Introducing Symmetry to Graph Rewriting Systems with Process Abstraction

Taichi Tomioka, Yutaro Tsunekawa, Kazunori Ueda
Waseda University, Tokyo

ICGT 2019, Eindhoven, July 14, 2019

Outline of the presentation

- **Model checking of graph rewriting systems** enjoys the synergy of two key features:

natural way to represent *inherent symmetry of models*	⟷	*symmetry reduction* based on graph isomorphism

- **Model abstraction** is another key technique for reducing state space and often strengthens model symmetry.
- To make these two ideas work together, we propose an abstraction technique, **UPE** (unused process elimination), that *automatically* simplifies models based on verification conditions.
- The whole framework has been developed in the graph rewriting language **LMNtal** and its model checker **SLIM**.

Topics

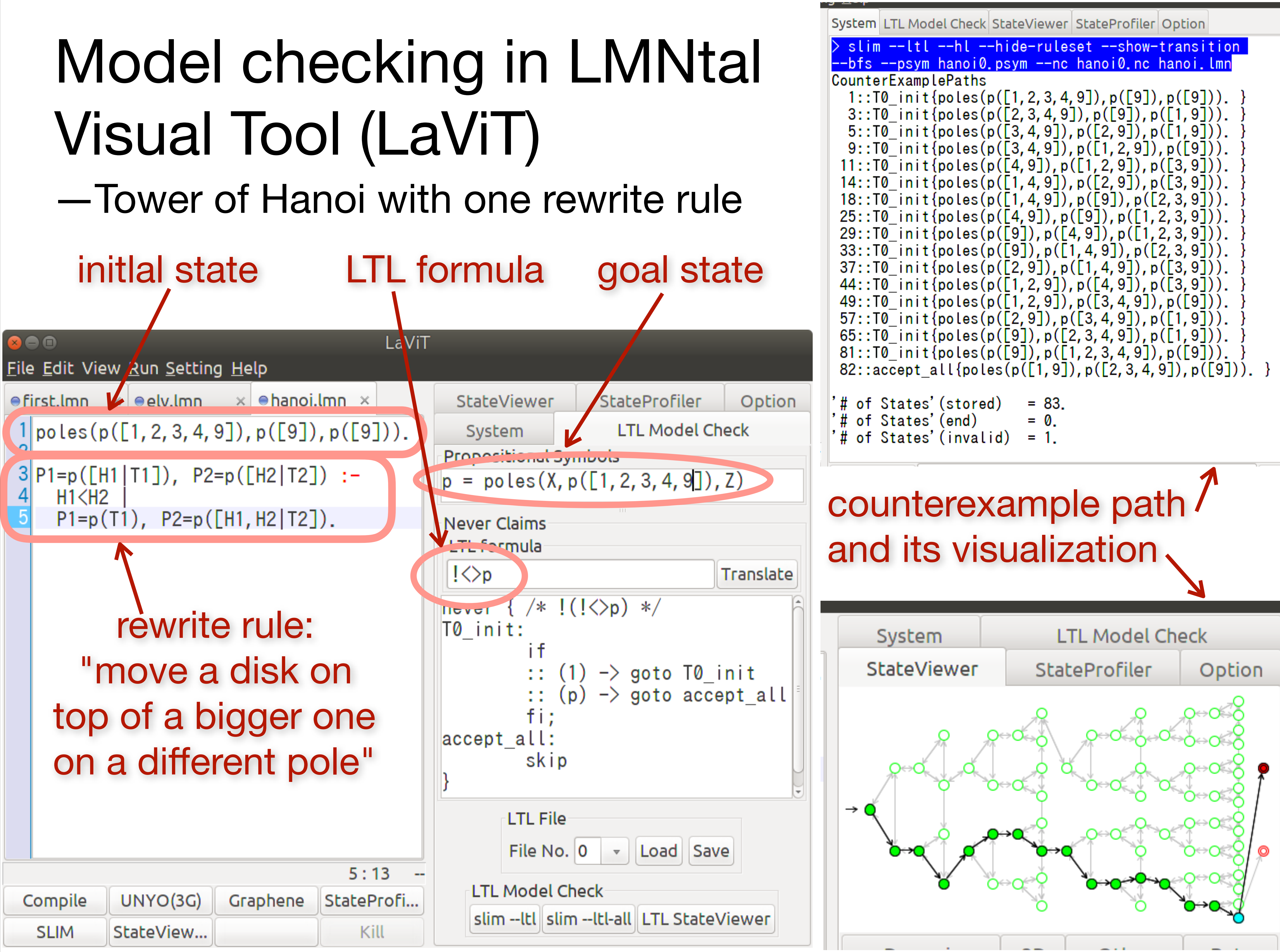
- Symmetry Reduction in Model Checking
- LMNtal
- Structural Congruence and Symmetry Reduction
- Process Abstraction
- Experiments
- Conclusion

Topics

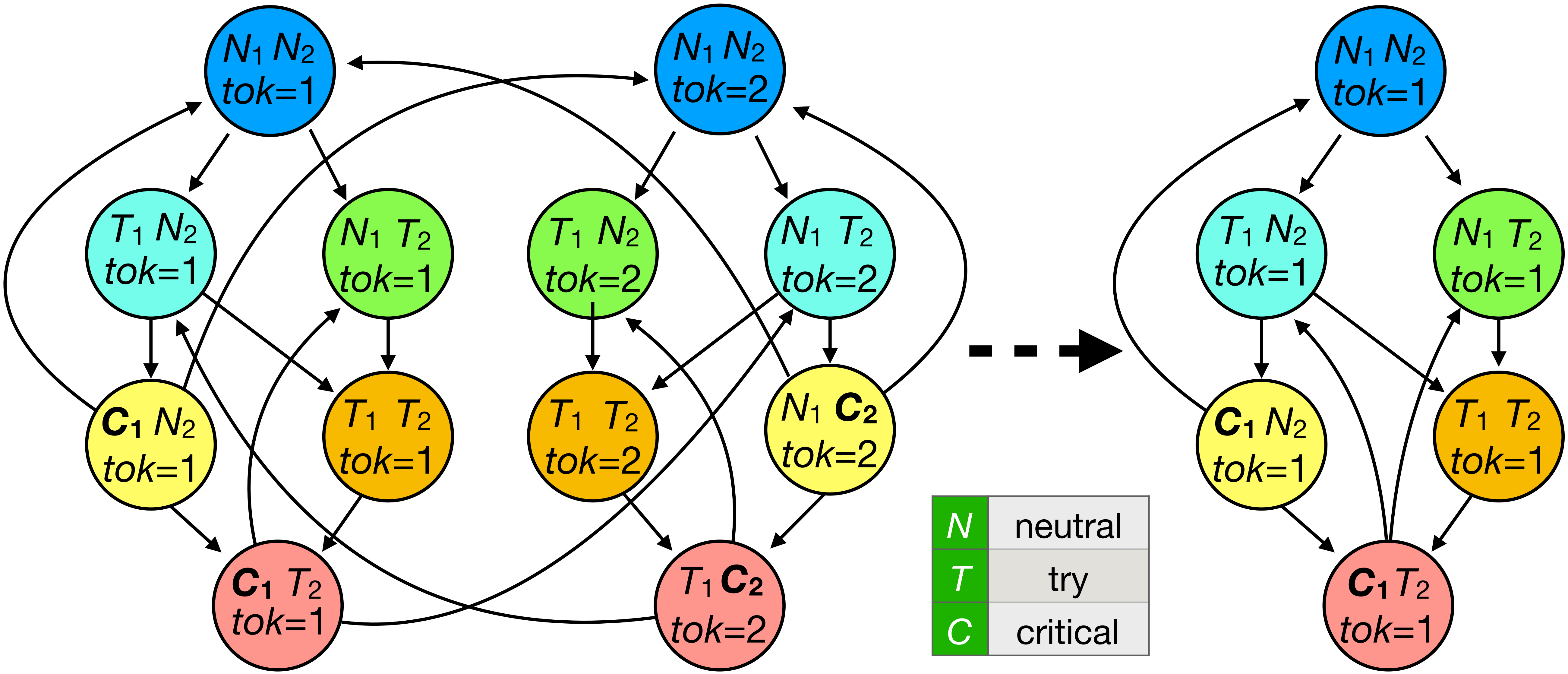
- Symmetry Reduction in Model Checking
- LMNtal
- Structural Congruence and Symmetry Reduction
- Process Abstraction
- Experiments
- Conclusion

Model checking in LMNtal Visual Tool (LaViT)

—Tower of Hanoi with one rewrite rule



Symmetry reduction in model checking



N	neutral
T	try
C	critical

Model : mutual exclusion problem (*)

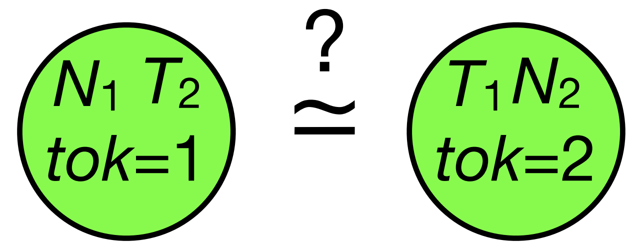
Spec. : two processes do not enter the critical section ("***C***") at the same time

- States in the same colors are symmetric to each other
- State space on the right is obtained by merging states

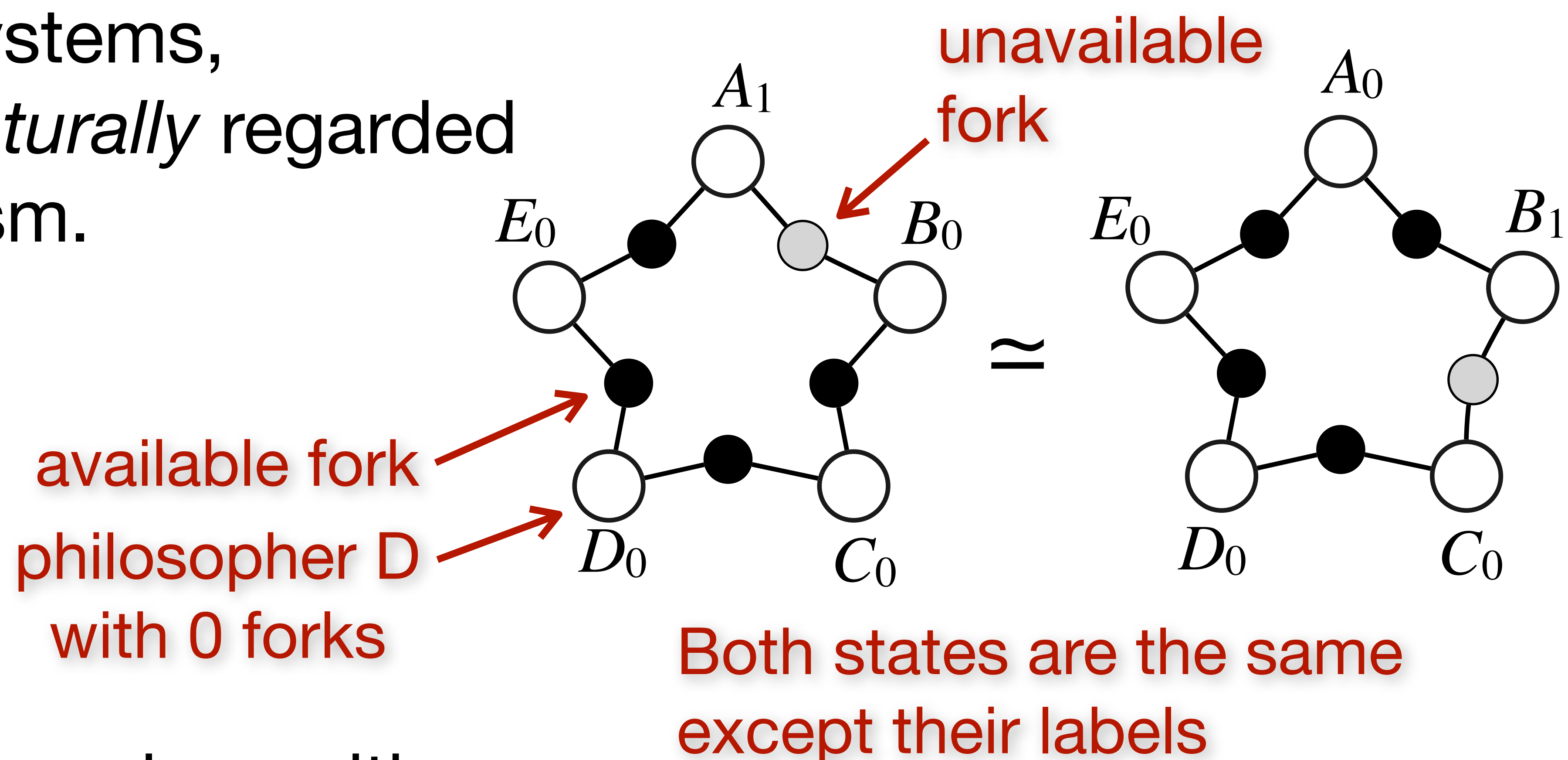
(*) A.Miller, A.Donaldson, M.Calder: Symmetry in Temporal Logic Model Checking, ACM CSUR **38**(3), 2006

Symmetry reduction + Graph rewriting

In symmetry reduction, we must define equivalence relation between states.



In graph rewriting systems, symmetry can be *naturally* regarded as graph isomorphism.



Implementations of graph rewriting systems featuring model checking include:

- GROOVE
- LMNtal + SLIM model checker (next slide)

Topics

- Symmetry Reduction in Model Checking
- LMNtal
- Structural Congruence and Symmetry Reduction
- Process Abstraction
- Experiments
- Conclusion

LMNtal, a unifying language for modeling and programming

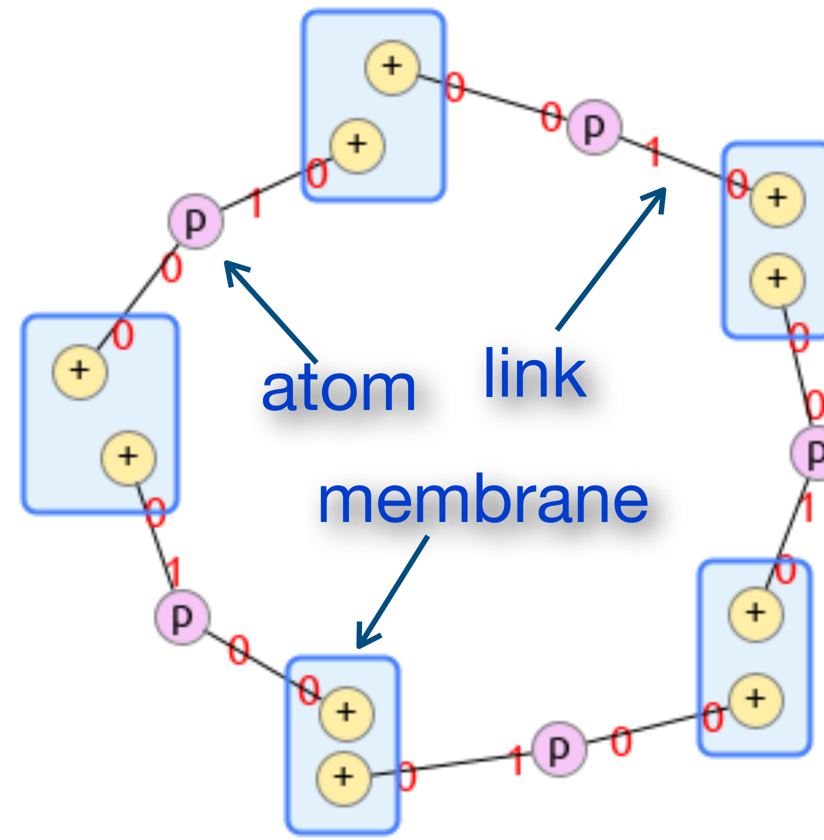
- Designed as a model of concurrency (2002) and implemented as a full-fledged *programming language*
  - unified view of processes, messages, functions and data structures by *atoms, links and membranes*
- Evolved into a *modeling tool* (2007) with
  - *parallel state-space search* and *LTL model checking* (up to 10^9 states) and
  - IDE (LaViT) with *state and state-space visualizers.*
  - Both extremely useful for *understanding* models with concurrency and nondeterminism
- Available open-source from GitHub;
  Portal at https://www.ueda.info.waseda.ac.jp/lmntal/

K. Ueda, LMNtal as a Hierarchical Logic Programming Language. Theor. Comput. Sci. **410**(46), 2009.

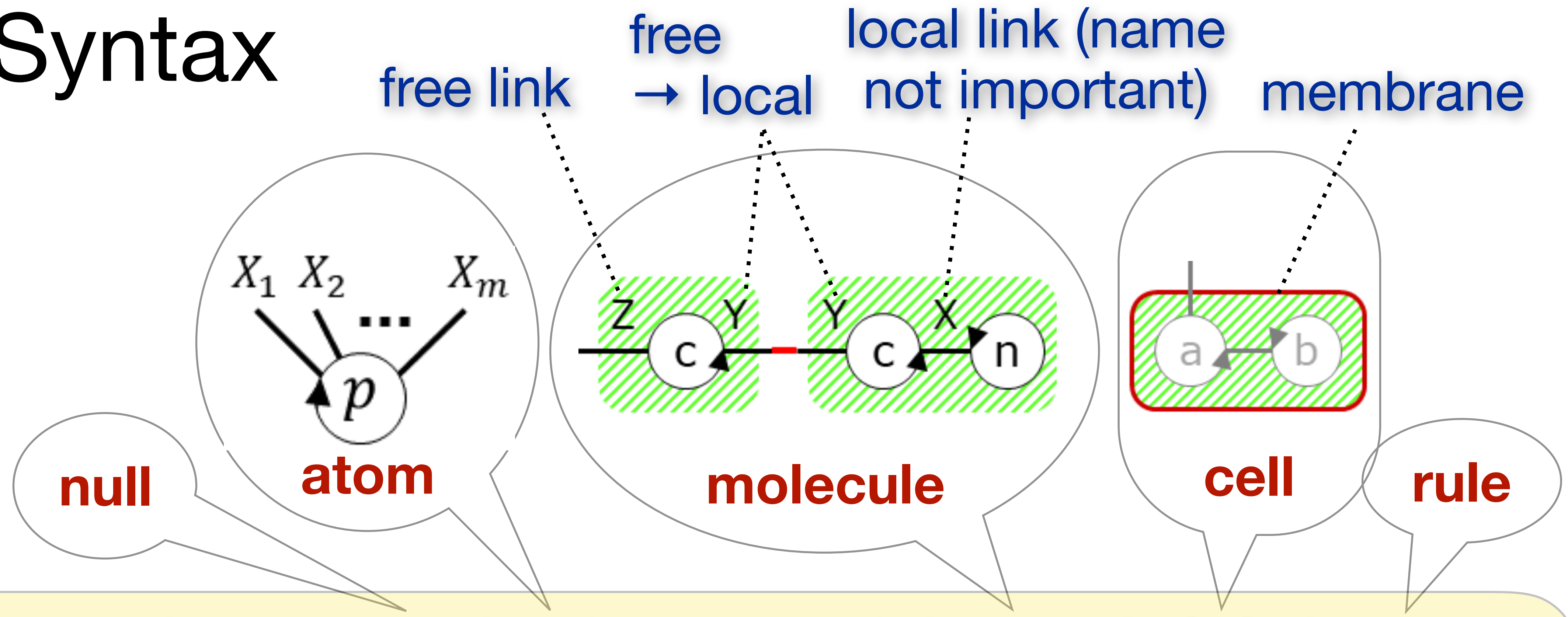
LMNtal in a nutshell

- A rule-based concurrent **language** for expressing and rewriting **connectivity** and **hierarchy**
- Computation is manipulation of (hierarchical) graphs consisting of:
  - **atoms** (simple nodes), each with its **name** and **arity**
  - **links** for **1-to-1 connectivity**
  - **hyperlinks** for **multipoint connectivity**
  - **membranes** (composite nodes) for **hierarchy**, **locality** and **first-class multisets**
- Well-defined notion of atomic actions (= rewrite steps)
- Allows concise encoding of the lambda calculus [RTA2008]

```
p(L1,R1), {+R1,+L2},
p(L2,R2), {+R2,+L3},
p(L3,R3), {+R3,+L4},
p(L4,R4), {+R4,+L5},
p(L5,R5), {+R5,+L1}.
```



LMNtal: Syntax



$$
\begin{array}{rl}
\text{(process)} & P ::= \mathbf{0} \mid p(X_1,\ldots,X_m) \mid P,P \mid \{P\} \mid T :- T \\
\text{(process template)} & T ::= \mathbf{0} \mid p(X_1,\ldots,X_m) \mid T,T \mid \{T\} \mid T :- T \\
 & \quad \mid \$p[X_1,\ldots,X_m|A] \\
\text{(residual)} & A ::= [\,] \mid *X
\end{array}
$$

Remarks:

- A well-formed process obeys *link conditions* (e.g., a link occurs at most twice in a process)
- A binary atom, **X=Y**, called a *connector*, fuses two links

LMNtal: Semantics (1/2)

- **Structural congruence** reflects the interpretation of LMNtal terms as *processes* and *graphs*

$$\text{(E1)}\quad \mathbf{0}, P \equiv P \qquad \text{(E2)}\quad P, Q \equiv Q, P \qquad \text{(E3)}\quad P, (Q, R) \equiv (P, Q), R$$

$$\text{(E4)}\quad P \equiv P[Y/X] \qquad \text{if } X \text{ is a local link of } P$$

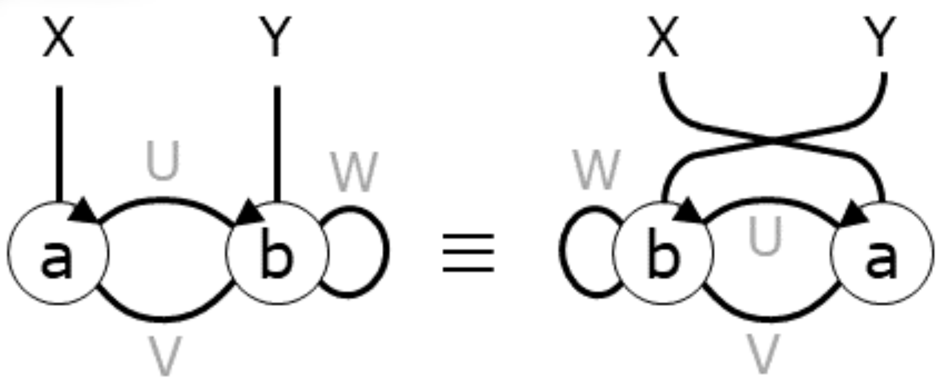
$$\text{(E5)}\quad P \equiv P' \Rightarrow P, Q \equiv P', Q \qquad \text{(E6)}\quad P \equiv P' \Rightarrow \{P\} \equiv \{P'\}$$

$$\text{(E7)}\quad X = X \equiv \mathbf{0} \qquad \text{(E8)}\quad X = Y \equiv Y = X$$

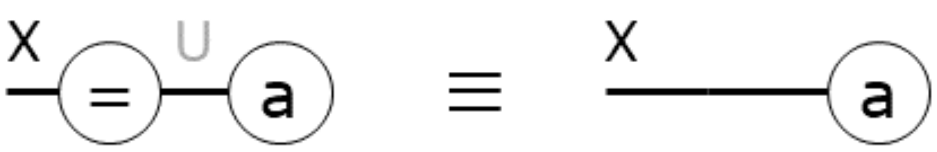
$$\text{(E9)}\quad X = Y,\ P \equiv P[Y/X] \qquad \text{if } P \text{ is an atom and } X \text{ occurs free in } P$$

$$\text{(E10)}\quad \{X = Y,\ P\} \equiv X = Y,\ \{P\} \qquad \text{if exactly one of } X \text{ and } Y \text{ occurs free in } P$$

Examples:



(E2) a(U,V,X), b(U,V,W,Y) ≡ b(U,V,W,Y), a(U,V,X)



(E9) X=U, a(U) ≡ a(X)

LMNtal: Semantics (2/2)

- **Reduction relation** defined in a standard small-step style

structural rules

$$\text{(R1)}\quad \frac{P \longrightarrow P'}{P,Q \longrightarrow P',Q} \qquad \text{(R2)}\quad \frac{P \longrightarrow P'}{\{P\} \longrightarrow \{P'\}} \qquad \text{(R3)}\quad \frac{Q \equiv P \quad P \longrightarrow P' \quad P' \equiv Q'}{Q \longrightarrow Q'}$$

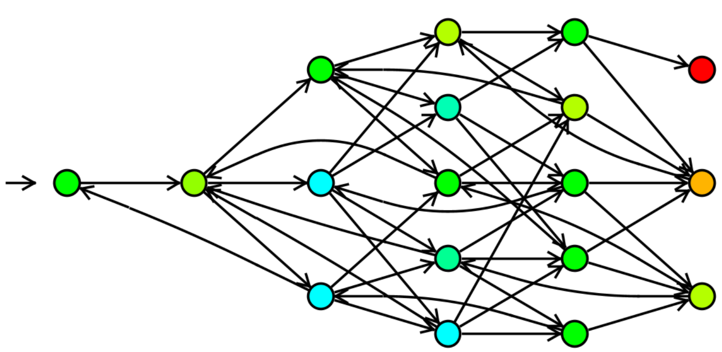
$$\text{(R4)}\quad \{X=Y,P\} \longrightarrow X=Y,\{P\} \quad \text{if } X \text{ and } Y \text{ occur free in } \{X=Y,P\}$$

$$\text{(R5)}\quad X=Y,\{P\} \longrightarrow \{X=Y,P\} \quad \text{if } X \text{ and } Y \text{ occur free in } P$$

$$\text{(R6)}\quad T\theta,(T\,{:}\text{-}\,U) \longrightarrow U\theta,(T\,{:}\text{-}\,U)$$

main reduction rule

- We regard the semantics $[\![P]\!]$ of a process P as the *state space* *(= state transition graph) of P*.



Topics

- Symmetry Reduction in Model Checking
- LMNtal
- Structural Congruence and Symmetry Reduction
- Process Abstraction
- Experiments
- Conclusion

Symmetry reduction in LMNtal

- Equivalence of processes written as LMNtal terms are *inductively* defined as ***structural congruence*** in a *syntax-directed* manner

  Symmetry in LMNtal = Structural Congruence

- Standard theory of symmetry reduction is stated using *permutation groups of **group theory***

Bridging these two formalisms is not straightforward.

Structural Congruence and Group

We need to construct a permutation group *E* that satisfies:

$$P \equiv Q \iff P \sim_E Q \overset{\text{def}}{\iff} \exists \sigma \in E.\ Q = \sigma P$$

syntactic mapping on P

— Not straightforward due to the inductive nature of *P* and ≡.

We defined *an underlying set of E* as *the least fixed point of an inductive function F* (details omitted) and established the bridge between $\equiv$ and $\sim_E$.

Now, from the known result on symmetry reduction, we have the *soundness of symmetry reduction*: For any LTL formula ϕ,

$$[\![P]\!]/\!\equiv\ \vDash \phi \implies [\![P]\!] \vDash \phi$$

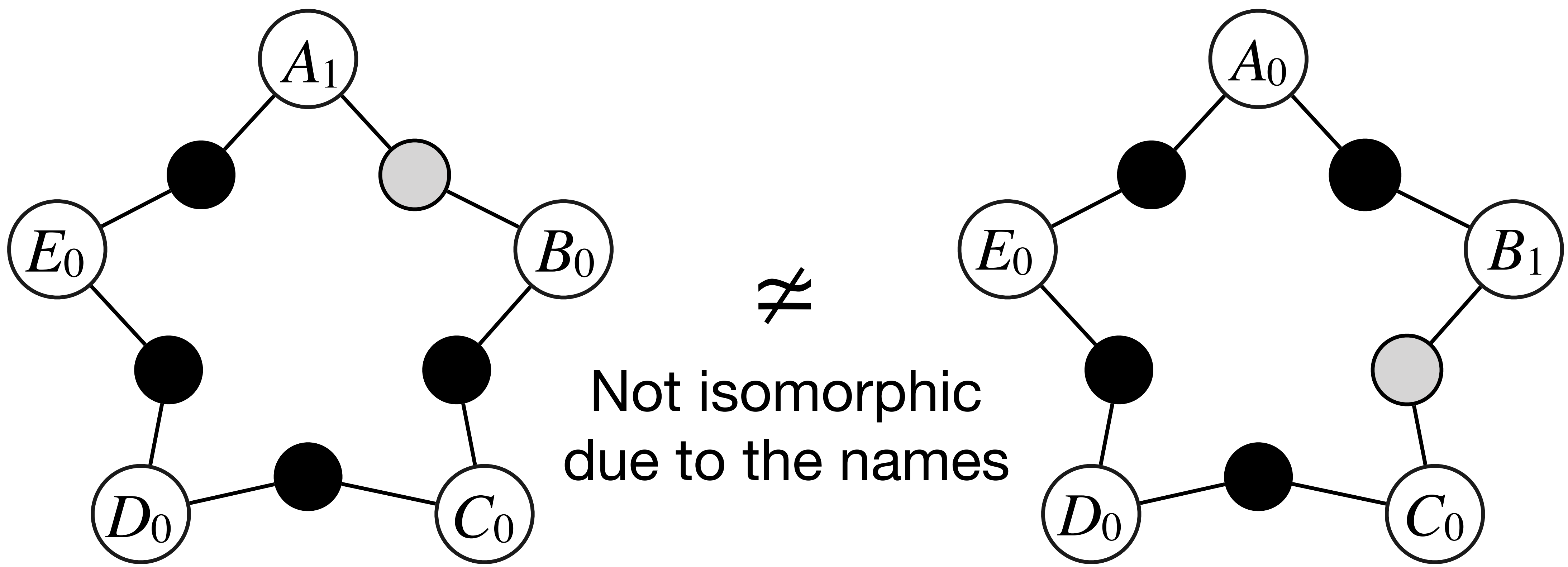
Topics

- Symmetry Reduction in Model Checking
- LMNtal
- Structural Congruence and Symmetry Reduction
- Process Abstraction
- Experiments
- Conclusion

Symmetry of graphs is not sufficient

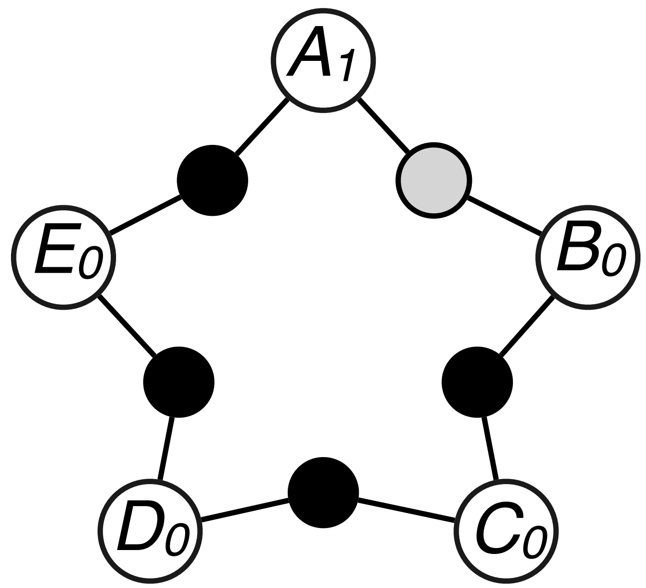
States of dining philosopher's problem with names are *not* isomorphic

→ Symmetry reduction does not work



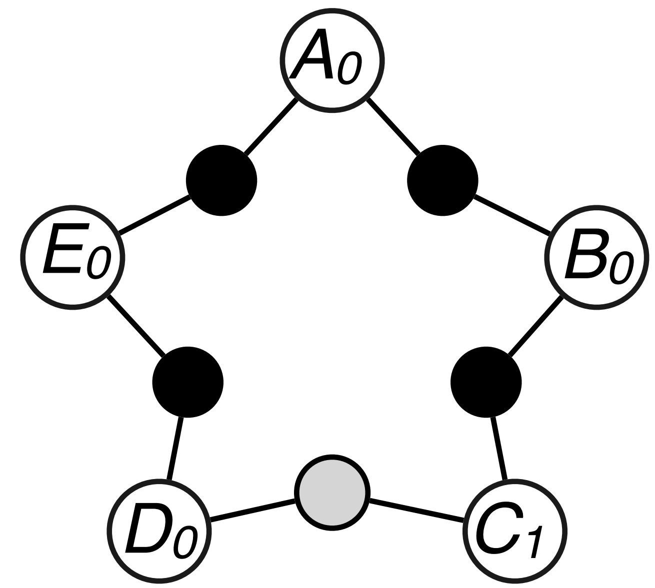
Process Abstraction

Abstracting their names reveals the symmetry of states



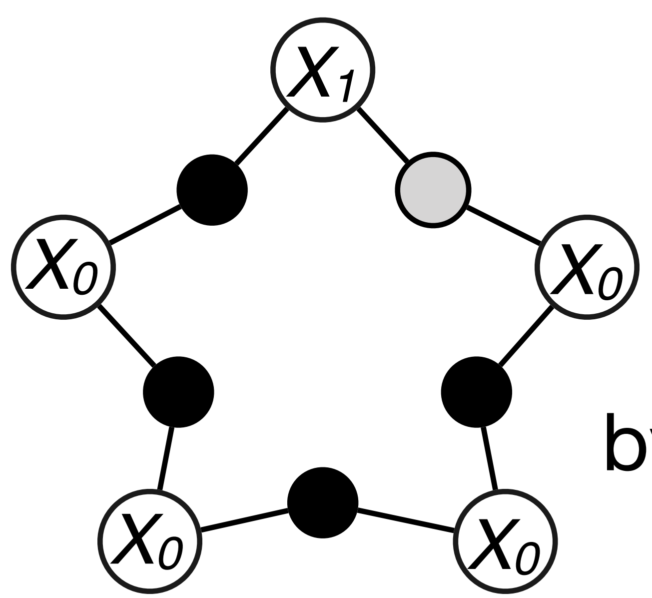
≠

Not isomorphic
due to the names



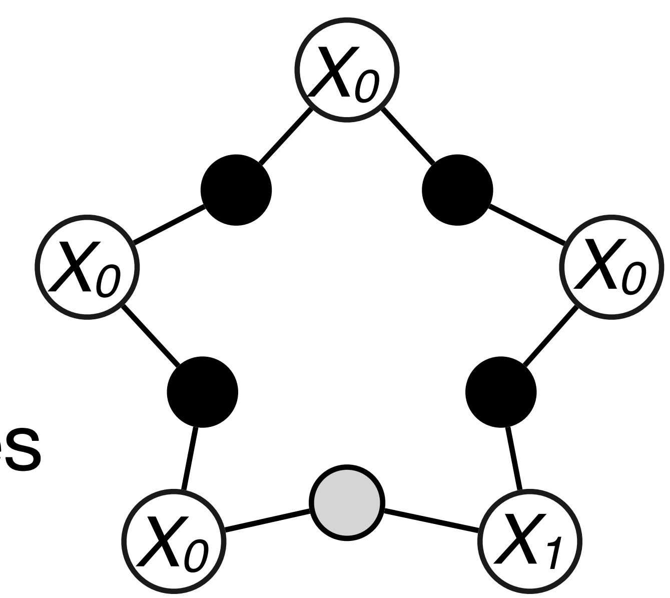
↓ **Abstraction**

↓ **Abstraction**



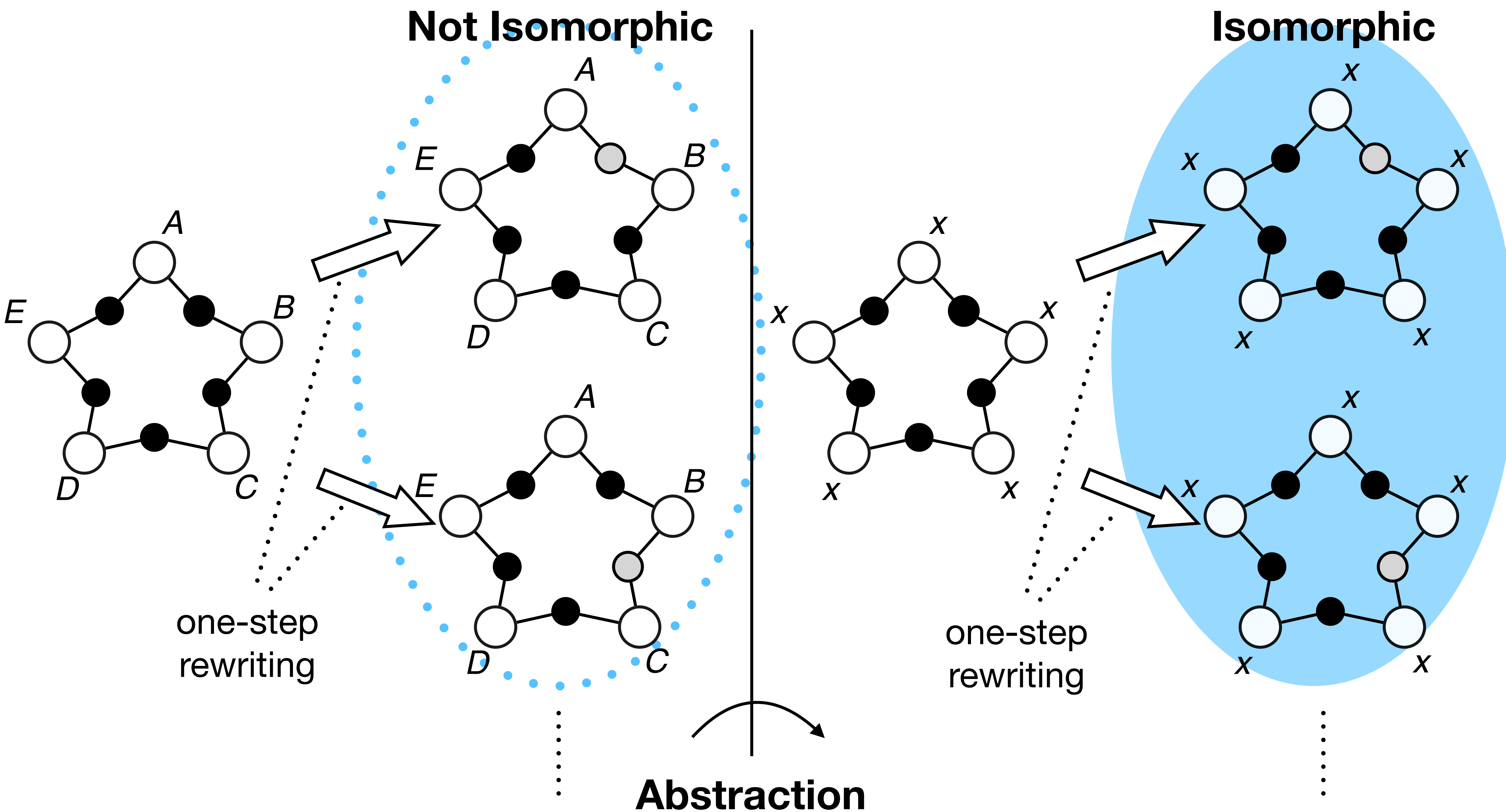
≃

Isomorphic
by ignoring the names



Process Abstraction

Symmetry reduction is done by abstracting all states in a state space



How to abstract processes?

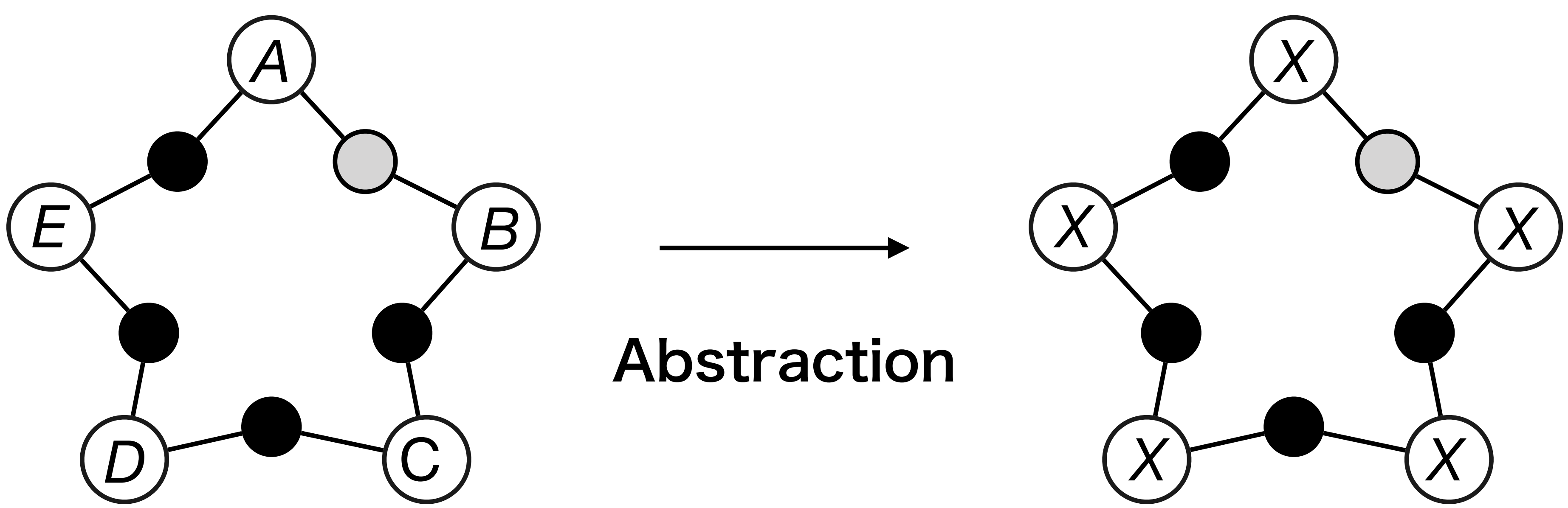
Sometimes abstraction loses the soundness of model checking.

For example, consider two specifications below:

(1) the model does not cause deadlock

(2) philosopher A eats before philosopher B

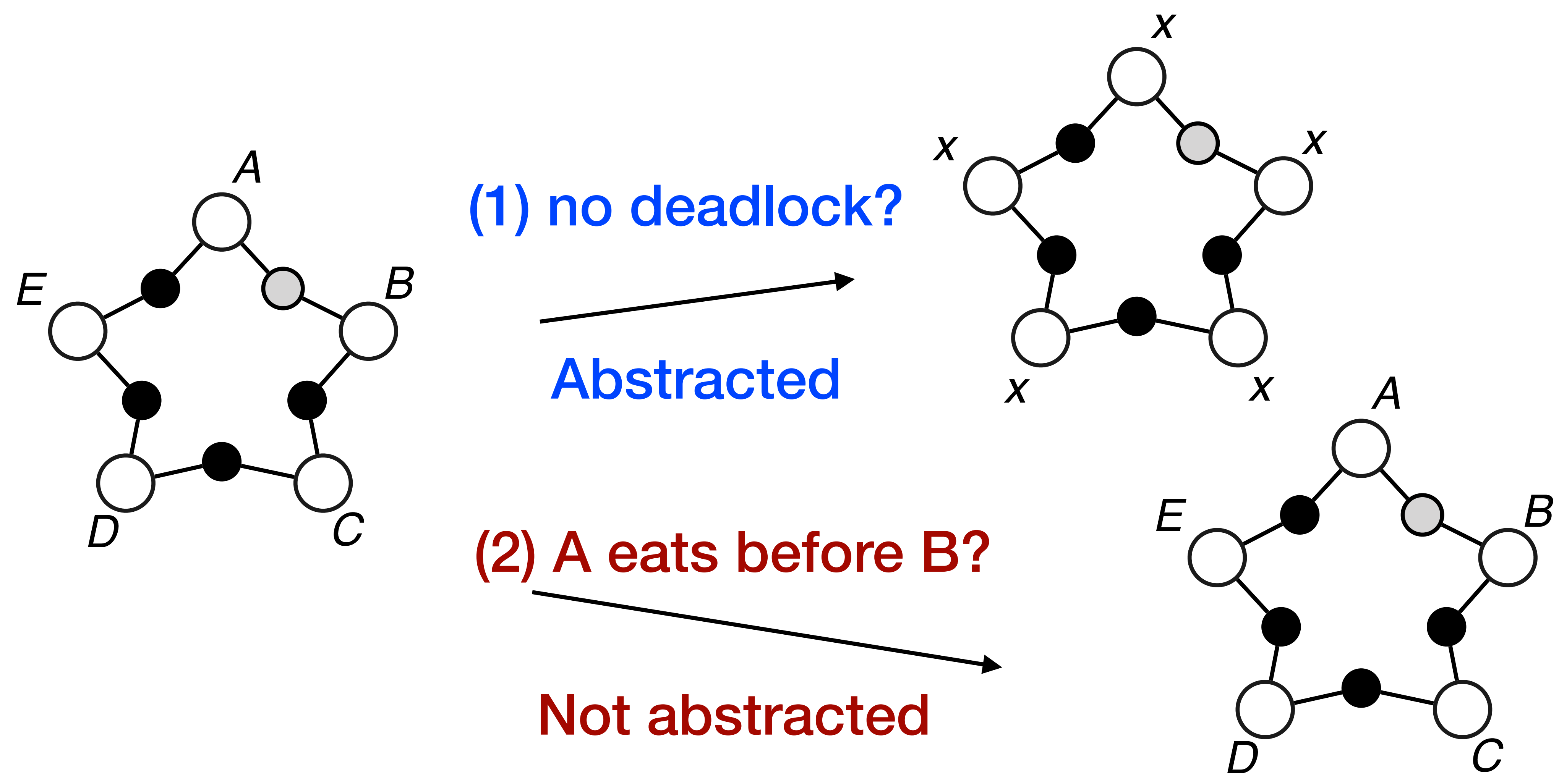
Spec (1) is verified in both models (abstracted and not abstracted), but Spec (2) cannot be verified in the abstracted model



Unused Process Elimination

UPE (Unused Process Elimination) abstracts part of a process not appearing in rewrite rules or specifications.

UPE automatically and statically decides whether a process should be abstracted.



Basics of LMNtal UPE

1. Mark removable atoms not appearing in rewrite rules or specifications
2. Delete removable atoms
3. Terminate dangling links with some atoms (we used '#')

```
%% grab a left fork
{+X,+L}, phi(N,L,R) :-
  {-X,+L}, phi(N,L,R).

%% grab a right fork
{-X,+L}, phi(N,L,R), {+R,+Y} :-
  {-X,+L}, phi(N,L,R), {+R,-Y}.

%% release forks
{-X,+L}, phi(N,L,R), {+R,-Y} :-
  {+X,+L}, phi(N,L,R), {+R,+Y}.
```

```
phi(a, R1,L1), {+R1,+L2},
phi(b, L2,R2), {+R2,+L3},
phi(c, L3,R3), {+R3,+L4},
phi(d, L4,R4), {+R4,+L5},
phi(e, L5,R5), {+R5,+L1}.
```

Specs

A_can_eat :=
```
{-X,+L},phi(a,L,R),{+R,-Y} :-
```
B_can_eat :=
```
{-X,+L},phi(b,L,R),{+R,-Y} :-
```
Someone_can_eat :=
```
{-X,+L},phi(_,L,R),{+R,-Y} :-
```

Named dining philosopher's problem in LMNtal

Basics of LMNtal UPE

1. Mark removable atoms not appearing in rewrite rules or specifications
2. Delete removable atoms
3. Terminate dangling links with some atoms (we used '#')

```
%% grab a left fork
{+X,+L}, phi(N,L,R) :-
  {-X,+L}, phi(N,L,R).

%% grab a right fork
{-X,+L}, phi(N,L,R), {+R,+Y} :-
  {-X,+L}, phi(N,L,R), {+R,-Y}.

%% release forks
{-X,+L}, phi(N,L,R), {+R,-Y} :-
  {+X,+L}, phi(N,L,R), {+R,+Y}.
```

```
phi(a, R1,L1), {+R1,+L2},
phi(b, L2,R2), {+R2,+L3},
phi(c, L3,R3), {+R3,+L4},
phi(d, L4,R4), {+R4,+L5},
phi(e, L5,R5), {+R5,+L1}.
```

Specs

A_can_eat :=
```
{-X,+L},phi(a,L,R),{+R,-Y} :-
```

B_can_eat :=
```
{-X,+L},phi(b,L,R),{+R,-Y} :-
```

Someone_can_eat :=
```
{-X,+L},phi(_,L,R),{+R,-Y} :-
```

Named dining philosopher's problem in LMNtal

Basics of LMNtal UPE

1. **Mark removable atoms not appearing in rewrite rules or specifications**
2. Delete removable atoms
3. Terminate dangling links with some atoms (we used '#')

```
%% grab a left fork
{+X,+L}, phi(N,L,R) :-
  {-X,+L}, phi(N,L,R).

%% grab a right fork
{-X,+L}, phi(N,L,R), {+R,+Y} :-
  {-X,+L}, phi(N,L,R), {+R,-Y}.

%% release forks
{-X,+L}, phi(N,L,R), {+R,-Y} :-
  {+X,+L}, phi(N,L,R), {+R,+Y}.
```

```
phi(a, R1,L1), {+R1,+L2},
phi(b, L2,R2), {+R2,+L3},
phi(c, L3,R3), {+R3,+L4},
phi(d, L4,R4), {+R4,+L5},
phi(e, L5,R5), {+R5,+L1}.
```

Specs

```
A_can_eat :=
  {-X,+L},phi(a,L,R),{+R,-Y} :-
B_can_eat :=
  {-X,+L},phi(b,L,R),{+R,-Y} :-
Someone_can_eat :=
  {-X,+L},phi(_,L,R),{+R,-Y} :-
```

Named dining philosopher's problem in LMNtal

Basics of LMNtal UPE

1. Mark removable atoms not appearing in rewrite rules or specifications
2. **Delete removable atoms**
3. Terminate dangling links with some atoms (we used '#')

```
%% grab a left fork
{+X,+L}, phi(N,L,R) :-
  {-X,+L}, phi(N,L,R).

%% grab a right fork
{-X,+L}, phi(N,L,R), {+R,+Y} :-
  {-X,+L}, phi(N,L,R), {+R,-Y}.

%% release forks
{-X,+L}, phi(N,L,R), {+R,-Y} :-
  {+X,+L}, phi(N,L,R), {+R,+Y}.
```

```
phi( , R1,L1), {+R1,+L2},
phi( , L2,R2), {+R2,+L3},
phi( , L3,R3), {+R3,+L4},
phi( , L4,R4), {+R4,+L5},
phi( , L5,R5), {+R5,+L1}.
```

Specs

A_can_eat :=
```
  {-X,+L},phi(a,L,R),{+R,-Y} :-
```
B_can_eat :=
```
  {-X,+L},phi(b,L,R),{+R,-Y} :-
```
Someone_can_eat :=
```
  {-X,+L},phi(_,L,R),{+R,-Y} :-
```

Named dining philosopher's problem in LMNtal

Basics of LMNtal UPE

1. Mark removable atoms not appearing in rewrite rules or specifications
2. Delete removable atoms
3. **Terminate dangling links with some atoms (we used '#')**

```
%% grab a left fork
{+X,+L}, phi(N,L,R) :-
  {-X,+L}, phi(N,L,R).

%% grab a right fork
{-X,+L}, phi(N,L,R), {+R,+Y} :-
  {-X,+L}, phi(N,L,R), {+R,-Y}.

%% release forks
{-X,+L}, phi(N,L,R), {+R,-Y} :-
  {+X,+L}, phi(N,L,R), {+R,+Y}.
```

```
phi(#, R1,L1), {+R1,+L2},
phi(#, L2,R2), {+R2,+L3},
phi(#, L3,R3), {+R3,+L4},
phi(#, L4,R4), {+R4,+L5},
phi(#, L5,R5), {+R5,+L1}.
```

Specs

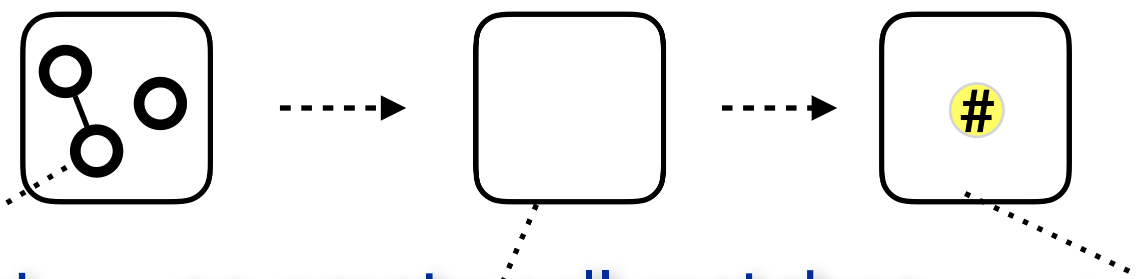
```
A_can_eat :=
  {-X,+L},phi(a,L,R),{+R,-Y} :-
B_can_eat :=
  {-X,+L},phi(b,L,R),{+R,-Y} :-
Someone_can_eat :=
  {-X,+L},phi(_,L,R),{+R,-Y} :-
```

Named dining philosopher's problem in LMNtal

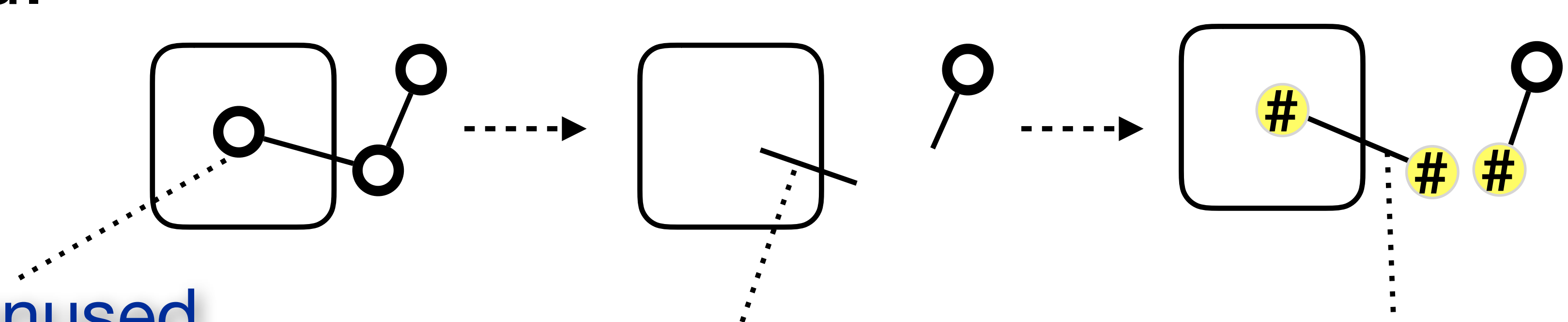
UPE and Membranes

Membranes of LMNtal needs further care in the design of UPE.

- UPE may add a special nullary atom in order to indicate that there were some atoms in the membrane.



- Free links crossing a membrane are terminated and not deleted.



UPE Commutes with state space construction

Theorem: The following two state spaces deduced from a process *P* are equal.

State space whose states are obtained by applying UPE to each states of ⟦*P*⟧

State space starting with UPE(*P*) and constructed in a standard way

$$\mathit{UPE}([\![P]\!]) = [\![\mathit{UPE}(P)]\!]$$

- It is practically an important property because applying UPE to all states of a model is very costly.

UPE: State space reduction

Theorem: UPE preserves structural congruence.

$$\forall s, t \in S_p, \;\; s \equiv t \implies \mathit{UPE}(s) \equiv \mathit{UPE}(t)$$

- The number of varieties of structurally congruent processes does not increase and may decrease.
- The quotient of a state space by structural congruence does not become larger after UPE.

UPE: Preservation of rewritablity

- If a process can be rewritten by some rule, the abstracted process obtained by UPE can be rewritten by the same rule.

Theorem: For state spaces $[\![P]\!]=(S_P, R_P, P)$ and $UPE([\![P]\!])=(S_P{}^{\#}, R_P{}^{\#}, UPE(P))$,

$$\forall s, t \in S_P, (s, t) \in R_P \implies (UPE(s), UPE(t)) \in R_P{}^{\#}$$

holds.

- UPE is a *homomorphism* between state spaces.

UPE: Soundness of Model Checking

- Because UPE preserves rewritability, it also preserves labeling functions for model checking.
- These two preservation properties lead to the soundness in model checking.

Theorem:

For any LTL formula ϕ, $\mathrm{UPE}([\![P]\!]) \vDash \phi \;\Rightarrow\; [\![P]\!] \vDash \phi$.

Topics

- Symmetry Reduction in Model Checking
- LMNtal
- Structural Congruence and Symmetry Reduction
- Process Abstraction
- Experiments
- Conclusion

Experiments

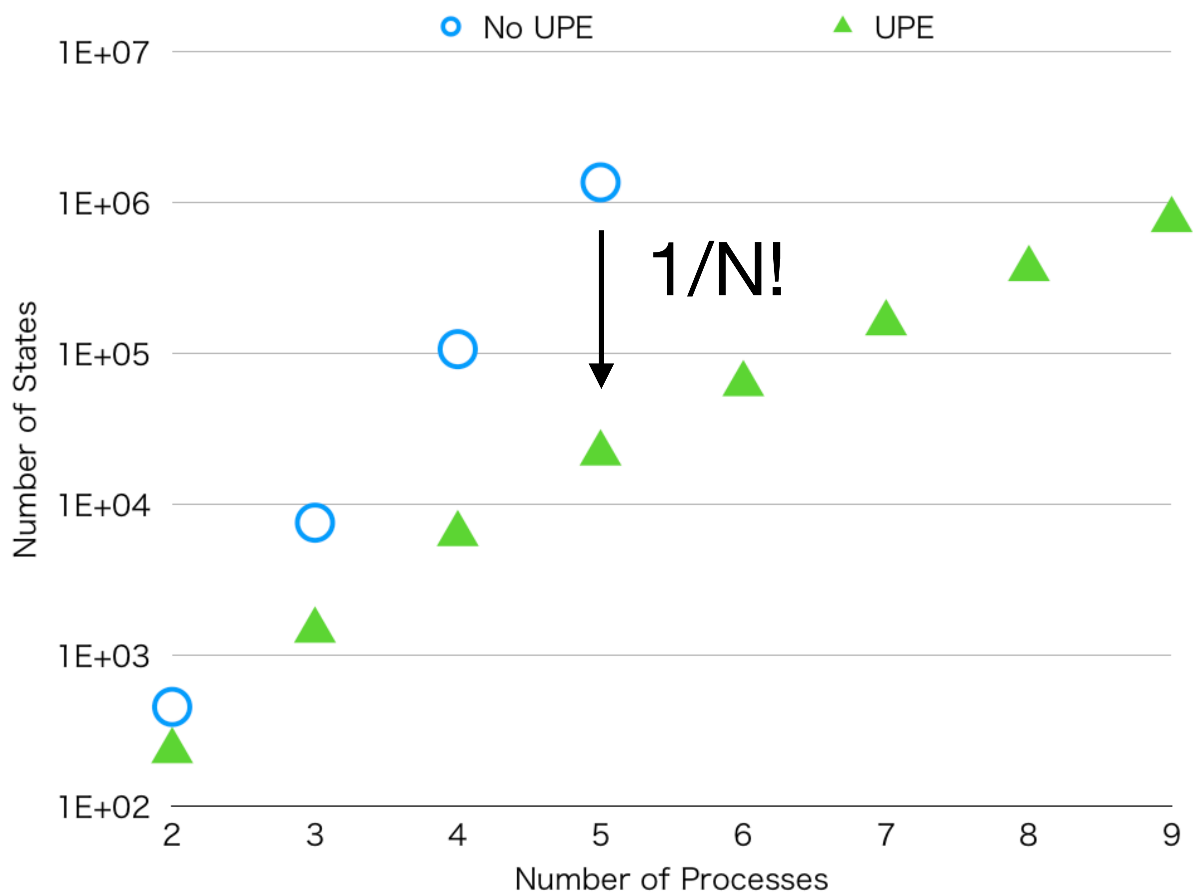
- Implemented various concurrent algorithms[2] in LMNtal

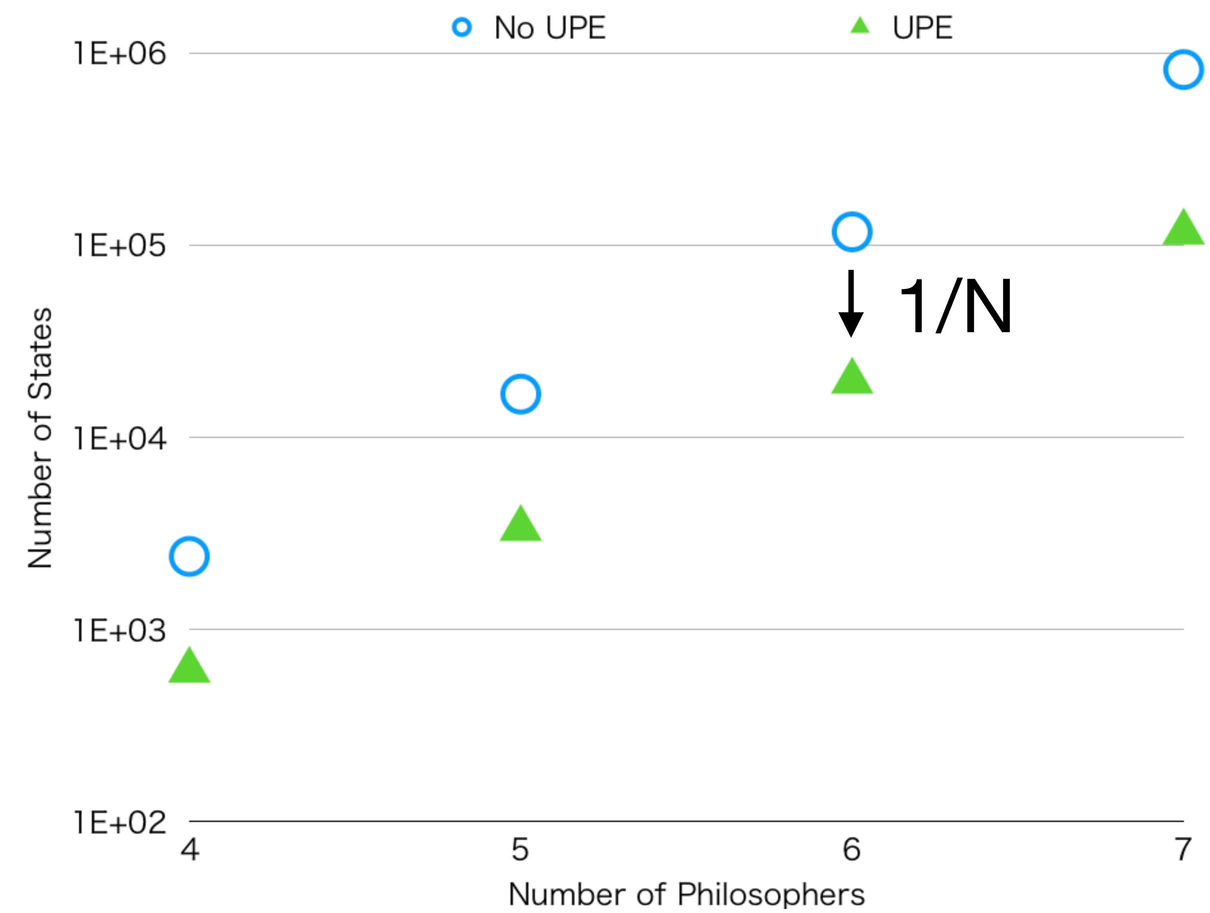
Problem	# of States	# of States (UPE)
Dekker	364	182
Peterson	190	95
Doran-Thomas	576	288
Udding's (3 processes)	7619	1478
Philosophers	16805	3365
Philosophers (no deadlock)	16806	16806

- ❖ Dekker, Peterson, Doran-Thomas runs in 2 processes
- ❖ Philosophers (no deadlock) has a philosopher who picks up an opposite fork first.

[2] Ben-Ari, M.: Principles of Concurrent and Distributed Programming. Addison- Wesley (2006)

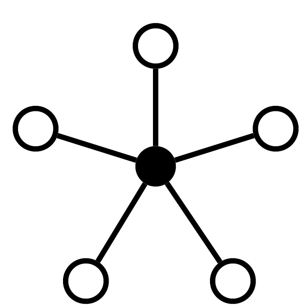
Experiments



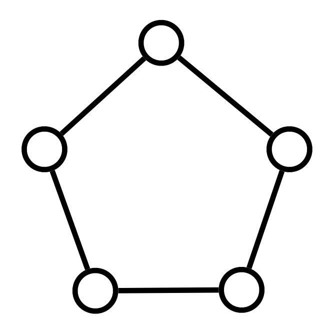


Udding's starvation-free algorithm

Dining Philosophers Problem



Star



Ring

The number of states are reduced depending on the symmetry of models.

Topics

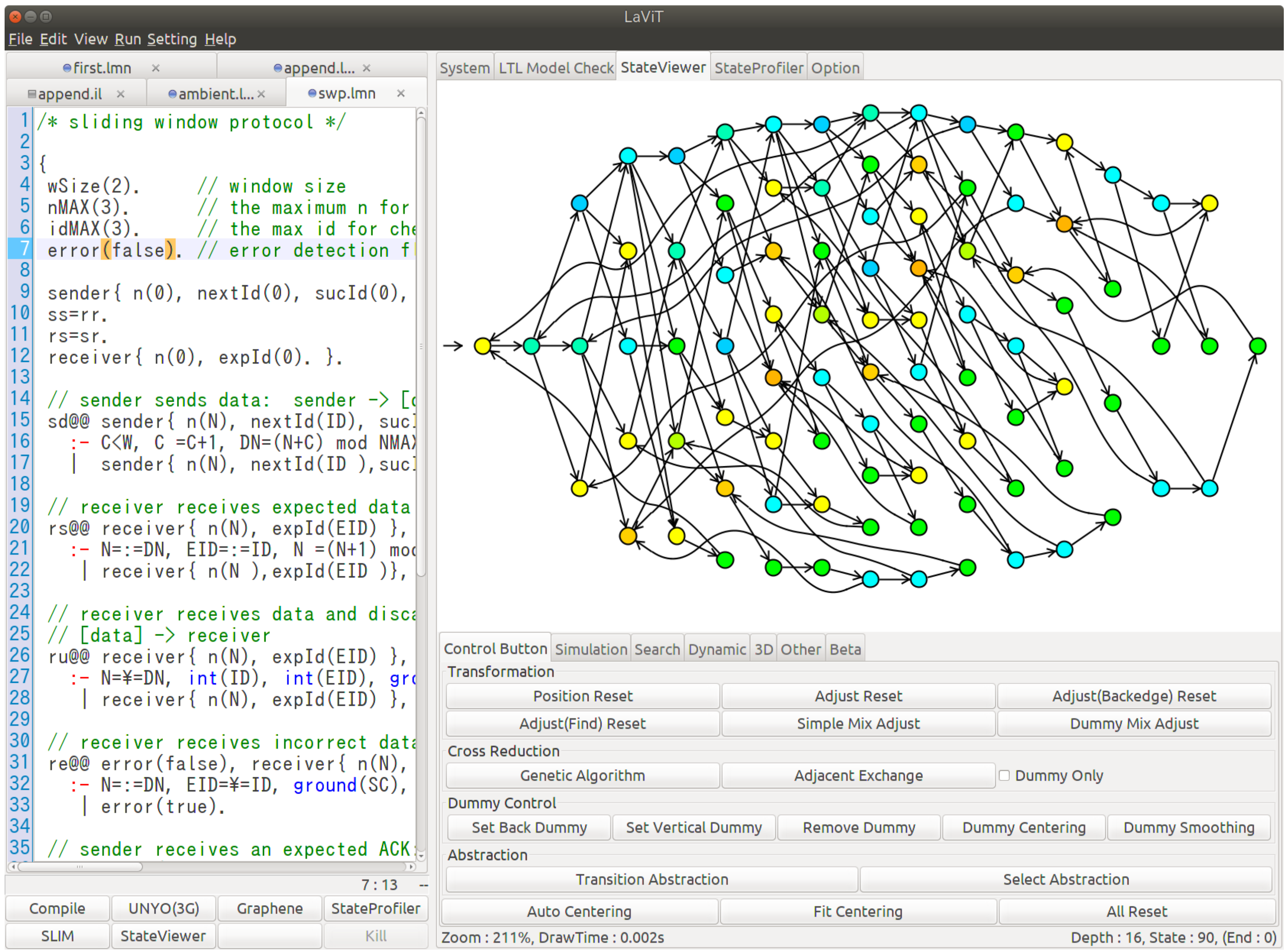
- Symmetry Reduction in Model Checking
- LMNtal
- Structural Congruence and Symmetry Reduction
- Process Abstraction
- Experiments
- Conclusion

Conclusion and Future Work

1. We developed a method for *automatically* reducing state space by *static* model abstraction that works in a concrete setting.
2. We showed the soundness of abstraction by reducing *equivalence relation induced by the abstraction* to *equivalence relation in the source language LMNtal*, i.e., without introducing additional formalisms.
3. We established a formal connection between
   - symmetry reduction grounded by the semantics of LMNtal and
   - standard theory of symmetry reduction based on symmetric group.

✦ We are interested in applying predicate abstraction to graph rewriting systems. It will allow us to more powerful symmetry reduction.

Thank you for the attention!



state transition diagram of a simple sliding windows protocol in LMNtal

(Various demos welcome during the conference; contact us.)