Pervasive Model Checking

-An introduction to PAT

LIU, Yang

liuyang@comp.nus.edu.sg

Senior Research Scientist
Temasek Laboratories
National University of Singapore

Outline

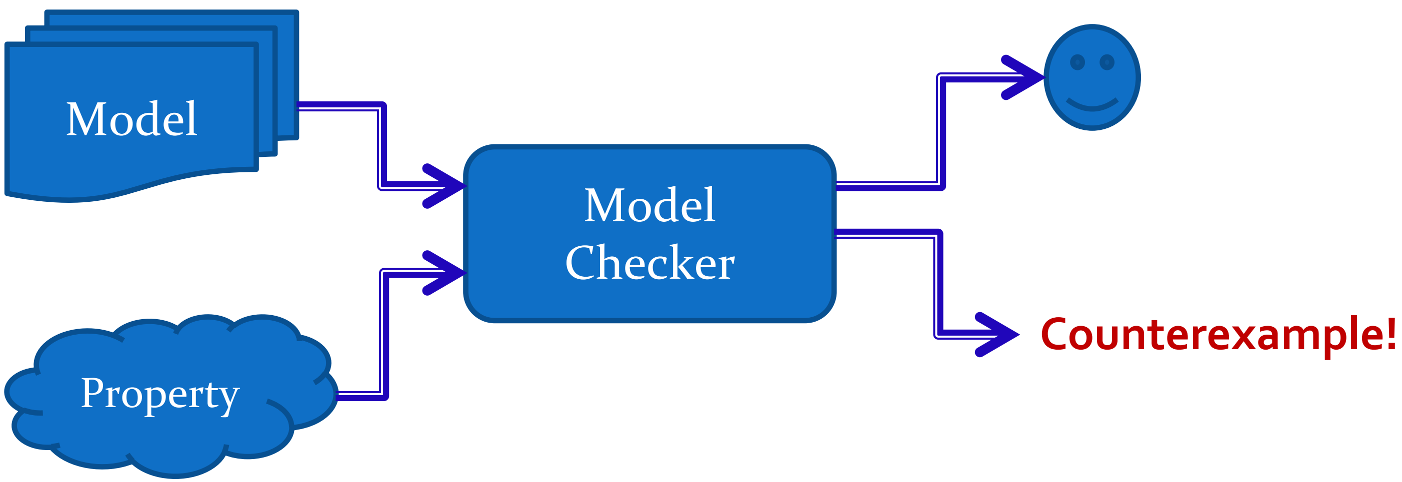
- Motivation and Background
- Introduction to PAT tool
  - in model checking techniques
    - Case study in handling fairness assumptions
  - in applying model checking
    - Case study in verify the correctness of concurrent objects
- On-going Works
- Vision in model checking

Motivations

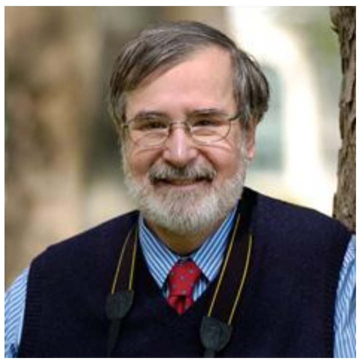
- Concurrent and real-time systems are everywhere
  - Internet, embedded systems, mobile devices…
- The design of concurrent and real-time systems are notoriously difficult problems
  - concurrent executions, shared resources, timing factors and so on
  - Increasing complexity
- Correctness is the one of the key problems
  - Mission critical systems accept no failure:
    Intel Pentium II bug, Ariane 5 failure, Therac-25 accident
- Principal validation methods
  - Simulation and Testing,
  - Deductive verification and
  - **Model checking**

Model Checking

- Determining whether a model satisfies a property by the means of exhaustive searching (fully automatic)



Model Checking Works!



SLAM

Challenges in Applying MC

- Using existing model checkers
  - Steep learning curve
  - Existing model checkers may be inefficient or insufficient
    - E.g. multi-party barrier synchronization is difficult in SPIN
  - How to express the properties
- Extending existing model checkers
  - Model checker's code is complicated
- Developing a new model checkers
  - Complicated functions:
    - language parsing, system simulation, verification algorithms, state reduction techniques and counterexample generation and display
  - Decades of efforts to build a solid model checker

PAT (Process Analysis Toolkit)

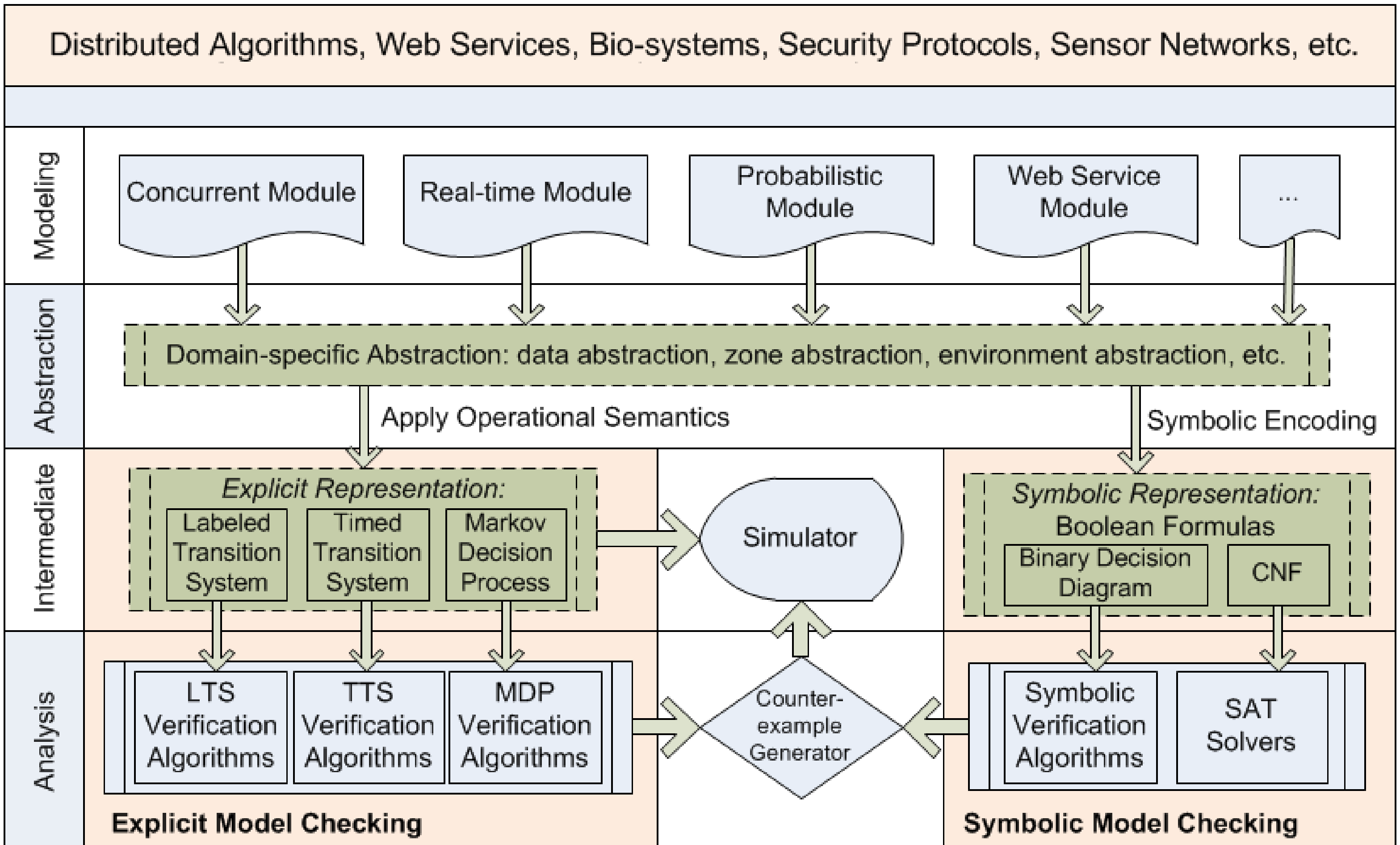
- An self-contained framework to support the development of formal verification tools
  - A wide range of systems
    - Concurrent, real-time and probabilistic systems
  - Extensible architecture
  - Modular design
    - 10+ Modules for different application domains
  - Various model checking techniques
    - Explicit model checking
    - Symbolic model checking
    - Assume-guarantee model checking
    - …

www.patroot.com

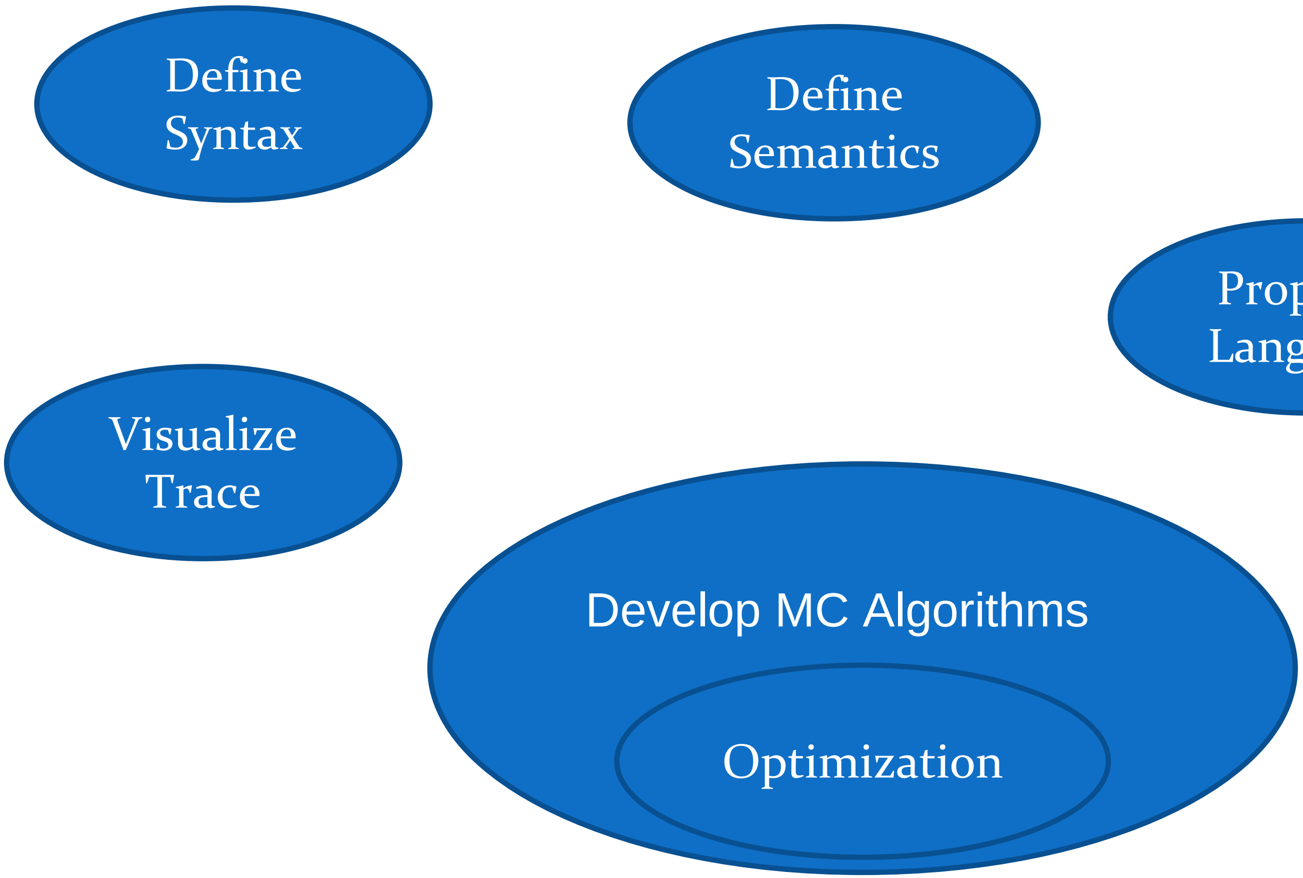
PAT (Process Analysis Toolkit)

www.patroot.com

- An self-contained framework to support the development of formal verification tools [ICSE 08, CAV 09, ATVA 10, ISSRE 11, CAV 12]



Build a Model Checker with PAT



The Current Status

- 5 Years Development
  - National University of Singapore
  - Singapore University of Technology and Design
- PAT research team (~ 30 persons):
  - 3 Faculty + 8 post doc + 15 ph.d. + 5 RA
- 1 Million lines of code, 10+ modules with 200+ build in examples
- Attracted more than 1900 registered users in the last 5 years from more than 400 organizations, e.g. Microsoft, HP, ST Elec, Oxford Univ., … Sony, Hitachi, Canon.
- Used as an educational tool in many universities.
- Japanese PAT user group formed in Sep 2009.

Research Contributions in MC

- **Featured modeling languages proposed** [TASE 09, ICFEM 09-b, ICFEM 11-b]

Concurrency + Real-time + Probability + Hierarchy

- **Novel MC algorithms developed**
  - Fast LTL model checking with fairness assumption [ICFEM 09, CAV 2009]
    - Multi-core version [ICFEM 10]
  - Fast trace refinement checking [Isola 08, icfem 12]
- **Novel MC techniques developed**
  - Real-time abstraction techniques [ICFEM 09-b, TOSEM 11, FM 12]
    - Zone abstraction, Non-zeno behaviors , BDD
  - Develop different reduction techniques
    - Symmetry reduction [FM 11]
    - Process counter abstraction [FM 09]
    - (Dynamic/compositional) partial order reduction [ICFEM 11-d]
  - Symbolic model checking libraries for hierarchical systems [ASE 11]
  - Compositional verification:
    - Assume-guarantee reasoning techniques [ATVA 11, FM 12]

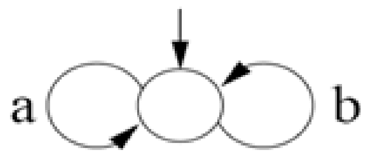
Case Study: Handling Fairness Assumptions [ICFEM 08, TASE 09-a]

- Fairness is important in concurrent systems to rule out un-realistic system behaviors
  - Enabled processes/choices can not be infinitely ignored

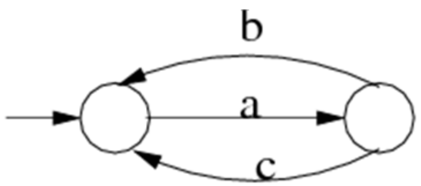
- Examples:
  - Peterson’s mutual exclusion protocol
    - weak fairness
  - Token circulation or leader election in network rings
    - Strong global fairness

Various Fairness Assumptions

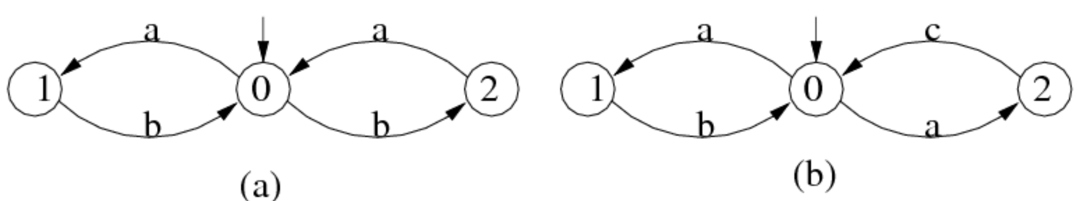
- Weak fairness
  - if an event eventually becomes enabled forever, infinitely many occurrences of the event must be observed.



- Strong fairness
  - if an event is infinitely often enabled (or in other words, repeatedly enabled), infinitely many occurrences of the event must be observed.

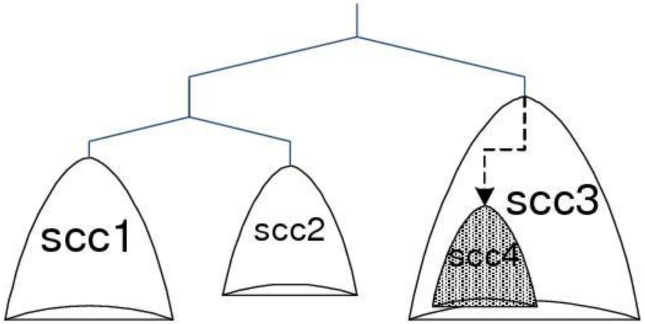


- Global fairness
  - If a *step* is infinitely often enabled, it must be taken infinitely



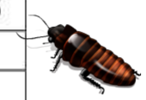
Verification under Fairness

- Previous Approaches
  - Naive: specify the fairness as part of the property
  - Different algorithms for different fairness are also available
- Our approach
  - Variant of Tarjan's SCC searching algorithm
    - Check whether the counterexample satisfy the constraints
  - Generic and Handling all types of fairness
    - weak fairness: SCC search
    - strong fairness: strongly connected sub-graph search
    - global fairness = terminal SCC search
  - Flexible
    - Fairness on individual events
  - Efficient
    - Linear to the number of edges
  - A parallel version for multi-core CPU
    - Scalable to N-cores



Experiment Results

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



Model Checking Applications using PAT

- Model Checking **Leariability** via Refinement [FM 09, TSE12]
  - Model Checking a Lazy Concurrent List-Based Set Algorithm [SSIRI 10-b]
- Analyzing **Multi-agent Systems** with Probabilistic Model Checking Approach [ICSE 12, PRIMA 12]
- An Automatic Approach to Model Checking **UML State Machines** [SSIRI 10-a]
- Verification of **Population Ring Protocols** in PAT [TASE 09-a]
- **Model Checking a Model Checker**: A Code Contract Combined Approach [ICFEM 10-a]

PAT Application Modules

- NesC module [ICFEM 11-c, Sensys 11]
  - Bug detected for Trickle algorithm
    - a code propagation algorithm which is intended to reduce network traffic
- Web service module [APSEC 10, ICFEM 11-d]
- Security protocol module [FCS 12]
- Stateflow module [STTT 12]
- Event recording automata module [ATVA 11]

Case Study: Verify the Correctness of Concurrent Objects [FM 09, SSIRI 10-2, TSE12]

- Concurrent objects (shared queue, stacks) are hard to design correctly
  - Exclusive access (correctness) vs. maximum interleaving (performance)

- Linearizability is an accepted correctness criterion for shared objects.
  - A shared object is linearizable if each operation on the object can be understood as occurring instantaneously at some point, (a.k.a. *linearization point)*

p0:	$W_{inv}(x,1)$		$W_{res}(x)$		$R_{inv}(y)$		$R_{res}(y,2)$	
p1:		$W_{inv}(y,2)$		$W_{res}(y)$		$R_{inv}(x)$		$R_{res}(x,1)$

p0:	$W(x,1)$		$R(y,2)$	
p1:		$W(y,2)$		$R(x,1)$

- Automatic verification of linearizability is challenging
  - Rely on the knowledge of linearization points
  - Linearization points are hard to be statically determined

Linearizability

- *Trace* σ is linearizable if there exists a sequential permutation π of σ such that
  - 1) for each object o_i, $\pi|_{oi}$ is a legal sequential history (i.e. π respects the sequential specification of the objects), and
  - 2) if $op1 <_\sigma op2$, then $op1 <_\pi op2$ (i.e., π respects the run-time ordering of operations).
- Examples

po: $W_{inv}(x,1)$ $W_{res}(x)$ $R_{inv}(y)$ $R_{res}(y,2)$
p1: $W_{inv}(y,2)$ $W_{res}(y)$ $R_{inv}(x)$ $R_{res}(x,1)$

po: $W_{inv}(x,1)$ $W_{res}(x)$ $R_{inv}(y)$ $R_{res}(y,2)$
p1: $W_{inv}(y,2)$ $W_{res}(y)$ $R_{inv}(x)$ $R_{res}(x,1)$

po: $W_{inv}(x,1)$ $W_{res}(x)$ $R_{inv}(y)$ $R_{res}(y,0)$
p1: $W_{inv}(y,2)$ $W_{res}(y)$ $R_{inv}(x)$ $R_{res}(x,1)$

Stack Example

Algorithm 3.3 Concurrent stack implementation

type $Node = \{val : T; next : Node\}$;
shared $Node\ H := null$;

Procedure push

```
n := new Node();
n.val := v;
repeat
    ss := H;
    n.next := ss;
until CAS(H, ss, n)
return
```

Procedure pop

```
repeat
    ss := H;
    if ss = null then
        return empty
    end if
    ssn := n.next;
    lv := ss.val;
until CAS(H, ss, n)
return lv
```

Create Specification Model

- Specify each operation *op* of a shared object *o* on a process p_i using three atomic steps:
  - the invocation action $inv(op)_i$,
  - the linearization action $lin(op)_i$, and (Invisible event)
  - the response action $res(op, resp)_i$.
- Event-base formalism: CSP#
- Is linearizable!

$$
\begin{aligned}
PushA(i) &= push\_inv.i \rightarrow \tau\{if(S < SIZE)\{S = S + 1;\ \}\ T_i = S;\ \} \\
&\qquad\qquad \rightarrow push\_res.i.T_i \rightarrow Skip; \\
PopA(i) &= pop\_inv.i \rightarrow \tau\{if(S > 0)\{S = S - 1;\ \}\ T_i = S;\ \} \\
&\qquad\qquad \rightarrow pop\_res.i.T_i \rightarrow Skip; \\
ProcessA(i) &= (PushA(i) \,\Box\, PopA(i));\ ProcessA(i); \\
StackA() &= ProcessA(0) \,|||\, \ldots \,|||\, ProcessA(N);
\end{aligned}
$$

Create Implementation

- Consider the implementment of object *o*.
  - The visible events of *impl* are also those *inv*(*op*)$_i$'s and *res*(*op*, *resp*)$_i$'s.

$$
\begin{aligned}
Push(i) &= push\_inv.i \rightarrow l_1 \rightarrow l_2 \rightarrow PushLoop(i);\\
PushLoop(i) &= l_4\{T_i = H;\} \rightarrow l_5 \rightarrow (l_6 \rightarrow PushLoop(i) \lhd T_i \neq H \rhd\\
&\qquad\qquad (l_6\{if(H < SIZE)\{H = H + 1;\} T_i = H;\} \rightarrow\\
&\qquad\qquad\qquad push\_res.i.T_i \rightarrow Skip));\\
Pop(i) &= pop\_inv.i \rightarrow PopLoop(i);\\
PopLoop(i) &= l_2\{T_i = H;\} \rightarrow (l_3 \rightarrow pop\_res.i.0 \rightarrow Skip \lhd T_i \neq 0 \rhd\\
&\qquad\qquad l_3 \rightarrow l_5 \rightarrow l_6 \rightarrow (l_7 \rightarrow PopLoop(i) \lhd T_i \neq H \rhd\\
&\qquad\qquad\qquad l_7\{H = H - 1;\ T_i = H;\} \rightarrow pop\_res.i.T_i \rightarrow Skip));\\
Process(i) &= (Push(i) \, \Box \, Pop(i));\ Process(i);\\
Stack() &= Process(0) \,|||\, \ldots \,|||\, Process(N);
\end{aligned}
$$

- Is linearizable?

Linearizability as Refinement

Theorem 1. *All traces of L_{im} are linearizable iff $L_{im} \sqsupseteq_T L_{sp}$.*

Theorem 2. *Let L'_{sp} and L'_{im} be the specification and implementation LTSs such that linearization events are specified as $lin(op, resp)_i$ and are the only visible events. If $L'_{im} \sqsupseteq_T L'_{sp}$, then the implementation is linearizable. Conversely, if the implementation is linearizable, and* it can be shown that no other actions in the implementation can be linearization actions, *then $L'_{im} \sqsupseteq_T L'_{sp}$.*

Algorithm and results

- A novel refinement checking algorithm to verify linearizability automatically
  - partial order reduction
  - symmetry reduction
- Substantial Experiments:
  - Stack,
  - Queue,
  - K-valued Register
  - Mailbox algorithm
  - SNZI.

Algorithm 4.1 A linearizability checking algorithm

Procedure Linearizability (L_{im}, L_{sp})

$checked := \emptyset;$
$pending.push((init_{im}, \tau^*(init_{sp})));$
while $pending \neq \emptyset$ **do**
    $(s, X) := pending.pop();$
    $checked := checked \cup (s, X);$
    if $X = \emptyset$ **then**
        return $false$
    end if
    for all $(s', X') \in next(s, X)$ **do**
        if $(s, X) \notin checked$ **then**
            $pending.push((s', X'));$
        end if
    end for
end while
return $true$

Outline

- Motivation and Background
- Research contributions
- On-going Works
  - MC in New domains
  - MC techniques
  - Others
- Vision in model checking

Ongoing Works – New Domains

- Web Service (Orc language /BPEL language) (in implementation)
- Sensor networks system written in NesC (in optimization)
  - Distributed algorithms
- Context-aware systems (in design phase)
- UML (or FUML) diagram (in design phase)
  - Merlion 2012 funding on "Software Verification from Design to Implementation"
- Software Architecture Description Language (in implementation)
  - Event Grammar/ADL
- Verification of C# Programs (in progress)
- Multi-agent Systems (in progress)
- Timed Transition Systems (in progress)

Temasek Research Fellow Project

- Research and Development in the Formal Verification of System Design and Implementation.
  - Principal Investigator
  - S$1,150,000.
  - 3 RA, 3 Post Docs
- Security protocol verification
  - Get models from implementations
  - Trusted Platform Module
- Assembly Code Verification (in implementation)
  - Model checking assembly code
  - Model Abstraction from assembly code
- Automatic generation of correct implementation (in implementation)
  - Code generation from PAT models to C/C++ code for embedded system or mobile applications (in testing phase)
  - Security protocol code generation

Ongoing Works – Model Checking Techniques

- Automatic symmetry detection and reduction (in design phase).
- Probabilistic Model Checking [ICFEM 2011] (in testing phase) / Statistical Model Checking (in implementation phase)
- Symbolic modeling checking library (using BDD) for hierarchical systems
  - CSP/LTS (in testing phase) [ASE 2011]
  - TA/RTS (in implementation phase) [FM12]
- Assume-guarantee verification for real-time [ATVA 11, FM12] (in optimization) and probabilistic systems (in implementation)
- Multi-core model checking algorithms [ICFEM 09-a] and swarm verification techniques (in implementation)

Vision: *Pervasive Model Checking*

- We not only aim to develop a verifier, but rather to build a framework for realizing system verification techniques
  - **Model checking techniques**: EMC, SMC (SAT/BDD/SMT), A-G, CEGAR
  - **Different domains**: web services, sensor network, distributed algorithms, security, multi-agent systems, bio…
  - **Different semantics model**: LTS/TTS/MDP/TA/PTA…
  - **Different algorithms**: LTL/Refinement/Multi-core..
  - **Different reduction techniques**: POR, Symmetry detection and reduction, process counter abstraction
  - **Applications of model checking**: testing, planning, SE (reliability/product line/software-eco systems…)

Thank You!

Questions?

References (1)

- [ICSE 12] Songzheng Song, Jianye Hao, Yang Liu, Jun Sun, Ho-Fung Leung, and Jin Song Dong. Analyzing Multi-agent Systems with Probabilistic Model Checking Approach. The 34th International Conference on Software Engineering, 2012. (Accepted)
- [FCS 12] Luu Anh Tuan, Jun Sun, Yang Liu, Jin Song Dong, Xiaohong Li, and Quan Thanh Tho. SEVE: Automatic Tool for Verification of Security Protocols. Frontiers of Computer Science, Special Issue on Formal Engineering Method, 6(1):57-75, 2012.
- [ASE 11] Truong Khanh Nguyen, Jun Sun, Yang Liu and Jin Song Dong. A Symbolic Model Checking Framework for Hierarchical Systems. The 26th IEEE/ACM International Conference On Automated Software Engineering, pages 633-636, Lawrence, Kan., USA, Nov 6 - 11, 2011.
- [TOSEM 11] 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 (Accepted)
- [Sensys 11] Manchun Zheng, Jun Sun, David Sanán, Yang Liu, Jin Song Dong, Yu Gu. Demo: Towards Bug-free Implementations for Wireless Sensor Networks. The 9th ACM Conference on Embedded Networked Sensor Systems, pages 407-408, Seattle, WA, USA, Nov 1 - 4, 2011.
- [ISSRE 11] 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, pages 190-199, Hiroshima, Japan, Nov 29 - Dec 2, 2011.
- [ATVA 11] Shang-Wei Lin, Etienne Andre, Jin Song Dong, Jun Sun, and Yang Liu. Efficient Algorithm for Learning Event-Recording Automata. The 9th International Symposium on Automated Technology for Verification and Analysis pages 463-472, Taipei, Taiwan, October 11 - 14, 2011.
- [ICFEM 11-a] Zhenchang Xing, Jun Sun, Yang Liu and Jin Song Dong. Differencing Labeled Transition Systems. The 13th International Conference on Formal Engineering Methods , pages 537-552, Durham, United Kingdom, October 25-28, 2011.
- [ICFEM 11-b] 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, pages 147-162, Durham, United Kingdom, October 25-28, 2011.
- [ICFEM 11-c] Manchun Zheng, Jun Sun, Yang Liu, Jin Song Dong and Yu Gu. Towards a Model Checker for NesC and Wireless Sensor Networks. The 13th International Conference on Formal Engineering Methods, pages 372-387, Durham, United Kingdom, October 25-28, 2011.
- [ICFEM 11-d] Tian Huat Tan, Yang Liu, Jun Sun and Jin Song Dong. Verification of Computation Orchestration System with Compositional Partial Order Reduction. The 13th International Conference on Formal Engineering Methods, pages 98-114, Durham, United Kingdom, October 25-28, 2011.
- [FM 11] 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, pages 432 - 447, Lero, Limerick, Ireland, June 20 - 24, 2011.
- [APSEC 10] Jun Sun, Yang Liu, Jin Song Dong, Geguang Pu and Tian Huat Tan. Model-based Methods for Linking Web Service Choreography and Orchestration. The 17th Asia Pacific Software Engineering Conference, pages 166-175, Sydney, Australia, 30 November – 3 December 2010.
- [FSE 10] Yang Liu, Jun Sun and Jin Song Dong. Analyzing Hierarchical Complex Real-time Systems. The ACM SIGSOFT International Symposium on the Foundations of Software Engineering, pages 365-366, Santa Fe, New Mexico, USA, 7-11 November 2010.
- [ICFEM 10-a] Jun Sun, Yang Liu and Bin Cheng. Model Checking a Model Checker: A Code Contract Combined Approach. The 12th International Conference on Formal Engineering Methods, pages 518-533, Shang Hai, China, 16-19 November 2010.
- [ATVA 10] Yang Liu, Jun Sun, Jin Song Dong. Developing Model Checkers Using PAT. 8th International Symposium on Automated Technology for Verification and Analysis, pages 371-377, Singapore, 2010.
- [ICFEM 10-b] Jun Sun, Songzheng Song and Yang Liu. Model Checking Hierarchical Probabilistic Systems. The 12th International Conference on Formal Engineering Methods, pages 388-403, Shang Hai, China, 16-19 November 2010.
- [ASE 10] Zhenchang Xing, Jun Sun, Yang Liu and Jin Song Dong. SpecDiff: Debugging Formal Specifications. The 25th IEEE/ACM International Conference on Automated Software Engineering, pages 353-354, Antwerp, Belgium, 20-24 September 2010.
- [SSIRI 10-1] Shao Jie Zhang, Yang Liu. An Automatic Approach to Model Checking UML State Machines. The 4th IEEE International Conference on Secure Software Integration and Reliability Improvement. pages 1-6, Singapore, June, 2010.
- [SSIRI 10-2] Shaojie Zhang and Yang Liu. Model Checking a Lazy Concurrent List-Based Set Algorithm. The 4th IEEE International Conference on Secure Software Integration and Reliability Improvement. pages 43-52, Singapore, June, 2010. **Best Paper Awards**

References (2)

- [ICFEM 09-a] Yang Liu, Jun Sun and Jin Song Dong. Scalable Multi-Core Model Checking Fairness Enhanced Systems. 11th International Conference on Formal Engineering Methods. pages 426-445, Rio de Janeiro, Brazil, December, 2009.
- [ICFEM 09-b] Jun Sun, Yang Liu, Jin Song Dong and Xian Zhang. Verifying Stateful Timed CSP using Implicit Clocks and Zone Abstraction. 11th International Conference on Formal Engineering Methods. pages 581-600, Rio de Janeiro, Brazil, December, 2009.
- [FM 09-a] Jun Sun, Yang Liu, Abhik Roychoudhury, Shanshan Liu and Jin Song Dong. Fair Model Checking with Process Counter Abstraction. The 16th International Symposium on Formal Methods. pages 123 - 139, Eindhoven, the Netherlands, November, 2009.
- [FM 09-b] Yang Liu, Wei Chen, Yanhong A. Liu and Jun Sun. Model Checking Lineariability via Refinement. The 16th International Symposium on Formal Methods. pages 321-337, Eindhoven, the Netherlands, November, 2009.
- [CAV 09] Jun Sun, Yang Liu, Jin Song Dong and Jun Pang. PAT: Towards Flexible Verification under Fairness. The 21th International Conference on Computer Aided Verification, pages 709-714, Grenoble, France, June, 2009.
- [TASE 09-a] Yang Liu, Jun Pang, Jun Sun and Jianhua Zhao. Verification of Population Ring Protocols in PAT. The 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering, pages 81 - 89, Tian Jing, China, July, 2009.
- [TASE 09-b] Jun Sun, Yang Liu, Jin Song Dong and Chun Qing Chen. Integrating Specification and Programs for System Modeling and Verification. The 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2009), pages 127 - 135, Tian Jing, China, July, 2009.
- [SEKE 09] Shao Jie Zhang, Yang Liu, Jun Sun, Jin Song Dong, Wei Chen and Yanhong A. Liu. **Formal Verification of Scalable NonZero Indicators**. The 21st International Conference on Software Engineering and Knowledge Engineering pages 406-411, Boston, Massachusetts, USA, July 1-3, 2009.
- [ICFEM 08] Jun Sun, Yang Liu, Jin Song Dong and Hai H. Wang. Specifying and Verifying Event-based Fairness Enhanced Systems. The 10th International Conference on Formal Engineering Methods (ICFEM 2008), pages 318-337, Japan, October, 2008.
- [ISoLA 08] Jun Sun, Yang Liu and Jin Song Dong. Model Checking CSP Revisited: Introducing a Process Analysis Toolkit. The Third International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, pages 307-322, Porto Sani, Greece, October 13-15, 2008.
- [TASE 08] Jun Sun, Yang Liu, Jin Song Dong and Jing Sun. Bounded Model Checking of Compositional Processes. The 2nd IEEE International Symposium on Theoretical Aspects of Software Engineering,pages 23-30, Nanjing, China, June 17-19, 2008.
- [ICSE 08] Yang Liu, Jun Sun and Jin Song Dong. An Analyzer for Extended Compositional Processes. ICSE Companion, pages 919-920, Leipzig, Germany, 2008.

Industrial Collaborations

- Security products design verification
- Verify Flash Memory Device Driver
- Japanese Industrial Workshop on 23rd Feb 2012

How to efficiently analyze MAS?

Challenge 1: The existence of multiple agents usually indicates complicated system structure

Challenge 2: Agents or environment may have random behaviors, which generates probabilistic characteristics

Extensive Simulation

- Simulating the system behaviors; quite convenient
- **Drawback**: results are usually inaccurate and some properties are not supported

Mathematical Model

- A good way to understanding the whole system and properties can be proved directly
- **Drawback:** very difficult to build a correct math model; need ingenuity

Model Checking and MAS

- Using general model checkers:
  - Cannot verify some specific properties in MAS, such as *knowledge* and *ATL*; not so convenient to build MAS systems with their languages since MAS has its own characteristics;
- Specific MAS model checkers
  - MCMAS: supports ATL; no probabilistic behaviors
  - MCK: supports probability; only supports knowledge; based on DTMC instead of MDP

Our Approach

- Designing an expressive modeling language to conveniently model MAS with probabilistic behaviors

```
Agent A {
   var state;
   ChooseAction = {1: {action=1}
                   2: {action=2}};
   Update = [action==1]{state=1}
            []
            [action==2]{state=2};
}
```

```
System = A and B and Environment;
```

- Supporting various properties in this kind of systems
  - **System level** : Reachability checking, LTL checking and reward checking are used to analyze the overall behaviors of the system;
  - **Agent level** : Knowledge reasoning is used to check agent's epistemic properties.
  - **Knowledge** reasoning in PMA

Preliminary Experiments

- Dispersion Game is the generalization of anti-coordination game to an arbitrary number of players and actions.
  - two strategies: *basic simple strategy (BSS) and extended simple strategies (ESS).*
- Two important properties of the system are considered
  - convergence $\mathcal{P}r(System \models \Diamond\Box\, MDO);$
  - convergence rate
  - average rounds to MDO
- Automatically verified using probabilistic model checking techniques
- Better understandings of the dynamics of the strategies compared with empirical evaluations in previous work
  - The system becomes more dynamic due to the increase in the number of agents, making it more difficult for the agents to coordinate their actions.
  - The local max points in terms of the average number of rounds before convergence always correspond to those cases when the convergence property holds.

Linearizability Experiments

Algorithm	Processes	Linearizable	No Reductions		With SR		With POR		With SR & POR	
			States	Time	States	Time	States	Time	States	Time
8-valued *register*	3	true	112000	2.70	57616	12.6	18885	3.29	10734	2.51
10-valued *register*	3	true	256209	5.95	131059	30.6	38402	8.22	21407	5.99
3-valued *register*	4	true	52381	2.72	10764	7.42	12028	1.63	3114	1.27
5-valued *register*	4	true	507082	22.7	96184	73.5	71021	12.7	16377	8.66
7-valued *register*	4	true	2352897	107	430595	370	245634	60.8	52908	31.4
3-valued *register*	6	true	14799133	3665	235635	3067	1649624	699	39416	447
4-valued *register*	7	true	-	-	-	-	-	-	484723	53110
stack of size 2	3	true	3590	0.26	660	0.26	2833	0.27	548	0.19
stack of size 2	4	true	112394	6.59	5557	6.36	91507	10.3	4522	5.11
stack of size 3	4	true	123190	7.49	5898	6.89	99939	11.3	4845	5.55
stack of size 4	4	true	124558	7.63	5935	6.93	101037	11.8	4879	5.59
stack of size 5	5	true	6002458	659	60124	398	4874975	1030	53670	391
stack of size 5	6	true	-	-	-	-	-	-	646665	52071
stack of size 2 (points)	3	true	535	0.06	104	0.05	198	0.05	62	0.06
stack of size 3 (points)	4	true	9165	0.31	536	0.37	2796	0.31	320	0.33
stack of size 4 (points)	5	true	190367	5.40	2495	9.13	52510	6.74	877	4.79
queue of size 3	3	true	181591	10.2	31637	14.6	76283	18.6	15267	8.45
queue of size 3	4	true	-	-	1417457	3173	-	-	773064	2068
buggy queue of size 2	3	false	86736	4.23	74920	33	74920	0.93	477	0.36
SNZI of size 2	2	true	17629	0.90	8824	1.6	6406	1.44	3461	1.03
SNZI of size 3	3	true	-	-	-	-	-	-	3524553	5224