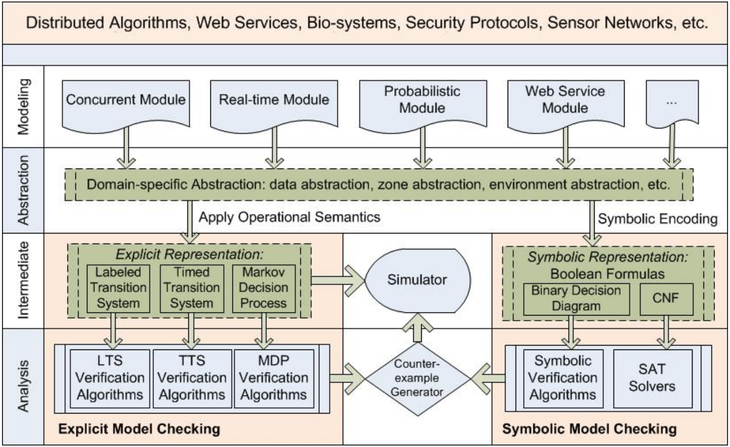
We propose **PRTS** for probabilistic real-time systems and it has been integrated into our framework **PAT**.

`http://www.patroot.com`

PAT

- 1400+ registered users;
- 40+ countries and regions;
- 300+ organizations;
- 10+ modules. PRTS is one of them.

PAT


Distributed Algorithms, Web Services, Bio-systems, Security Protocols, Sensor Networks, etc.
Modeling
Abstraction
Intermediate
Analysis
Concurrent Module
Real-time Module
Probabilistic Module
Web Service Module
...
Domain-specific Abstraction: data abstraction, zone abstraction, environment abstraction, etc.
Apply Operational Semantics
Symbolic Encoding
Explicit Representation:
Labeled Transition System
Timed Transition System
Markov Decision Process
Simulator
Symbolic Representation: Boolean Formulas
Binary Decision Diagram
CNF
LTS Verification Algorithms
TTS Verification Algorithms
MDP Verification Algorithms
Counter-example Generator
Symbolic Verification Algorithms
SAT Solvers
Explicit Model Checking
Symbolic Model Checking

Outline

1 Motivation

2 Language Syntax of PRTS

3 Operational Semantics
- Concrete Configurations
- Abstraction

4 Verification

5 Evaluation

6 Conclusion

Based on C. A. R. Hoare's CSP

$P = Stop$	– in-action
$\mid Skip$	– termination
$\mid e \rightarrow P$	– event prefixing
$\mid a\{program\} \rightarrow P$	– data operation prefixing
$\mid [b]P$	– guard condition
$\mid \textbf{if}\ (b)\ \{P\}\ \textbf{else}\ \{Q\}$	– conditional choice
$\mid P \Box Q$	– external choice
$\mid P \sqcap Q$	– internal choice
$\mid P \setminus X$	– hiding
$\mid P;\ Q$	– sequential composition
$\mid P \parallel Q$	– parallel composition
$\mid Q$	– process referencing

Based on C. A. R. Hoare's CSP

$$
\begin{array}{lll}
P = & Wait[d] & \text{– delay} \\
\mid & P\ timeout[d]\ Q & \text{– timeout} \\
\mid & P\ interrupt[d]\ Q & \text{– timed interrupt} \\
\mid & P\ within[d] & \text{– timed responsiveness} \\
\mid & P\ deadline[d] & \text{– deadline}
\end{array}
$$

d is a non-negative integer.

Based on C. A. R. Hoare's CSP

$$
\begin{array}{rll}
P = & Wait[d] & \text{– delay} \\
\mid & P\ timeout[d]\ Q & \text{– timeout} \\
\mid & P\ interrupt[d]\ Q & \text{– timed interrupt} \\
\mid & P\ within[d] & \text{– timed responsiveness} \\
\mid & P\ deadline[d] & \text{– deadline}
\end{array}
$$

d is a non-negative integer.

$$P = pcase\{pr_0 : P_0;\ pr_1 : P_1;\ \cdots;\ pr_k : P_k\}$$

pr_i is defined as a positive integer. It means with probability $\frac{pr_i}{pr_0+pr_1+\cdots+pr_k}$, P behaves as P_i.

Multi-lift System

```
#define NoOfFloors 2;
#define NoOfLifts 2;
#import "PAT.Lib.Lift";
var<LiftControl> ctrl = new LiftControl(NoOfFloors,NoOfLifts);
Users() = pcase {
        1 : extreq.0.1{ctrl.Assign_External_Up_Request(0)} -> Skip
        1 : intreq.0.0.1{ctrl.Add_Internal_Request(0,0)} -> Skip
        1 : intreq.1.0.1{ctrl.Add_Internal_Request(1,0)} -> Skip
        1 : extreq.1.0{ctrl.Assign_External_Down_Request(1)} -> Skip
        1 : intreq.0.1.1{ctrl.Add_Internal_Request(0,1)} -> Skip
        1 : intreq.1.1.1{ctrl.Add_Internal_Request(1,1)} -> Skip
    } within[1]; Users();
Lift(i, level, direction) = ...;
System = (||| x:{0..NoOfLifts-1} @ Lift(x, 0, 1)) ||| Users();
```

Multi-lift System

```
#define NoOfFloors 2;
#define NoOfLifts 2;
#import "PAT.Lib.Lift";
var<LiftControl> ctrl = new LiftControl(NoOfFloors,NoOfLifts);
Users() = pcase {
      1 : extreq.0.1{ctrl.Assign_External_Up_Request(0)} -> Skip
      1 : intreq.0.0.1{ctrl.Add_Internal_Request(0,0)} -> Skip
      1 : intreq.1.0.1{ctrl.Add_Internal_Request(1,0)} -> Skip
      1 : extreq.1.0{ctrl.Assign_External_Down_Request(1)} -> Skip
      1 : intreq.0.1.1{ctrl.Add_Internal_Request(0,1)} -> Skip
      1 : intreq.1.1.1{ctrl.Add_Internal_Request(1,1)} -> Skip
    } within[1]; Users();
Lift(i, level, direction) = ...;
System = (||| x:{0..NoOfLifts-1} @ Lift(x, 0, 1)) ||| Users();
```

Property: what is the probability that a lift passes by?

Outline

1 Motivation

2 Language Syntax of PRTS

3 Operational Semantics
- Concrete Configurations
- Abstraction

4 Verification

5 Evaluation

6 Conclusion

Definition (Markov Decision Process)

An MDP is a tuple $\mathcal{D} = (S, init, Act, Pr)$ where

- S is a set of states;
- $init \in S$ is the initial state;
- Act is a set of actions and Act_τ is $Act \cup \tau$;
- $Pr : S \times (Act_\tau \cup \mathbb{R}_+) \times Distr(S)$ is a transition relation.

Definition (Markov Decision Process)

An MDP is a tuple $\mathcal{D} = (S, init, Act, Pr)$ where

- S is a set of states;
- $init \in S$ is the initial state;
- Act is a set of actions and Act_τ is $Act \cup \tau$;
- $Pr : S \times (Act_\tau \cup \mathbb{R}_+) \times Distr(S)$ is a transition relation.

A Markov Chain can be defined given an MDP $\mathcal{D}$ and a scheduler δ, which is denoted as $\mathcal{D}^\delta$.

Definition (Markov Decision Process)

An MDP is a tuple $\mathcal{D} = (S, init, Act, Pr)$ where

- S is a set of states;
- $init \in S$ is the initial state;
- Act is a set of actions and Act_τ is $Act \cup \tau$;
- $Pr : S \times (Act_\tau \cup \mathbb{R}_+) \times Distr(S)$ is a transition relation.

A Markov Chain can be defined given an MDP $\mathcal{D}$ and a scheduler δ, which is denoted as $\mathcal{D}^\delta$.

A *path* of $\mathcal{D}^\delta$ is defined as $\omega = s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} s_2 \xrightarrow{x_2} \ldots$

Given a property ϕ:

$$\mathcal{P}_{\mathcal{D}}^{max}(\phi) = \text{sup}_{\delta}\, \mathcal{P}_{\mathcal{D}}(\{\pi \in paths(\mathcal{D}^{\delta}) \mid \pi \text{ satisfies } \phi\})$$

$$\mathcal{P}_{\mathcal{D}}^{min}(\phi) = \text{inf}_{\delta}\, \mathcal{P}_{\mathcal{D}}(\{\pi \in paths(\mathcal{D}^{\delta}) \mid \pi \text{ satisfies } \phi\})$$

Definition (Concrete System Configuration)

A concrete system configuration is a tuple $s = (\sigma, P)$ where σ is a variable valuation and P is a process.

Definition (Concrete System Configuration)

A concrete system configuration is a tuple $s = (\sigma, P)$ where σ is a variable valuation and P is a process.

The probabilistic transition relation of a model's MDP semantics is defined by a set of firing rules with every process construct.

- $Wait[d]$
- *pcase*

$Wait[d]$

$$\frac{\epsilon \leq d}{(\sigma, Wait[d]) \xrightarrow{\epsilon} (\sigma, Wait[d - \epsilon])} \; [\; wait_1 \;]$$

$$\frac{}{(\sigma, Wait[0]) \xrightarrow{\tau} (\sigma, Skip)} \; [\; wait_2 \;]$$

pcase

$$\frac{}{(\sigma, pcase\ \{pr_0 : P_0;\ pr_1 : P_1;\ \cdots\ ;\ pr_k : P_k\}) \xrightarrow{\tau} \mu}\ [\ pcase\ ]$$

$$\mu((\sigma, P_i)) = \frac{pr_i}{pr_0 + pr_1 + \cdots + pr_k} \text{ for all } i \in [0,\ k]$$

pcase

$$\frac{}{(\sigma, pcase\ \{pr_0 : P_0;\ pr_1 : P_1;\ \cdots;\ pr_k : P_k\}) \xrightarrow{\tau} \mu}\ [\ pcase\ ]$$

$$\mu((\sigma, P_i)) = \frac{pr_i}{pr_0 + pr_1 + \cdots + pr_k} \text{ for all } i \in [0,\ k]$$

pcase transitions are not time-consuming!

Using the firing rules, the transition relation of the MDP could be defined. However, since PRTS model has a dense-time semantics, the underlying MDP has infinite states.

Using the firing rules, the transition relation of the MDP could be defined. However, since PRTS model has a dense-time semantics, the underlying MDP has infinite states.

$$(\sigma, \ \mathit{Wait}[1]) \stackrel{0.1}{\rightarrow} (\sigma, \ \mathit{Wait}[0.9]) \stackrel{0.01}{\rightarrow} (\sigma, \ \mathit{Wait}[0.89]) \stackrel{0.001}{\rightarrow} \ldots$$

Using the firing rules, the transition relation of the MDP could be defined. However, since PRTS model has a dense-time semantics, the underlying MDP has infinite states.

$$(\sigma,\ \mathit{Wait}[1]) \stackrel{0.1}{\to} (\sigma,\ \mathit{Wait}[0.9]) \stackrel{0.01}{\to} (\sigma,\ \mathit{Wait}[0.89]) \stackrel{0.001}{\to} \ldots$$

Abstraction is required!

Dynamic Zone Abstraction

The first step of abstraction is to associate timed process constructs with implicit **clocks**.

- $P\ timeout[d]\ Q \rightarrow P\ timeout[d]_c\ Q$
- Constraint over clock : $c \leq 5$ represents any process $P\ timeout[d']\ Q$ with $d' \leq 5$

Dynamic Zone Abstraction

The first step of abstraction is to associate timed process constructs with implicit **clocks**.

- $P\ timeout[d]\ Q \rightarrow P\ timeout[d]_c\ Q$
- Constraint over clock : $c \leq 5$ represents any process $P\ timeout[d']\ Q$ with $d' \leq 5$

A **zone** D is the conjunction of multiple primitive constraints over a set of clocks.

- $c \sim d$ or $c_i - c_j \sim d$ where c, c_i, c_j are values of clocks and d is a constant integer. $\sim$ represents $\geq, \leq, =$

Abstract Configurations

Definition (Abstract System Configuration)

Given a concrete system configuration (σ, P), the corresponding abstract system configuration is a triple (σ, P_T, D) such that P_T is a process obtained by associating P with a set of clocks; and D is a zone over the clocks.

Abstract Configurations

Definition (Abstract System Configuration)

Given a concrete system configuration (σ, P), the corresponding abstract system configuration is a triple (σ, P_T, D) such that P_T is a process obtained by associating P with a set of clocks; and D is a zone over the clocks.

Abstract firing rules are defined in order to get the abstract MDP. $Wait[d]$ and *pcase* are listed as examples.

Wait[*d*]

$$\frac{}{(\sigma, \mathit{Wait}[d]_c, D) \overset{\tau}{\rightsquigarrow} (\sigma, \mathit{Skip}, D^{\uparrow} \wedge c = d)} \; [\; \mathit{await} \;]$$

- $D^{\uparrow}$ denotes the zone obtained by delaying arbitrary amount of time. e.g. $(c \leq 5)^{\uparrow}$ is $c \leq \infty$.

pcase

$$\frac{}{(\sigma, pcase\ \{pr_0 : P_0;\ pr_1 : P_1;\ \cdots;\ pr_k : P_k\}, D) \overset{\tau}{\rightsquigarrow} \mu}\ [\ apcase\ ]$$

$\mu((\sigma, P_i, D)) = \frac{pr_i}{pr_0 + pr_1 + \cdots + pr_k}$ for $i \in [0, k]$; **zone is unchanged.**

Outline

1. Motivation
2. Language Syntax of PRTS
3. Operational Semantics
   - Concrete Configurations
   - Abstraction
4. Verification
5. Evaluation
6. Conclusion

verification

1. abstract model is suitable for standard probabilistic model checking techniques.
2. abstract model preserves the verification result of given property ϕ in concrete model.

- M : PRTS model;
- $\mathcal{D}_M$: concrete MDP of M;
- $\mathcal{D}_M^a$: abstract MDP of M.

Theorem 1

Theorem

$\mathcal{D}^a_M$ is finite for any model M. □

Theorem 1

Theorem

$\mathcal{D}^a_M$ is finite for any model M. □

1. Variable valuations are finite[**by assumption**].
2. Process expressions are finite[**by assumption and clock reuse**].
3. Zones are finite.

J. Bengtsson and Y. Wang.
Timed Automata: Semantics, Algorithms and Tools.
In *Lectures on Concurrency and Petri Nets*, pages 87-124, 2003.

Theorem 2

Theorem

$\mathcal{P}^{max}_{\mathcal{D}^a_M}(\phi) = \mathcal{P}^{max}_{\mathcal{D}_M}(\phi)$ *and* $\mathcal{P}^{min}_{\mathcal{D}^a_M}(\phi) = \mathcal{P}^{min}_{\mathcal{D}_M}(\phi)$. □

Theorem 2

Theorem

$\mathcal{P}^{max}_{\mathcal{D}^a_M}(\phi) = \mathcal{P}^{max}_{\mathcal{D}_M}(\phi)$ and $\mathcal{P}^{min}_{\mathcal{D}^a_M}(\phi) = \mathcal{P}^{min}_{\mathcal{D}_M}(\phi)$. □

1. For any scheduler δ in $\mathcal{D}^a_M$, there is a scheduler ξ in $\mathcal{D}_M$ such that $(\mathcal{D}^a_M)^\delta$ and $(\mathcal{D}_M)^\xi$ are equivalent Markov Chains.
2. For any scheduler η in $\mathcal{D}_M$, there is a scheduler ϑ in $\mathcal{D}^a_M$ such that $(\mathcal{D}_M)^\eta$ and $(\mathcal{D}^a_M)^\vartheta$ might not be equivalent but they still have the same result of verifying ϕ.

Theorem 2

Theorem

$\mathcal{P}^{max}_{\mathcal{D}^a_M}(\phi) = \mathcal{P}^{max}_{\mathcal{D}_M}(\phi)$ *and* $\mathcal{P}^{min}_{\mathcal{D}^a_M}(\phi) = \mathcal{P}^{min}_{\mathcal{D}_M}(\phi)$. □

1. For any scheduler δ in $\mathcal{D}^a_M$, there is a scheduler ξ in $\mathcal{D}_M$ such that $(\mathcal{D}^a_M)^\delta$ and $(\mathcal{D}_M)^\xi$ are equivalent Markov Chains.
2. For any scheduler η in $\mathcal{D}_M$, there is a scheduler ϑ in $\mathcal{D}^a_M$ such that $(\mathcal{D}_M)^\eta$ and $(\mathcal{D}^a_M)^\vartheta$ might not be equivalent but they still have the same result of verifying ϕ.

pcase transitions are not time-consuming!

After abstraction, we obtain the finite states system and standard probabilistic model checking are applied to solve the linear program in MDP.

C. Baier and J. Katoen.
Principles of Model checking.
The MIT Press, 2008.

Outline

1. Motivation
2. Language Syntax of PRTS
3. Operational Semantics
   - Concrete Configurations
   - Abstraction
4. Verification
5. Evaluation
6. Conclusion

Multi-lift System

Property: one user presses the lift button, but one lift traveling on the same direction passes by without serving him/her.

System	Random			Nearest		
	Result(pmax)	Time(s)	States	Result(pmax)	Time(s)	States
lift=2; floor=2; user=2	0.21875	3.262	10398	0.13889	2.385	4299
lift=2; floor=2; user=3	0.47656	38.785	127384	0.34722	18.061	59267
lift=2; floor=2; user=4	0.6792	224.708	737447	0.53781	78.484	276709
lift=2; floor=2; user=5	0.81372	945.853	3941883	0.68403	223.036	743973
lift=2; floor=3; user=2	0.2551	12.172	37263	0.18	6.757	19726
lift=2; floor=3; user=3	0.54009	364.588	831334	0.427	119.810	339630
lift=2; floor=3; user=4	0.74396	11479.966	17022359	0.6335	1956.041	6428704
lift=2; floor=4; user=2	0.27	27.888	87676	0.19898	13.693	44316
lift=3; floor=2; user=2	0.22917	208.481	262588	0.10938	88.549	122604
lift=3; floor=2; user=3	OOM	-	-	0.27344	3093.969	6708763

OOM means Out Of Memory.

Outline

1. Motivation
2. Language Syntax of PRTS
3. Operational Semantics
   - Concrete Configurations
   - Abstraction
4. Verification
5. Evaluation
6. Conclusion

Conclusion

1. Modeling language PRTS is proposed for hierarchical probabilistic real-time systems.
2. Zone abstraction is used in order to apply probabilistic model checking techniques.
3. Model checker PAT is extended to support PRTS.

References I

C. A. R. Hoare.
Communicating Sequential Processes.
Prentice-Hall, 1985.

R. Alur, C. Courcoubetis, and D. L. Dill.
Model-checking for Probabilistic Real-time Systems.
ICALP, pages 115-126, 1991.

A. Hinton, M. Kwiatkowska, G. Norman and D. Parker.
PRISM: A Tool for Automatic Verification of Probabilistic Systems.
TACAS, pages 441-444, 2006.

References II

C. Baier and J. Katoen.
Principles of Model checking.
The MIT Press, 2008.

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.

S. Schnerder.
Concurrent and Real-time Systems.
John Wiley and Sons, 2000.

References III

J. Ouaknine and J. Worrell.
Timed CSP = Closed Timed Safety Automata.
Electrical Notes Theoretical Computer Sciences, 68(2), 2002.

J. Sun, Y. Liu, J. S. Dong and X. Zhang.
Verifying Stateful Timed CSP Using Implicit Clocks and Zone Abstraction.
ICFEM, pages 581-600, 2009.

J. Sun, S. Song and Y. Liu
Model Checking Hierarchical Probabilistic Systems.
ICFEM, pages 388-403, 2010.

THANK YOU!


CLAP!
CLAP!
CLAP!
CLAP!
CLAP!
CLAP!
CLAP!
CLAP!
CLAP!
CLAP!
CLAP!
CLAP!
CLAP!
CLAP!
CLAP!