Pervasive Model Checking: PAT Approach

Jin Song Dong
National University of Singapore (NUS) and Griffith University
Joint work with Jun Sun (SUTD), Yang Liu (NTU) and 20 other PhD/Postdocs

Thanks: Prof. Honiden, Dr. Yoshioka, Dr. Fujikura

Intel Pentium 1994 Bug

- Try 4195835 – 4195835 / 3145727 * 3145727 = 0 ?

  In ’94 Pentium, it doesn’t return 0, but 256.

- The bug is found by a mathematics professor back in 1994
- Intel uses the SRT (Sweeney, Robertson, and Tocher) algorithm for floating point division. Five entries in the lookup table are missing.

- Cost: **~$500 million (~$800 million in today context)**

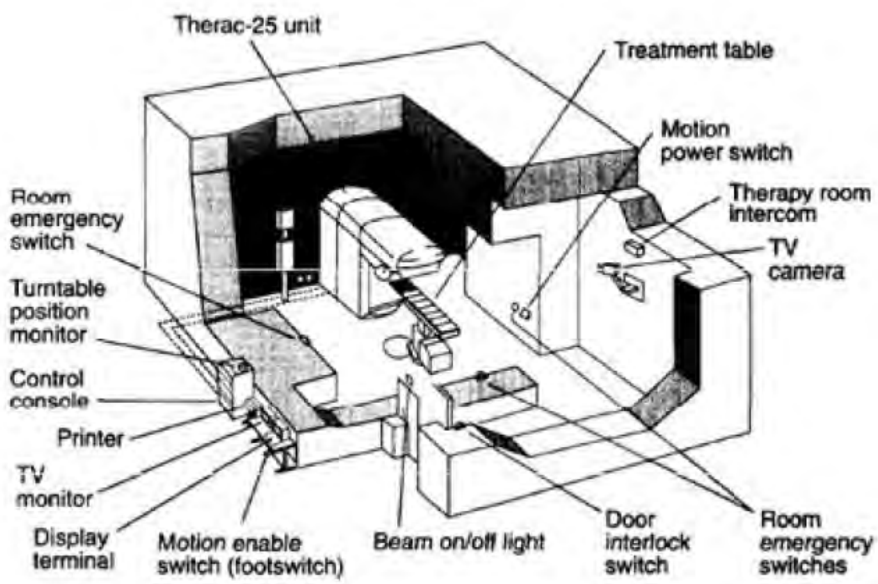

The explosion of the Ariane 5 (1996)

- In 1996, Europe's unmanned satellite-launching rocket, the Ariane 5, was blown up just seconds after taking off on its maiden flight from Kourou, French Guiana. The European Space Agency estimated that total development of Ariane 5 cost more than $8bn. On board Ariane 5 was a $500 million set of four scientific satellites created to study how the Earth's magnetic field interacts with Solar Winds.
- According to a piece in the New York Times Magazine, the self-destruction was triggered by software trying to stuff "a 64-bit number into a 16-bit space."

Airbus A380 suffers from incompatible software issues (2006)

- The Airbus issue of 2006 highlighted a problem many companies can have with software: What happens when one program doesn't talk to the another. In this case, the problem was caused by two halves of the same program, the CATIA software that is used to design and assemble one of the world's largest aircraft, the Airbus A380.

Therac-25 Radiation Overdosing (1985-87)



- Radiation machine for treatment of cancer patients
- At least 6 cases of overdoses in period 1985–1987 (100-times doses)
- Three cancer patients died
- Source: Design error in the control software (race condition)

Proton Therapy Machine Crashes (2017)

- Radiation machine for treatment of cancer patients in Massachusetts General Hospital, Boston.
- Symptoms:

  1) Un-expected system crashes

  2) Does given to the patient is different from the subscription amount
- Source: **still unknown**

Early History of Formal Methods

- a 1949 paper "Checking Large Routine" presented by Alan Turing at a conference on High Speed Automatic Calculating Machines at Cambridge University in 1949.

  Turing is regarded as father of computer science, Nobel Prize in computing is named after Turing, called Turing Award

- a 1967 paper by R. W. Floyd (1976 Turing Winner) "Assigning meanings to programs" In Proc. Symp. In Applied Mathematics.

- a 1969 paper by C. A. R. Hoare (1980 Turing Winner). "An axiomatic basis for computer programming." Communications of the ACM. Another great contribution from Hoare is CSP (Communicating Sequential Process) which is an elegant formalism for concurrency.

Model Checking Works!

- Three researchers won ACM Turing Award 2007 for their pioneer work on model checking

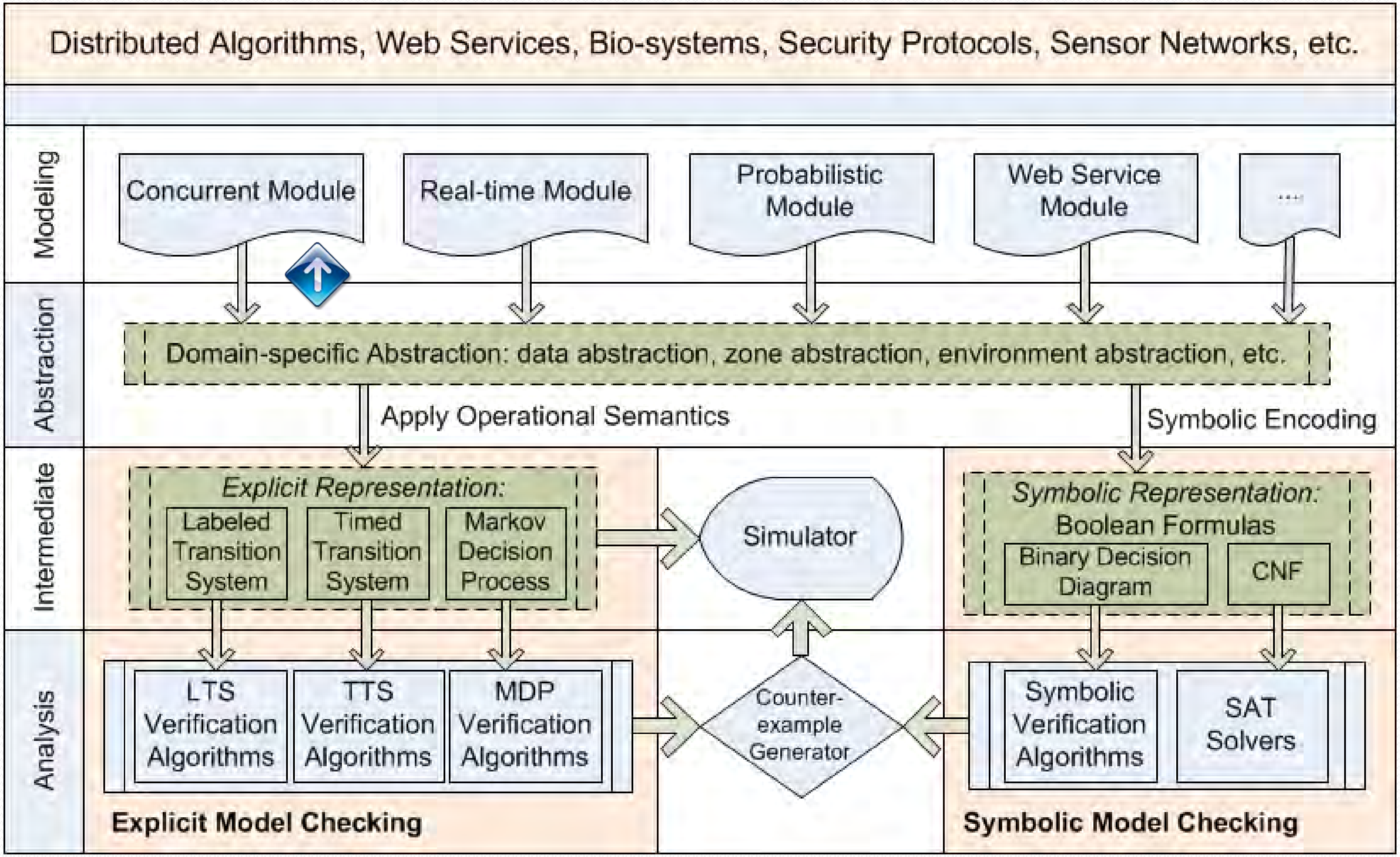


- In 2009, Intel i7 processor is verified by symbolic model checking completely without executing a single test case!

- The Slam project from Microsoft successfully detected many bugs in many driver software!

SLAM

PAT System (ICSE’08’12’13, CAV’09’12’13, FM’11’12’14, TOSEM’13, TSE’13, FMSD’13)



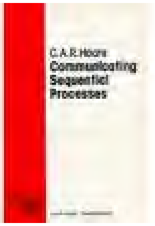
Core Members:
Sun Jun (SUTD), Liu Yang (NTU), Dong Jin Song (NUS)
Rest of the team: 20 PhD/Postdoc/RAs

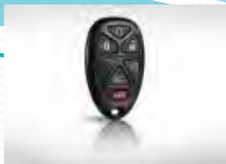
PAT’ concurrent system module is based on CSP:

Communicating Sequencial Processes

Hoare’s CSP (Communicating Sequential Processes) an *event* based notation primarily aimed at de scribing the sequencing of behaviour within a process and the synchronisation of behaviour (or *communication*) between processes. Events represent a co-operative synchronisation between process and environment. Both process and environment may control the behaviour of the other by *enabling* or *refusing* certain events or sequences of events.

- A.W. Roscoe. The Theory and Practice of Concurrency. Prentice-Hall, 1997.
- C. A. R. Hoare. Communicating Sequential Processes. Prentice-Hall International, 1985.






Case Study: Keyless System

- One of the latest automotive technologies, push-button keyless system, allows you to start your car's engine without the hassle of key insertion and offers great convenience.

- Push-button keyless system allows owner with key-fob in her pocket to unlock the door when she is very near the car. The driver can slide behind the wheel, with the key-fob in her pocket (briefcase or purse or anywhere inside the car), she can push the start/stop button on the control panel. Shutting off the engine is just as hassle-free, and is accomplished by merely pressing the start/stop button.

- These systems are designed so it is impossible to start the engine without the owner's key-fob and it cannot lock your key-fob inside the car because the system will sense it and prevent the user from locking them in.

- However, the keyless system can also surprise you as it may allow you to drive the car without key-fob. E.g. you can drive without key!

Constant and variables of KCS in PAT

```
#define N 2;          // number of owners

#define far 0;        // owner is out and far away from the car
#define near 1;       // owner is close enough to open/lock the door if she has the keyfob
#define in 2;         // owner is in the car

#define off 0;        // engine is off
#define on 1;         // engine is on

#define unlock 0;     // door is unlocked but closed
#define lock 1;       // door is locked (must be closed)
#define open 2;       // door is open

#define incar -1;     // keyfob is put inside car
#define faralone -2;  // keyfob is put outside and far

var owner[N];         //owners' position. initially, all users are far away from the car
var engine = off;     // engine status, initially off
var door = lock;      // door status, initially locked
var key = 0;          // key fob position, initially, it is with first owner
var moving = 0;       // car moving status, 0 for stop and 1 for moving
var fuel = 10;        // energy costs, say 1 for a short drive and 5 for a long driving
```

Owner positions

car = (||i:{0..N-1} @ (owner_pos(i) || motor(i) || door_op(i) || key_pos(i)));

```
owner_pos(i) =
  [owner[i] == far]towards.i{owner[i] = near} -> owner_pos(i)
        []
  [owner[i] == near]goaway.i{owner[i] = far} -> owner_pos(i)
        []
  [owner[i] == near && door == open && moving==0]getin.i{owner[i] = in}
   -> owner_pos(i)
        []
  [owner[i] == in && door == open && moving==0]goout.i{owner[i] = near}
   -> owner_pos(i);
```

Key-fob position

```
key_pos(i) =
   [key == i && owner[i] == in]putincar.i{key = incar} -> key_pos(i)
       []
   [key == i && owner[i] == far]putaway.i{key = faralone} -> key_pos(i)
       []
   [(key == faralone && owner[i] == far) || (key == incar &&
   owner[i] == in)]getkey.i{key = i} -> key_pos(i);
```

Door operation


```
door_op(i) =
      [key == i && owner[i]==near && door ==lock &&
         moving==o]unlockopen.i{door = open} -> door_op(i)
      []
      [owner[i]==near && door==unlock &&
         moving==o]justopen.i{door = open} -> door_op(i)
      []
      [door != open && owner[i] == in]insideopen.i{door = open}
         -> door_op(i)
      []
      [door == open]close.i{door = unlock} -> door_op(i)
      []
      [door==unlock&&owner[i]==in]insidelock.i{door=lock} -> door_op(i)
      []
      [door == unlock && owner[i]==near && key==i]outsidelock.i{door=lock}
  -> door_op(i);
```

Motor

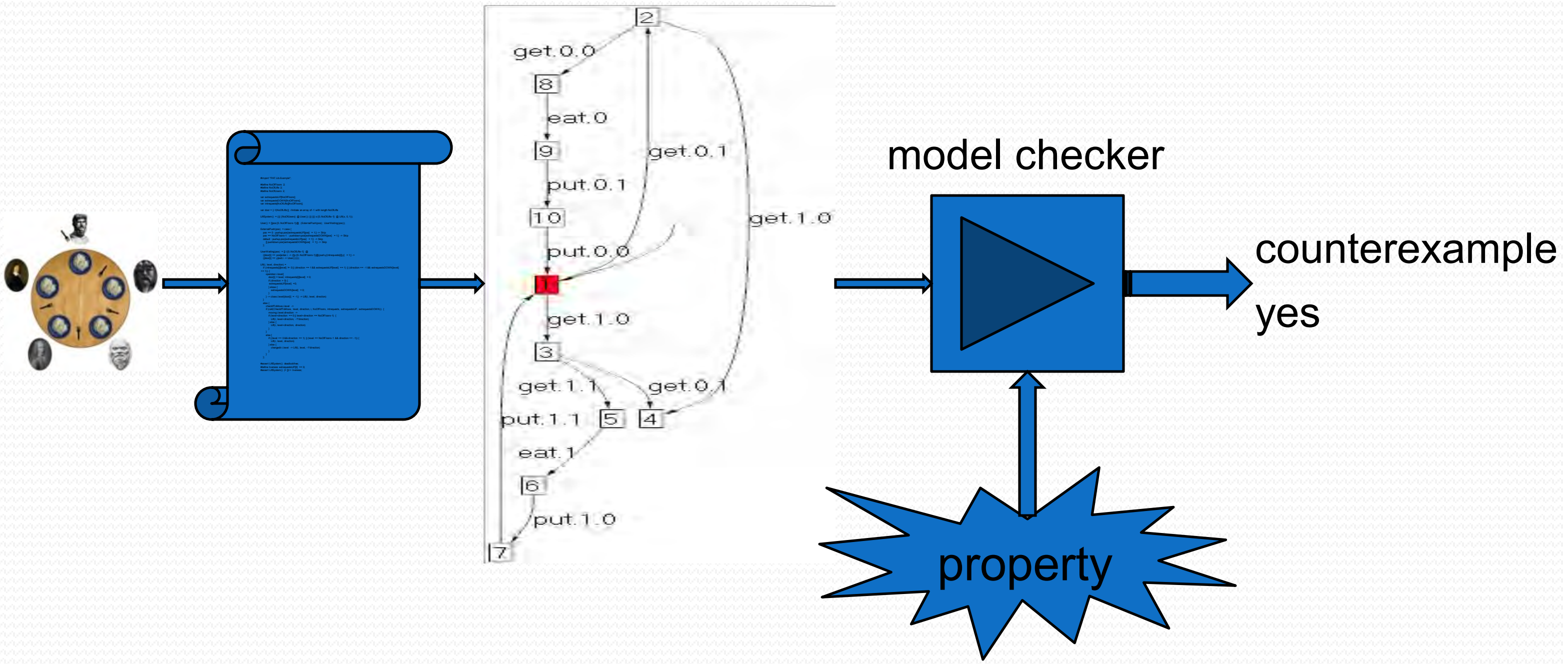
```
motor(i) =
    [owner[i]==in&&(key==i||key==incar)&&engine==off && fuel!= 0]turnon.i{engine = on} -> motor(i)
    []
    [engine==on&&owner[i]==in&&moving==0]startdrive.i{moving=1} -> motor(i)
    []
    [moving==1&&fuel!=0]shortdrive.i{fuel=fuel-1;if (fuel==0) {engine=off; moving =0}} -> motor(i)
    []
    [moving==1&&fuel > 5]longdrive.i{fuel=fuel-5;if (fuel==0) {engine=off; moving =0}} -> motor(i)
    []
    [engine==on&&moving==1&&owner[i]==in]stop.i{moving=0} -> motor(i)
    []
    [fuel==0&&engine==off]refill{fuel=10} -> motor(i)
    []
    [engine==on&&moving==0&&owner[i]==in]turnoff.i{engine = off;} -> motor(i);

car =     (||i:{0..N-1} @ (motor(i) || door_op(i) || key_pos(i) || owner_pos(i)));
```

Reasoning

```
#define keylockinside (
        key == incar && door == lock && owner[0] != in && owner[1] != in);

#define drivewithoutengineon (moving==1 && engine==off);

#define drivewithoutkeyholdbyother (
        moving ==1 && owner[1] == in && owner[0] == far && key == 0);

#assert car deadlockfree;

#assert car reaches keylockinside;

#assert car reaches drivewithoutengineon;

#assert car reaches drivewithoutkeyholdbyother;

#assert car reaches drivewithoutkeyholdbyother with max(fuel);
```

PAT Demonstration

- http://www.patroot.com
- http://pat.comp.nus.edu.sg



Tutorial question: extending system

- Your tasks is to extend the current system with two more operations: *window(i)* which can open (totally or partially) the car door window
- *throwKey(i)* which captures that the key-fob can be throw in/out of the car.

```
car =     (|||i:{0..N-1} @ (motor(i) ||| door_op(i) ||| key_pos(i) ||| owner_pos(i)) ||| window(i) ||| throwKey(i));
var window_state= 0;
// 0 for closed, 1 for down (partial/full) open

window(i) = ?

throwKey(i) = ?
```

Tutorial Solution

```
window(i) =
          [window_state == 0 && owner[i] == in && engine
==on]open_window.i{window_state = 1} -> window(i)
         []
         [window_state == 1 && owner[i] == in && engine
==on]close_window.i{window_state = 0} -> window(i);

throwKey(i) =
              [window_state == 1 && owner[i] == in &&
key == i]throw_key_out.i{key = faralone} -> throwKey(i)
             []
               [window_state == 1 && owner[i] != in &&
key == i]throw_key_in.i{key = incar} -> throwKey(i);
```

PAT Recent Work:

Pervasive Model Checking

- *Wide application domains, including Real-Time and Probabilistic systems; supporting many formalisms* *[FM'12, CAV'13, TSE'13, TOSEM'13]*

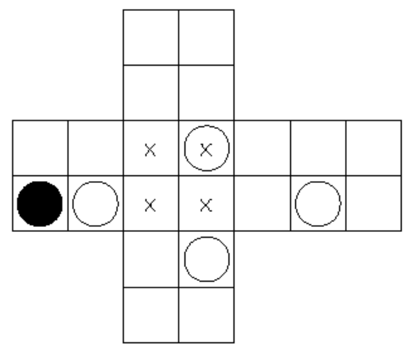
Towards Event Analytics

- *Model Checking as Services for Event Planning, Scheduling, Predication, Decision-Making* [FMSD'13]

Problem Solving: Shunting Example

The figure below gives the board and starting position for a game of *Shunting*. A move consists of the black piece (the shunter) moving one position either vertically or horizontally provided either

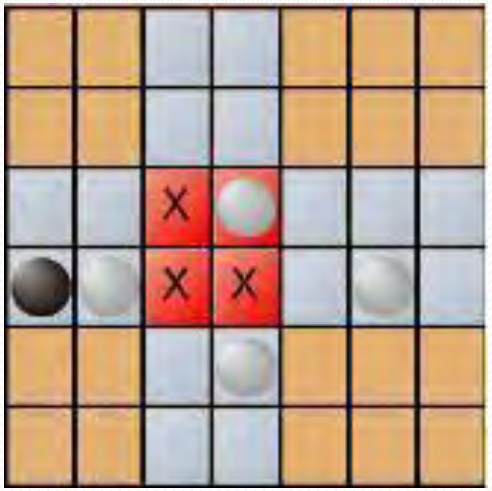
- the position moved to is empty, or
- the position moved to is occupied by a white piece but the position beyond the white piece is empty, in which case the white piece is pushed into the empty position.



In fact, let's make it more interesting that the black(dog) has limited energy and PushUp(the hill) requires extra energy.

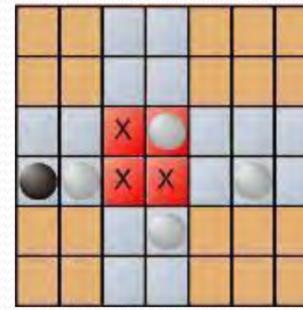
The shunter can not push two white pieces at the same time. At each stage a score is kept of the number of moves made so far. The game ends when the white pieces occupy the four positions marked with a cross.

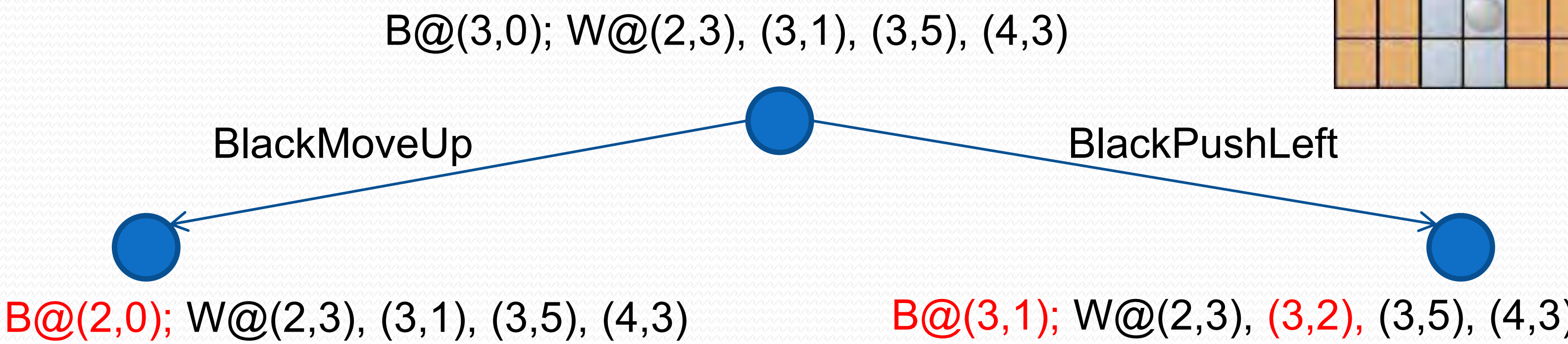
Example: Shunting Game



- A state consists of the positions of the black one and the white ones. Initially, it is
  - Black at (3,0); Whites at (2,3), (3,1), (3,5), (4,3)

Example: Shunting Game

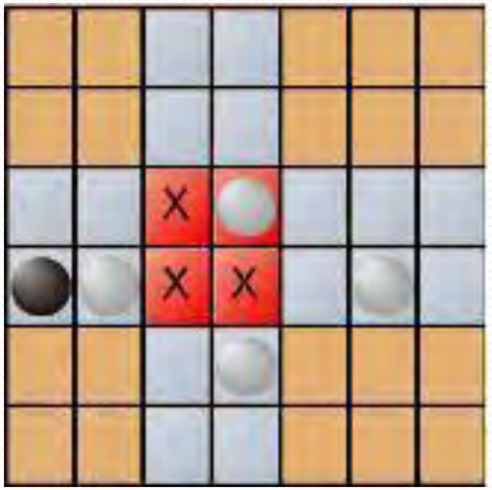




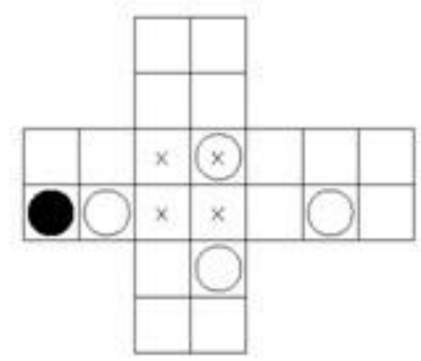
- A transition is caused by the movement of the black.

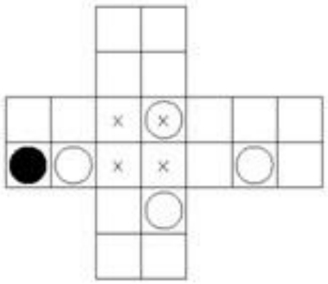
Reachability Analysis

- Goal: to determine whether there is a reachable state such that certain condition is satisfied.
  - e.g., searching for a state such that the white ones are at (2,2), (2,3),(3,2), (3,3)

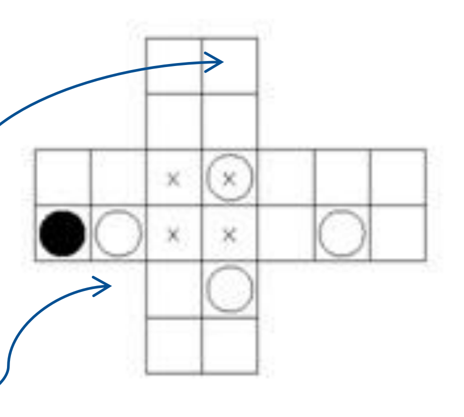


```
//The following are constants of the shunting game
#define M 7;
#define N 6;
#define o -1; //off board
#define a 1;  // available
#define w 0;  // white occupied
//       col number:  0 1 2 3 4 5 6
var board[N][M] = [o,o,a,a,o,o,o, //0 row number
                   o,o,a,a,o,o,o, //1
                   a,a,a,w,a,a,a, //2
                   a,w,a,a,a,w,a, //3
                   o,o,a,w,o,o,o, //4
                   o,o,a,a,o,o,o]; //5
// Black position:
var r = 3; var c = 0; work = 0;

Game = [work <= 10]okay -> ([r-1>=0]MoveUp [] [r-2>=0]PushUp
                    [] [r+1<N]MoveDown [] [r+2<N]PushDown
                    [] [c-1>=0]MoveLeft [] [c-2>=0]PushLeft
                    [] [c+1<M]MoveRight [] [c+2<M]PushRight)
        [] [work > 10]overworked -> Skip;
```





```
MoveUp = [board[r-1][c]==a]go_up{r=r-1} -> Game;
PushUp = [board[r-2][c]==a && board[r-1][c]==w]
      push_up{board[r-2][c]=w; board[r-1][c]=a; r=r-1;work=work+2} -> Game;

MoveDown = [board[r+1][c]==a]go_down{r=r+1} -> Game;
PushDown = [board[r+2][c]==a && board[r+1][c]==w]
            push_down{board[r+2][c]=w; board[r+1][c]=a; r=r+1} -> Game;

MoveLeft = [board[r][c-1]==a]go_left{c=c-1} -> Game;
PushLeft = [board[r][c-2]==a && board[r][c-1]==w]
            push_left{board[r][c-2]=w; board[r][c-1]=a; c=c-1} -> Game;

MoveRight = [board[r][c+1]==a]go_right{c=c+1} -> Game;
PushRight = [board[r][c+2]==a && board[r][c+1]==w]
            push_right{board[r][c+2]=w; board[r][c+1]=a; c=c+1} -> Game;
```



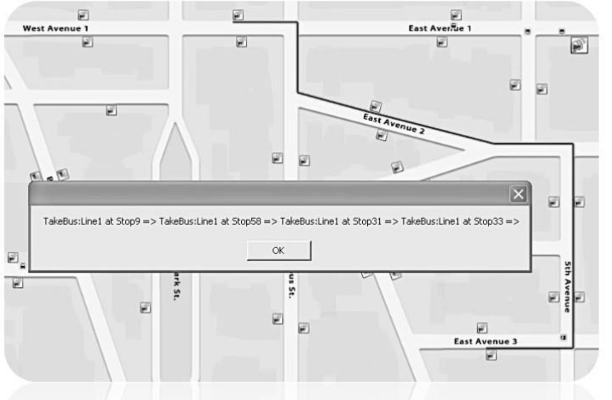
```
//one particular potential trouble position
#define trouble board[0][3] == w;

//testing if a white can be pushed to ourside
#define outside board[4][1] == w;

#assert Game reaches trouble;
#assert Game reaches outside;

#define goal board[2][2] == w && board[2][3] == w
             && board[3][2] == w  && board[3][3] == w;
#assert Game reaches goal;
#assert Game reaches goal with min(work);  //optimisation,
                                            //towards a problem solving tool

#assert Game |= [] (trouble -> !<> goal);
//show the trouble position will prevent the goal
```

Model Checking as Planning/Scheduling/Service:

Transport4You, an intelligent public transportation manager

ICSE 2011 SCORE Competition Project (PAT won FM Award)

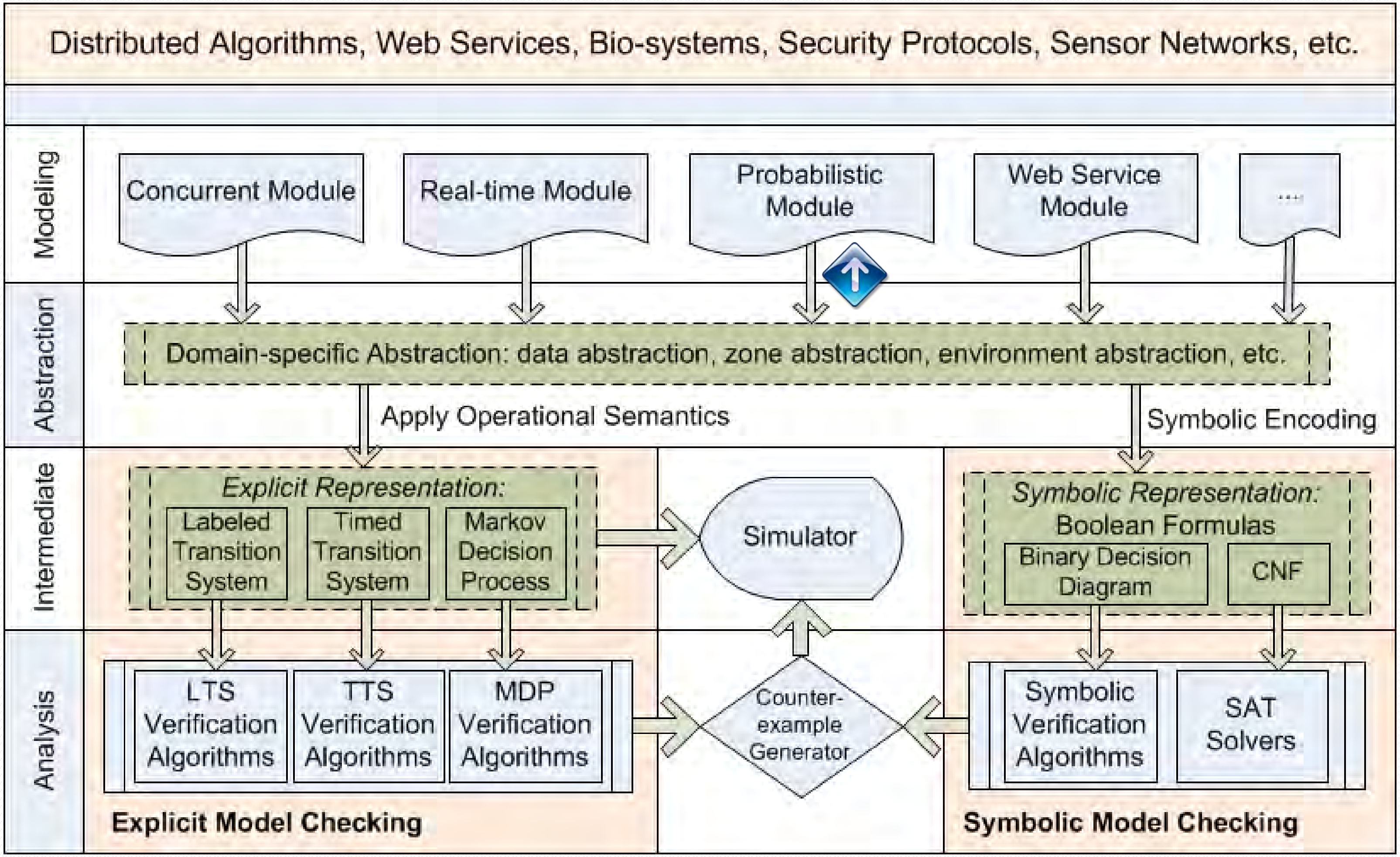
- PAT model checker is used not only as a verification tool for the system design but also as a service that computes an optimal travel plan.



- 94 teams from 48 universities in 22 countries started the competition; Two winners (Formal Methods Award and Overall Award) were selected during the conference.

  PAT student team at CS4211 won Formal Method Award in 2011 with a free trip to Hawaii!

PAT System (ICSE'08'12'13, CAV'09'12'13, FM'11'12'14, TOSEM'13, TSE'13, FMSD'13)

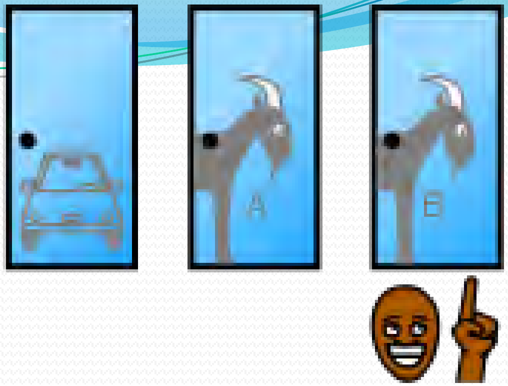


Core Members:
Sun Jun (SUTD), Liu Yang (NTU), Dong Jin Song (NUS)
Rest of the team: 20 PhD/Postdoc/RAs

Probabilistic Model Checking [CAV'12]

- 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

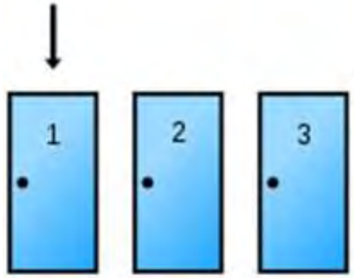


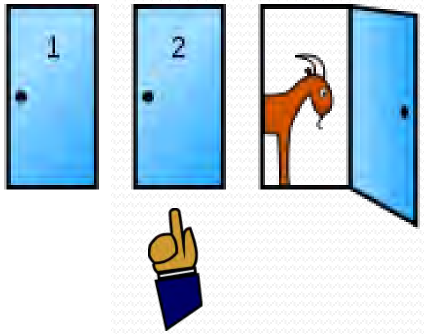
The **Monty Hall problem** is based on the American television game show *Let's Make a Deal* and named after the show's original host, Monty Hall.
The problem was originally posed in a letter by Steve Selvin to the *American Statistician* in 1975.

- 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;
```

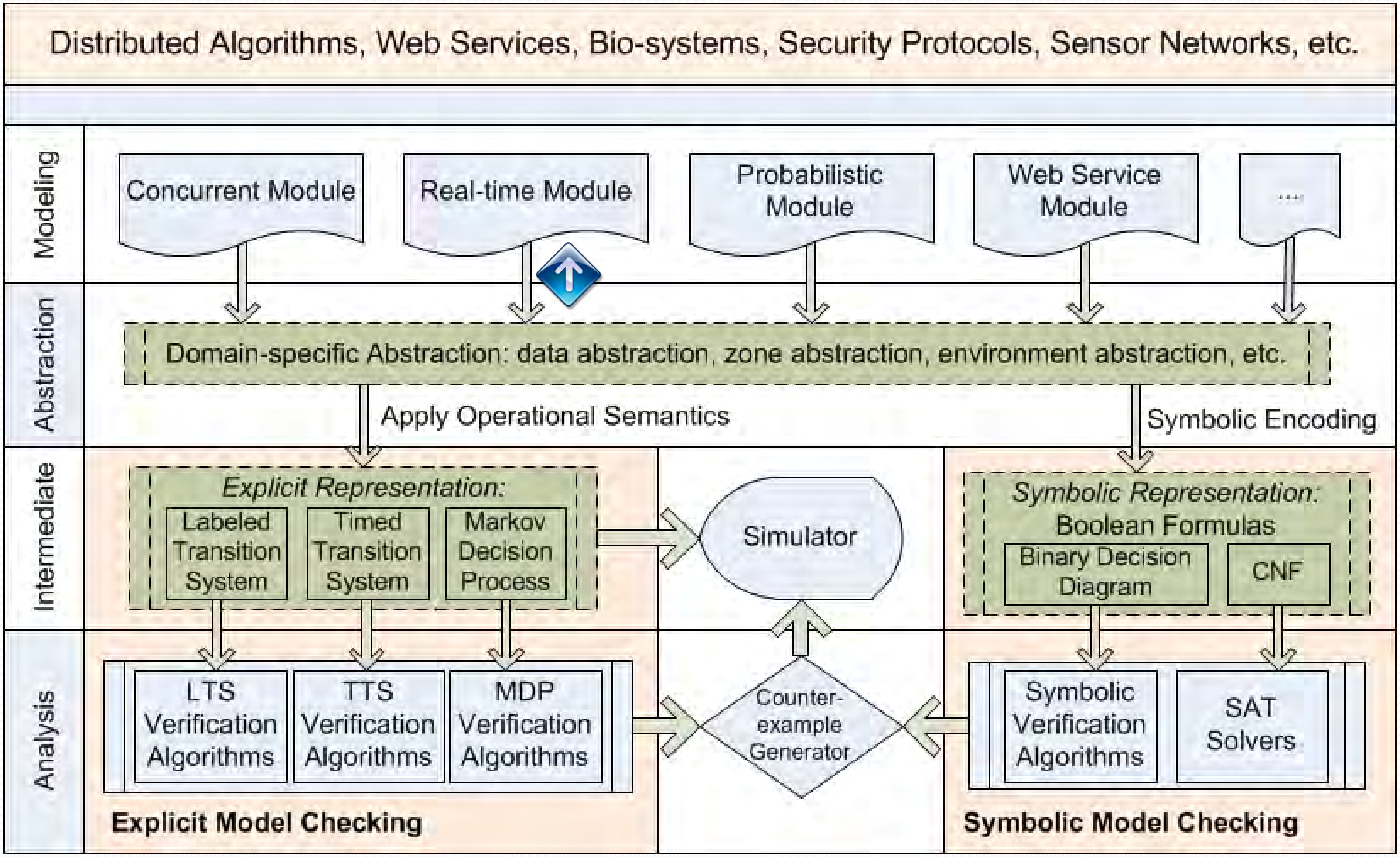




What if the host is Dishonest?

```
//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;
```

PAT System (ICSE'08'12'13, CAV'09'12'13, FM'11'12'14, TOSEM'13, TSE'13, FMSD'13)



Core Members:
Sun Jun (SUTD), Liu Yang (NTU), Dong Jin Song (NUS)
Rest of the team: 20 PhD/Postdoc/RAs

Light Control System

```
//Light Control System
var dim : {0..100}; var on = false;
channel button 0; channel dimmer 0; channel motion 0;

//the light
TurningOn = turnOn{on=true; dim=100} -> Skip;
TurningOff = turnOff{on=false; dim=0} -> Skip;

ButtonPushing = button?1 ->
       atomic{if (dim > 0) { TurningOff } else { TurningOn }};

DimChange = dimmer?n -> atomic{setdim{dim = n} -> Skip}};

ControlledLight = (ButtonPushing [] DimChange); ControlledLight;
```

Motion Detector and Controller

```
//the motion detector
NoUser = move -> motion!1 -> User [] nomove -> Wait[1]; NoUser;
User = nomove -> motion!0 -> NoUser [] move -> Wait[1]; User;
MotionDetector = NoUser;
//the room controller
Ready = motion?1 -> button!1 -> On;
Regular = adjust -> (sunshine -> dimmer!50 -> Regular
                  [] cloudy -> dimmer!70 -> Regular
                  [] night -> dimmer!100 -> Regular );
On = Regular interrupt motion?0 -> OnAgain;
OnAgain = (motion?1 -> On) timeout[20] Off;
Off = button!1 ->> Ready;
Controller = Ready;
//the system
System = MotionDetector ||| ControlledLight ||| Controller;
```

Reasoning

```
//the system
System = MotionDetector ||| ControlledLight ||| Controller;

#assert System deadlockfree;

#define inv ((on && dim > 0) || (!on && dim == 0));
#assert System |= []inv;

#assert System |= [](turnOn -> <> turnOff);

#define test1  (!on && dim == 50);
#assert System reaches test1;

#define test2  (on && dim == 50);
#assert System reaches test2;
```

Current Status of PAT

- PAT is available at http://pat.comp.nus.edu.sg
- 1Million lines of C# code, 20 verification systems with 200+ build in examples, 100+ publications (CAV, FM, ICSE, ASE, TSE, TOSEM …).
- Used as an educational tool in many universities.
- Attracted 3000+ registered users in the last 5 years from 800+ organizations in 71 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

Commercialised in multiple countries, esp. in Japan, thanks to CATS!

PAT is used by other research institutes

- Model Checking Linearizability (with Wei Chen at MSR-Asia)
- Sensor networks systems
- Context-aware systems for health care systems
- Embedded systems
  - High Speed Train
  - Singapore Car IU
- Software Models
  - UML (or FUML) diagram
  - Software Architecture Description Language
- Multi-agent Systems
- Timed Transition Systems
- Finance Software
- Security
  - Security Protocols
  - Trust Based Platform

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)
- Y. Liu, W. Chen, Y. A. Liu, J. Sun, S. Zhang and J. S. Dong.**Verifying Linearizability via Optimized Refinement Checking.** IEEE Transactions on Software Engineering (TSE), (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**. ICFEM'11, pages 147-162, Durham, UK, 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**. FM'11, pages 432 - 447, Lero, Limerick, Ireland, June 20 - 24, 2011.
- Y. Liu, J. Sun and J. S. Dong. **Analyzing Hierarchical Complex Real-time Systems**. *FSE '10*, 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.** CAV '09, 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**, ICSE'08, 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
- 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 and J. S. Dong, **Synthesis of Distributed Processes from Scenario-based Specifications.** FM'05, pp 415-431, Newcastle, UK. 2005.
- S. C. Qin, J. S. Dong and W. N. Chin. **A Semantic Foundation of TCOZ in Unifying Theory of Programming.** FM'03. 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.

Institute for Integrated Intelligent Systems (Cybersecurity)

Jin-Song Dong

Professor and Director

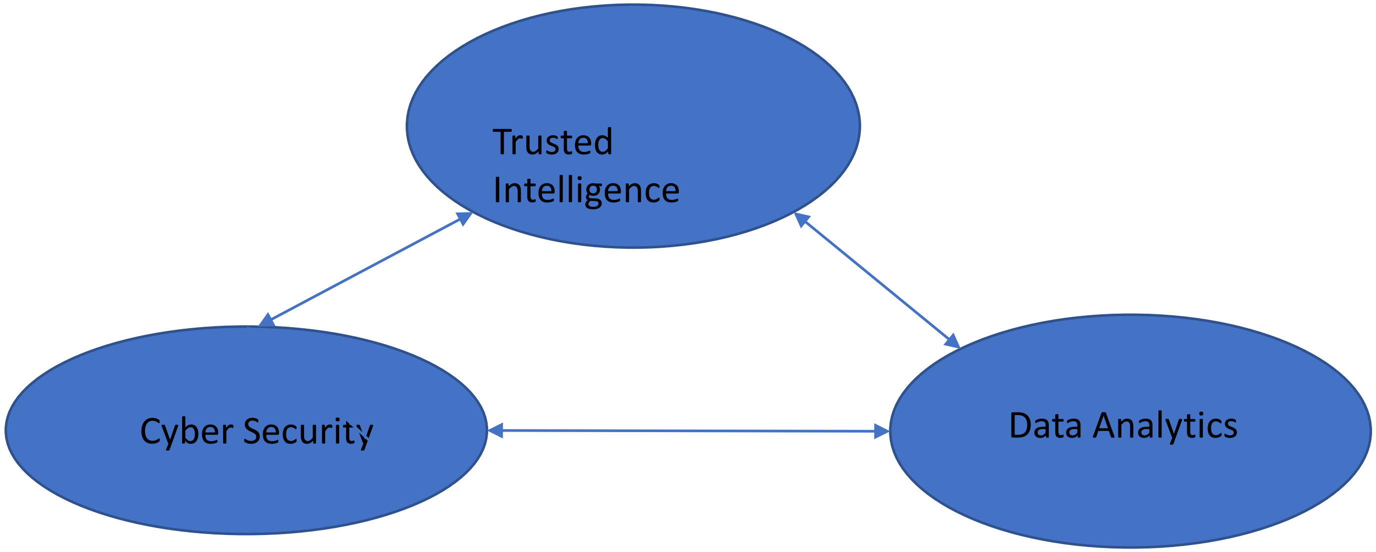
Institute for Integrated Intelligent Systems (IIIS), Griffith University
j.dong@griffith.edu.au

IIIS has about 40 professors and 70 PhD students across ICT, Eng, Sci and Biz and has expertise in AI, Computer Vision, Robotic, Data Analytics, Cybersecurity, Formal Verification

IIIS Strategic Research Teams/Labs

- AI and Semantic Technologies Leader: Prof. Kewen Wang
- Big Data Prof. Bela Stantic
- Enterprise Architecture A/Prof. Peter Bernus
- Environmental Informatics & Image Processing Prof. Yongsheng Gao
- Idea Lab & Information Systems Leigh Ellen Potter
- Interactive Robotics & Networked Control Systems Prof. Vlad Estivill-Castro
- Logic & Optimisation Prof. Abdul Sattar
- Network Security A/Prof. Vallipuram Muthukkumarasamy
- Speech Processing and Deep Learning Prof. Kuldip Paliwal

IIIS Long term focus direction/goals



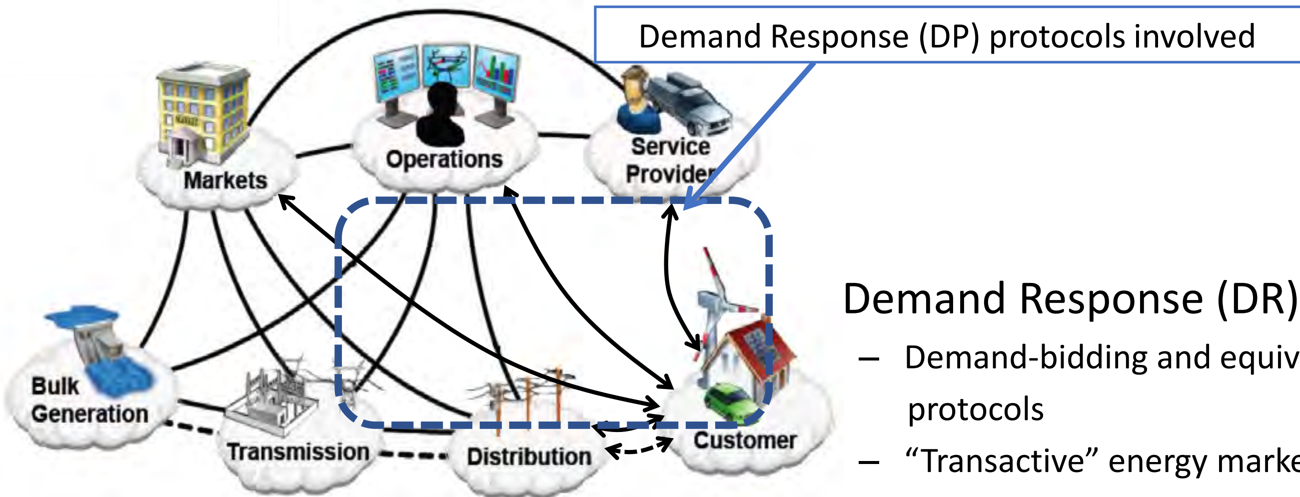
Current International Cybersecurity Collaboration Projects

- Singapore-UK joint cyber security project on smart grid security and privacy (with Prof. Andrew Martin from Oxford University).
- Trustworthy systems from untrusted Components ($6M)
- Securify: A Compositional Approach of Building Security Verified System “($6M)
- Singtel-NUS Cyber Security joint lab ($43M).

Security and Privacy in Smart Grid Systems: Countermeasure and Formal Verification

Key PI & Collaborators	Name	Designation	Institution
Co-PI	Dr. Jin Song Dong	Professor	NUS -> Griffith University
Co-PI	Dr. Andrew Martin	Professor	University of Oxford
Collaborator	Dr. Guangdong Bai	Research Fellow	NUS (Computing)

New challenges in smart grid: **Bi-directional communication**



Demand Response (DR)

- Demand-bidding and equivalent protocols
- "Transactive" energy markets
- Closed-loop feedback control

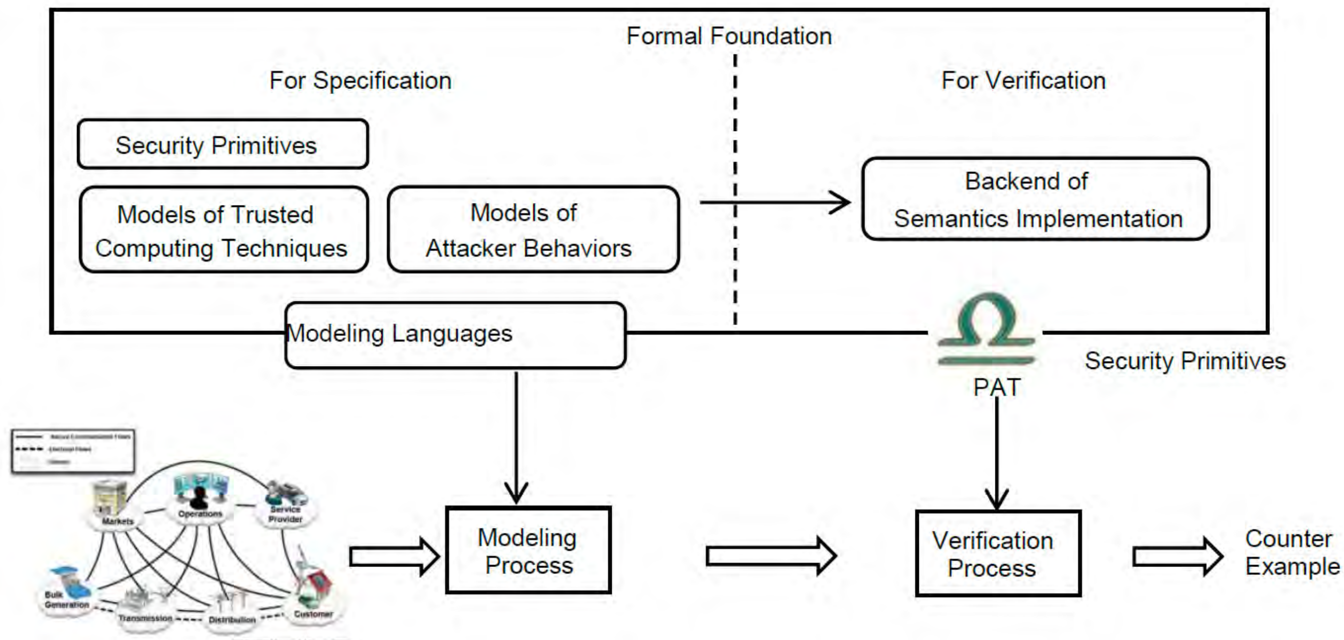
- Requires full bi-directional communication:
  - Consumers ↔ Demand Manager (DM)

Security and privacy threats and goals in DR protocols

- Security Threats
  - Modification or falsification of data
- Privacy Threats
  - Inference of private information

- Security goals
  - For consumers: verify the **authenticity** and **integrity** of all DR events and bid notifications
  - For DR manager: verify the **authenticity** and **integrity** of all DR bids
- Privacy goals
  - Untrusted entities must not be able to **link** DR bids to individual consumers.
  - Untrusted entities must not be able to **infer private informatio**n about individual consumers from the DR system.

Formal analysis of DR protocols



Trustworthy systems from untrusted component ($6M)

Binary analysis, Binary hardening, System Security

Best paper awards – FSE09, USENIX Security 07, ICECCS 14
Tool deployment – BitBlaze, JSlice.
Many related past grants including DIRP.

Abhik Roychoudhury (WP1)

Liang Zhenkai (WP2)

Formal Verification.

Originator of research effort in PAT model checker

Dong Jin Song (WP3)

System Security, Data Protection

Best paper award ICECCS 14,
Deployed past research to Google+ and Chrome.

Prateek Saxena (WP4)

Systems security

Grant from T-labs, FSTD

Roland Yap

Network Security, Applied Cryptography

Grant from TDSI, MINDEF

Chang Ee-chien

Collaborator (1)	Dawn Song	UC Berkeley	Professor
Collaborator (2)	Ruby B. Lee	Princeton	Professor
Collaborator (3)	Alessandro Orso	Georgia Tech	Professor
Collaborator (4)	Andrew Martin	Oxford	Professor

TSUNAMi

Formalize Protocols Using PAT

Modelling language: CSP#

Security properties :

- Secrecy:

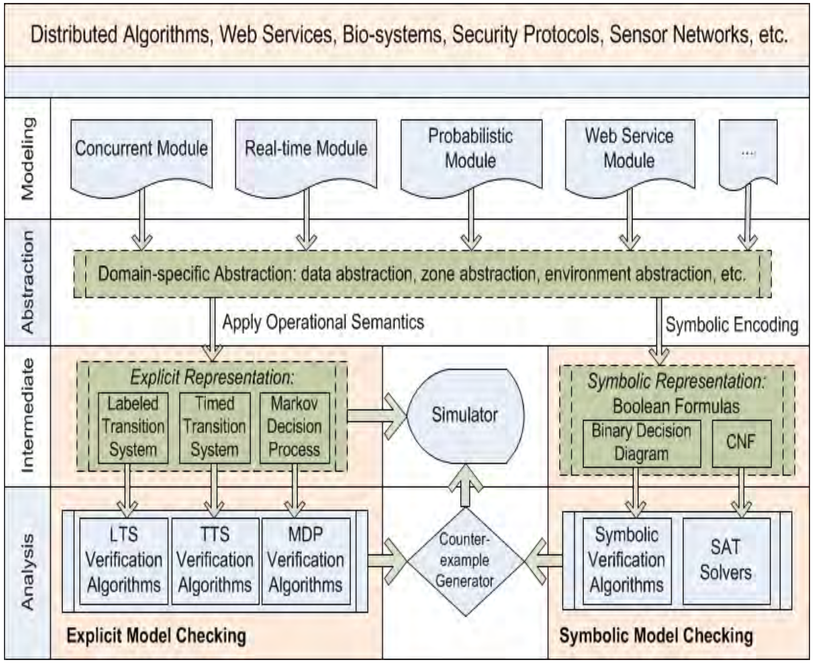
assertions on reachability

- Authentication:

LTL (Linear Temporal Logic) formula

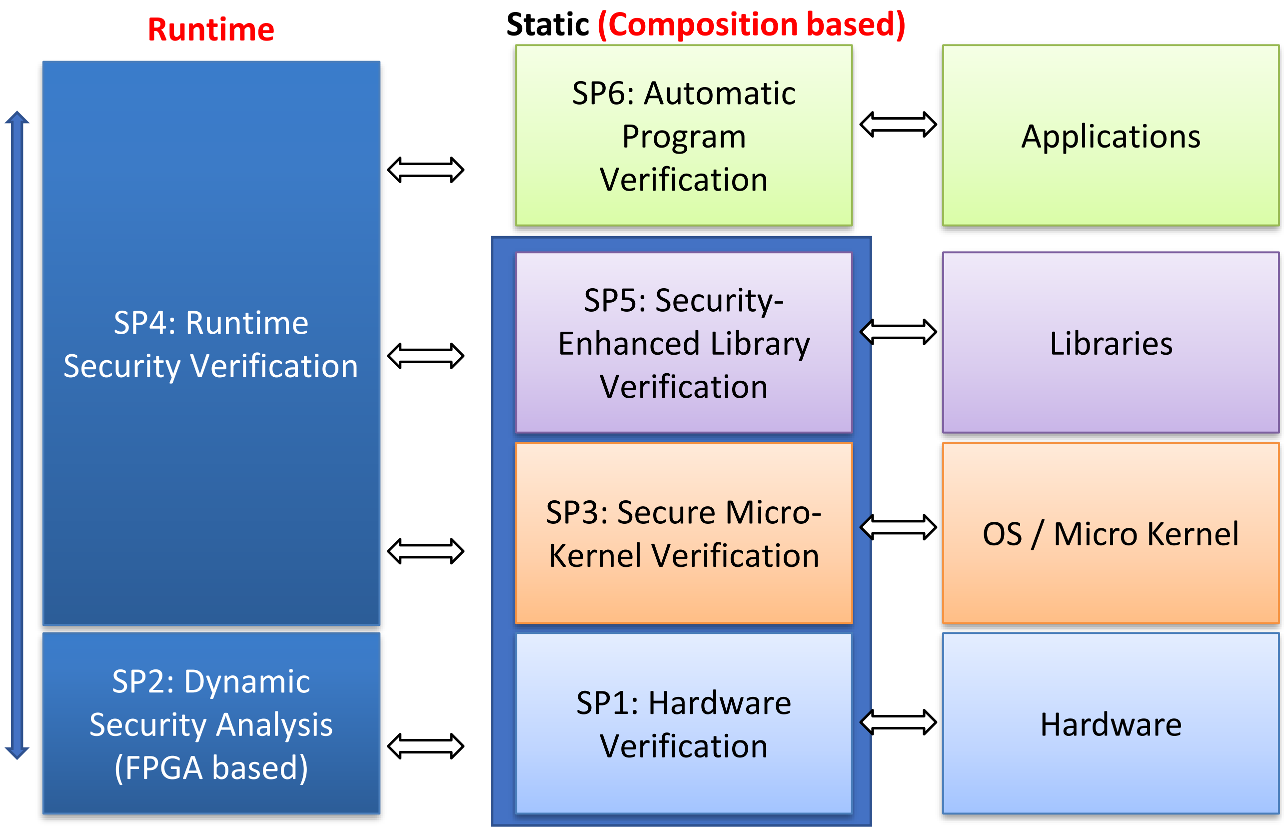
Modelling tasks:

1. message (crypto primitives)
2. protocol behavior
3. attacker behavior
4. attacker knowledge



TSUNAMi

Securify: Compositional Approach to Security Verification ($6M)



Securify Team Members

SP8: Compositional Security Reasoning with Untrusted Components

SP7: Model-based Secure Code Generation

SP6: Automatic Program Verification

SP4: Runtime Security Verification

SP5: Security-Enhanced Library Verification

SP3: Secure Micro-Kernel Verification

SP2: Hardware-aided Dynamic Security Analysis

SP1: Hardware Verification

ST Electronics Info-Security

WINCOR NIXDORF EXPERIENCE MEETS VISION.

Srikanthan Thambipilla
i
NTU


SP6, 8

Liu Yang
NTU


David Basin
ETH

SP4, 7

Alwen Tiu
NTU


Kenny Paterson
RHUL

SP5

Chin Wei Ngan
NUS


Sjouke Mauw
Uni. of Lux

SP3, 7

Sun Jun
SUTD


Luke Ong
Oxford

SP3, 5

Dong Jin Song
Griffith


Wei Zhang
HKUST

SP1, 2

- In this example, we model a railway control system to automatically control trains passing a critical point such a bridge. The idea is to use a computer to guide trains from several tracks crossing a single bridge instead of building many bridges. Obviously, a safety-property of such a system is to avoid the situation where more than one train are crossing the bridge at the same time.

- Intuitively, when a train, Train-i , approaches the bridge it sends a signal to the controller within a certain distance. If the bridge is occupied the controller immediately sends a stop signal stop-i to prevent the train from entering the bridge. Otherwise, if the approaching train does not receive a stop signal within 10 time units, it will start to cross the bridge within 20 time units (but it will take at least 10 time units for a train to enter the bridge). The crossing train is assumed to leave the bridge within 3 to 5 time units; a stopped train will slow down and eventually stop after some delay. When the bridge is free again and the controller signals (by sending go-i) the rst train in the waiting list to cross.