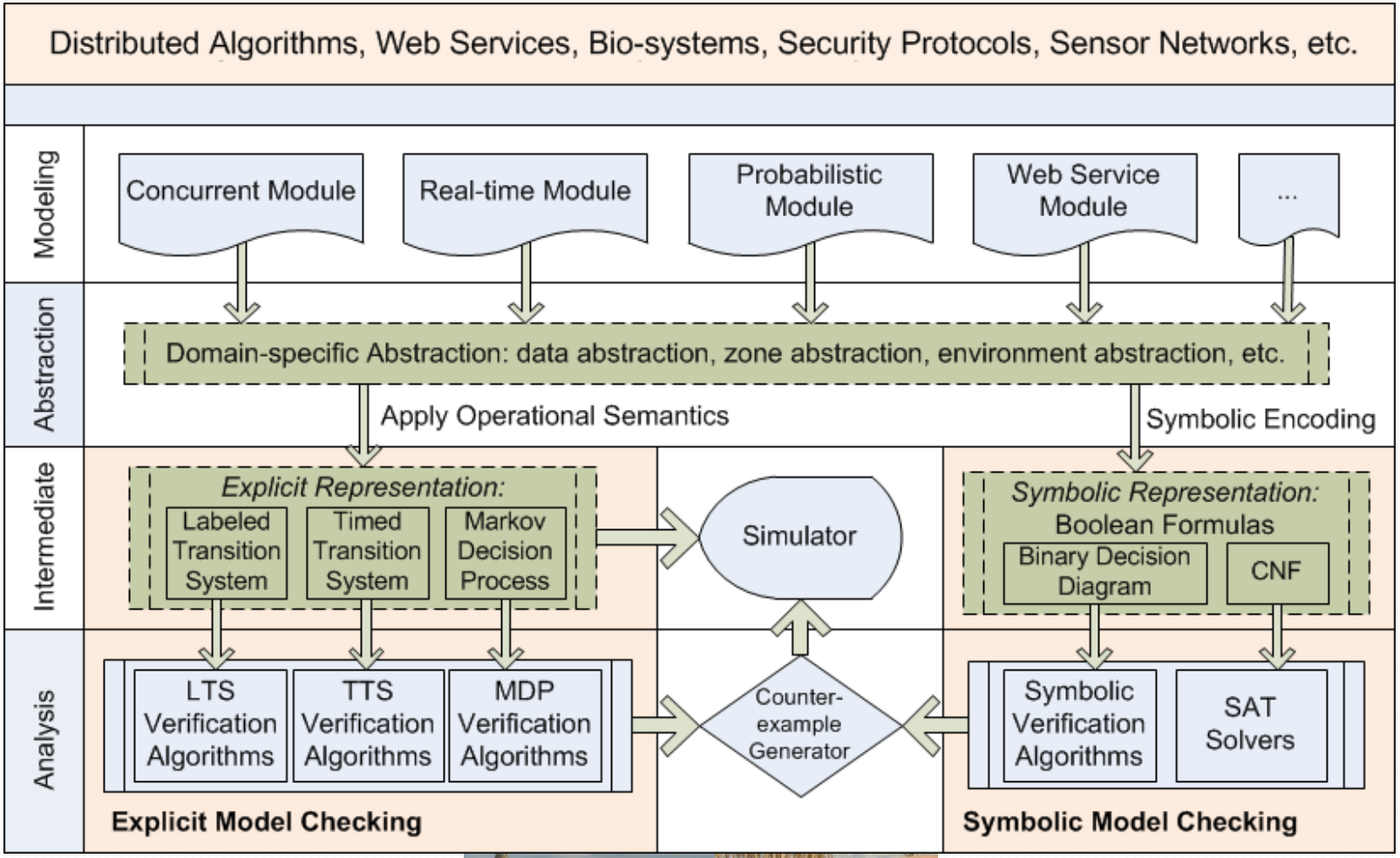
Pervasive Model Checking

Jin Song Dong
National University of Singapore
(joint work with two formal PhD students: Jun Sun and Yang Liu and
11 other current PhD students and 3 postdocs)

Overview

- Model checking has made excellent progress in recent years, i.e., Microsoft SLAM project and *Intel i7 processor*
  - At CAV 2009, Intel reported that Intel i7 CPU is verified using model checking without a single test case!
- There are a number of model checkers like SPIN, SMV and FDR which are designed for specialized domains and are therefore based on restrictive modeling languages.
- PAT is a self-contained, extensible and modularized multi-domain model checking systems for composing, simulating and reasoning of concurrent, real-time, probabilistic systems and other possible domains (i.e. distributed algorithms, security protocols, web services, sensor networks, etc).

PAT System Design (ICSE'08'12, CAV'09'12, FM'11'12, TOSEM'12)



Two formal PhD students:

11 current PhD students
3 postdocs

PAT Languages features

- Global variables: Boolean, Integer, Multi-dimensional arrays, etc.
- Data/State operations: a sequential program in PAT's language or a C# external method.
- Event Control flow: CSP process constructs (choice, parallel, interrupt, etc.) + timed patterns (delay, timeout, timed interrupt, deadline, etc.) + probabilistic choices …
- Assertions: reachability, refinement relationship (trace, timed trace, failures, failures/divergence), state/event LTL, min/max probability.

PAT Vision: ***Pervasive Model Checking***

- *Model Checking as Planning/Problem-Solving/Scheduling/Services*

- *Wide application domains, including Real-Time and Probabilistic systems.*

Model checking as planning/problem-solving

```
//@@Sliding Game@@
//The following models the sliding game with the extra 'costs' complexity

var board[9]:{0..8} = [3,5,6,  // 0,1,2 :index
                       0,2,7,  // 3 4 5 :index
                       8,4,1]; // 6,7,8 :index
hvar empty:{0..8} = 3; //empty position is a secondary variable, no need to put it in the state space

var c = 0; // cost utility, e.g. costs 1 for left and right move, 2 for up, 0 for down

Game() = Left() [] Right() [] Up() [] Down();

Left() = [empty!=2 && empty!=5 && empty!=8] left                    // c=c+1
         {board[empty]=board[empty+1]; board[empty+1]=0; empty=empty+1; c++} -> Game();

Right() = [empty!=0 && empty!=3 && empty!=6] right
          {board[empty]=board[empty-1]; board[empty-1]=0; empty=empty-1; c++} -> Game();

Up() = [empty!=6&&empty!=7&&empty!=8] up
       {board[empty]=board[empty+3]; board[empty+3]=0; empty=empty+3; c=c+2} -> Game();

Down() = [empty!=0&&empty!=1&&empty!=2] down
         {board[empty]=board[empty-3]; board[empty-3]=0; empty=empty-3} -> Game();


#define goal board[0] == 1 && board[1] == 2 && board[2] == 3 &&
             board[3] == 4 && board[4] == 5 && board[5] == 6 &&
             board[6] == 7 && board[7] == 8 && board[8] == 0;

#assert Game() reaches goal with min(c);
```

3	5	6
	2	7
8	4	1

1	2	3
4	5	6
7	8	

The sliding game problem cont'd

8	7	6
0	4	1
2	5	3

(a) Hard1

8	0	6
5	4	7
2	3	1

(b) Hard2

8	5	6
7	2	3
4	1	0

(c) Most1

8	5	4
7	6	3
2	1	0

(d) Most2

8	2	1
3	6	4
0	5	7

(e) Rand1

4	1	7
8	0	3
5	6	2

(f) Rand2

Figure: Initial configurations of *the sliding game problem* instances

Experimental Results

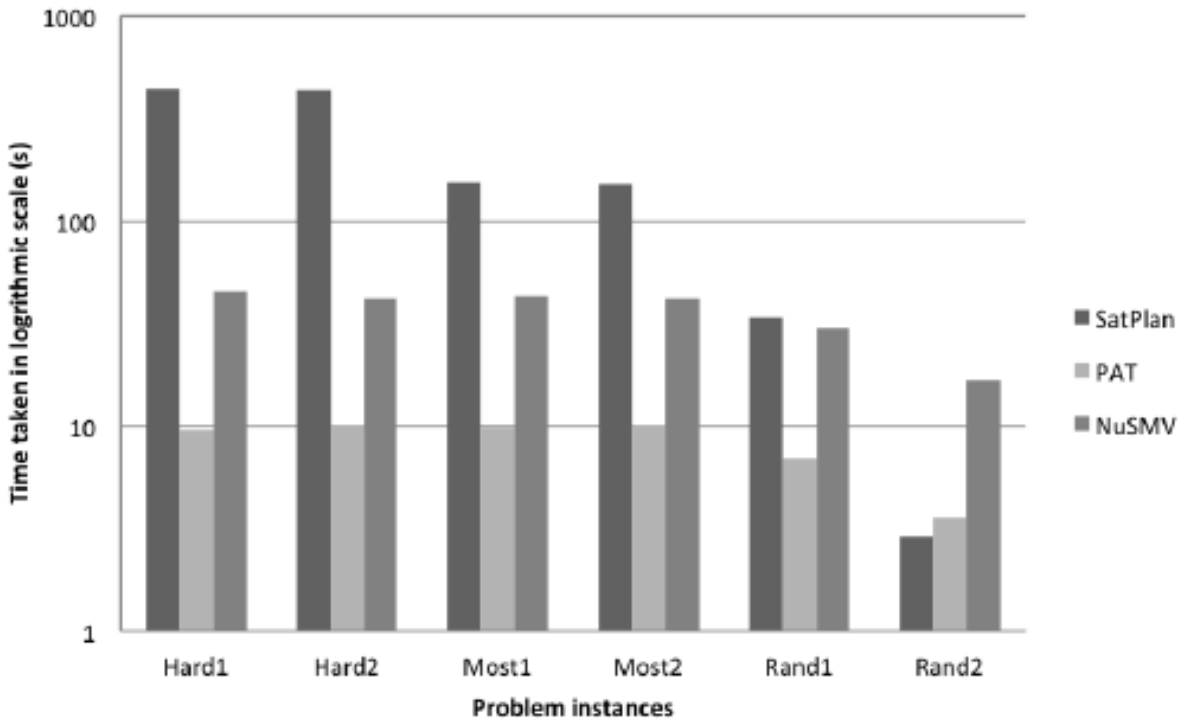


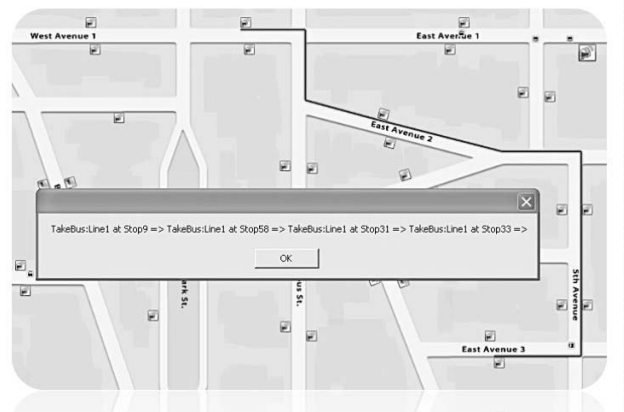
Figure: Execution time comparison of PAT, NuSMV and SatPlan on *the sliding game problem*, shown on a logarithm scale

Model Checking as Planning/Scheduling/Service:

Transport4You, an intelligent public transportation manager

ICSE 2011 SCORE Competition Project (PAT won FM Award)

- PAT model checker is used not only as a verification tool for the system design but also as a service that computes an optimal travel plan.



- 94 teams from 48 universities in 22 countries started the competition; 55 finished and made final submission; 18 teams were selected for the second round; 5 finalist teams invited to Hawaii with 2000USD travel award for each team. Two winners (Formal Methods Award and Overall Award) were selected during the conference.

PAT student team won Formal Method Award

Model Checking Timed Systems

- A language for modeling compositional real-time systems using implicit clocks.
  - Concurrency + Hierarchy + Data
  - Real-time constructs: wait, within, deadline, timeout …

- A method for abstracting and verifying the models.
  - Zone abstraction
  - Reachability checking, LTL, trace refinement checking and timed refinement checking.

Real-Time Concurrent Processes

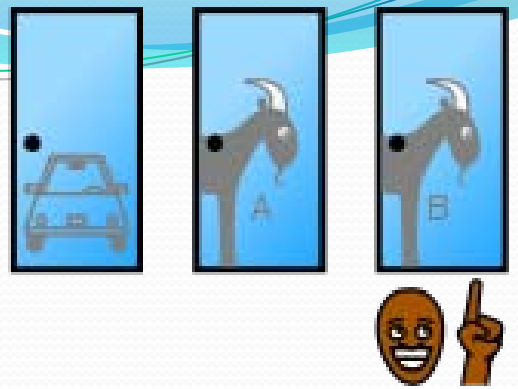
This mutual exclusion protocol is proposed by Fischer in 1985. Mutual exclusion in Fischer's Protocol is guaranteed by carefully placing bounds on the execution times of the instructions, leading to a protocol which is very simple, and relies heavily on time aspects.

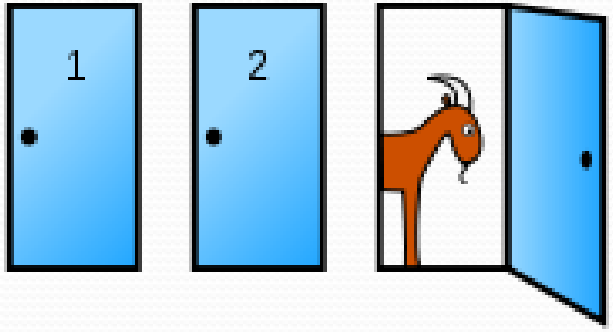
```
#define N 4;
#define Delta 3;
#define Epsilon 4;
#define Idle -1;

var x = Idle;
var counter;

//timed version
P(i) = ifb(x == Idle) {
        ((update.i{x = i} -> Wait[Epsilon]) within[Delta]);
            if (x == i) {
                cs.i{counter++} -> exit.i{counter--; x=Idle} -> P(i)
            } else {
                P(i)
            }
        };

FischersProtocol = ||| i:{0..N-1}@P(i);

//verifying mutual exclusion by reachability analysis
#define MutualExclusionFail counter > 1;
#assert FischersProtocol reaches MutualExclusionFail;
```

Probabilistic Model Checking

- Syntax
  - Hierarchical concurrent systems with probabilistic choices
- Semantics
  - Markov decision processes
- Given a property, probabilistic model checking returns, instead of true or false
  - the maximum and minimum probability of satisfying the property.

Monty Hall Problem



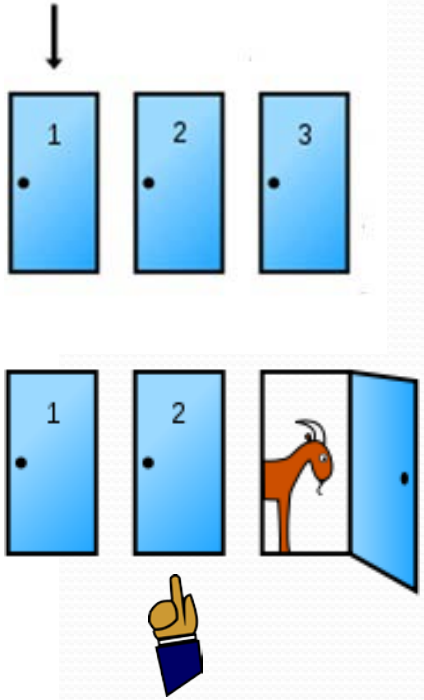


The **Monty Hall problem** is based on the American television game show *Let's Make a Deal* and named after the show's original host, Monty Hall.
The problem was originally posed in a letter by Steve Selvin to the *American Statistician* in 1975.

- In search of a new car, the player picks a door, say 1. The game host then opens one of the other doors, say 3, to reveal a goat and offers to let the player pick door 2 instead of door 1. Should the player take the offer?
- What if the host is dishonest, e.g., place car after 1st guess or host do a switch 33% time after the guess?

PAT Model

```
enum{Door1, Door2, Door3};

var car = -1;
var guess = -1;
var goat = -1;
var final = false;

#define goal guess == car && final;

PlaceCar = []i:{Door1,Door2,Door3}@ placecar.i{car=i} -> Skip;

Guest = pcase {
        1 : guest.Door1{guess=Door1} -> Skip
        1 : guest.Door2{guess=Door2} -> Skip
        1 : guest.Door3{guess=Door3} -> Skip
};

Goat = []i:{Door1,Door2,Door3}@
        ifb (i != car && i != guess) {
                hostopen.i{goat = i} -> Skip
        };

TakeOffer = []i:{Door1,Door2,Door3}@
        ifb (i != guess && i != goat) {
                changeguess{guess = i; final = true} -> Stop
        };

NotTakeOffer = keepguess{final = true} -> Stop;

Sys_Take_Offer = PlaceCar; Guest; Goat; TakeOffer;

#assert Sys_Take_Offer reaches goal with prob;

Sys_Not_Take_Offer = PlaceCar; Guest; Goat; NotTakeOffer;

#assert Sys_Not_Take_Offer reaches goal with prob;
```



What if the host is Dishonest?

```
//place after guessing
Sys_With_Dishonest_Program = Guest; PlaceCar; Goat; NotTakeOffer;

#assert Sys_With_Dishonest_Program reaches goal with prob;

HostSwitch = pcase {
                1 : switch{car = guess} -> Skip
                2 : Skip
        };

Sys_With_Cheating_Host_Switch = PlaceCar; Guest; Goat; HostSwitch; TakeOffer;

#assert Sys_With_Cheating_Host_Switch reaches goal with prob;

Sys_With_Cheating_Host_Not_Switch = PlaceCar; Guest; Goat; HostSwitch; NotTakeOffer;

#assert Sys_With_Cheating_Host_Not_Switch reaches goal with prob;
```

Combine Real-Time and Probability

Passing me without stopping!

Given the C# Program of a lift algorithm

C# Library Editor and Compiler

New Library | New DataType | Open C# Code | Save | Release

```
16  public class LiftControl : ExpressionValue
17  {
18      //-1; for not assigned; i for assigned to i-lift;
19      int[] ExternalRequestsUp;
20      int[] ExternalRequestsDown;
21      //0; for not pressed, 1 for pressed
22      int[][] InternalRequests;
23      //0 for stopped at ground level; ready to go up.
24      int[] LiftStatus;
25
26      public LiftControl()
27      {
28          ExternalRequestsUp = new int[2];
29          ExternalRequestsDown = new int[2];
30          InternalRequests = new int[2][];
31          InternalRequests[0] = new int[2];
32          InternalRequests[1] = new int[2];
33          LiftStatus = new int[2];
34      }
35
36      public LiftControl(int levels, int lifts)
37      {
38          ExternalRequestsUp = new int[levels];
39          ExternalRequestsDown = new int[levels]; ;
40
41          for (int i = 0; i < levels; i++)
42          {
43              ExternalRequestsUp[i] = -1;
44              ExternalRequestsDown[i] = -1;
45          }
46          ;
47          InternalRequests = new int[lifts][];
48          LiftStatus = new int[lifts];
```

C# Library Editor and Compiler

New Library | New DataType | Open C# Code | Save | Release | Build DLL

```
273
274     public int PassBy (int lift, int level, int up)
275     {
276         //if (isToOpenDoor(lift, level) == 0)
277         //{
278             if (up > 0)
279             {
280                 if (ExternalRequestsUp[level] != lift && ExternalRequestsUp[level] >= 0)
281                 {
282                     return 1;
283                 }
284             }
285             else
286             {
287                 if (ExternalRequestsDown[level] >= 0 && ExternalRequestsDown[level] == lift)
288                 {
289                     return 1;
290                 }
291             }
292         //}
293
294         return 0;
295     }
296
297     public void AddInternalRequest(int lift, int level)
298     {
299         InternalRequests[lift][level] = 1;
300     }
301
302     public int UpdateLiftStatus(int lift, int level, int direction)
303     {
304         LiftStatus[lift] = LiftStatus[lift] + 1;
305
306         return PassBy(lift, level, direction);
307     }
```

PAT checking the C# program with time+probability

```
#import "PAT.Lib.Lift";
#define NoOfFloors 2;
#define NoOfLifts 2;
var<LiftControl> ctrl = new LiftControl(NoOfFloors,NoOfLifts);
var passby = 0;

aSystem = (||| x:{0..NoOfLifts-1} @ Lift(x, 0, 1)) ||| Requests();

Requests() = Request();Request();
Request() = pcase {
            1 : extreq.0.1{ctrl.AssignExternalRequest(0,1)} -> Skip
            1 : intreq.0.0.1{ctrl.AddInternalRequest(0,0)} -> Skip
            1 : intreq.1.0.1{ctrl.AddInternalRequest(1,0)} -> Skip
            1 : extreq.1.0{ctrl.AssignExternalRequest(1,0)} -> Skip
            1 : intreq.0.1.1{ctrl.AddInternalRequest(0,1)} -> Skip
            1 : intreq.1.1.1{ctrl.AddInternalRequest(1,1)} -> Skip
        } within[1];

Lift(i, level, direction) = case {
            ctrl.isToOpenDoor(i, level) == 1 : (serve.level.direction{ctrl.ClearRequests(i, level, direction)}
                                                -> Lift(i, level, direction))
            ctrl.KeepMoving(i, level, direction) == 1 : (reach.level+direction.direction
                                                         {passby = ctrl.UpdateLiftStatus(i, level, direction)}
                                                         -> Lift(i, level+direction, direction))
            ctrl.HasAssignment(i) == 1 : changedirection.i{ctrl.ChangeDirection(i)}
                                         -> Lift(i, level, -1*direction)
            default : idle.i -> Lift(i, level, direction)
        } within[2];

#define goal passby == 1;
#assert aSystem reaches goal with prob;
```

The Current Status

- PAT is available at http://pat.comp.nus.edu.sg
- 1Million lines of code, 15 modules with 200+ build in examples
- Used as an educational tool in many universities.
- Attracted 2000+ registered users in the last 4 years from 400+ organizations in 52 countries, e.g. Microsoft, HP, ST Elec, … Sony, Hitachi, Canon.
- Japanese PAT User group formed in Sep 2009:

Founding Members:
Hiroshi Fujimoto
Nobukazu Yoshioka
Toshiyuki Fujikura
Kenji Taguchi
Masaru Nagaku
Kazuto MATSUI

Current and Ongoing Works

- Security systems (FCS'12)
  - Security Protocols
  - Trusted Platform Module

- Web Service (Orc language /BPEL language) (APSEC'10, ICFEM'11)
- Sensor networks system written in NesC (SenSys'11)
  - Distributed algorithms

- Context-aware systems (ICOST'10)

- Model Driving Development MDA: UML diagram, StateFlow (ITTT'12)
  - Merlion 2011 funding on "Software Verification from Design to Implementation"

- Software-System Architecture Description Language (in implementation)
  - Event Grammar/ADL

- Verification of C# Programs (in progress)

- Multi-agent Systems (ICSE'12)

- Timed Transition Systems (TOSEM'12)

Some related and background papers

- Jun Sun, Yang Liu, Jin Song Dong, Yan Liu, Ling Shi, Etienne, Andre. **Modeling and Verifying Hierarchical Real-time Systems using Stateful Timed CSP**. The ACM Transactions on Software Engineering and Methodology (TOSEM). (Accepted)
- Yang Liu, Jun Sun and Jin Song Dong. **PAT 3: An Extensible Architecture for Building Multi-domain Model Checkers**. The 22nd annual International Symposium on Software Reliability Engineering (ISSRE 2011), Hiroshima, Japan, Nov 29 - Dec 2, 2011.
- Jun Sun, Yang Liu, Songzheng Song and Jin Song Dong. **PRTS: An Approach for Model Checking Probabilistic Real-time Hierarchical Systems**. The 13th International Conference on Formal Engineering Methods (ICFEM 2011), pages 147-162, Durham, United Kingdom, October 25-28, 2011.
- Shaojie Zhang, Jun Sun, Jun Pang, Yang Liu and Jin Song Dong. **On Combining State Space Reductions with Global Fairness Assumptions**. The 17th International Symposium on Formal Methods (FM 2011), pages 432 - 447, Lero, Limerick, Ireland, June 20 - 24, 2011.
- Y. Liu, J. Sun and J. S. Dong. **Analyzing Hierarchical Complex Real-time Systems**. *The ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2010)*, Santa Fe, New Mexico, USA, 7-11 November 2010.
- C. Chen, J. S. Dong, J. Sun and A. Martin. **A Verification System for Interval-based Specification Languages,** ACM Transactions on Software Engineering and Methodology, Volume 19(4), pages 1 - 36, ACM. 2010
- C. Chen, J. S. Dong and J. Sun. **A Formal Framework for Modeling and Validating Simulink Diagrams.** Formal Aspects of Computing. 21(5), pages 451-483, Springer. Oct, 2009.
- J. Sun, Y. Liu, J. S. Dong and J. Pang. **PAT: Towards Flexible Verification under Fairness.** The 21th International Conference on Computer Aided Verification (CAV 2009), Grenoble, France, June 2009.
- J. S. Dong, P. Hao, S. C. Qin, J. Sun and Y. Wang, **Timed Automata Patterns**. IEEE Transactions on Software Engineering, vol. 34(6), pp 844-859, Nov./Dec. 2008.
- C. Chen, J. S. Dong and J. Sun. **A Verification System for Timed Interval Calculus**, the 30th International Conference on Software Engineering (ICSE'08), 2008.
- K. Taguchi and J. S. Dong. **Formally Specifying and Verifying Mobile Agents : Model Checking Mobility,** International Journal of Agent-Oriented Software Engineering, 2008.
- J. Sun and J. S. Dong, **Design Synthesis from Interaction and State-based Specifications.** IEEE Transactions on Software Engineering, Vol-32(6):349-364, 2006
- H. Wang, J. S. Dong, J. Sun and J. Sun. **Reasoning Support for Semantic Web Ontology Family Languages Using Alloy.** International Journal of Multiagent and Grid Systems, IOS press, Vol-2(4):455-471, 2006.
- L. Yuan, J. S. Dong, J. Sun and H. A. Basit. **Generic Fault Tolerant Software Architecture Reasoning and Customization.** IEEE Transactions on Reliability. Vol-55(3):421-435, 2006
- J. Sun, J. S. Dong, S. Jarzabek and H. Wang. **CAD System Family Architecture and Verification: An Integrated Formal Approach.** IEE Proceedings Software. Vol-153(3):102-112, 2006.
- J. Sun and J. S. Dong, **Synthesis of Distributed Processes from Scenario-based Specifications.** Formal Methods 2005 (FM'05), LNCS, pp 415-431, Newcastle, UK. 2005.
- J.S. Dong and P. Hao and B. Mahony, **Formal Designs for Embedded and Hybrid Systems.** International Journal on Software Engineering and Knowledge Engineering, vol 15(2): 373-378, 2005
- X. Wang, J. S. Dong, C. Chin, S. R. Hettiarachchi and D. Zhang. **Semantic Space: A Semantic Web Infrastructure for Smart Spaces.** IEEE Pervasive Computing, 3(3):32-39, July-September 2004.
- S. C. Qin, J. S. Dong and W. N. Chin. **A Semantic Foundation of TCOZ in Unifying Theory of Programming.** FM'03. LNCS, pages 321-340, 2003.
- B. Mahony and J.S. Dong. **Deep Semantic Links of TCSP and Object-Z: TCOZ Approach**. Formal Aspects of Computing journal, 13:142-160, Springer, 2002.
- B. Mahony and J.S. Dong. **Timed Communicating Object Z.** IEEE Transactions on Software Engineering, 26(2):150-177, Feb 2000.
- B. Mahony and J.S. Dong. **Sensors and Actuators in TCOZ**. World Congress on Formal Methods (FM'99), LNCS, Vol 1709, pages 1166-1185, Springer-Verlag, Toulouse, 1999.
- B. Mahony and J.S. Dong. **Blending Object-Z and Timed CSP: an introduction to TCOZ**. In Proceedings of the 20th International Conference on Software Engineering (ICSE'98), editors: K. Futatsugi and R. Kemmerer and K. Torii, IEEE Press, pages 95-104, Kyoto, 1998.

Thank you!

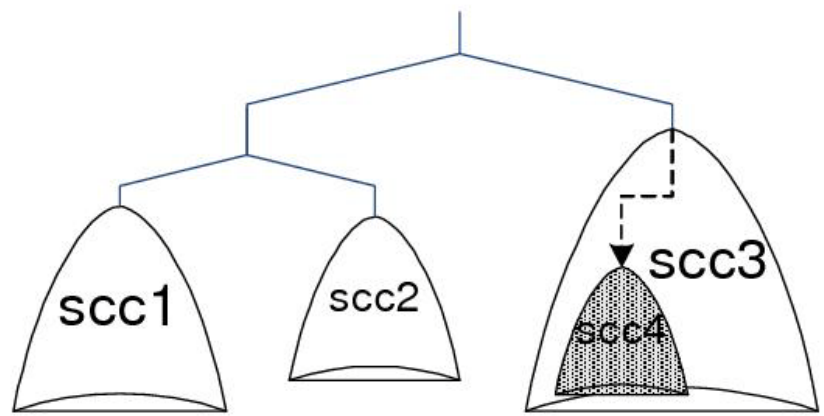
- Additional slides …

Monty hall: why switch?

Door 1	Door 2	Door 3	result if switching	result if staying
Car	Goat	Goat	**Goat**	**Car**
Goat	Car	Goat	**Car**	**Goat**
Goat	Goat	Car	**Car**	**Goat**

Verification under Fairness

- Automata-based LTL model checking
  - weak fairness: SCC search
  - strong fairness: strongly connected sub-graph search
  - strong global fairness = terminal SCC search



Experiment

Model	Size	EWF			ESF		SGF	
		Result	PAT	SPIN	Result	PAT	Result	PAT
LE_C	5	Yes	4.7	35.7	Yes	4.7	Yes	4.1
LE_C	6	Yes	26.7	229	Yes	26.7	Yes	23.5
LE_C	7	Yes	152.2	1190	Yes	152.4	Yes	137.9
LE_C	8	Yes	726.6	5720	Yes	739.0	Yes	673.1
LE_T	5	Yes	0.2	0.7	Yes	0.2	Yes	0.2
LE_T	7	Yes	1.4	7.6	Yes	1.4	Yes	1.4
LE_T	9	Yes	10.2	62.3	Yes	10.2	Yes	9.6
LE_T	11	Yes	68.1	440	Yes	68.7	Yes	65.1
LE_T	13	Yes	548.6	3200	Yes	573.6	Yes	529.6
LE_OR	3	No	0.2	0.3	No	0.2	Yes	11.8
LE_OR	5	No	1.3	8.7	No	1.8	–	–
LE_OR	7	No	15.9	95	No	21.3	–	–
LE_R	3	No	0.1	< 0.1	No	0.2	Yes	1.5
LE_R	4	No	0.3	< 0.1	No	0.7	Yes	19.5
LE_R	5	No	0.8	< 0.1	No	2.7	Yes	299.0
LE_R	6	No	1.8	0.2	No	4.6	–	–
LE_R	7	No	4.7	0.6	No	9.6	–	–
LE_R	8	No	11.7	1.7	No	28.3	–	–
TC_R	3	Yes	< 0.1	< 0.1	Yes	< 0.1	Yes	< 0.1
TC_R	5	No	< 0.1	< 0.1	No	< 0.1	Yes	0.6
TC_R	7	No	0.2	0.1	No	0.2	Yes	13.7
TC_R	9	No	0.4	0.2	No	0.4	Yes	640.2

Comparing with LTSA

- PAT supports a variety of fairness (process-level/event-level weak strong fairness and strong global fairness), LTSA supports only event-level strong fairness.
- PAT supports shared variables and external C# library, while LTSA doesn't support that.
- PAT supports both DFS and BFS search for deadlock-freeness check, while LTSA supports only BFS.
- PAT supports verification of LTL formulae made up of variable predicates and events, while LTSA supports LTL constituted by events only.
- PAT supports real-time systems, while LTSA supports the ad-hoc tick event.
- LTSA supports message sequence charts, and UML2, while PAT has not yet.

Example C: Pacemaker

AATpace =

(atomic{senseA -> paceA{SA = 0} -> Skip}
timeout[LRI]

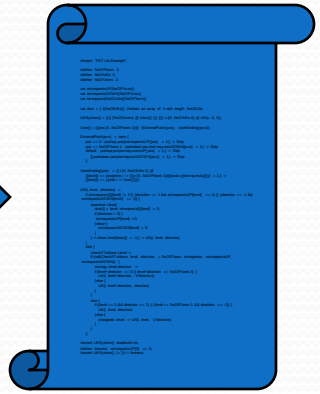
((paceA{SA = 0} -> Skip) within[0]);

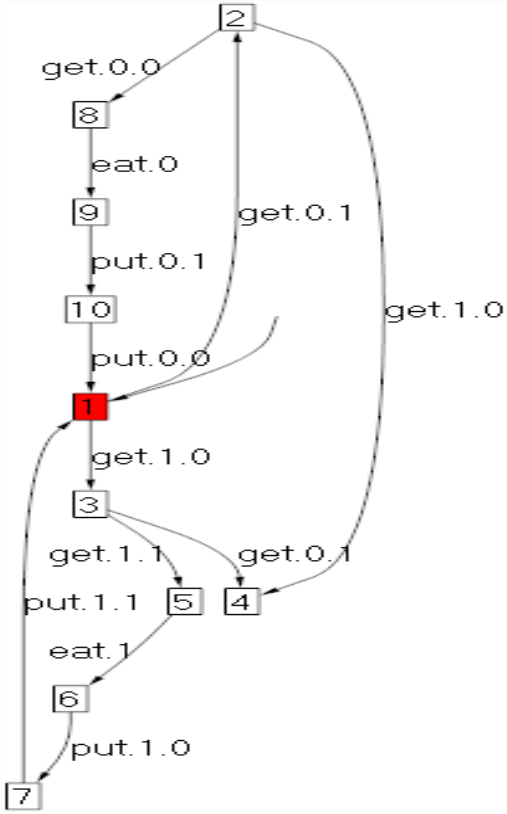
Wait[URI];

(enableSA{SA = 1} -> AATpace1) within[0]);

The behaviors of the composition of the pacemaker and a abnormal heart must refine a normal heart!

Operational Semantics





Operational Semantics

$$\frac{}{(V, e\{prog\} \rightarrow P) \xrightarrow{e} (upd(V, prog), P)} \; [\; prefix \;]$$

$$\frac{c \text{ is not empty in } V}{(V, c?x \rightarrow P) \xrightarrow{c?top(c)} (pop(V, c?x), P)} \; [\; in \;]$$

$$\frac{V \not\models b, (V, Q) \xrightarrow{e} (V', Q')}{(V, if\ b\ \{P\}\ else\ \{Q\}) \xrightarrow{e} (V', Q')} \; [\; cond2 \;]$$

$$\frac{(V, P) \xrightarrow{x} (V', P'), x \in \alpha P, x \notin \alpha Q}{(V, P \parallel Q) \xrightarrow{x} (V', P' \parallel Q)} \; [\; par1 \;]$$

Abstraction

$$\frac{(V, P, D) \overset{\tau}{\hookrightarrow} (V', P', D')}{(V, P\ timeout[d]_{tm}\ Q, D) \overset{\tau}{\hookrightarrow} (V', P'\ timeout[d]_{tm}\ Q, D' \wedge tm \leq d)}\ [\ ato1\ ]$$

$$\frac{(V, P, D) \overset{x}{\hookrightarrow} (V', P', D')}{(V, P\ timeout[d]_{tm}\ Q, D) \overset{x}{\hookrightarrow} (V', P', D' \wedge tm \leq d)}\ [\ ato2\ ]$$

$$\frac{}{(V, P\ timeout[d]_{tm}\ Q, D) \overset{\tau}{\hookrightarrow} (V, Q, tm = d \wedge \iota(V, P, D))}\ [\ ato3\ ]$$

Abstraction: Example

(a → Wait[5]; b → Stop) interrupt[3] (c → Stop)

- Introduce clock t1,

(a → Wait[5]; b → Stop) interrupt[3]_t1 (c → Stop), t1 = 0

- Event a occurs,

(Wait[5]; b → Stop) interrupt[3]_t1 (c → Stop), 0 <=t1 <= 3

- Introduce t2,

(Wait[5]_t2; b → Stop) interrupt[3]_t1 (c → Stop), 0 <=t1 <= 3 and t2 = 0

- Event τ occurs,

(b → Stop) interrupt[3]_t1 (c → Stop), 0 <=t1 <= 3 and t2 =5

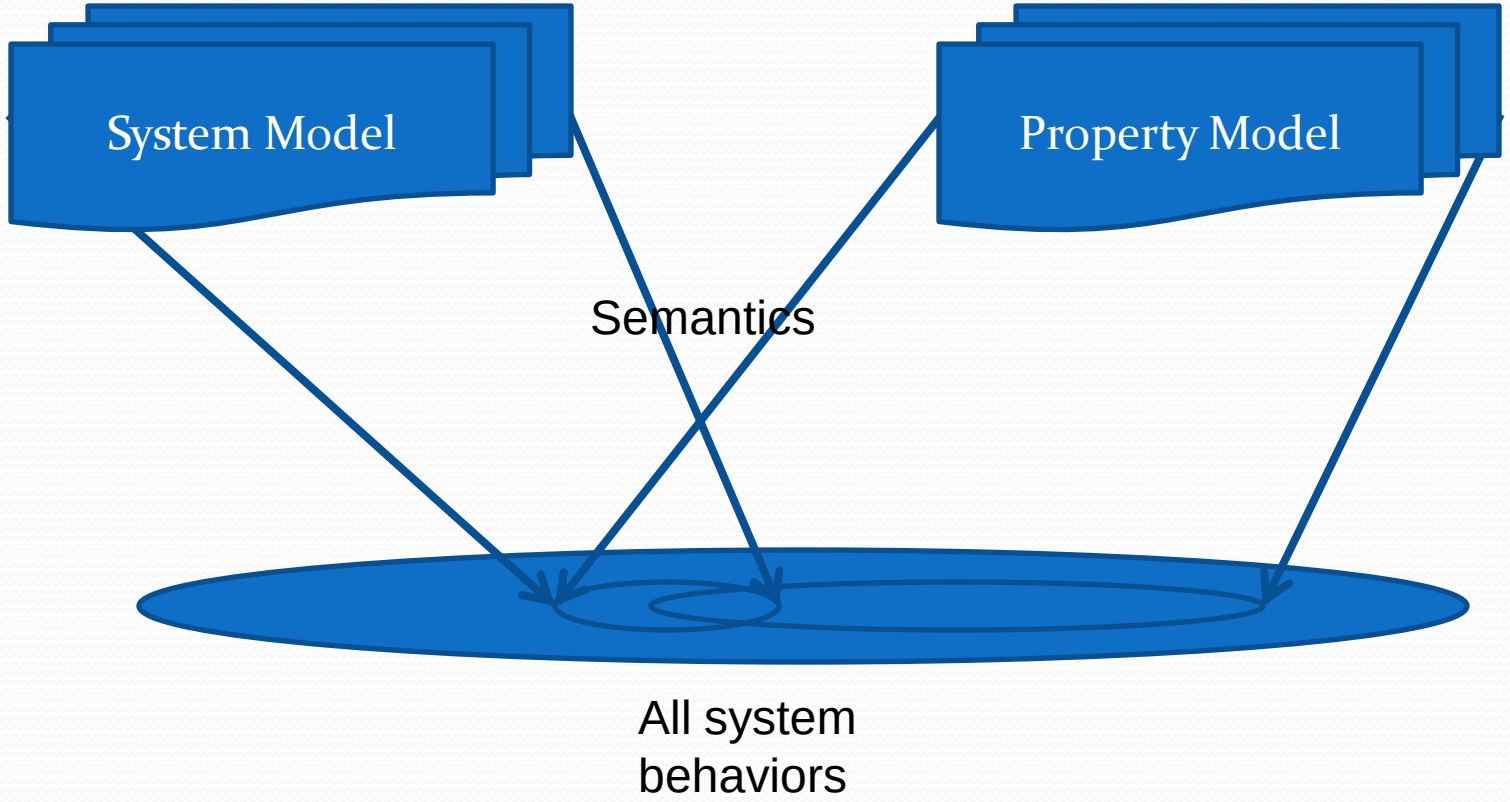
Experiment

Model	Size	Property	States/Transitions	PAT (s)
Fischer	4	$\Box ct \leq 1$	3452/8305	0.22
Fischer	5	$\Box ct \leq 1$	26496/73628	2.49
Fischer	6	$\Box ct \leq 1$	207856/654776	27.7
Fischer	7	$\Box ct \leq 1$	1620194/5725100	303
Fischer	4	$\Box(x = i \Rightarrow \Diamond cs.i)$	5835/16776	0.53
Fischer	5	$\Box(x = i \Rightarrow \Diamond cs.i)$	49907/169081	5.83
Fischer	6	$\Box(x = i \Rightarrow \Diamond cs.i)$	384763/1502480	70.5
Fischer	4	$Protocol\ refines\ uProtocol$	7741/18616	5.22
Fischer	5	$Protocol\ refines\ uProtocol$	72140/201292	126.3
Fischer	6	$Protocol\ refines\ uProtocol$	705171/2237880	3146
Railway Control	4	deadlock-free	853/1132	0.11
Railway Control	5	deadlock-free	4551/6115	0.42
Railway Control	6	deadlock-free	27787/37482	3.07
Railway Control	7	deadlock-free	195259/263641	24.2
Railway Control	8	deadlock-free	1563177/2111032	223.1
Railway Control	4	$\Box(appr.1 \rightarrow \Diamond leave.1)$	1504/1985	0.16
Railway Control	5	$\Box(appr.1 \rightarrow \Diamond leave.1)$	8137/10862	0.95
Railway Control	6	$\Box(appr.1 \rightarrow \Diamond leave.1)$	50458/67639	6.58
Railway Control	7	$\Box(appr.1 \rightarrow \Diamond leave.1)$	359335/482498	58.63

Refinement Checking

- The property is given as a model (often in the same language).
- A property is proved by showing a refinement relationship (i.e. language inclusion) from the system model to the model capturing the property.
  - Trace refinement checking,
  - Stable failures refinement checking,
  - Failures/divergence refinement checking,
  - Timed trace refinement checking,
  - and etc.

Refinement Checking



Example B: Parallel Objects

- Sequential stack
  - call Push → put item on the top → finish Push
- Lock-free concurrent stack
  - call Push → read stack → make local modification → check if the stack has been updated → if not, commit; else retry → finish Push

A concurrent stack must refine the sequential stack.
The sequential stack must refine the concurrent stack.

PAT's Approach

- Given two transition systems S and T, to show that S refines T,
  - Build pair (s, X) on-the-fly where s is a reachable state of S and X is the set of states which can be reached via the same trace.
  - If X becomes empty, then S doesn't refine T;

Experiment D

Model	Size	Property	States/Transitions	Result	Time (s)
Pacemaker	-	deadlock-free	302442/2405850	true	92.1
Pacemaker	-	correctness	986342/2608226	true	122
Fischer	4	mutual exclusion	9941/34244	true	0.78
Fischer	5	mutual exclusion	141963/599315	true	17.2
Fischer	6	mutual exclusion	2144610/10795380	true	401
Fischer	6	bounded bypass	2429/8065	false	0.36
Fischer	7	bounded bypass	9213/34611	false	1.47
Fischer	8	bounded bypass	32785/137417	false	6.16
Fischer	9	bounded bypass	91665/425966	false	21.1
Fischer	10	bounded bypass	300129/1542020	false	79.8
Fischer	11	bounded bypass	693606/3880577	false	214
Railway Control	4	bounded waiting	918/1359	true	0.45
Railway Control	5	bounded waiting	4764/7199	true	3.21
Railway Control	6	bounded waiting	28782/43795	true	26.2
Railway Control	7	bounded waiting	201444/307071	true	238