Verification of Orchestration Systems using Compositional Partial Order Reduction

Tian Huat Tan[1], Yang Liu[1], Jun Sun[2] and Jin Song Dong[1]

[1]National University of Singapore
[2]Singapore University of Technology and Design

Outline

- Introduction of Orc Language
- Compositional Partial Order Reduction
- PAT – Process Analysis Toolkit
- Conclusion and Future Works

Orc Language

- Proposed by Jayadev Misra at University of Texas at Austin (UT Austin) in 2004.
- Orc is a task orchestration language, which can be used as:
  - Executable specification language
  - General purpose programming language

Overview of Orc Language

- Site – Basic service or component
  - Operator sites: +, −, *, &&, ||, < =
    - 1+1→(+)(1,1)
  - Timer Sites
    - Rtimer(5000)
  - External Sites
    - Google ("Orc")

Structure of Orc Expression

- Simple: just a site call, eg. *CNN(d)*
  - Publishes the value returned by the site.

- Composition of two Orc expressions:

  f and g can be simple expression like CNN(d), or composite expression like CNN(d) | BBC(d), x is a variable to be bounded.

  f | g   Parallel Combinator
  f >x>g   Sequential Combinator
  f <x< g   Pruning Combinator
  f ; g   Otherwise Combinator

- Orc is about the theory of combinators.

Parallel Combinator: $f \mid g$

- Evaluate f and g independently.
- Publish all values from both.
- No direct communication or interaction between f and g.

Example: $CNN(d) \mid BBC(d)$

Calls both *CNN* and *BBC* simultaneously. Publishes values returned by both sites. (0, 1 or 2 values)

Pruning Combinator: *g <x< f*

For some values published by *g* do *f* .

- Evaluate *g* and *f* in parallel.
  - Site calls in *g* that need *x* are suspended.
  - see (*M*() | *N*(*x*)) <*x*< *f*
- When *f* returns a (first) value:
  - Bind the value to *x*.
  - Terminate *g*.
  - Resume suspended calls in *f*.
- Values published by (*f* <*x*< g) are the values returned by f.
- Example:

  Email(address, x) <x< (CNN(d) | BBC(d))

Notation: *f* <<*g* for *f* <*x*<*g*, if *x* is unused in *g*.

Challenges of Verifying Orc

- State explosion problem
  - Many normal operations such as declaration of variable, or application of function are designed to run in parallel.
  - Example, in this simple expression

    val a=2+2

    1+1+a

    $\Rightarrow$ ((+)((+)(1,1),a)) < a < (+)(2,2)

    (+)(1,1) and (+)(2,2) are running in parallel.

Observation 1 - Independency

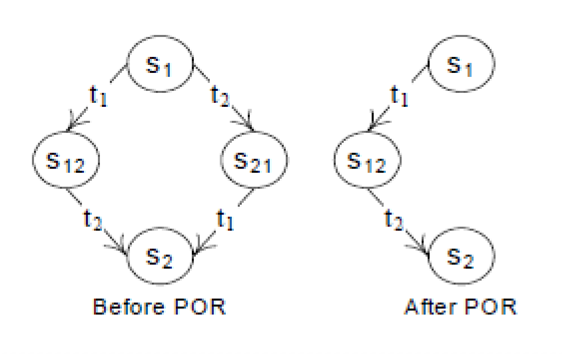
- Nature of Sites
  - Stateless sites – Sites that do not have any states
    - e.g. Plus site (+),
      (+)(1,2)=3
  - Stateful sites – Sites that have states, stored in some *state objects*
    - e.g. Buffer site, *Buffer* (State Object: a FIFO queue)
      (*userdb.put("item1")* < *userdb* < *Buffer()*)

- Many site calls are independent – their order of execution is irrelevant
  - Any two stateless site calls are independent
  - Any two stateful site calls are independent iff they do not share common state object.

(Solution: Partial Order Reduction)

Partial Order Reduction (POR)

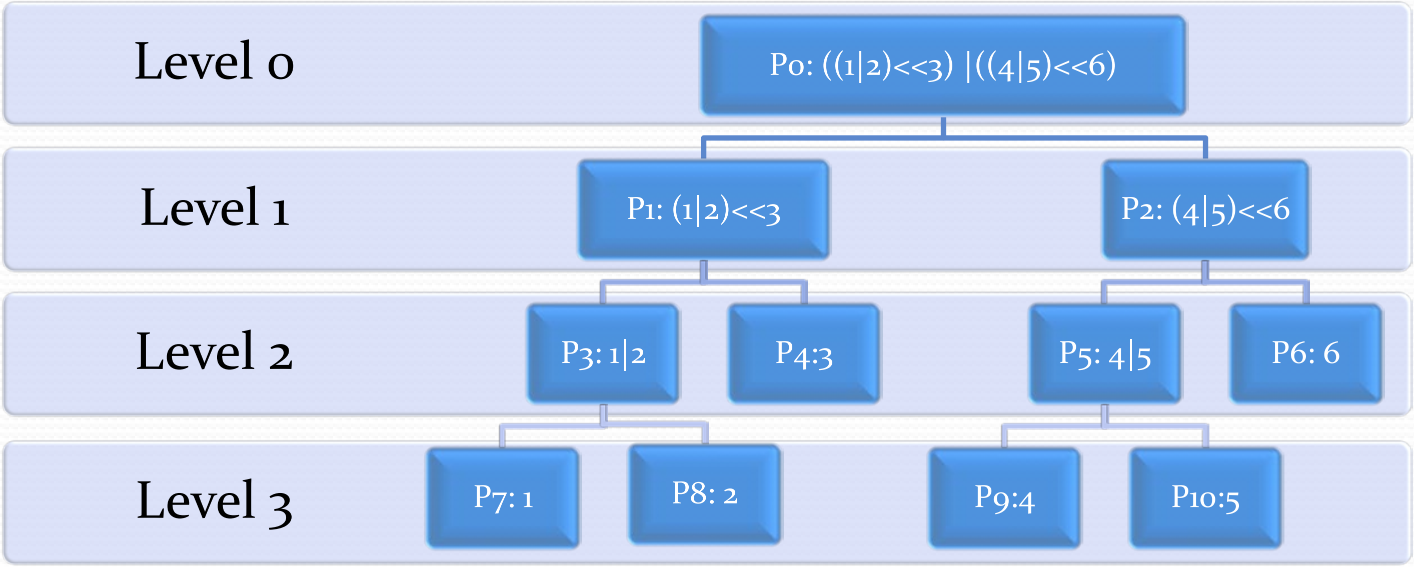
- Reduce the number of possible orderings for checking for certain properties.



- Algorithms
  - Identifying a subset of outgoing transitions of a state, call ample set, that is sufficient for verification.

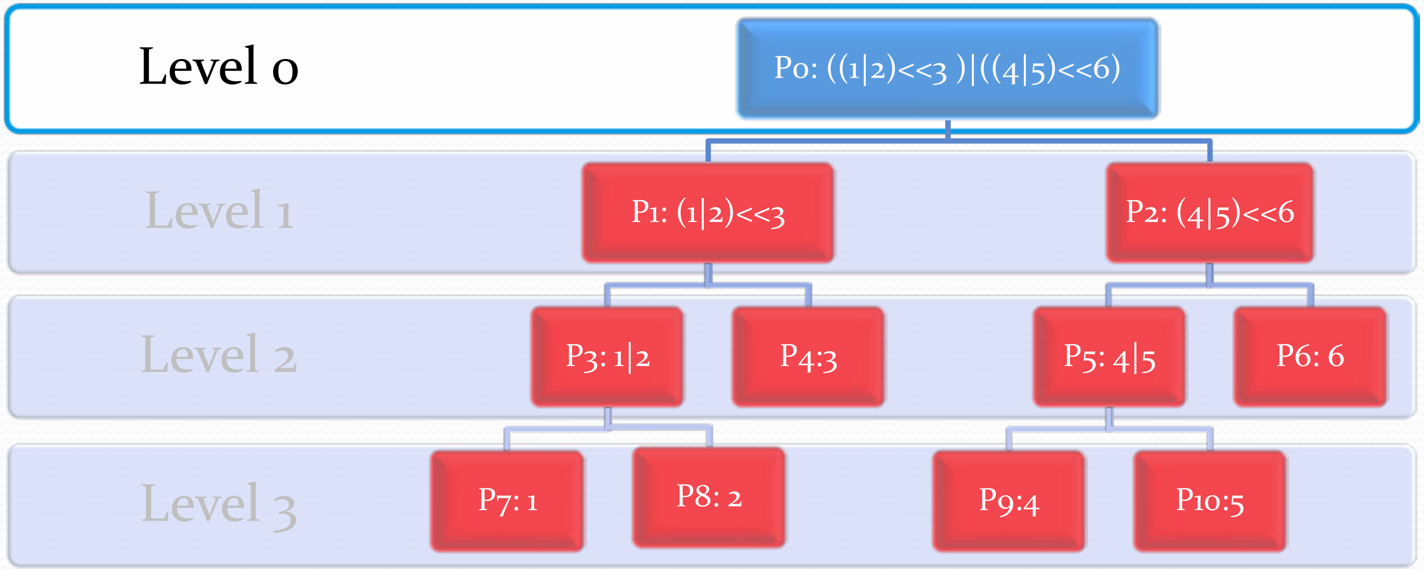
Observation 2 - Hierarchical Concurrent Processes(HCP)

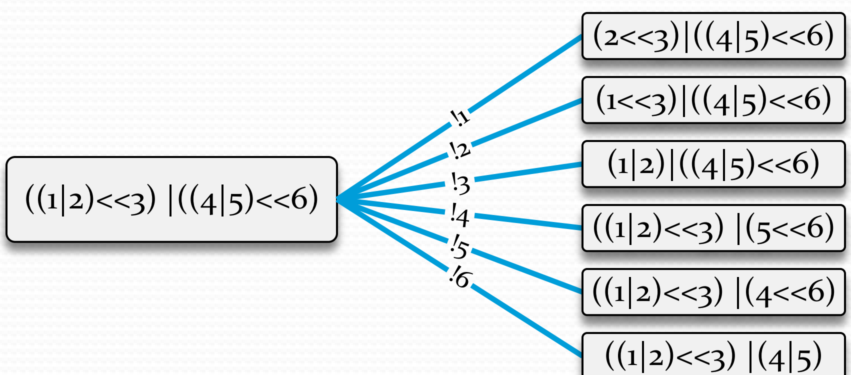
- The structure of Orc program can be viewed as hierarchical concurrent processes
  - e.g. *((1|2) << 3) | ((4|5) << 6)*



No Partial Order Reduction

HCP Graph





Labeled Transition System:

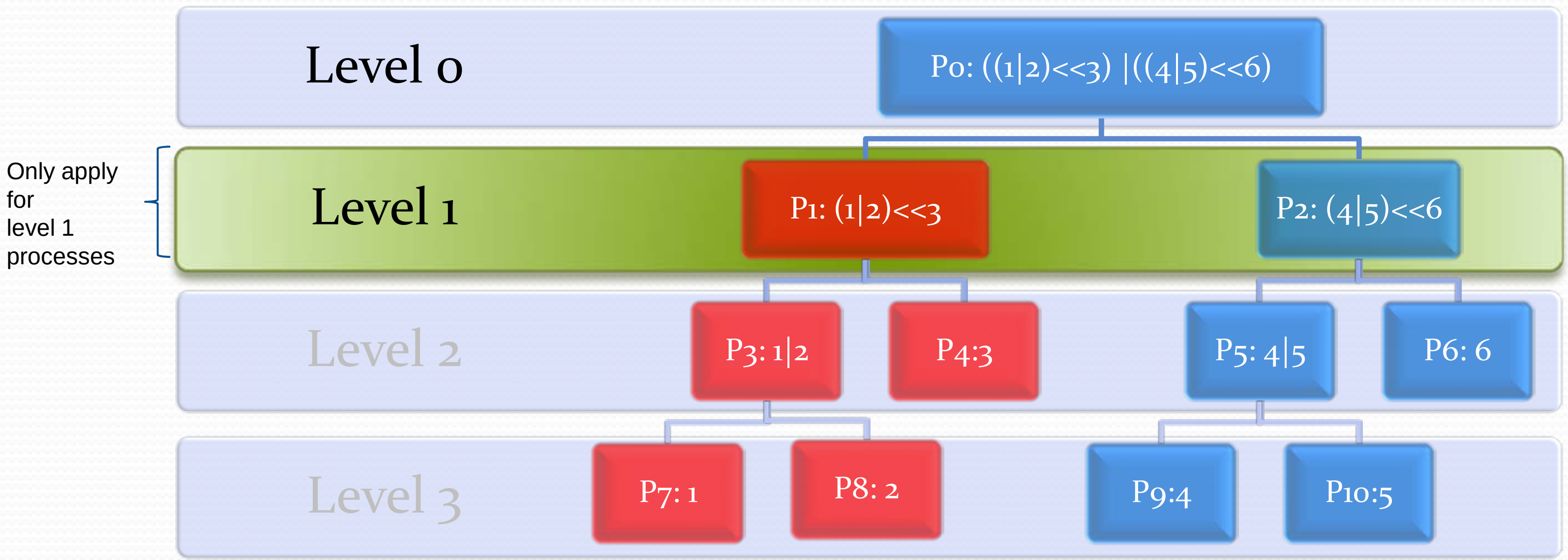
Number of possible transitions from P_0 is 6.

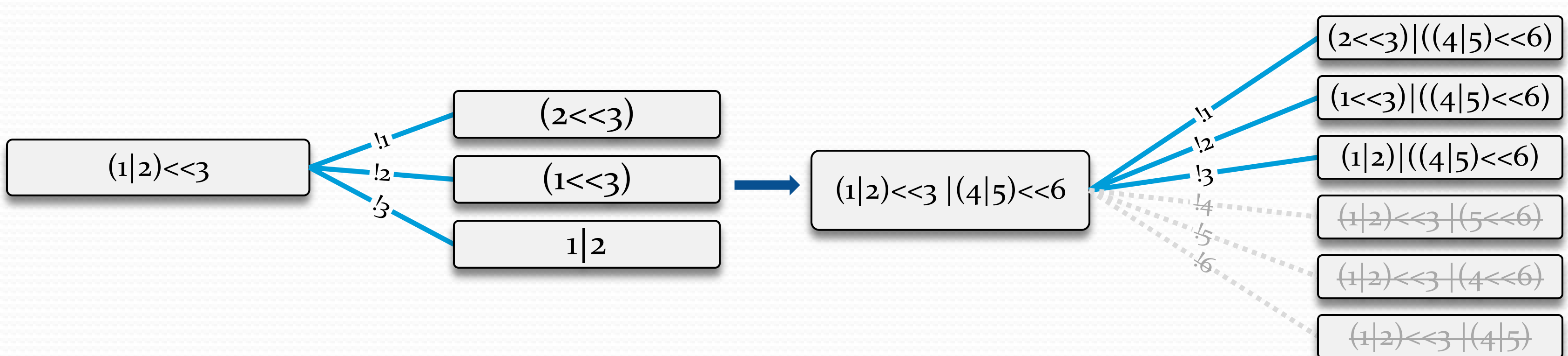
Classic Partial Order Reduction - Algorithms

Given a state s, how to find an ample set,

1. Checking four conditions for each level 1 process.
2. If any of the processes satisfies all four conditions, the transitions from that process (which is a subset of all possible transitions) could be used as the ample set.
3. Otherwise, all possible transitions are taken. (Same as no partial order reduction)

Classic Partial Order Reduction





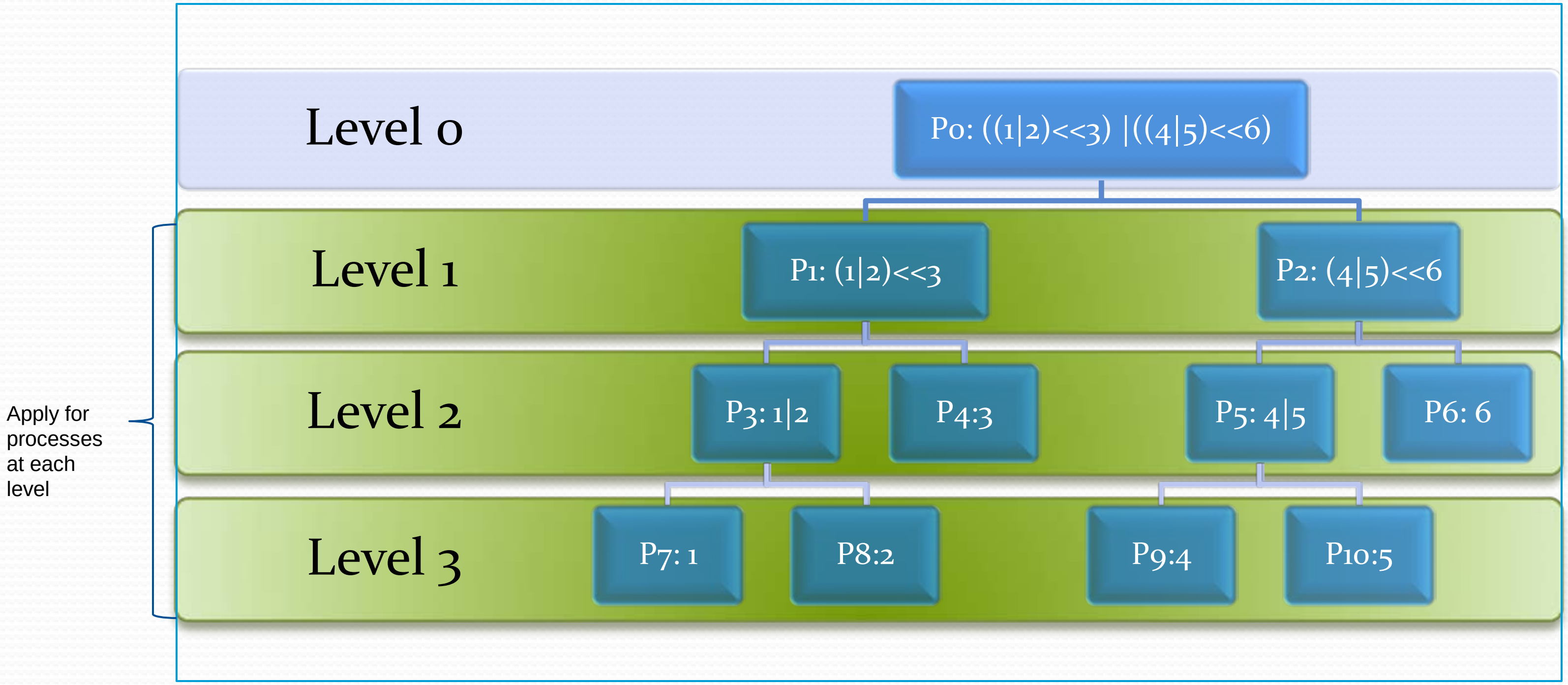
A solution for Hierarchical Concurrent Processes - Compositional Partial Order Reduction

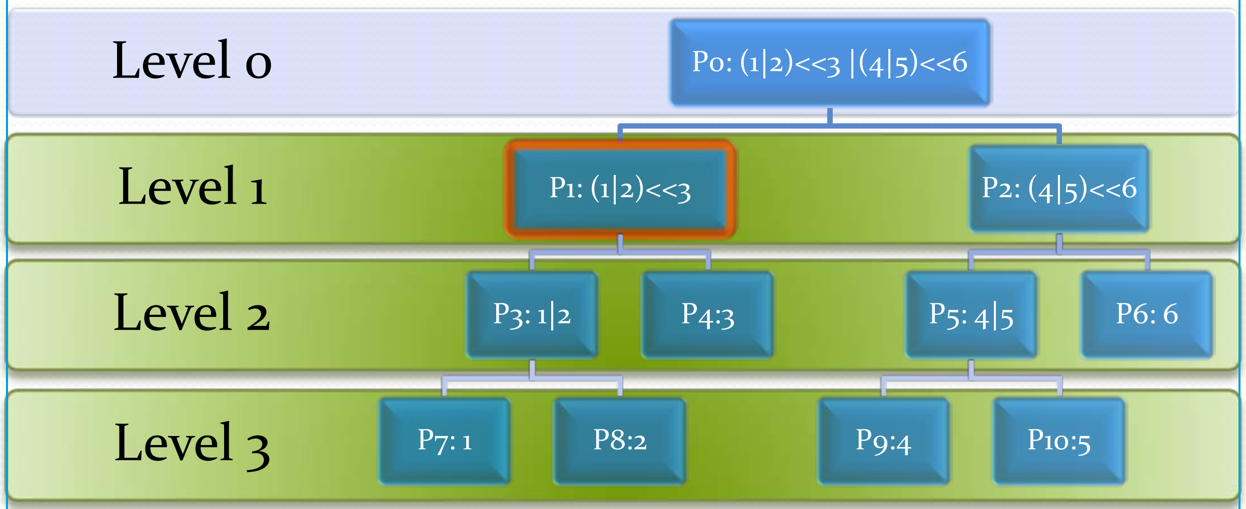
How to find an ample set, given a state s.

1. Categorized four conditions into two global transitions and two local transitions.
2. Checking two local transitions recursively for the processes at each level and collect all potential ample sets that satisfies the conditions.
3. Filter the collected potential ample sets with two global transitions.
4. Returned one of the ample sets that satisfies all four conditions.
5. Otherwise, all the possible transitions are taken. (Same as no partial order reduction)

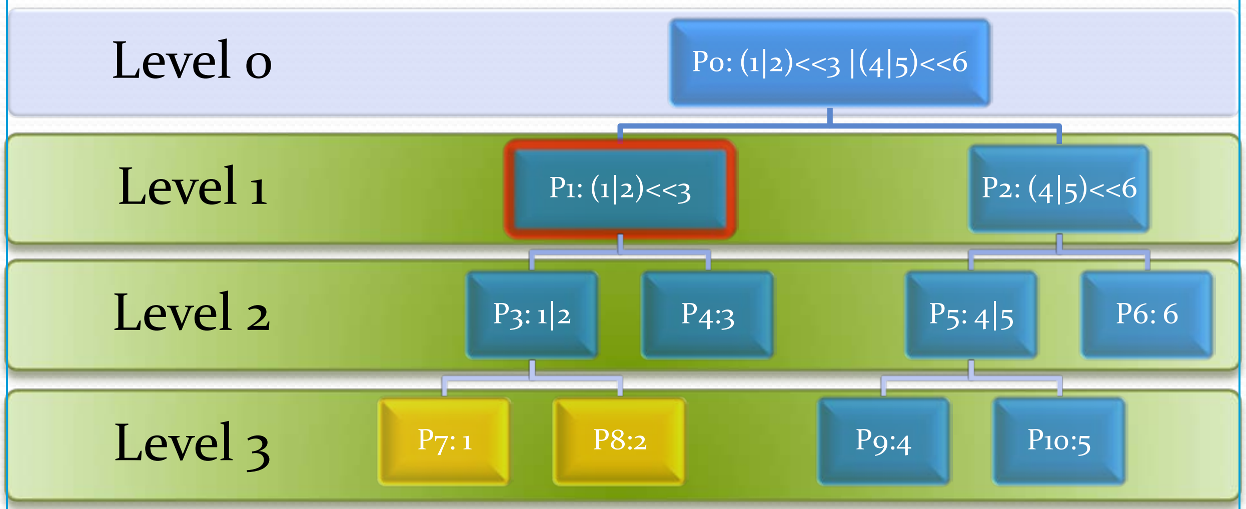
A solution for Hierarchical Concurrent Processes -

Compositional Partial Order Reduction





Processes in Level 1 is analyzed one by one.
Assume P_1 is analyzed first.



After choosing P1, traverses until the process at the bottom level.

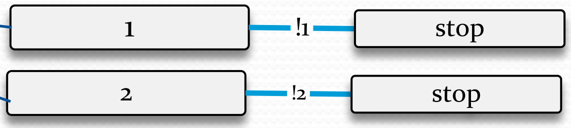
Notation

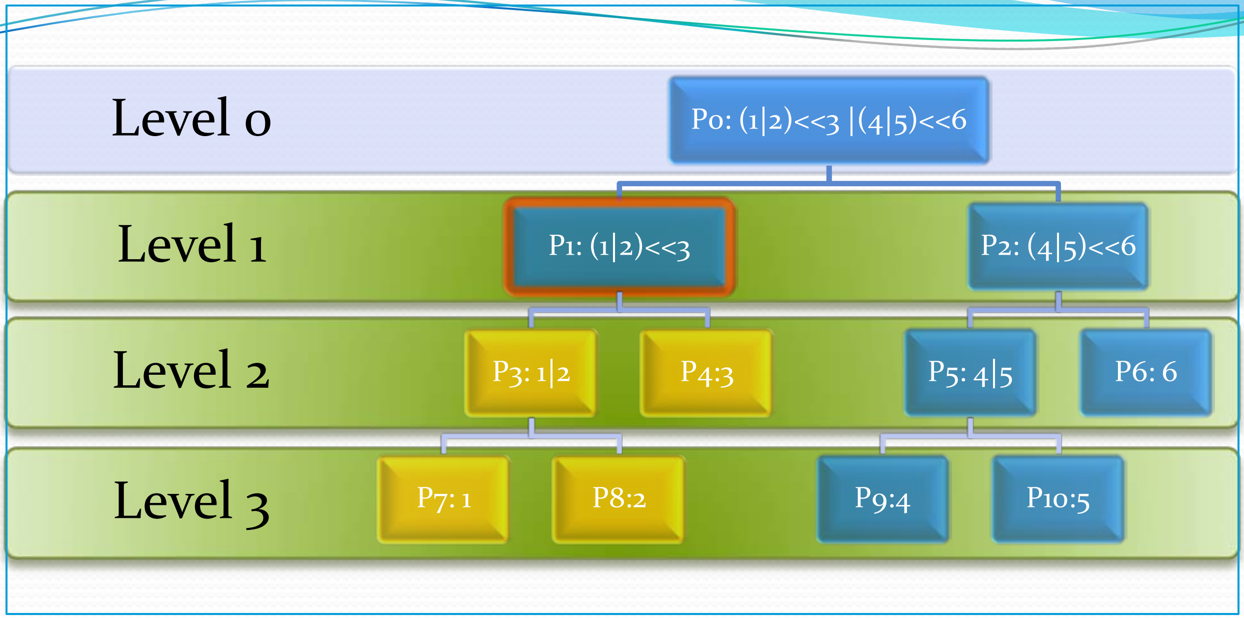
Potential Ample sets, denoted as amples , are set of the ample set that satisfies local conditions.

amples={$ample_1$, $ample_2$,…}

For amples(P_7) = {enable(P_7)} = {{(!1,stop)}}

For amples(P_8) = {enable(P_8)} = {{(!2,stop)}}





$$\frac{1 \xrightarrow{!1} stop}{1 \mid 2 \xrightarrow{!1} 2} \qquad \frac{2 \xrightarrow{!2} stop}{1 \mid 2 \xrightarrow{!2} 1}$$

For amples(P_3)

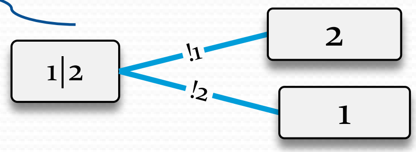
= reform(amples(P_7),P_3) U reform(amples(P_8), ,P_3) U {enable(P_3)}

= reform({{(!1, stop)}},P_3) U reform({{(!2, stop)}},P_3) U {enable(P_3)}

={{(!1,2)}} U {{(!2,1)}} U {{(!1,2), (!2,1)}}

= {{(!1,2)}, {(!2,1)}, {(!1,2),(!2,1)}}

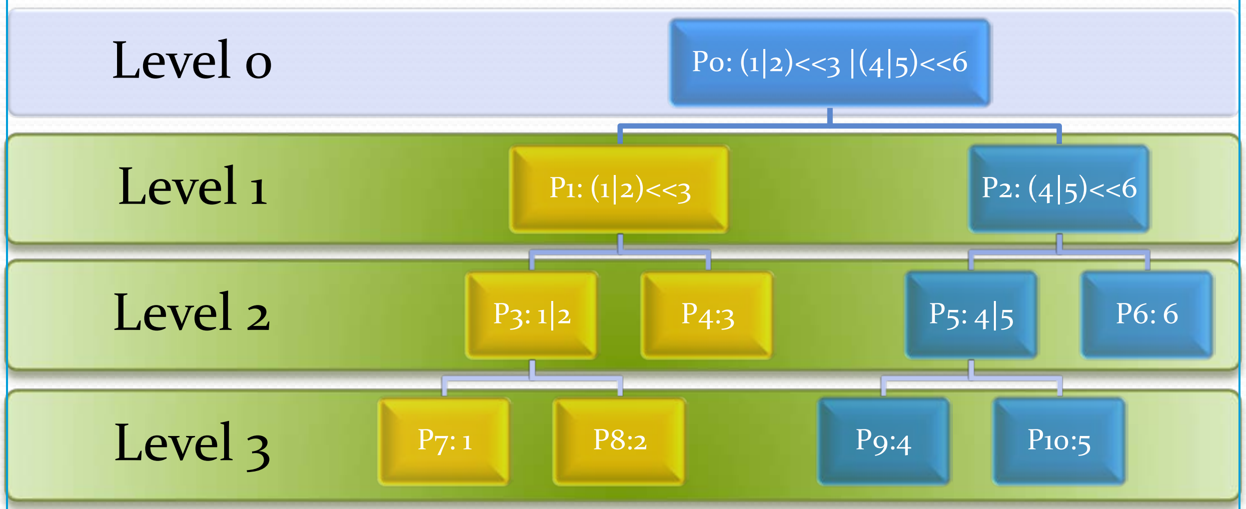
(Ample sets with 3 possible ample set)



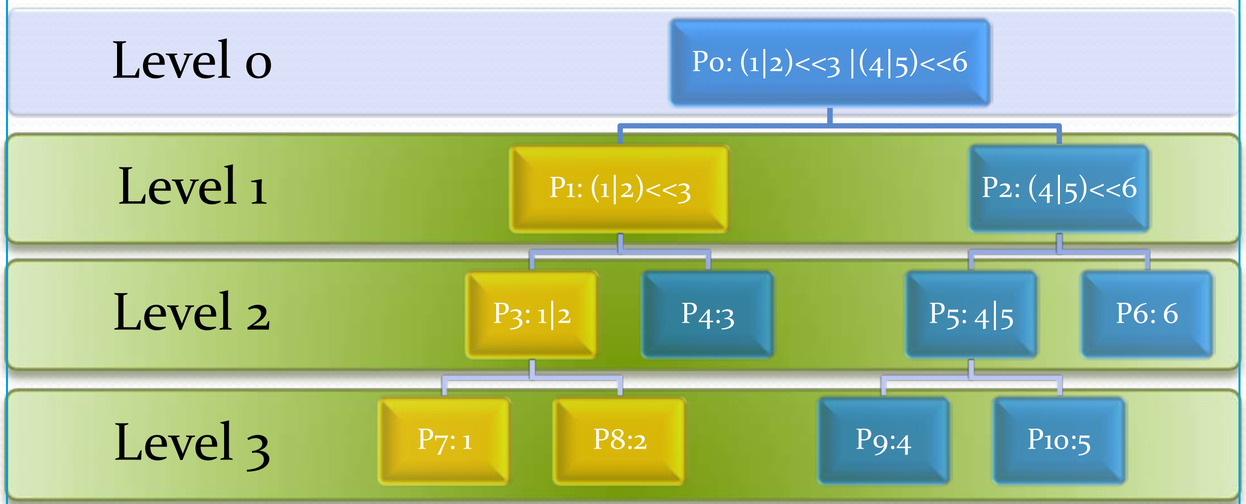
For amples(P_4)

= {enable(P_4)}

= {{(!3,stop)}}



For amples(P_1)
= reform(amples(P_3),P_1) U reform(amples(P_4),P_1) U {enable(P_1)}
= reform({{(!1,2)}, {(!2,1)}, {(!1,2),(!2,1)}} ,P_1) U
reform({{(!3,stop)},P_1) U
{enable(P_1)}



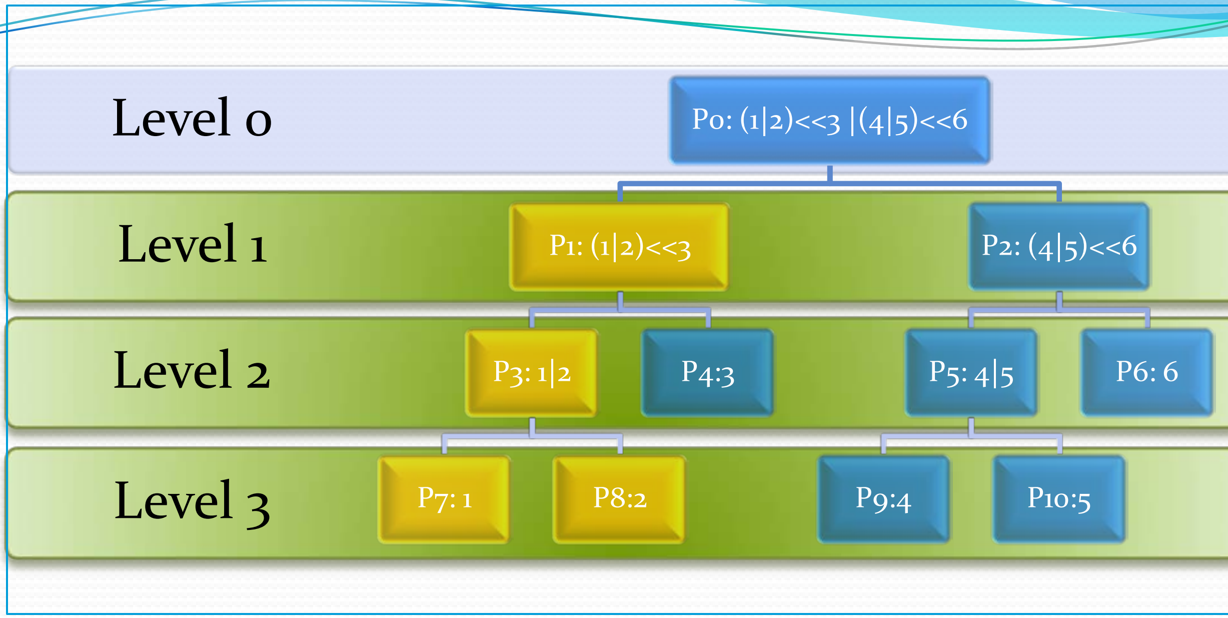
For amples(P_1)

= reform(amples(P_3),P_1) U reform(amples(P_3),P_1) U {enable(P_1)}

= reform({{(!1,2)}, {(!2,1)}, {(!1,2),(!2,1)}} ,P_1) U

~~reform({{(!3,stop)}},P_1) U~~ (Local condition violation - RHS of pruning operator not allowed)

{enable(P_1)}



For amples(P_1)

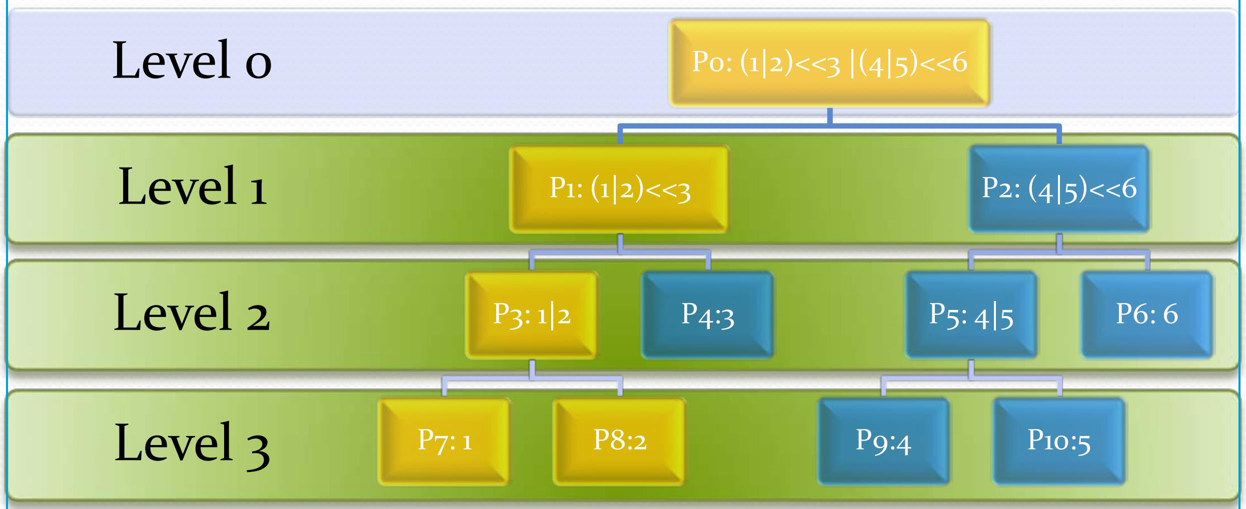
= reform(amples(P_3),P_1) U reform(amples(P_3),P_1) U {enable(P_1)}

= reform({{(!1,2)}, {(!2,1)}, {(!1,2),(!2,1)}} ,P_1) U

~~reform({{(!3,stop)},P_1) U~~ (Local condition violate-RHS of pruning operator not allowed)

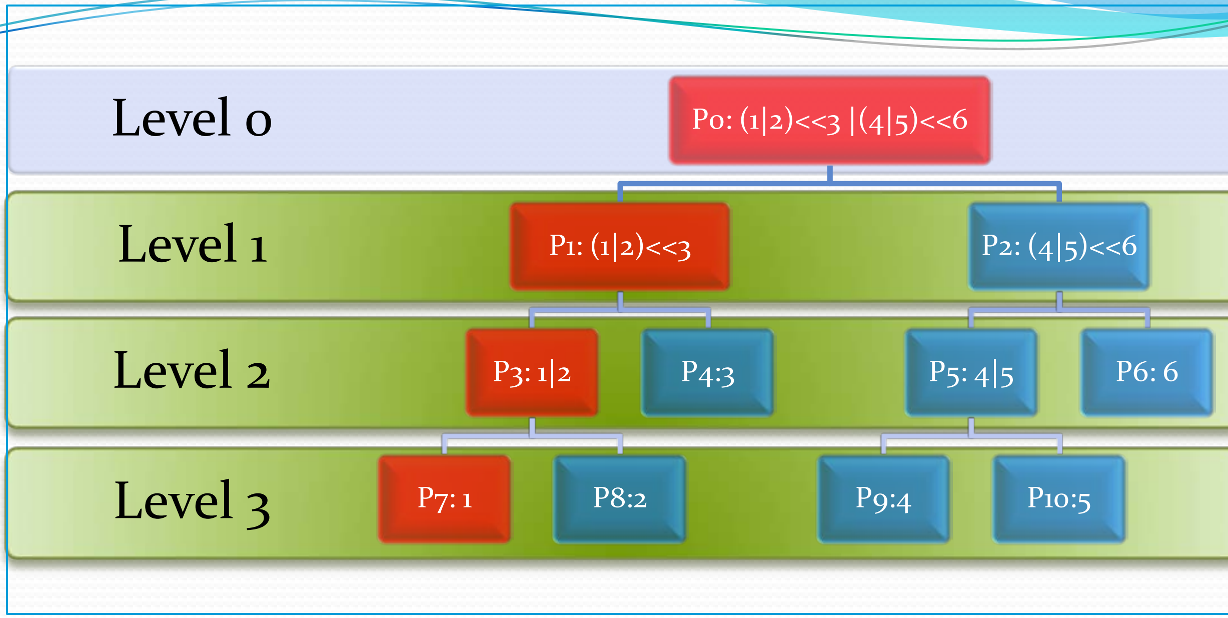
{enable(P_1)}

={{(!1,2<<3)}, {(!2,1<<3)}, {(!1,2<<3),(!2,1<<3)},{(!1,2<<3), (!2,2<<3), (!3,1|2)}}

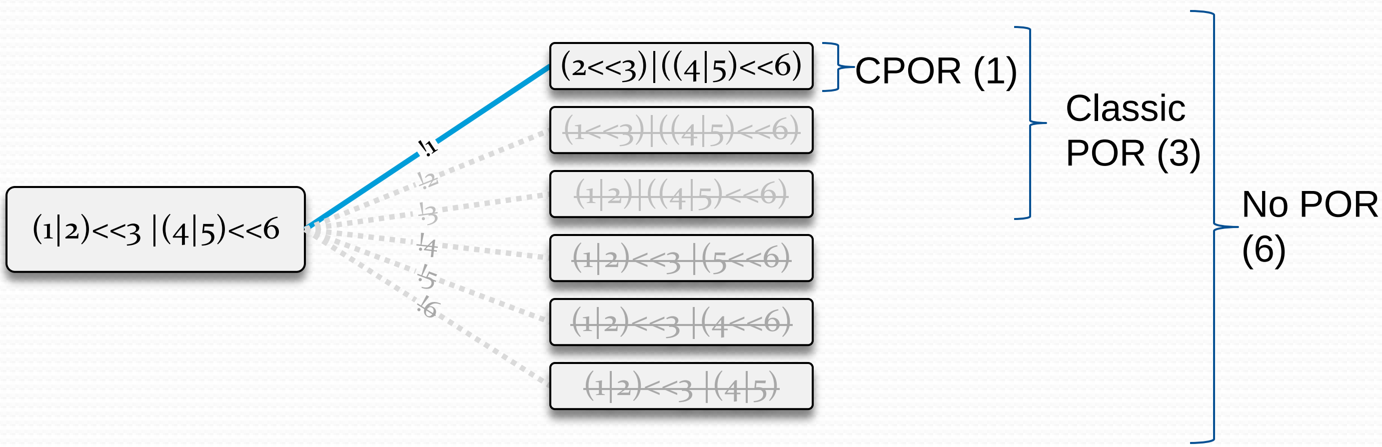


For amples(P_0)
= reform(amples(P_1),P_1)
= {{(!1,2<<3|P},
{!2,1<<3| P},
{(!1,2<<3|P),(!2,1<<3| P)},
{(!1,2<<3| P), (!2,2<<3| P), (!3,1|2| P)}}

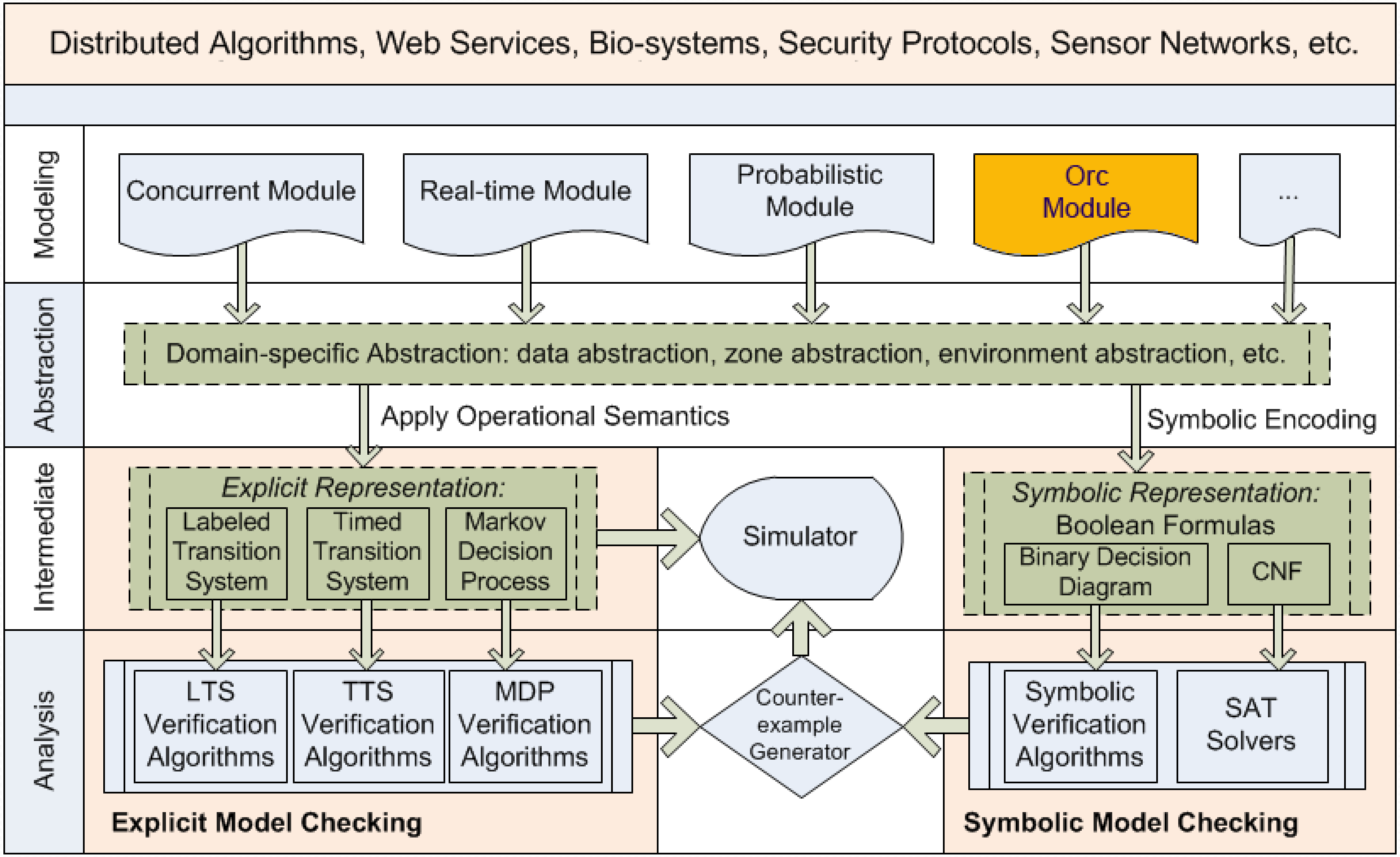
All four possible ample sets are checked for two global conditions, and all four turned up to be valid.
Ample set with smallest number of element is chosen. If they are multiple of them, choose one non-deterministically.



Assume {(!1,2<<3|(4|5)<<6} is chosen



PAT Architecture Design



The Current Status

- PAT is available at http://pat.comp.nus.edu.sg
- 1M lines of C# code, 11 modules with 100+ build in examples
- Used as an educational tool in e.g. York Univ., Univ. of Auckland, NII (Japan), NUS …
- Attracted more than 1400+ registered users in the last 3 years from more than 300+ organizations, e.g. Microsoft, HP, ST Elec, Oxford Univ., … Sony, Hitachi, Canon, Samsung.
- Japanese PAT User group formed in Sep 2009:

Founding Members:
Hiroshi Fujimoto
Kenji Taguchi
Masaru Nagaku
Toshiyuki Fujikura

Evaluation

(A) Comparing difference POR methods

Model	Property	Size		States			Time(s)		
				CPOR	POR	No POR/CPOR	CPOR	POR	No POR/CPOR
Concurrent Quicksort	(1.1)	2	✓	58	1532	10594	0.08	1.13	5
		3	✓	69	3611	36794	0.11	8.48	74
		5	✓	237	-	-	0.68	-	-
Readers-Writers Problem	(2.1)	2	✗	106	1645	7620	0.07	1.12	4
		3	✗	152	18247	142540	0.11	14.86	101
		10	✗	472	-	-	0.49	-	-
Auction Management	(3.1)	N.A.	✓	869	-	-	0.6	-	-
	(3.2)	N.A.	✓	883	-	-	0.75	-	-

(B) Comparing Our Model Checker and Maude

Model	Property		States/Rewrites		Time(s)	
			Our	Maude	Our	Maude
Auction Management	(3.1)	✓	869	7052663	0.6	14.4
	(3.2)	✓	883	8613539	0.75	19.8

Table 1. Performance evaluation on model checking *Orc*'s model

Conclusion and Future Works

- Contribution:
  - A new technique of Compositional Partial Order Reduction (CPOR) is proposed.
  - Verification for *Orc* language by directly using its operational semantics is supported.
- Future Works:
  - Extends CPOR to other languages.

Demo

Thanks!