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
  {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 &&

#assert Game() reaches goal with min(c);
The sliding game problem cont’d

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.
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.

```c
#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?
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?

```plaintext
//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

```csharp
public class LiftControl : ExpressionValue
{
    // -1; for not assigned; i for assigned to i-lift;
    int[,] ExternalRequestsUp;
    int[,] ExternalRequestsDown;
    // 0; for not pressed, 1 for pressed
    int[,][] InternalRequests;
    // 0 for stopped at ground level; ready to go up.
    int[] LiftStatus;

    public LiftControl()
    {
        ExternalRequestsUp = new int[2];
        ExternalRequestsDown = new int[2];
        InternalRequests = new int[2][,];
        InternalRequests[0] = new int[2];
        InternalRequests[1] = new int[2];
        LiftStatus = new int[2];
    }

    public LiftControl(int levels, int lifts)
    {
        ExternalRequestsUp = new int[levels];
        ExternalRequestsDown = new int[levels];

        for (int i = 0; i < levels; i++)
        {
            ExternalRequestsUp[i] = -1;
            ExternalRequestsDown[i] = -1;
        }

        InternalRequests = new int[lifts][,];
        LiftStatus = new int[lifts];
    }

    public int PassBy (int lift, int level, int up)
    {
        // [IsToOpenDoor(lift, level) == 0]
        if (up > 0)
        {
            if (ExternalRequestsUp[level] != lift && ExternalRequestsUp[level] > 0)
            {
                return 1;
            }
        }
        else
        {
            if (ExternalRequestsDown[level] > 0 & ExternalRequestsDown[level] == lift)
            {
                return 1;
            }
        }

        return 0;
    }

    public void AddInternalRequest(int lift, int level)
    {
        InternalRequests[lift][level] = 1;
    }

    public int UpdateLiftStatus(int lift, int level, int direction)
    {
        LiftStatus[lift] = LiftStatus[lift] + 1;
        return PassBy(lift, level, direction);
    }
}
```
PAT checking the C# program with time+probability

```csharp
#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](http://pat.comp.nus.edu.sg)
- 1 Million 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.
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)
Thank you!
• Additional slides ...
Monty hall: why switch?

<table>
<thead>
<tr>
<th>Door 1</th>
<th>Door 2</th>
<th>Door 3</th>
<th>result if switching</th>
<th>result if staying</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
<td>Goat</td>
<td>Goat</td>
<td>Goat</td>
<td>Car</td>
</tr>
<tr>
<td>Goat</td>
<td>Car</td>
<td>Goat</td>
<td>Car</td>
<td>Goat</td>
</tr>
<tr>
<td>Goat</td>
<td>Goat</td>
<td>Car</td>
<td>Car</td>
<td>Goat</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
<th>EWF</th>
<th>ESF</th>
<th>SGF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Result</td>
<td>PAT</td>
<td>SPIN</td>
</tr>
<tr>
<td>LE_C</td>
<td>5</td>
<td>Yes</td>
<td>4.7</td>
<td>35.7</td>
</tr>
<tr>
<td>LE_C</td>
<td>6</td>
<td>Yes</td>
<td>26.7</td>
<td>229</td>
</tr>
<tr>
<td>LE_C</td>
<td>7</td>
<td>Yes</td>
<td>152.2</td>
<td>1190</td>
</tr>
<tr>
<td>LE_C</td>
<td>8</td>
<td>Yes</td>
<td>726.6</td>
<td>5720</td>
</tr>
<tr>
<td>LE_T</td>
<td>5</td>
<td>Yes</td>
<td>0.2</td>
<td>0.7</td>
</tr>
<tr>
<td>LE_T</td>
<td>7</td>
<td>Yes</td>
<td>1.4</td>
<td>7.6</td>
</tr>
<tr>
<td>LE_T</td>
<td>9</td>
<td>Yes</td>
<td>10.2</td>
<td>62.3</td>
</tr>
<tr>
<td>LE_T</td>
<td>11</td>
<td>Yes</td>
<td>68.1</td>
<td>440</td>
</tr>
<tr>
<td>LE_T</td>
<td>13</td>
<td>Yes</td>
<td>548.6</td>
<td>3200</td>
</tr>
<tr>
<td>LE.OR</td>
<td>3</td>
<td>No</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>LE.OR</td>
<td>5</td>
<td>No</td>
<td>1.3</td>
<td>8.7</td>
</tr>
<tr>
<td>LE.OR</td>
<td>7</td>
<td>No</td>
<td>15.9</td>
<td>95</td>
</tr>
<tr>
<td>LE.R</td>
<td>3</td>
<td>No</td>
<td>0.1</td>
<td>&lt; 0.1</td>
</tr>
<tr>
<td>LE.R</td>
<td>4</td>
<td>No</td>
<td>0.3</td>
<td>&lt; 0.1</td>
</tr>
<tr>
<td>LE.R</td>
<td>5</td>
<td>No</td>
<td>0.8</td>
<td>&lt; 0.1</td>
</tr>
<tr>
<td>LE.R</td>
<td>6</td>
<td>No</td>
<td>1.8</td>
<td>0.2</td>
</tr>
<tr>
<td>LE.R</td>
<td>7</td>
<td>No</td>
<td>4.7</td>
<td>0.6</td>
</tr>
<tr>
<td>LE.R</td>
<td>8</td>
<td>No</td>
<td>11.7</td>
<td>1.7</td>
</tr>
<tr>
<td>TC.R</td>
<td>3</td>
<td>Yes</td>
<td>&lt; 0.1</td>
<td>&lt; 0.1</td>
</tr>
<tr>
<td>TC.R</td>
<td>5</td>
<td>No</td>
<td>&lt; 0.1</td>
<td>&lt; 0.1</td>
</tr>
<tr>
<td>TC.R</td>
<td>7</td>
<td>No</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>TC.R</td>
<td>9</td>
<td>No</td>
<td>0.4</td>
<td>0.2</td>
</tr>
</tbody>
</table>
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

\[ \text{AATpace} = \]
\[
(\text{atomic}\{\text{senseA} \rightarrow \text{paceA}\{\text{SA} = 0\} \rightarrow \text{Skip}\}
\]
\text{timeout}[\text{LRI}]
\]
\[
((\text{paceA}\{\text{SA} = 0\} \rightarrow \text{Skip}) \text{ within}[0])
\]
\text{Wait}[\text{URI}];
\]
\[
(\text{enableSA}\{\text{SA} = 1\} \rightarrow \text{AATpace}_1) \text{ within}[0]);
\]

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

```c
#include "PAT.Lib.Example"

#define NoOfFloors 2;
#define NoOfLifts 2;
#define NoOfUsers 2;

var extrequestsUP[NoOfFloors];
var extrequestsDOWN[NoOfFloors];
var intrequests[NoOfLifts][NoOfFloors];

var door = [-1(NoOfLifts)]; //initiate an array of -1 with length NoOfLifts

LiftSystem() = (||| { NoOfUsers} @ User()) ||| (||| x:{0..NoOfLifts-1} @ Lift(x, 0, 1));

User() = \[
    pos:{0..NoOfFloors-1}@ ( ExternalPush(pos); UserWaiting( pos));
\]

ExternalPush(pos) = case {
    pos = = 0 : pushup.pos{extrequestsUP[pos] = 1;} -> Skip
    pos = = NoOfFloors-1 : pushdown.pos {extrequestsDOWN[pos] = 1;} -> Skip
    default : pushup.pos{extrequestsUP[pos] = 1;} -> Skip
    [] pushdown.pos{extrequestsDOWN[pos] = 1;} -> Skip
};

UserWaiting(pos) = \
    i:{0..NoOfLifts-1} @
    (\[door[i] = = pos\]enter.i -> (\[y:{0..NoOfFloors -1}\]@ (push.y{intrequests[i][y] = 1;} ->
        (\[door[i] == y\] exit.i -> User()))));

Lift(i, level, direction) =
    if (intrequests[i][level] != 0 || (direction = = 1 && extrequestsUP[level] = = 1) || (direction = = -1 && extrequestsDOWN[level] == 1)) {
        opendoor.i.level{
            door[i] = level; intrequests[i][level] = 0;
            if (direction > 0) {
                extrequestsUP[level] = 0;
            } else {
                extrequestsDOWN[level] = 0;
            }
        } -> close.i.level{door[i] = -1;} -> Lift(i, level, direction)
    } else {
        checkIfToMove.i.level ->
            if (call(CheckIfToMove, level, direction, i, NoOfFloors, intrequests, extrequestsUP, extrequestsDOWN)) {
                moving.i.level.direction ->
                    if (level + direction == 0 || level + direction = = NoOfFloors-1) {
                        Lift(i, level + direction, -1*direction)
                    } else {
                        Lift(i, level + direction, direction)
                    }
            } else {
                if ((level = = 0 && direction = = 1) || (level = = NoOfFloors-1 && direction = = -1)) {
                    Lift(i, level, direction)
                } else {
                    changedir.i.level -> Lift(i, level, -1*direction)
                }
            }
    };

#define liveness extrequestsUP[0] = = 0;
#define liveness extrequestsU[1] = = 0;

#assert LiftSystem() deadlockfree;
#assert LiftSystem() |= [[]<> liveness];
```
Operational Semantics

\[
\begin{align*}
(V, e\{\text{prog}\} \rightarrow P) & \xrightarrow{e} (\text{upd}(V, \text{prog}), P) & \text{[prefix]} \\
\text{c is not empty in } V & \hline \\
(V, c?x \rightarrow P) & \xrightarrow{c?\text{top}(c)} (\text{pop}(V, c?x), P) & \text{[in]} \\
V \not\ni b, (V, Q) & \xrightarrow{e} (V', Q') & \text{[cond2]} \\
(V, \text{if } b \{P\} \text{ else } \{Q\}) & \xrightarrow{e} (V', Q') \\
(V, P) & \xrightarrow{x} (V', P'), x \in \alpha P, x \not\in \alpha Q & \text{[par1]} \\
(V, P \parallel Q) & \xrightarrow{x} (V', P' \parallel Q)
\end{align*}
\]
Abstraction

\[
(V, P, D) \xrightarrow{\tau} (V', P', D') \quad [\text{ato1}]
\]

\[
(V, P \ \text{timeout}[d]_{tm} \ Q, D) \xrightarrow{\tau} (V', P' \ \text{timeout}[d]_{tm} \ Q, D' \wedge tm \leq d)
\]

\[
(V, P, D) \xrightarrow{x} (V', P', D') \quad [\text{ato2}]
\]

\[
(V, P \ \text{timeout}[d]_{tm} \ Q, D) \xrightarrow{x} (V', P', D' \wedge tm \leq d)
\]

\[
(V, P \ \text{timeout}[d]_{tm} \ Q, D) \xrightarrow{\tau} (V, Q, tm = d \wedge \nu(V, P, D)) \quad [\text{ato3}]
\]
Abstraction: Example

(a $\rightarrow$ Wait[5]; b $\rightarrow$ Stop) interrupt[3] (c $\rightarrow$ Stop)

- Introduce clock $t_1$,
  (a $\rightarrow$ Wait[5]; b $\rightarrow$ Stop) interrupt[3]$_{t_1}$ (c $\rightarrow$ Stop), $t_1 = 0$
- Event $a$ occurs,
  (Wait[5]; b $\rightarrow$ Stop) interrupt[3]$_{t_1}$ (c $\rightarrow$ Stop), $0 \leq t_1 \leq 3$
- Introduce $t_2$,
  (Wait[5]$_{t_2}$; b $\rightarrow$ Stop) interrupt[3]$_{t_1}$ (c $\rightarrow$ Stop), $0 \leq t_1 \leq 3$ and $t_2 = 0$
- Event $\tau$ occurs,
  (b $\rightarrow$ Stop) interrupt[3]$_{t_1}$ (c $\rightarrow$ Stop), $0 \leq t_1 \leq 3$ and $t_2 = 5$
### Experiment

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

System Model

Property Model

Semantics

All system behaviors
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

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