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 &&

#define Game() / Game

#define min(c) / c

#define Game() reaches goal with min(c);
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 Concurrent 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.
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?

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.
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!
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
- 1 Million lines of code, 11 modules with 100+ build in examples
- Used as an educational tool in many universities.
- Attracted more than 1700 registered users in the last 3 years from more than 350 organizations, e.g. Microsoft, HP, ST Elec, Oxford Univ., ... 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
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)


