Topology and Property-Specific Verification and Synthesis (V&S) of *Parameterized Distributed Protocols (PDP)*

Ali Ebnenasir
aebnenas@mtu.edu

Department of Computer Science
College of Computing
Michigan Technological University
Houghton MI 49931

http://asd.cs.mtu.edu/

Modeling Parameterized Distributed Protocols (PDP)

Dijkstra's token passing:

π_2: Template process 2

Action$_0$: $\mathbf{x_0 = x_{N-1}}$ $\rightarrow$ $\mathbf{x_0 := x_{N-1} + 1}$

- Process P_i has a variable $x_i \in \mathbb{Z}_N = \{0, 1, \dots, N-1\}$
- N denotes the total number of processes
- Addition and subtraction are done in modulo N

self-disabling actions

π_1: Template process 1

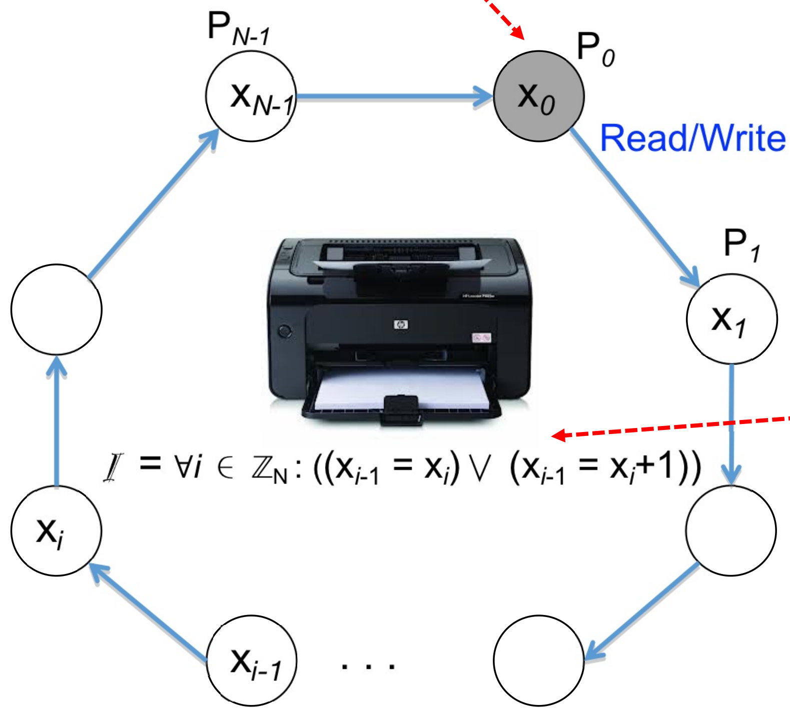
Action$_i$: $\mathbf{x_i \neq x_{i-1}}$ $\rightarrow$ $\mathbf{x_i := x_{i-1}}$

Legend:

Process/Node

Read from

Family 2: just one process



Set of good states; a.k.a. invariant.

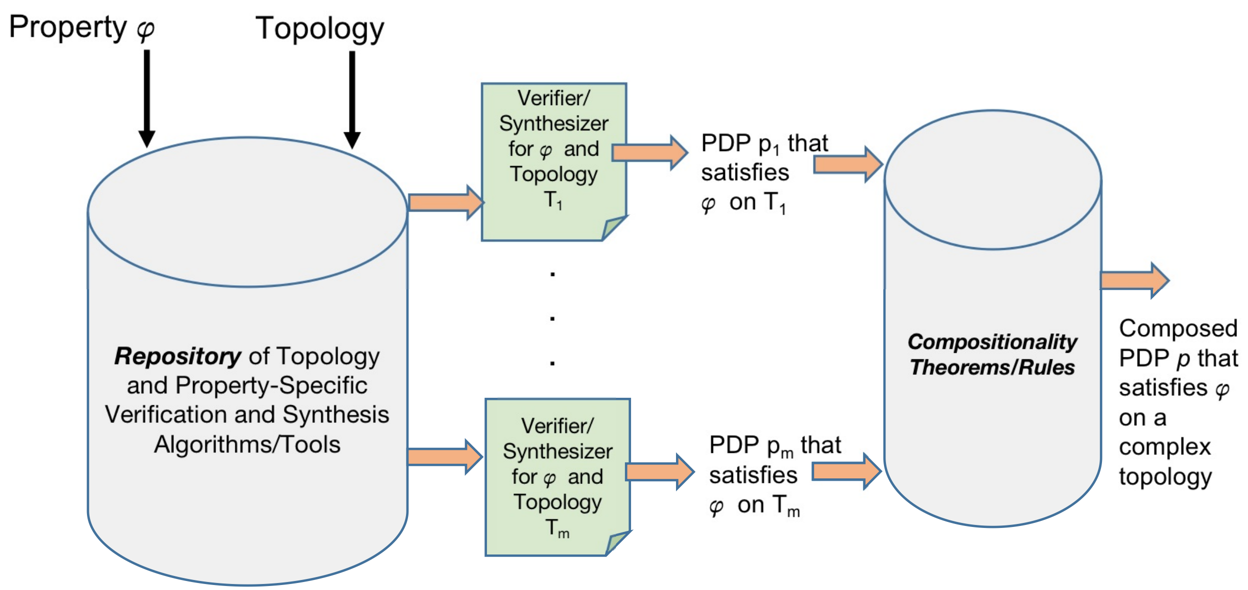
Family 1: N-1 *symmetric* processes

Significance of PDPs

From System on Chip, to multithreaded programs and large scale network protocols.

Vision: Topology and Property-Specific Verification and Synthesis (V&S) of PDP

- Solve V&S for a set of elementary topologies and determine necessary and/or sufficient conditions for their property-preserving composition.
  - Elementary topologies such as ring, chain, tree



Start With Self-Stabilizing Uni-directional Rings

- Topology = Uni-directional Ring (Uni-Ring)
  - Uni-directional topologies are important in wireless (mobile) networks
    - Some communication links may become uni-directional due to RF range constraints
  - Uni-directional ring (uni-ring) is a simple but useful model of computation
    - Information flows only in one direction.
  - Results can be useful for any topology that contains (uni-)rings
- Property = Self-stabilization (which entails livelock-freedom, deadlock-freedom)
  - Important applications in networks, multi-agent systems and socioeconomics

Related Work

- Verification of temporal logic properties for parameterized protocols is undecidable. [Apt and Kozen 1986]
- Verification problem remains undecidable even for uni-rings. [Suzuki 1988]

- What if we make the model stronger and focus on a specific property?
  - self-disabling, constant-space and deterministic processes
  - property: self-stabilization of symmetric uni-rings
  - conjunctive invariants

- Decidability of the V&S problems?

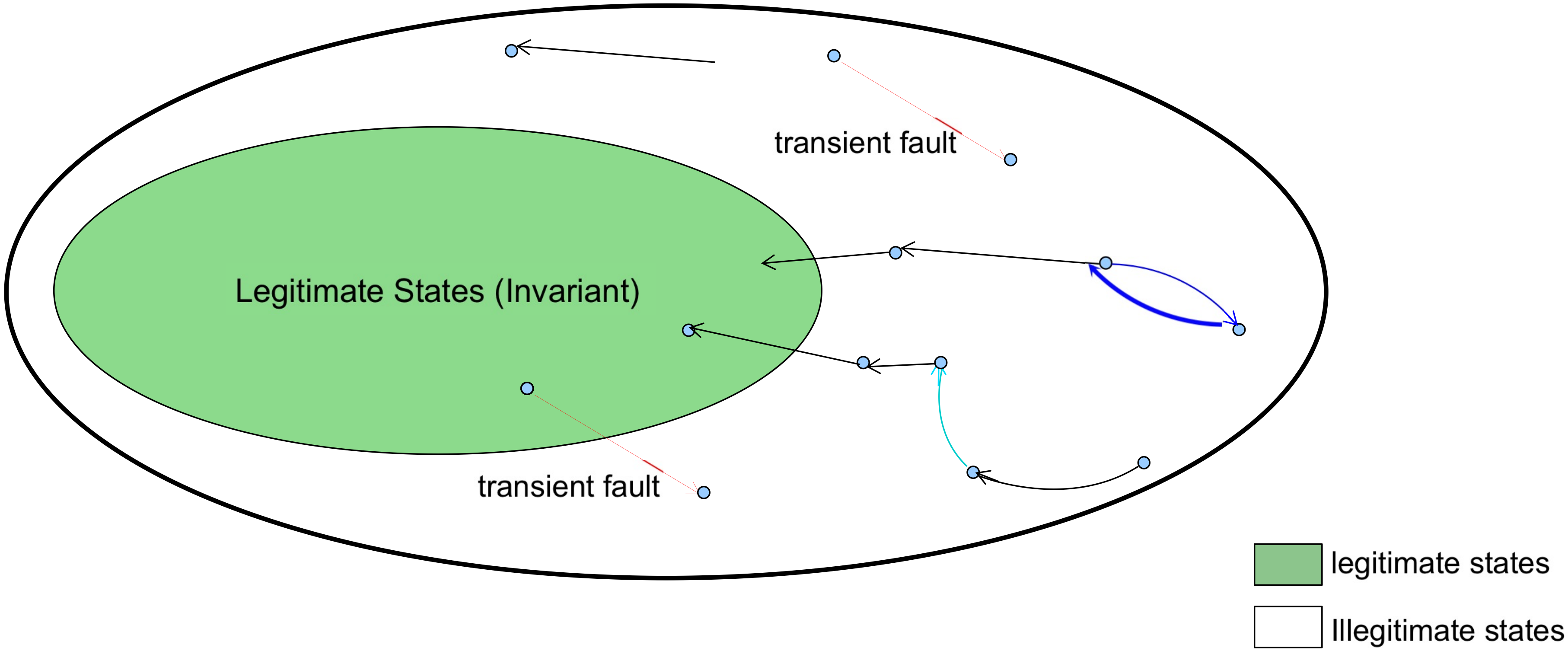
[Apt and Kozen 1986] K. R. Apt and D. C. Kozen. **Limits for automatic verification of finite-state concurrent systems**. Inf. Process. Lett. 22, 6 (1986), pp. 307–309.
[Suzuki 1988] I.Suzuki. 1988. Proving properties of a ring of finite-state machines. Inform. Process. Lett. 28, 4 (Jul. 1988), 213–214.

V&S of Parameterized Self-Stabilizing Symmetric Uni-Directional Rings with Constant-Space Processes

Self-Stabilization (SS)

"The ability of a distributed system to resume its legal behavior in a finite number of steps regardless of its initial configuration/state" [Dijkstra'74, Arora and Gouda'93]

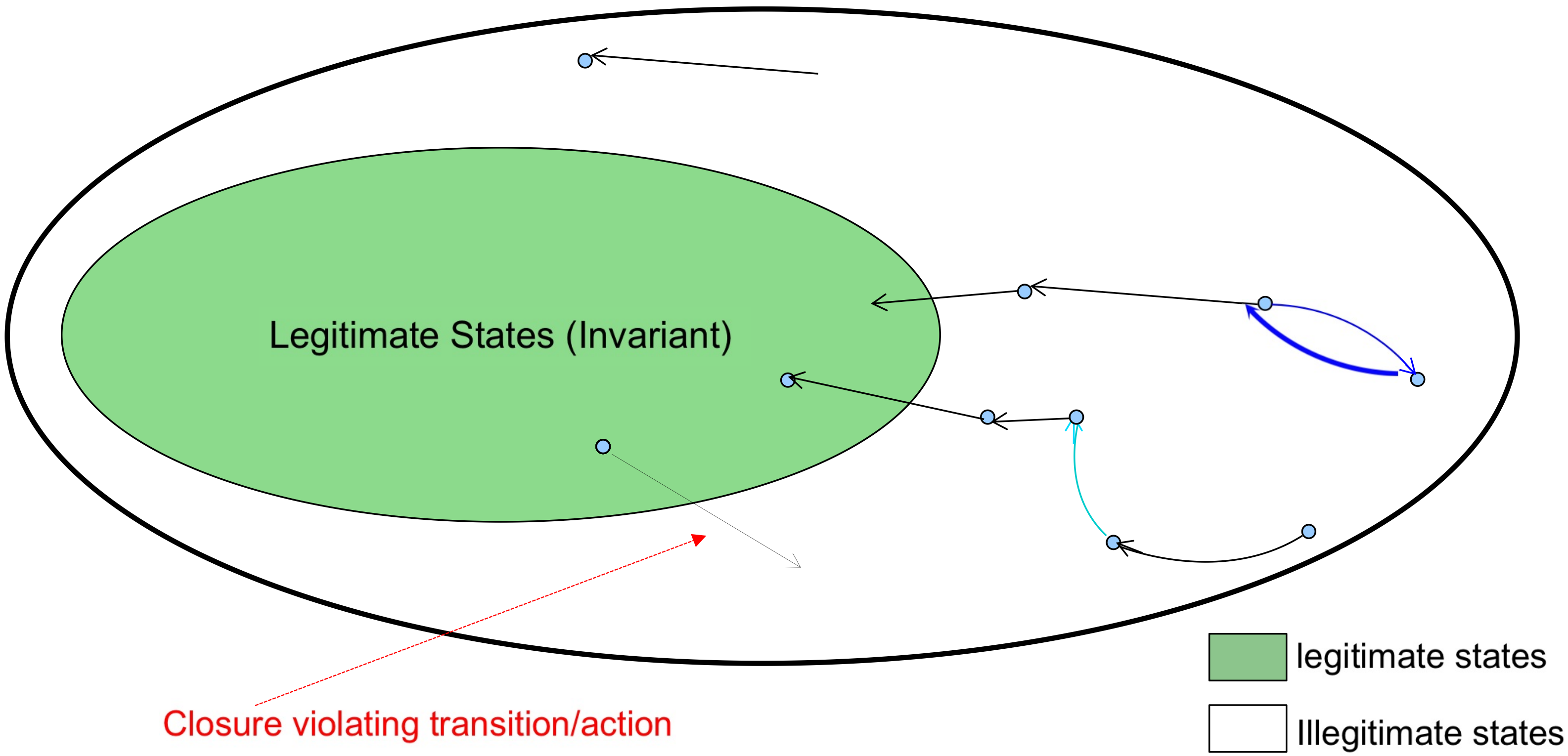
Self-stabilization = closure + convergence



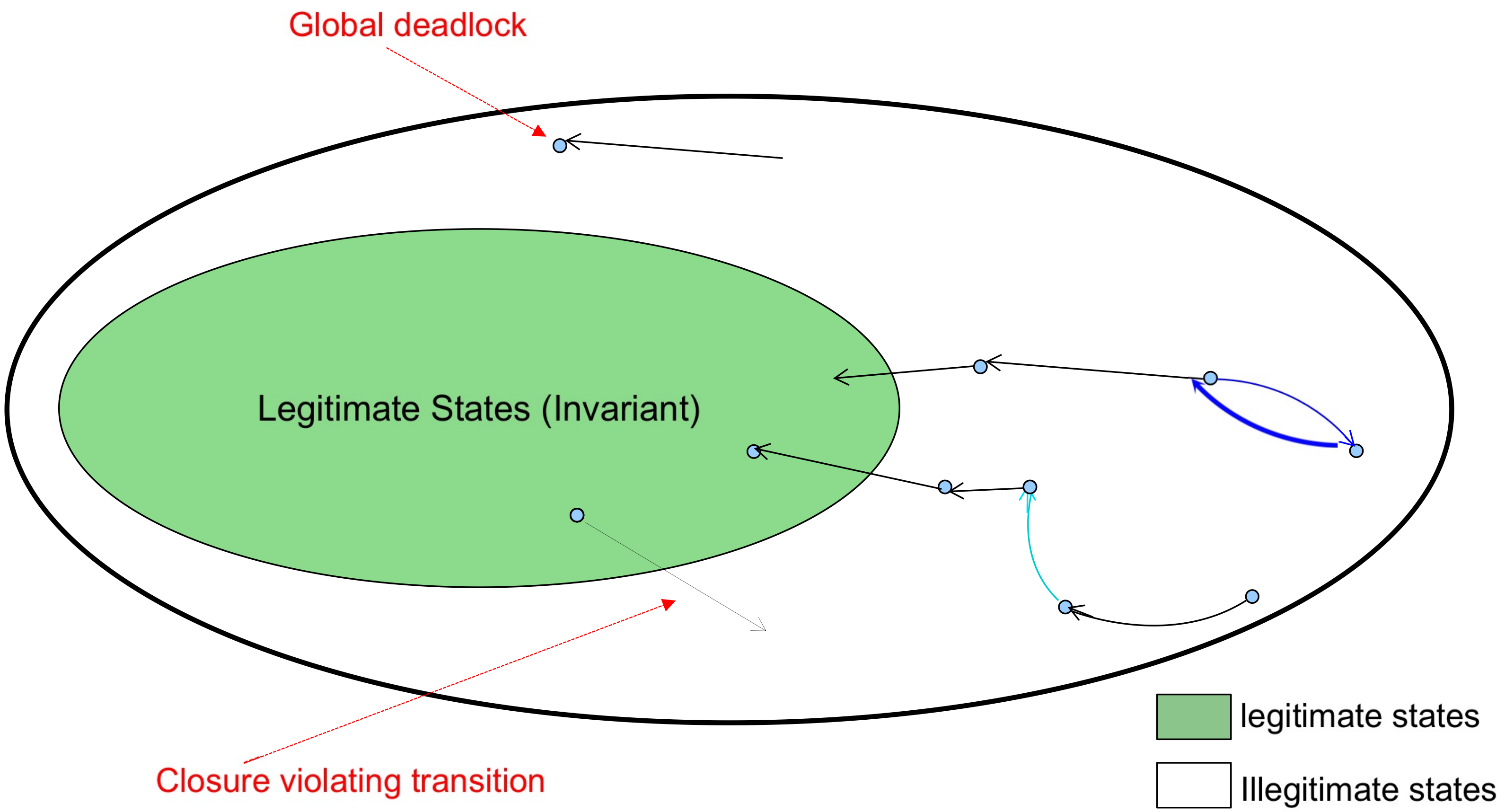
[1] E. W. Dijkstra, **Self-stabilizing systems in spite of distributed control**. *Communications of the ACM*, vol. 17, no. 11, pp. 643-644, 1974

[2] A. Arora and M. Gouda, **Closure and Convergence: A foundation of fault-tolerant computing**. *IEEE Transactions on Software Engineering*, vol 19, no. 11, pp. 1015-1027, 1993.

Design Complexity: Closure and Convergence



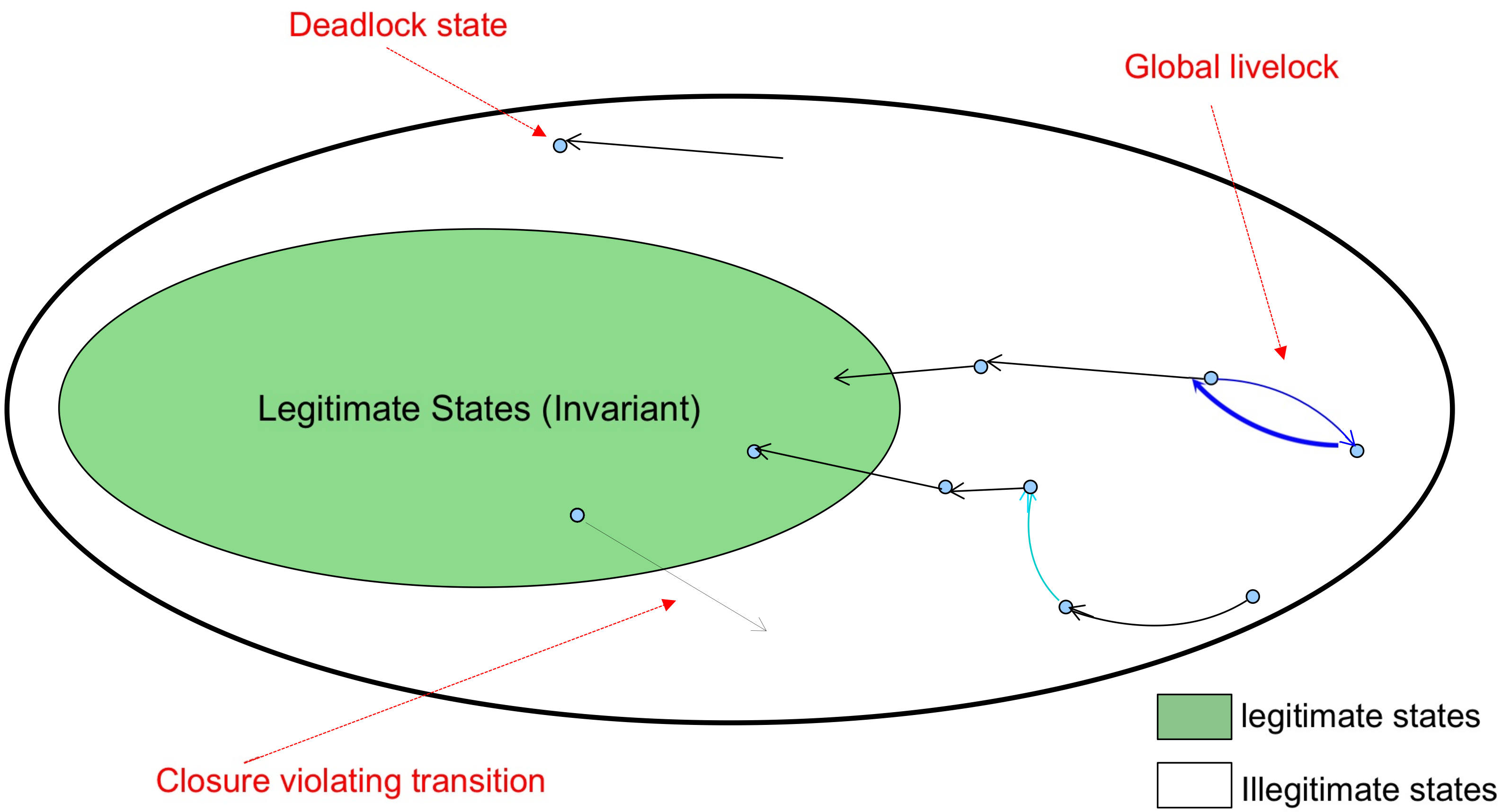
Design Complexity: Closure and Convergence



Verifying deadlock-freedom is decidable in rings. [Farahat & Ebnenasir, ICDCS'12]

Aly Farahat and Ali Ebnenasir, **Local Reasoning for Global Convergence in Parameterized Rings**, In Proceedings of the 32nd International Conference on Distributed Computing Systems (ICDCS), pages 496-505, 2012.

Design Complexity: Closure and Convergence



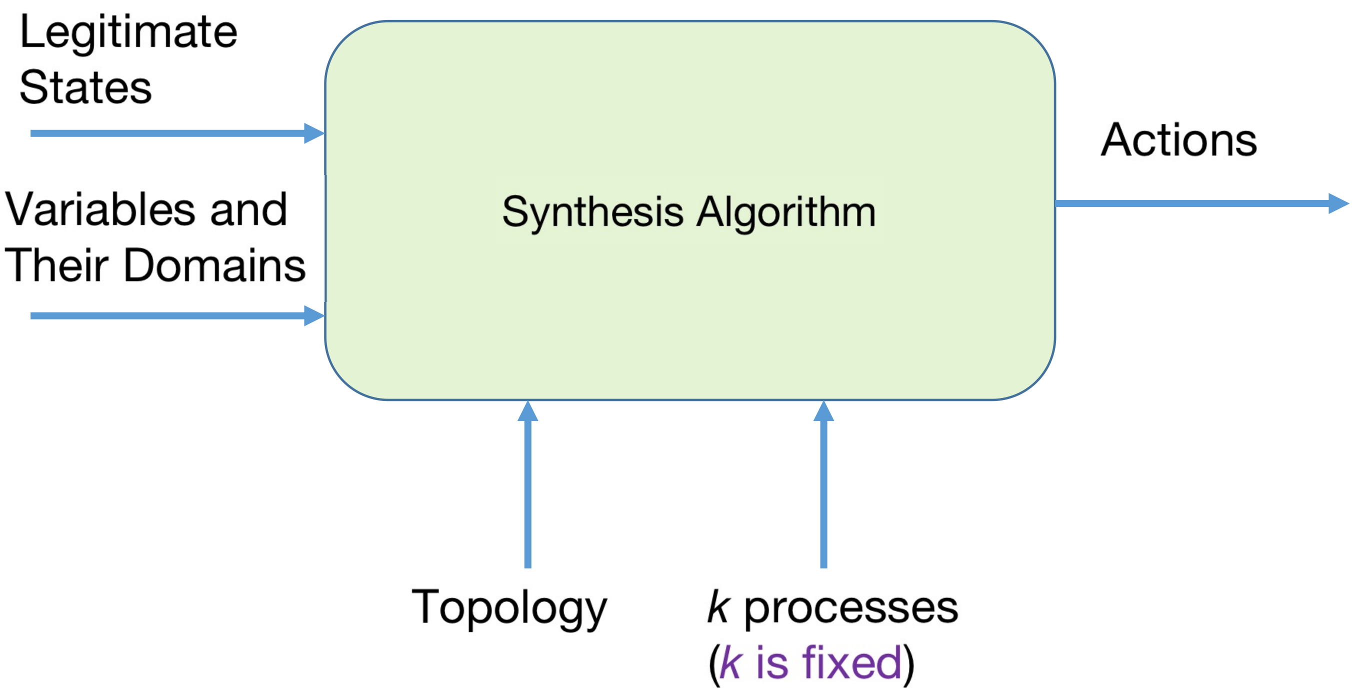
Verifying deadlock-freedom is decidable in rings. [Farahat & Ebnenasir, ICDCS'12]

Aly Farahat and Ali Ebnenasir, **Local Reasoning for Global Convergence in Parameterized Rings**, In Proceedings of the 32nd International Conference on Distributed Computing Systems (ICDCS), pages 496-505, 2012.

Challenges of Verification and Synthesis of SS

- To design self-stabilization, three intertwined problems must be solved:
  - Closure
  - Deadlock Freedom
  - Livelock Freedom

Our Previous Work on Synthesis



Protocon: A Framework for Verification and Synthesis (V&S) of Self-Stabilization

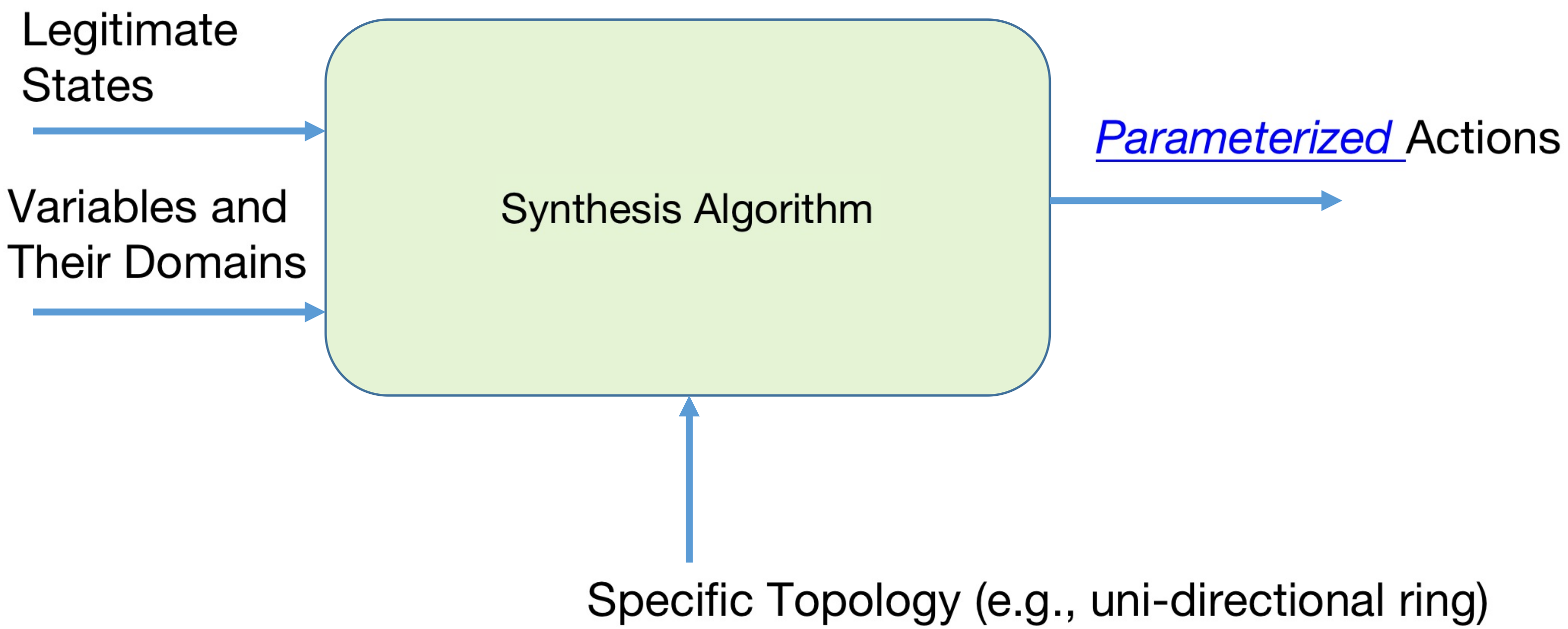
http://asd.cs.mtu.edu/projects/protocon/

Example: Coloring on Trees

```
// L = number of levels in the tree.
constant L := 3;
variable x[ (2^L-1) ] < 3;
process Root [i < 1] {
        read:  x[1];   read:  x[2];
        write: x[0];
   (future & silent)    ( x[0] != x[1] && x[0] != x[2] ) ;    }
process internalProcess[ j < (2^(L-1)-2) ] {
        let i := j + 1;  let parent_idx := (i-1)/2;
        let left_idx := 2*(i+1)-1;  let right_idx := 2*(i+1);
        read:  x[parent_idx];  read:  x[left_idx];
        read:  x[right_idx];
        write: x[i];
    (future & silent)
 ( x[parent_idx] != x[i] && x[i] != x[left_idx] && x[i] != x[right_idx] );
 synthesized action: ( x[i]==x[parent_idx] --> x[i]:=x[i]+1; );    }

process Leaf [j < (2^(L-1))]{
        let i := j + (2^(L-1)-1);
        let parent_idx := (i-1)/2;
        read:  x[parent_idx];
        write: x[i];
 synthesized action:
    ( x[i]==x[parent_idx] -->
                x[i]:=x[i]+1; );  }
```

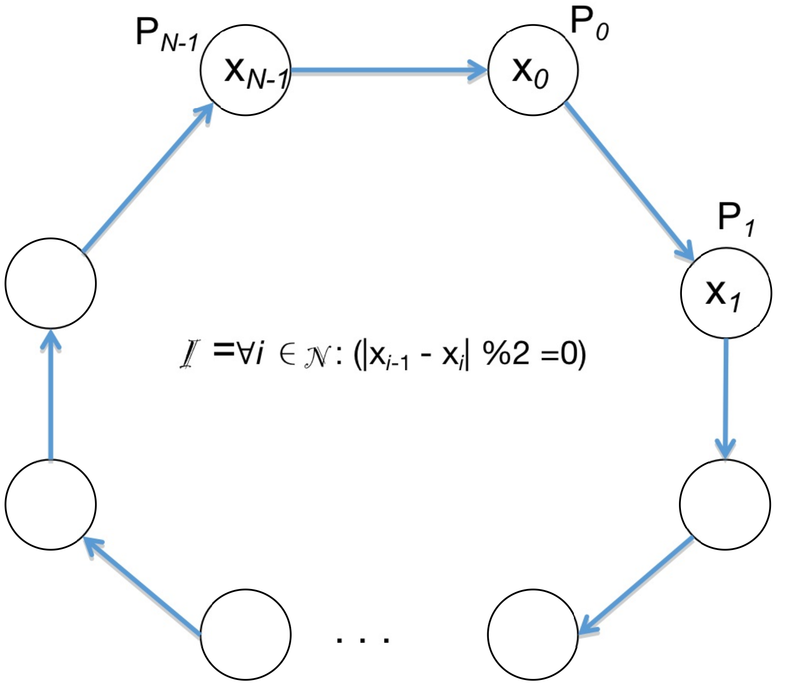
Synthesis of Self-Stabilizing PDP

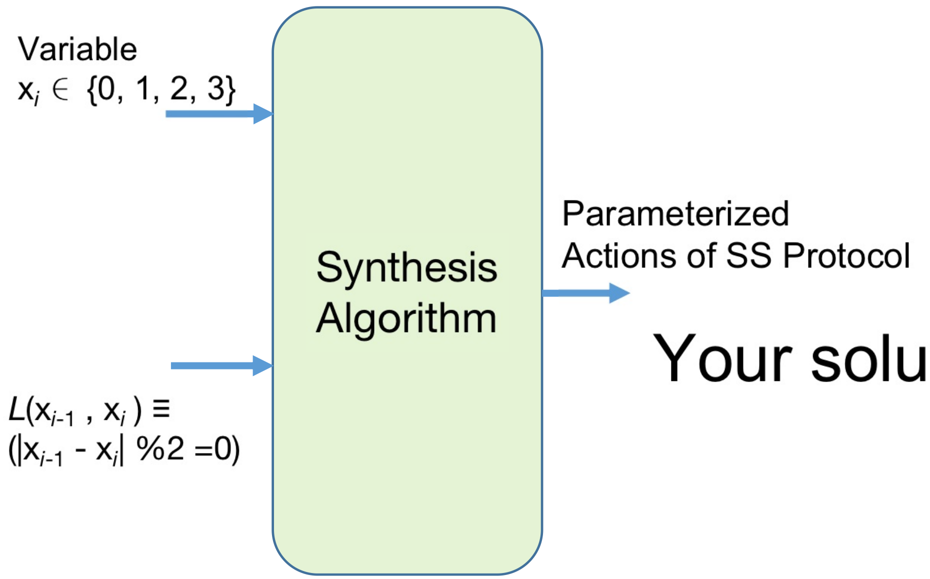


Example: Parity Protocol

Starting from any state, the *symmetric uni-ring* reaches states where all processes agree on a common odd/even parity.

$\mathcal{I} = \forall i \in \mathcal{N} : L(x_{i-1}, x_i)$ where $L(x_{i-1}, x_i) \equiv (|x_{i-1} - x_i| \,\%2 = 0)$ and $x_i \in \{0, 1, 2, 3\}$





Your solution?

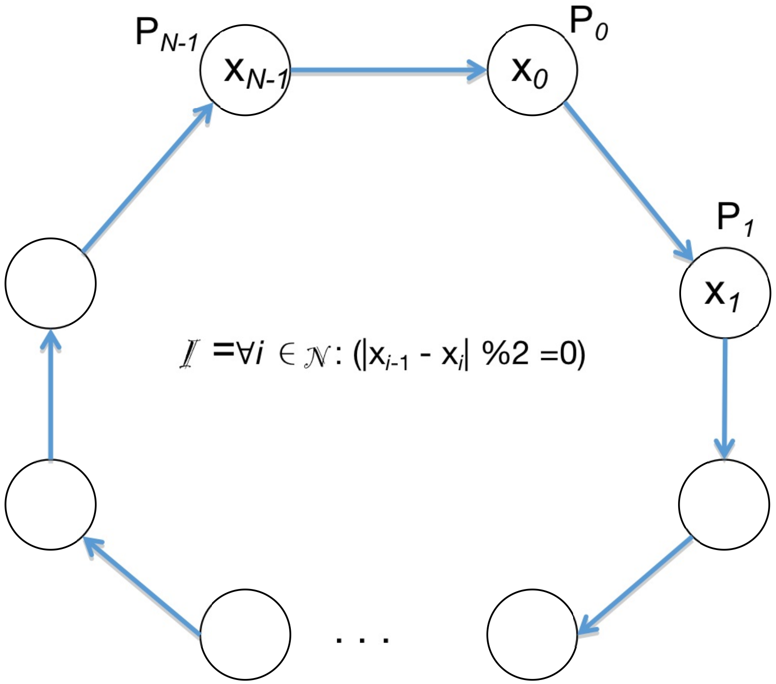
You might be tempted to say

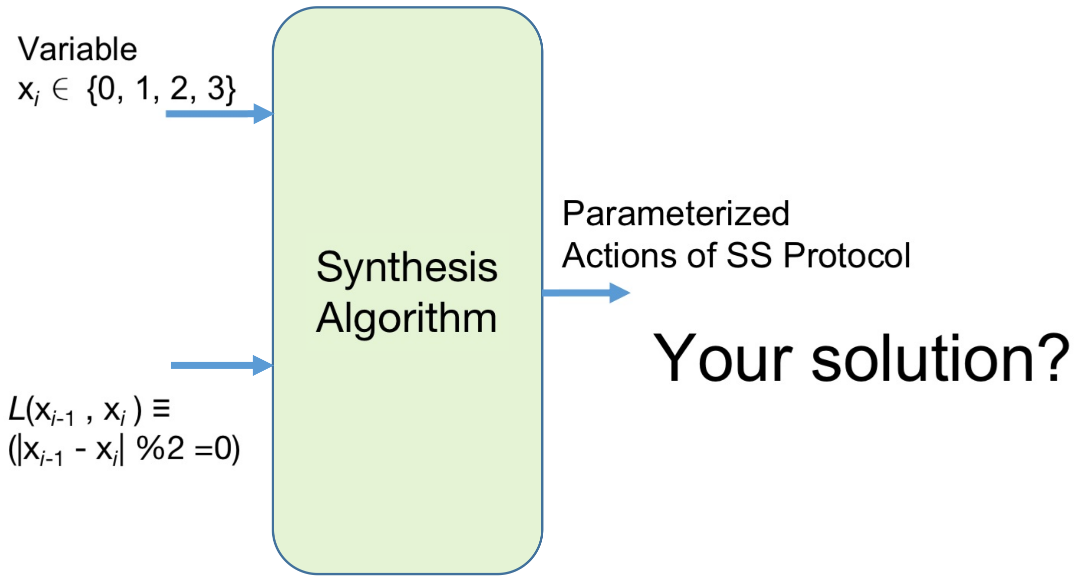
$(|x_{i-1} - x_i| \bmod 2) \neq 0 \rightarrow$ do something;

Example: Parity Protocol

Starting from any state, the *symmetric uni-ring* reaches states where all processes agree on a common odd/even parity.

$\mathcal{I} = \forall i \in \mathcal{N} : L(x_{i-1}, x_i)$ where $L(x_{i-1}, x_i) \equiv (|x_{i-1} - x_i| \,\%2 = 0)$ and $x_i \in \{0, 1, 2, 3\}$





You might be tempted to say

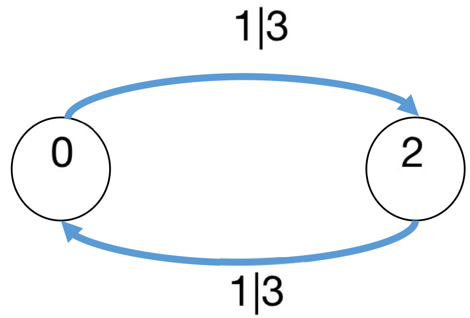
$(|x_{i-1} - x_i| \bmod 2) \neq 0 \rightarrow x_i := x_{i-1} \oplus_4 2$

Is it deadlock-free for all ring sizes outside $\mathcal{I}$?

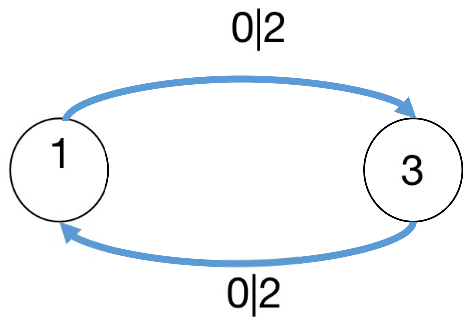
Is it livelock-free for all ring sizes outside $\mathcal{I}$?

Livelock in a Ring Size Four

- $(|x_{i-1} - x_i| \bmod 2) \neq 0 \rightarrow x_i := x_{i-1} \oplus_4 2$



- Initial state of livelock: < **2**, **0**, **3**, **1** >
- Interleaving: P_0, P_2, P_1, P_3, P_0, P_2, P_1, P_3



< **2**, **0**, **3**, **1** >,

<3, 0, 3, 1>,

<3, 0, 2, 1>,

<3, 1, 2, 1>, <3, 1, 2, 0>, <2, 1, 2, 0>, <2, 1, 3, 0>, <2, 0, 3, 0>

Undecidability of Verifying Livelock-Freedom

- **Theorem**: [SSS'13, ACM TOCL'19] **Verification of livelock-freedom** of PDPs with constant-space, self-disabling and deterministic processes on symmetric uni-ring is undecidable.

- Observation: States are repeated in a livelock
  - i.e., Sequences of actions taken in each segment of the ring must set the stage for the execution of another sequence of actions, and this goes forever.

[SSS'13] Alex Klinkhamer and Ali Ebnenasir, **Verfiying Livelock Freedom of Parameterized Rings and Chains**, 15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2013).
[ACM TOCL'19] A. Klinkhamer and A. Ebnenasir, "**On the verification of livelock-freedom and self-stabilization on parameterized rings**," ACM Transactions on Computational Logic, vol. 20, no. 3, pp. 1–36, 2019.

Proposed Approach:
Local Characterization of Global Failures

Characterize global failures (e.g., livelock) in local state space of the template process in a topology-specific fashion.

Absence of local characterizations may imply correctness of PDP

- *Methodology:* Search for local characterization of global failures in local state space of template processes.

Graph-Theoretic Representations

- Facilitate reasoning in the local state space of the template process; i.e., local reasoning for global correctness.

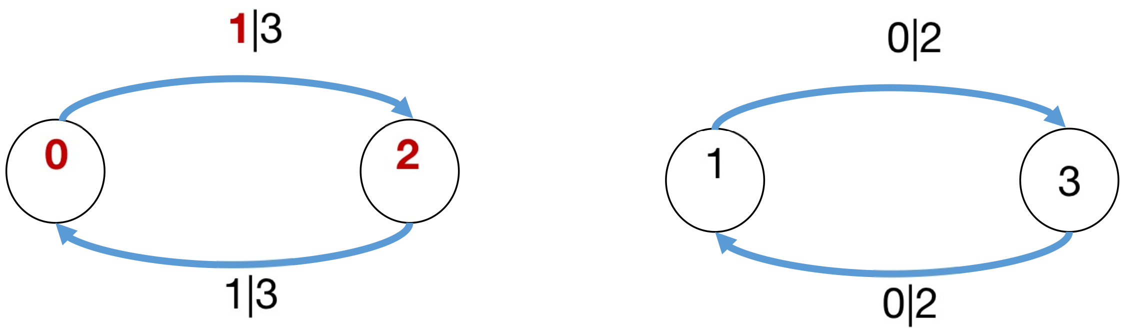
- Parameterized Actions $\rightarrow$ Action Graph
- State predicates $\rightarrow$ Locality Graph

Actions as Action Graphs

Action Graph

- Acions of a protocol can be represented as a *labeled directed multi-graph* in the local state space of the template process
- *Vertices*: values in the domain of $x_i \in \{0, 1, 2, 3\}$
- *Arcs*: each arc (a, b, c) represents a local update of x_i to c if x_{i-1}=a and x_i = b
  - E.g., (**0**, **1**, **2**) means if x_{i-1}=0 and x_i = 1 then update x_i to 2

$$(|x_{i-1} - x_i| \bmod 2) \neq 0 \rightarrow x_i := x_{i-1} \oplus_4 2$$

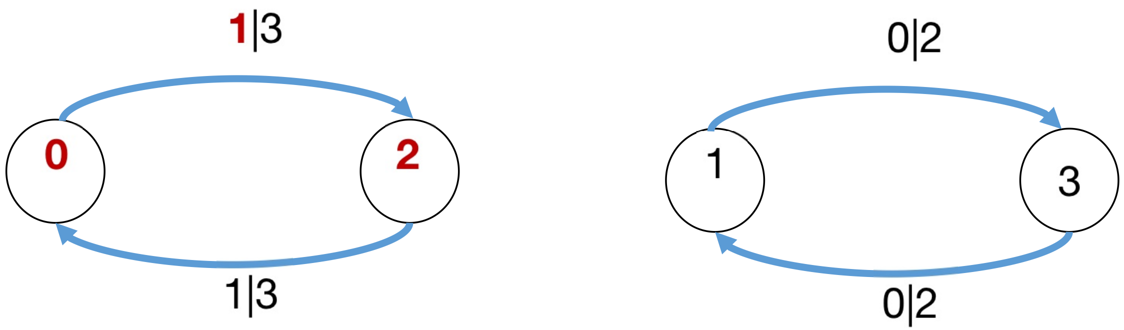


Action Graph

- Acions of a protocol can be represented as a *labeled directed multi-graph* in the local state space of the template process
- *Vertices*: values in the domain of $x_i \in \{0, 1, 2, 3\}$
- *Arcs*: each arc (a, b, c) represents a local update of x_i to c if x_{i-1}=a and x_i = b
  - E.g., (**0**, **1**, **2**) means if $x_{i-1}=0$ and $x_i = 1$ then update x_i to 2

$$(|x_{i-1} - x_i| \bmod 2) \neq 0 \rightarrow x_i := x_{i-1} \oplus_4 2$$

(**0**, **1**, **2**)



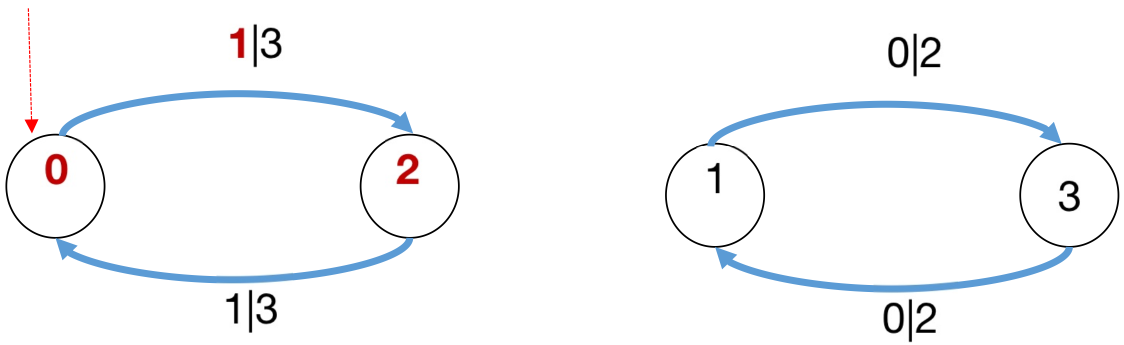
Action Graph

- Acions of a protocol can be represented as a *labeled directed multi-graph* in the local state space of the template process
- *Vertices*: values in the domain of $x_i \in \{0, 1, 2, 3\}$
- *Arcs*: each arc (a, b, c) represents a local update of x_i to c if x_{i-1}=a and x_i = b
  - E.g., (**0**, **1**, **2**) means if x_{i-1}=0 and x_i = 1 then update x_i to 2

$$(|x_{i-1} - x_i| \bmod 2) \neq 0 \rightarrow x_i := x_{i-1} \oplus_4 2$$

(**0**, **1**, **2**)

x_{i-1}



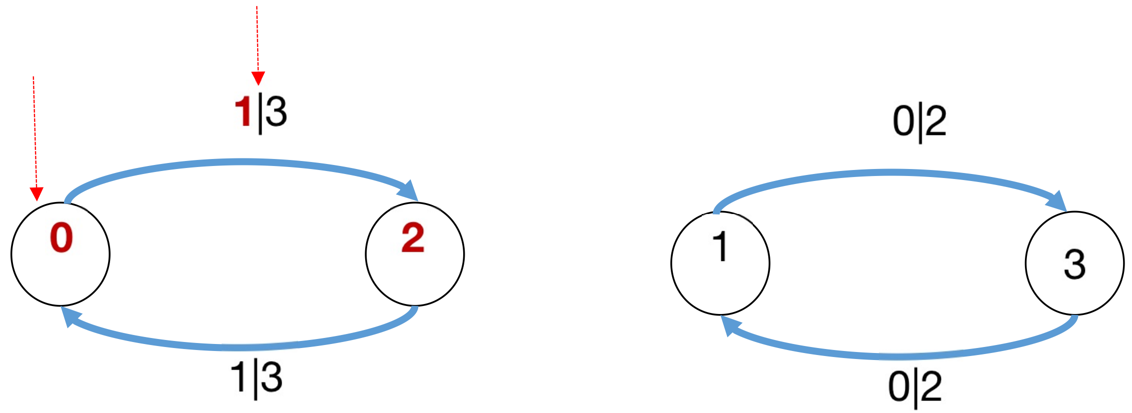
Action Graph

- Acions of a protocol can be represented as a *labeled directed multi-graph* in the local state space of the template process
- *Vertices*: values in the domain of $x_i \in \{0, 1, 2, 3\}$
- *Arcs*: each arc (a, b, c) represents a local update of x_i to c if x_{i-1}=a and x_i = b
  - E.g., (**0**, **1**, **2**) means if x_{i-1}=0 and x_i = 1 then update x_i to 2

$$(|x_{i-1} - x_i| \bmod 2) \neq 0 \rightarrow x_i := x_{i-1} \oplus_4 2$$

(**0**, **1**, **2**)

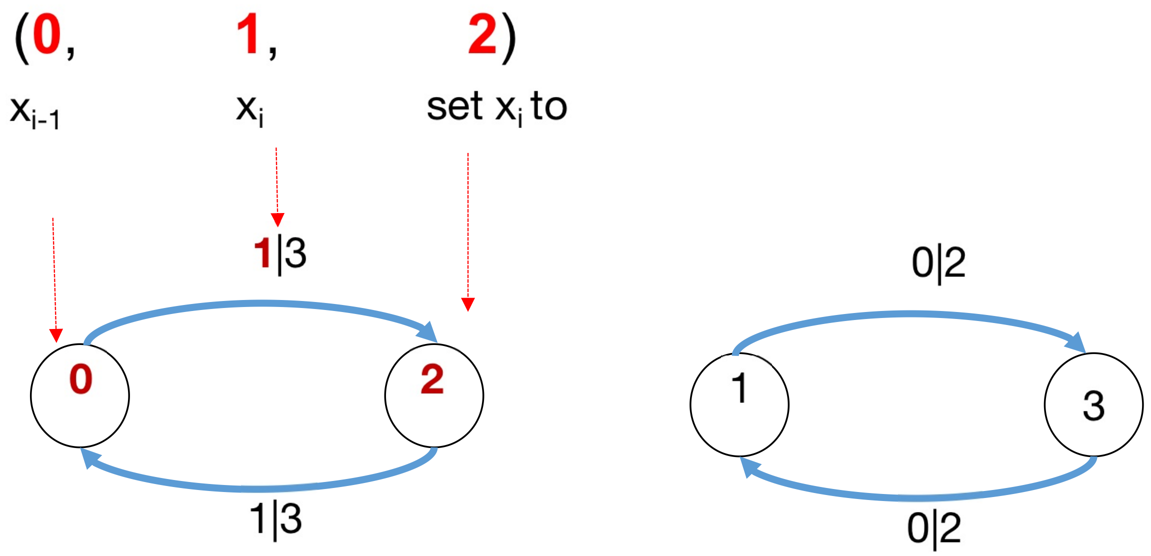
x_{i-1} x_i



Action Graph

- Acions of a protocol can be represented as a *labeled directed multi-graph* in the local state space of the template process
- *Vertices*: values in the domain of $x_i \in \{0, 1, 2, 3\}$
- *Arcs*: each arc (a, b, c) represents a local update of x_i to c if x_{i-1}=a and x_i = b
  - E.g., (**0**, **1**, **2**) means if x_{i-1}=0 and x_i = 1 then update x_i to 2

$$(|x_{i-1} - x_i| \bmod 2) \neq 0 \rightarrow x_i := x_{i-1} \oplus_4 2$$

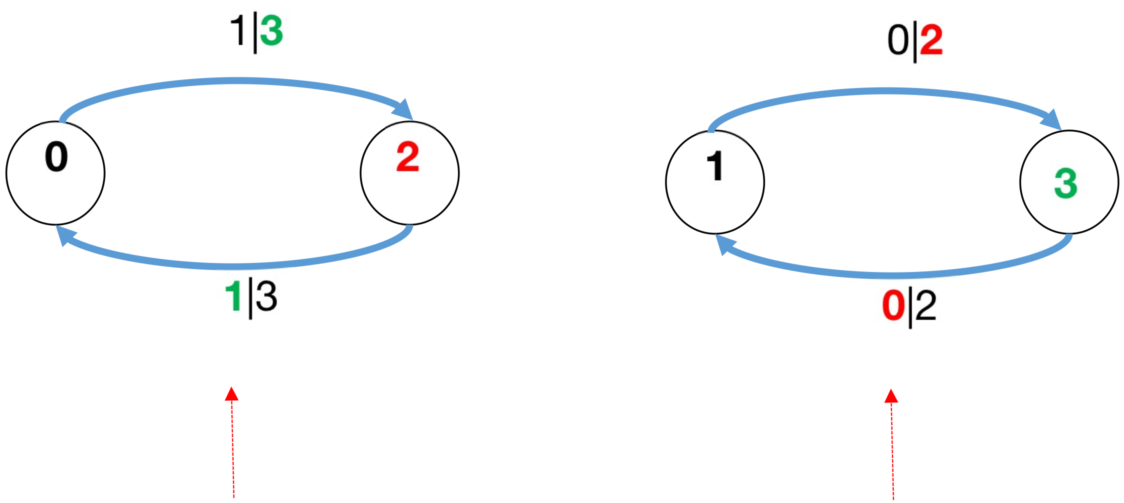


Propagations as Closed Walks

Closed Walks in Action Graph

- Closed walk/Propagation: sequence of consecutive actions

$$A_0 : \ (|x_{i\text{-}1} - x_i \,|\ \mathrm{mod}\ 2) \neq 0 \ \rightarrow x_i := \ x_{i\text{-}1} \oplus_4 \ 2$$

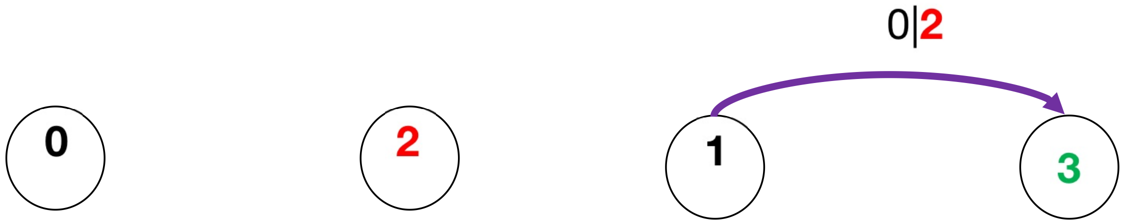


Closed Walks 2

Closed Walks 1

Enabling Closed Walks

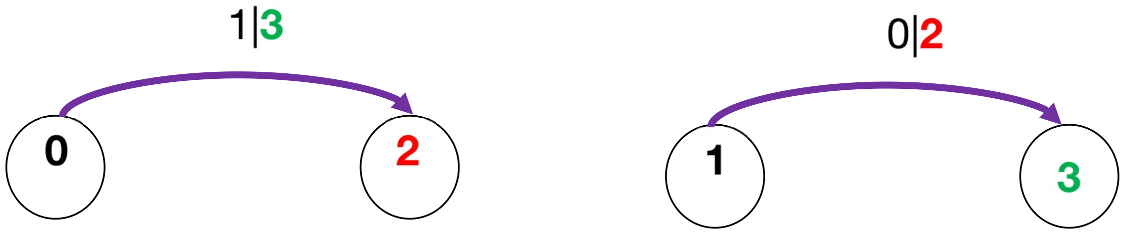
- A closed walk enabling another



Closed walk 1: (1, **2**, **3**),
Closed walk 2:

Enabling Closed Walks

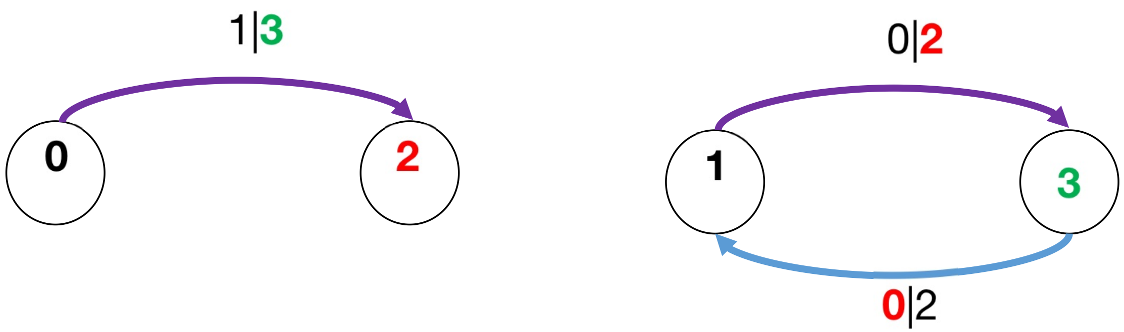
- A closed walk enabling another



Closed walk 1: (1, **2**, **3**),
Closed walk 2: (0, **3**, **2**),

Enabling Closed Walks

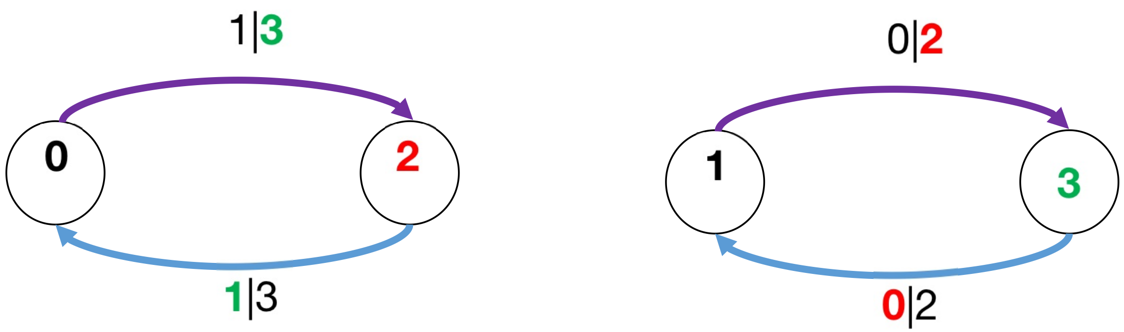
- A closed walk enabling another



Closed walk 1: (1, 2, 3), (3, 0, 1)
Closed walk 2: (0, 3, 2),

Enabling Closed Walks

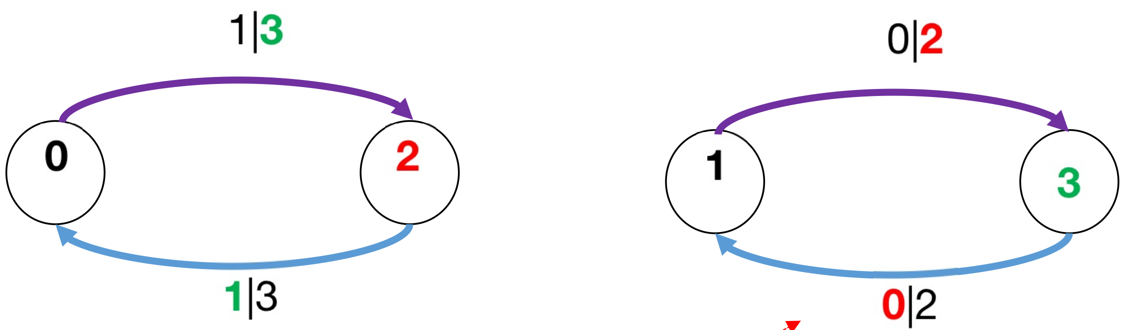
- A closed walk enabling another



Closed walk 1: (1, **2**, **3**), (3, **0**, **1**)
Closed walk 2: (0, **3**, **2**), (2, **1**, **0**)

Enabling Closed Walks

- A closed walk enabling another

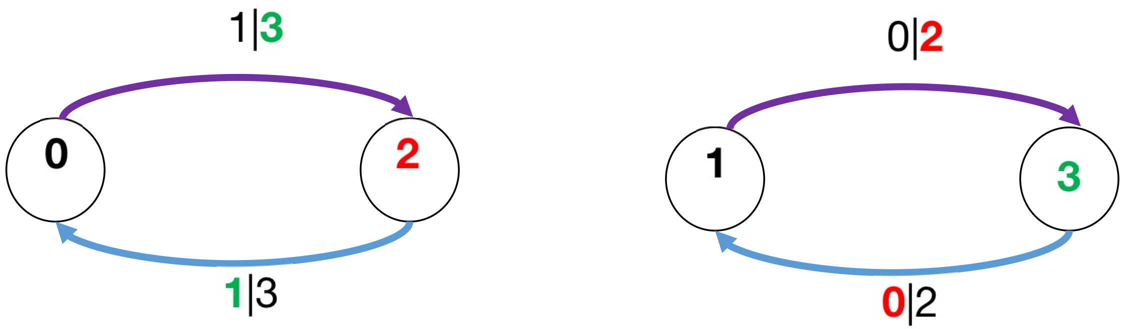


Closed walk 1 enables closed walk 2.

Closed walk 1: (1, **2**, **3**), (3, **0**, **1**)
Closed walk 2: (0, **3**, **2**), (2, **1**, **0**)

Enabling Closed Walks

- A closed walk enabling another

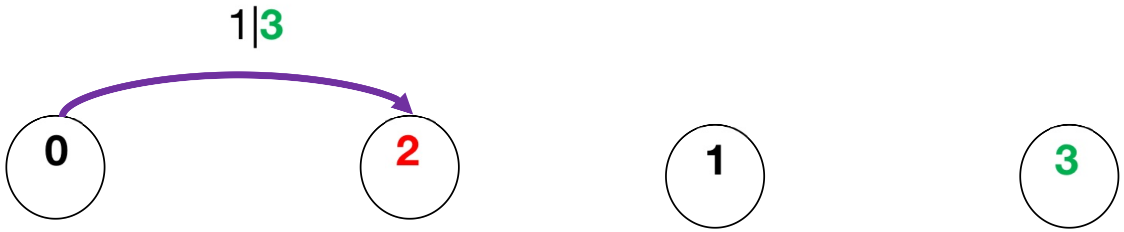


A closed walk of length *n* enables another closed walk of length *n*
iff
j-th action of the first walk enables the *j*-th action of
the second walk, for 1≤ j ≤ *n*

Closed walk 1: (1, **2**, **3**), (3, **0**, **1**)
Closed walk 2: (0, **3**, **2**), (2, **1**, **0**)

Circularly Enabling Closed Walks

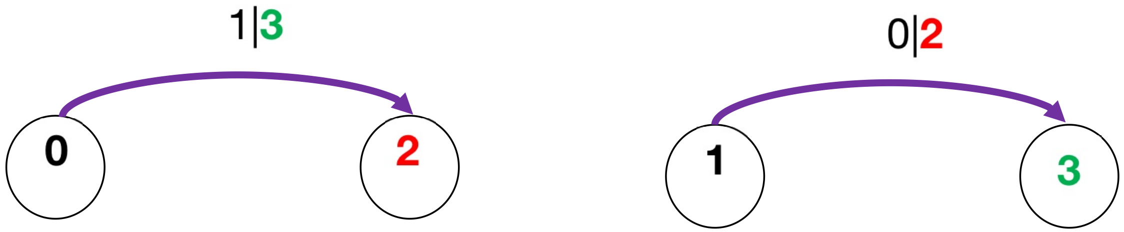
- Closed walk 2 also enables closed walk 1.



Closed walk 1:
Closed walk 2: (0, 3, 2),

Circularly Enabling Closed Walks

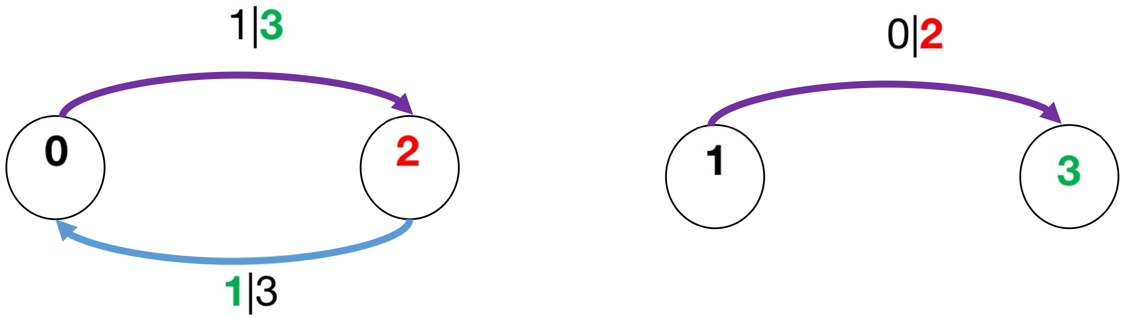
- Closed walk 2 also enables closed walk 1.



Closed walk 1: (1, **2**, **3**),
Closed walk 2: (0, **3**, **2**),

Circularly Enabling Closed Walks

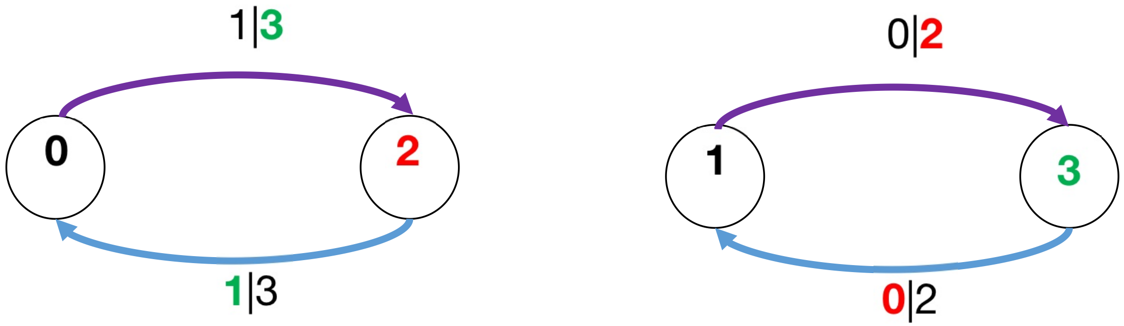
- Closed walk 2 also enables closed walk 1.



Closed walk 1: (1, **2**, **3**),
Closed walk 2: (0, **3**, **2**), (2, **1**, **0**)

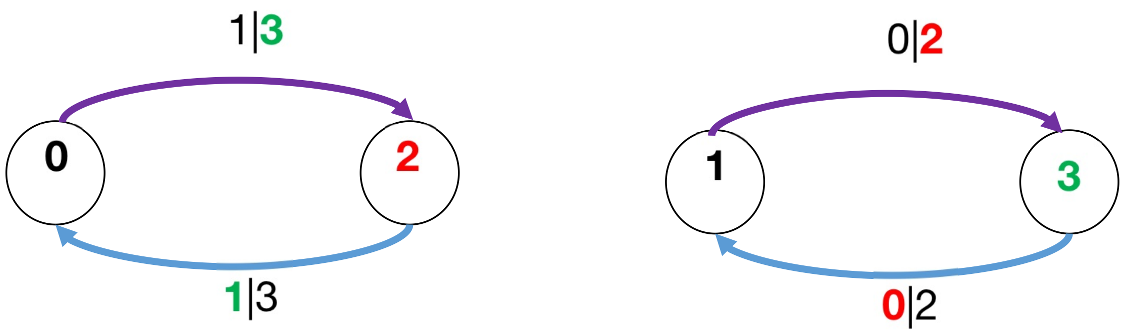
Circularly Enabling Closed Walks

- Closed walk 2 also enables closed walk 1.



Closed walk 1: (1, **2**, **3**), (3, **0**, **1**)
Closed walk 2: (0, **3**, **2**), (2, **1**, **0**)

Circularly Enabling Closed Walks



2 *circularly* enabling closed walks, each of length 2.

Closed walk 1: (1, **2**, **3**), (3, **0**, **1**)
Closed walk 2: (0, **3**, **2**), (2, **1**, **0**)

- A set of updates in a segment of the ring enables another set of updates and vice versa.
- Intuitively, we are observing same states being repeated.

Local Characterization of Global Livelocks

- **Theorem**: [SSS'13, ACM TOCL'19]

  A unidirectional ring of symmetric processes has a livelock for a ring size ($m \times n$)

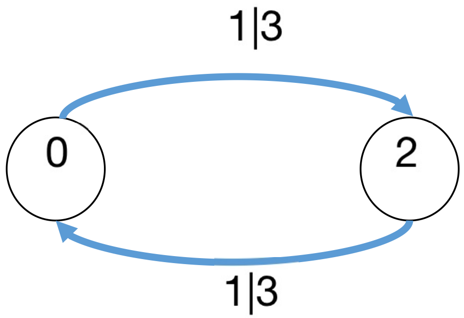
  if and only if

  There are m closed walks, each of length n, in the action graph that enable each other circularly

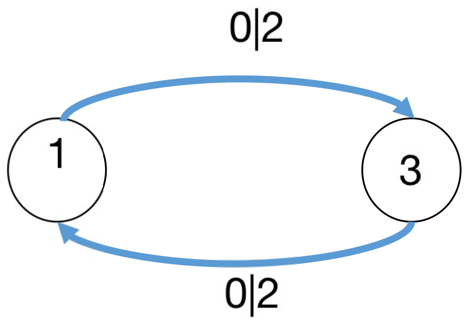
[SSS'13] Alex Klinkhamer and Ali Ebnenasir, **Verfiying Livelock Freedom of Parameterized Rings and Chains**, 15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2013).
[ACM TOCL'19] A. Klinkhamer and A. Ebnenasir, "**On the verification of livelock-freedom and self-stabilization on parameterized rings**," ACM Transactions on Computational Logic, vol. 20, no. 3, pp. 1–36, 2019.

Semi-Algorithm for Livelock Detection and Construction

- A_0 : $(|x_{i-1} - x_i| \bmod 2) \neq 0 \rightarrow x_i := x_{i-1} \oplus_4 2$
- Closed walk 1: (1, **2**, 3), (3, **0**, 1)
- Closed walk 2: (0, **3**, 2), (2, **1**, 0)
  - **m=n=2; ring size is 4.**



- Initial state of livelock: < **2**, **0**, **3**, **1** >
- Interleaving: P_0, P_2, P_1, P_3, P_0, P_2, P_1, P_3



< **2**, **0**, **3**, **1** >,

<3, 0, 3, 1>,

<3, 0, 2, 1>,

<3, 1, 2, 1>, <3, 1, 2, 0>, <2, 1, 2, 0>, <2, 1, 3, 0>, <2, 0, 3, 0>

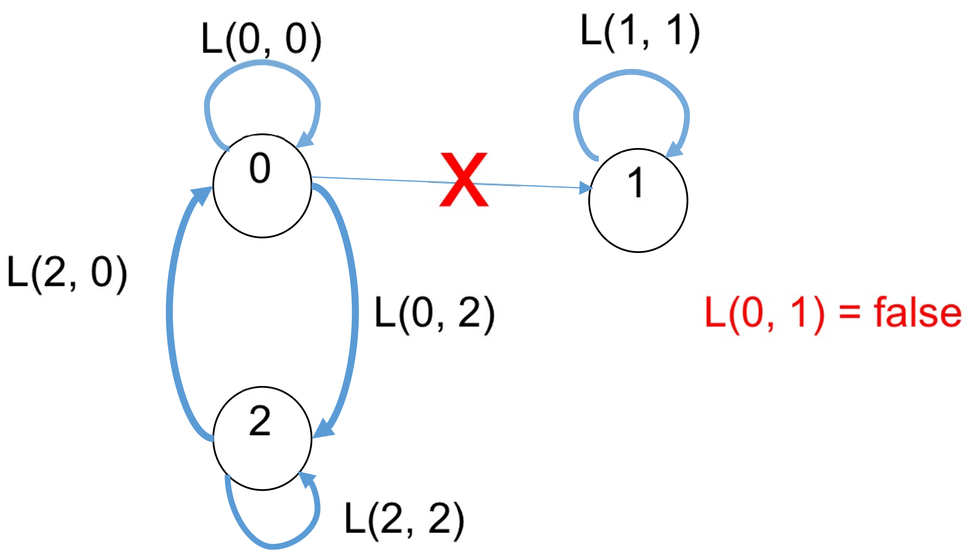
State Predicates as Locality Graphs

Locality Graph of Parity Protocol

- *Vertices*: values in domain of x_i
- *Arcs*: there is an arc from vertex *a* to *b iff L(a, b)* holds.

$\mathcal{I} = \forall i \in \mathbb{Z}_N : L(x_{i-1}, x_i)$ where $L(x_{i-1}, x_i) \equiv (|x_{i-1} - x_i| \% 2 = 0)$

$x_i \in \mathbb{Z}_3 = \{0, 1, 2\}$; i.e., constant-space processes

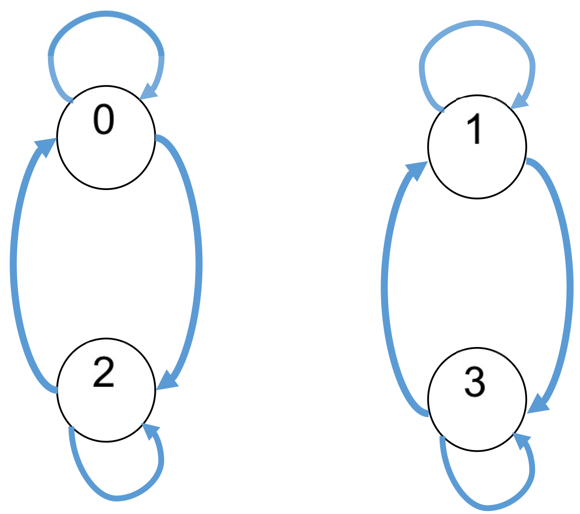


Locality Graph of Parity Protocol

- *Vertices*: values in domain of x_i
- *Arcs*: there is an arc from vertex *a* to *b iff L(a, b)* holds.

$\mathcal{I} = \forall i \in \mathbb{Z}^+ : L(x_{i-1}, x_i)$ where $L(x_{i-1}, x_i) \equiv (|x_{i-1} - x_i| \% 2 = 0)$

$x_i \in \mathbb{Z}_4 = \{0, 1, 2, 3\}$



Synthesis of SS on Uni-Ring

$\mathcal{I} = \forall i \in \mathcal{N} : L(x_{i-1}, x_i)$
Specifies legitimate states.
Variable x_i has a
constant domain.

Create Locality Graph of $L(x_{i-1}, x_i)$

Synthesize Action Graph of a PDP Self-stabilizing to $\forall i \in \mathcal{N} : L(x_{i-1}, x_i)$

Parameterized Actions

self-stabilizing for

1. an arbitrary number of processes, and
2. **domain size remains constant**.

Decidability of Synthesis

- Theorem: [IEEE TSE 2019]

  Synthesizing SS PDPs on symmetric uni-rings <u>is decidable</u> for deterministic, *constant-space* and self-disabling processes.

- Theorem: (necessary and sufficient condition) [IEEE TSE 2019]

There is a PDP p that self-stabilizes to $\mathcal{I} = \forall i \in \mathcal{N} : L(x_{i-1}, x_i)$

<u>*if and only if*</u>

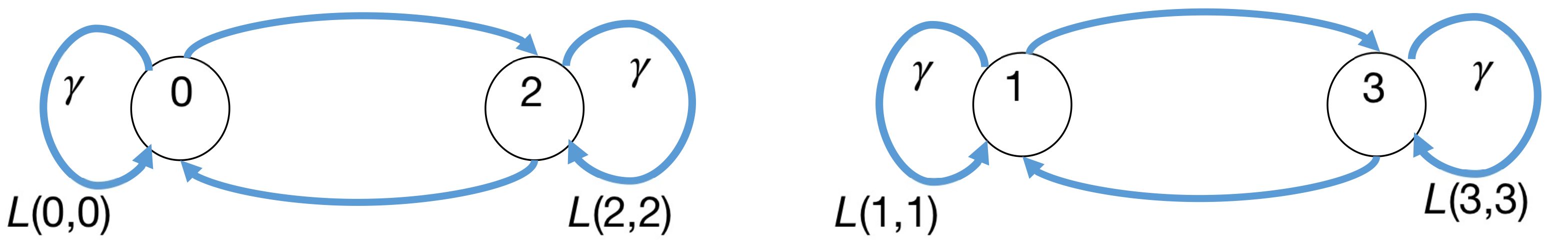
There is some value γ in the domain of x_i such that $L(\gamma, \gamma)$ holds (i.e., self-loops), and the action graph of p is a directed spanning tree rooted at γ.

[TSE 2019] Ali Ebnenasir and Alex Klinkhamer, **Topology-specific synthesis of self stabilizing parameterized systems with constant-space processes**, *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 614–629, 2019.

Synthesis Algorithm

- **Step 1**: Create the locality graph of

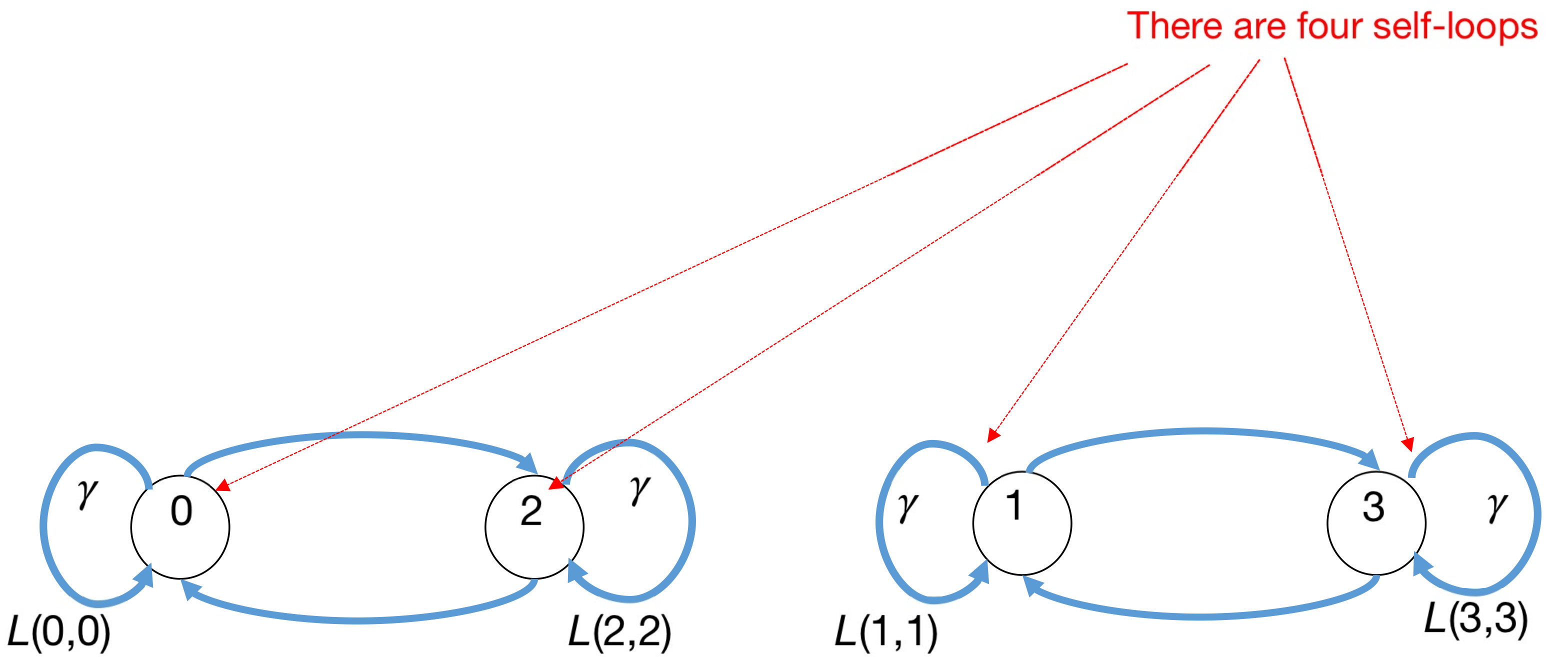
  $L(x_{i-1}, x_i) = ((|x_{i-1} - x_i| \bmod 2) = 0)$, where $x_i \in \{0, 1, 2, 3\}$



Synthesis Algorithm

- **<u>Step 1</u>**: Create the locality graph of

  $L(x_{i-1}, x_i) = ((|x_{i-1} - x_i| \bmod 2) = 0)$, where $x_i \in \{0, 1, 2, 3\}$



Synthesis Algorithm

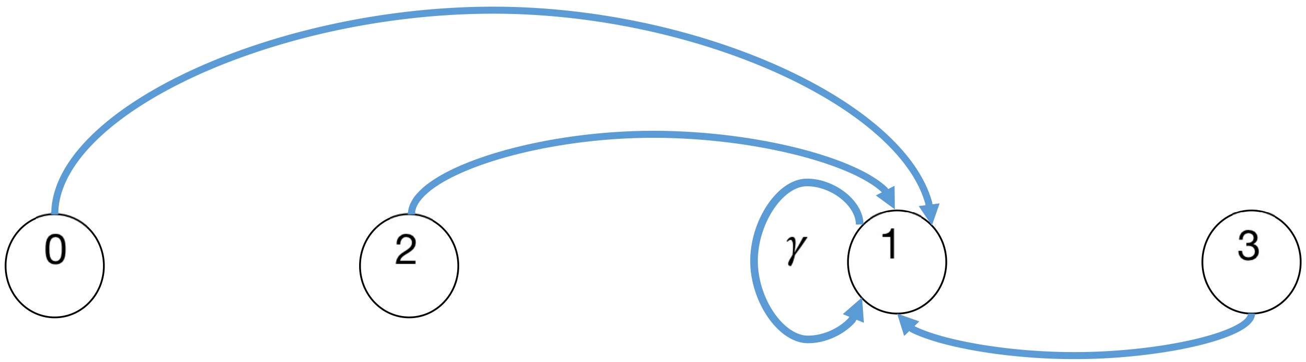
- **Step 2**: Induce subgraph *L'* using arcs that participate in some cycle
  - E.g., in the case of parity, all arcs participate in some cycle; hence kept

Synthesis Algorithm

- **<u>Step 3</u>**: arbitrarily pick a node γ and form a spanning tree with γ as its root
  - Backward reachability from root

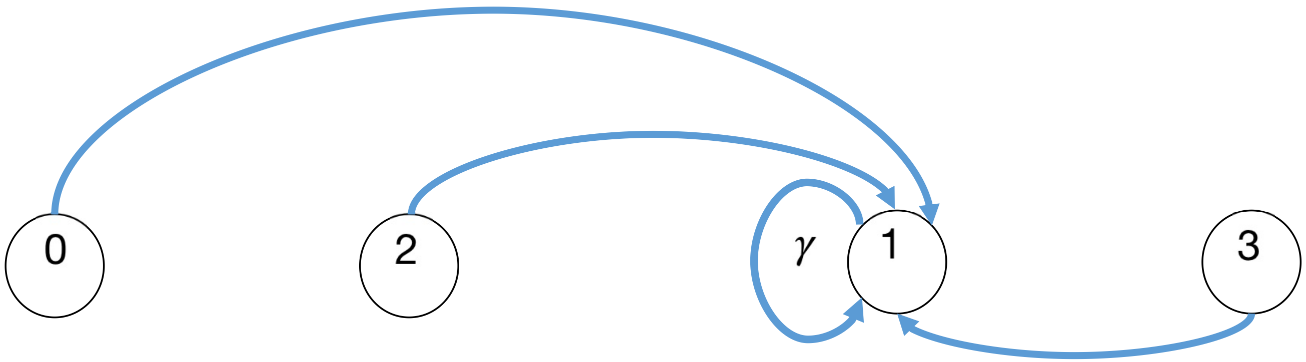
Synthesis Algorithm

- **Step 4**: add arcs from unreachable nodes to γ



Synthesis Algorithm

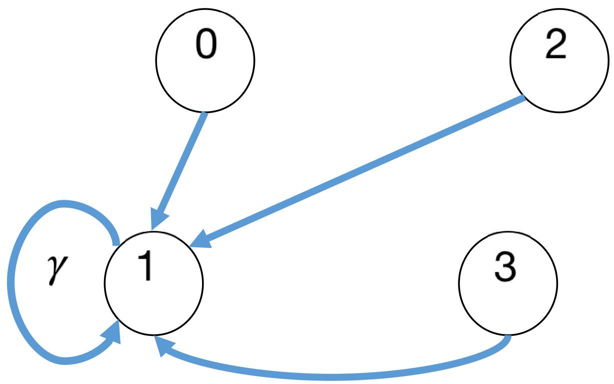
- **Step 4**: add arcs from unreachable nodes to γ



Intuitively, this spanning tree captures how "local updates" should be performed to ensure "global stabilization".

Synthesis Algorithm

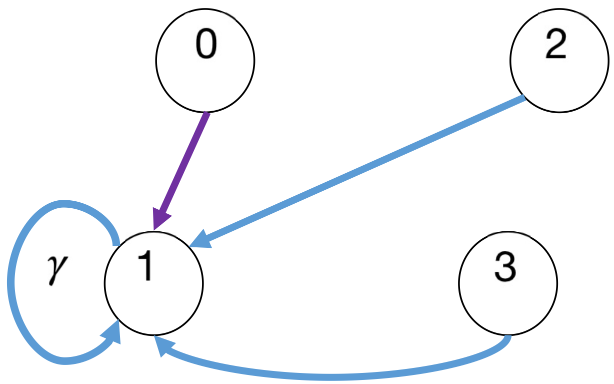
- **<u>Step 5</u>**: transform the spanning tree to an action graph by labeling its arcs
- *Labeling Method*:
    - For each arc (a, c), label it with a value b *iff* L(a,b) is false and b is not a parent of a in the spanning tree



Synthesis Algorithm

- **Step 5**: transform the spanning tree to an action graph by labeling its arcs
- *Labeling Method*:
  - For each arc (a, c), label it with a value b *iff* L(a,b) is false and b is not a parent of a in the spanning tree

a = 0 and b = 1 and c=1 ⇒ (|0-1| mod 2) != 0, but b=c; unacceptable

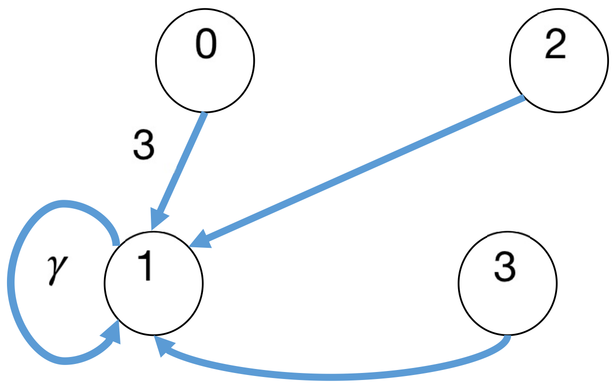


Synthesis Algorithm

- **Step 5**: transform the spanning tree to an action graph by labeling its arcs
- *Labeling Method*:
  - For each arc (*a*, *c*), label it with a value *b iff* L(*a*,*b*) is false and *b* is not a parent of *a* in the spanning tree

a = 0 and b = 1 and c=1 ⇒ |0-1| mod 2 != 0, but b=c; unacceptable

a = 0 and b = 3 and c=1 ⇒ |0-3| mod 2 != 0; acceptable



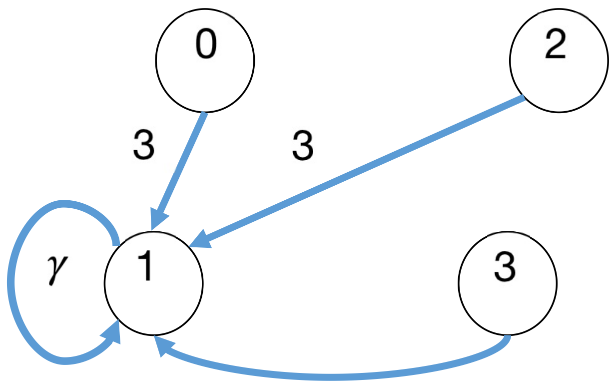
Synthesis Algorithm

- **Step 5**: transform the spanning tree to an action graph by labeling its arcs
- *Labeling Method*:
  - For each arc (a, c), label it with a value b *iff* L(a,b) is false and b is not a parent of a in the spanning tree

a = 0 and b = 1 and c=1 ⇒ |0-1| mod 2 != 0, but b=c; unacceptable

a = 0 and b = 3 and c=1 ⇒ |0-3| mod 2 != 0; acceptable

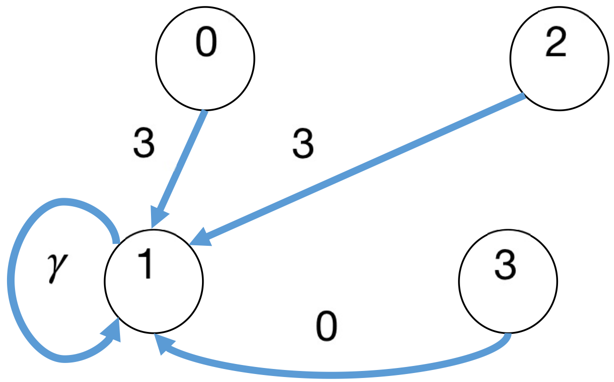
a = 2 and b = 3 and c=1 ⇒ |2-3| mod 2 != 0; acceptable



Synthesis Algorithm

- **<u>Step 5</u>**: transform the spanning tree to an action graph by labeling its arcs
- *Labeling Method*:
    - For each arc (a, c), label it with a value b *iff* L(a,b) is false and b is not a parent of a in the spanning tree

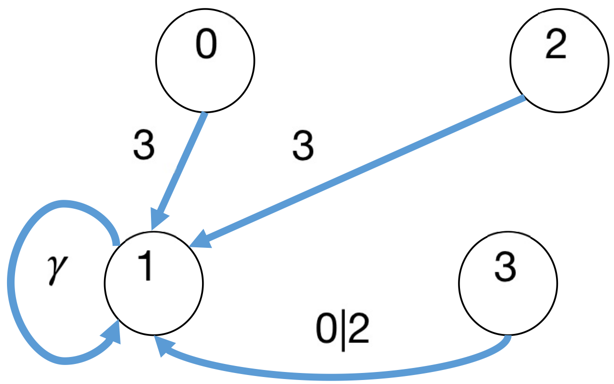
a = 0 and b = <u>1</u> and c=1 ⇒ |0-1| mod 2 != 0, but b=c; unacceptable
a = 0 and b = <u>3</u> and c=1 ⇒ |0-3| mod 2 != 0; acceptable
a = 2 and b = <u>3</u> and c=1 ⇒ |2-3| mod 2 != 0; acceptable
a = 3 and b = <u>0</u> and c=1 ⇒ |3-0| mod 2 != 0; acceptable



Synthesis Algorithm

- **Step 5**: transform the spanning tree to an action graph by labeling its arcs
- *Labeling Method*:
  - For each arc (a, c), label it with a value b *iff* L(a,b) is false and b is not a parent of a in the spanning tree

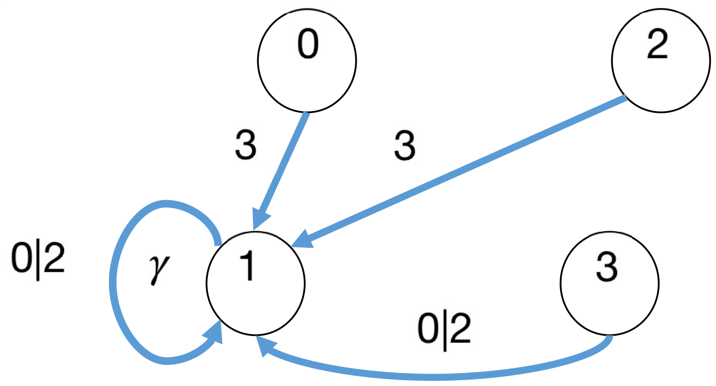
a = 0 and b = 1 and c=1 ⇒ |0-1| mod 2 != 0, but b=c; unacceptable
a = 0 and b = 3 and c=1 ⇒ |0-3| mod 2 != 0; acceptable
a = 2 and b = 3 and c=1 ⇒ |2-3| mod 2 != 0; acceptable
a = 3 and b = 0 and c=1 ⇒ |3-0| mod 2 != 0; acceptable
a = 3 and b = 2 and c=1 ⇒ |3-2| mod 2 != 0; acceptable



Synthesis Algorithm

- **<u>Step 5</u>**: transform the spanning tree to an action graph by labeling its arcs
- *Labeling Method*:
    - For each arc (a, c), label it with a value b *iff* L(a,b) is false and b is not a parent of a in the spanning tree

a = 0 and b = <u>1</u> and c=1 ⇒ |0-1| mod 2 != 0, but b=c; unacceptable
a = 0 and b = <u>3</u> and c=1 ⇒ |0-3| mod 2 != 0; acceptable
a = 2 and b = <u>3</u> and c=1 ⇒ |2-3| mod 2 != 0; acceptable
a = 3 and b = <u>0</u> and c=1 ⇒ |3-0| mod 2 != 0; acceptable
a = 3 and b = <u>2</u> and c=1 ⇒ |3-2| mod 2 != 0; acceptable
a = 1 and b = <u>0</u> and c=1 ⇒ |1-0| mod 2 != 0; acceptable
a = 1 and b = <u>2</u> and c=1 ⇒ |1-2| mod 2 != 0; acceptable



Synthesis Algorithm

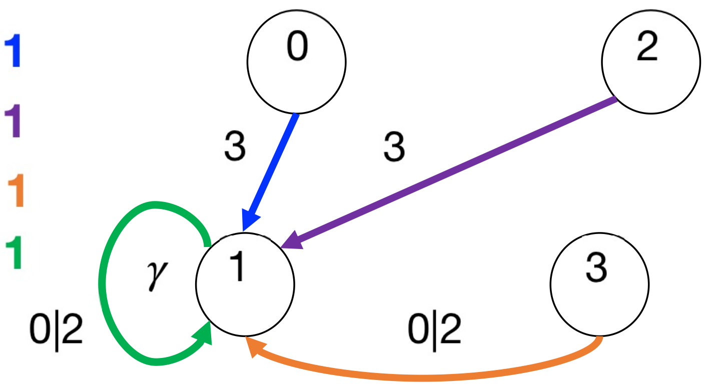
- Proof of stabilization:
  - Deadlock-freedom outside *I* :
    - Each process is enabled *iff* $L'(x_{i-1}, x_i)$ is false
  - Closure of *I* in protocol actions:
    - no action is enabled where $L'(x_{i-1}, x_i)$ is true
  - Livelock-freedom outside *I* :
    - The only type of closed walk includes **(γ, b, γ)** , which does not enable itself circularly

$(x_{i-1} = 0) \wedge (x_i = 3) \rightarrow x_i := 1$

$(x_{i-1} = 2) \wedge (x_i = 3) \rightarrow x_i := 1$

$(x_{i-1} = 3) \wedge ((x_i = 0) \vee (x_i = 2)) \rightarrow x_i := 1$

$(x_{i-1} = 1) \wedge ((x_i = 0) \vee (x_i = 2)) \rightarrow x_i := 1$

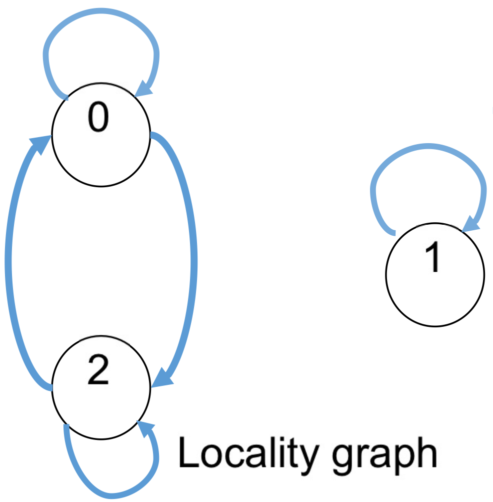


Synthesis for Constant Space

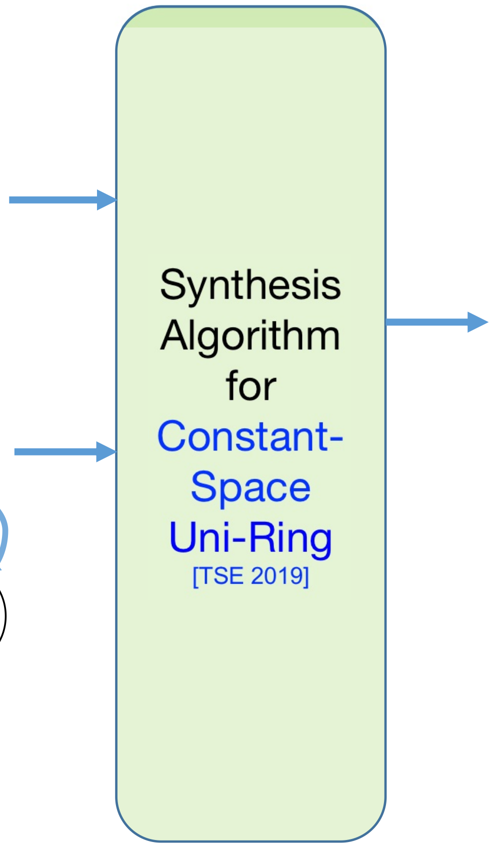
Example: Agree on a common Parity in uni-ring

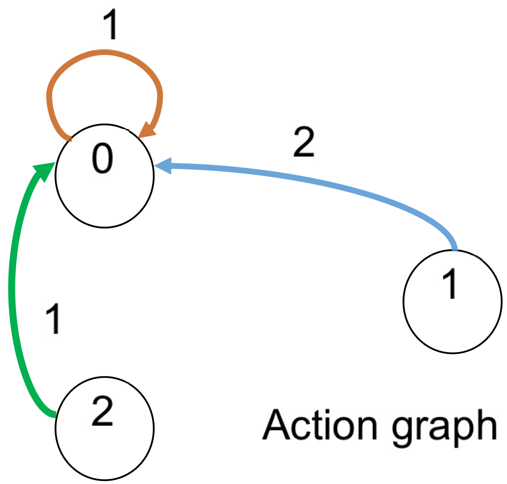
$\mathcal{I} = \forall i \in \mathbb{Z}^{+} : L(x_{i-1}, x_i)$ where $L(x_{i-1}, x_i) \equiv (|x_{i-1} - x_i| \,\%2 = 0)$ $x_i \in \mathbb{Z}_3 = \{0, 1, 2\}$

$x_i \in \mathbb{Z}_3 = \{0, 1, 2\}$



Locality graph





Action graph

Parameterized actions:

$x_{i-1} = 1 \wedge x_i = 2 \rightarrow x_i := 0$

$x_{i-1} = 2 \wedge x_i = 1 \rightarrow x_i := 0$

$x_{i-1} = 0 \wedge x_i = 1 \rightarrow x_i := 0$

[TSE 2019] Ali Ebnenasir and Alex Klinkhamer, **Topology-specific synthesis of self stabilizing parameterized systems with constant-space processes**, *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 614–629, 2019.

Related Work on V&S of Parameterized Protocols

- **Verification and Synthesis (V&S)** of PDP are in general undecidable problems.
  - *Pairwise synthesis*: safety properties and local liveness in symmetric systems [Attie and Emerson 1998]
  - *Abstraction methods*: create finite approximations of PDP (e.g., counter abstraction) and conduct verification [Pnueli et al. 2002]
  - *Regular model checking*: use regular languages to model PDP [Abdulla et al. 2004]
  - *Invisible invariants/ranking*: generate implicit local invariants and generalize [Fang et al. 2006]
  - *Network invariants*: prove safety by parallel compositions that are invariant to correctness [Wolper and Lovinfosse 1989]
    - Neo [Matthews, Bingham, Sorin 2016] expands this idea for topology-specific verification of safety properties
  - *Parameterized synthesis*: based on small model theorems (i.e., cutoff) and SMT-based bounded synthesis [Jacobs and Bloem 2012]
  - *Well-founded proof spaces*: prove safety and liveness of infinite traces by showing that traces terminate [Farzan et al. 2016]
  - *Population protocols*: anonymous processes; invariants are formed of counting constraints; mostly consider clique topology [Esparza et al. 2018]
  - *Synthesis of Threshold Automata (TA)*: complete sketches of TA using counter abstraction [Lazi et al. 2018]

- General topology (in some cases a clique) and temporal properties.
- *Correctness of a finite abstract model implies correctness of PDP.*
- Mostly focus on safety properties and local liveness.
- Few of them focus on self-stabilization.

V&S of Unbounded Protocols

Domain Size of Variables \ Number of Processes	Fixed	Unbounded
Fixed	Fixed-size Protocols	Unbounded Variable Protocols
Unbounded	Constant-space Parameterized Protocols	Unbounded Parameterized Protocols (FMCAD 2022)

[FMCAD 2022] Ali Ebnenasir, **Synthesizing Self-stabilizing Parameterized Protocols with Unbounded Variables**, *FMCAD* 2022.

Protocol	Topology	Property	Verified/Synthesized	Unboundedness
Leader Election	Uni-Ring	Livelock-freedom	Verified	Processes
Token Passing	Uni-Ring	Livelock-freedom	Verified	Processes
Agreement	Uni-Ring	Livelock-freedom	Verified	Processes
Coloring	Uni-Ring	Livelock-freedom	Verified	Processes
Parity	Uni-Ring	Self-Stabilization	Synthesized	Processes/Variable Domain
Agreement	Uni-Ring	Self-Stabilization	Synthesized	Processes/Variable Domain
Sum-Not-2	Uni-Ring	Self-Stabilization	Synthesized	Processes
Broadcast	Tree	Self-Stabilization	Synthesized	Processes
Coloring	Tree	Self-Stabilization	Synthesized	Processes
MIS	Tree	Self-Stabilization	Synthesized	Processes
Min/Max	Tree	Self-Stabilization	Synthesized	Processes
Sum-Not-2	Uni-Ring	LeadsTo	Synthesized	Processes
Agreement	Uni-Ring	LeadsTo	Synthesized	Processes
Parity	Uni-Ring	LeadsTo	Synthesized	Processes

Open Problems

- V&S for
  - Protocols with multiple symmetric families
  - Property:
    - Fault tolerance (e.g., failsafe, nonmasking, masking)
      - Characterize faults in action graphs?
      - Calculate fault-span locally?
    - Security and privacy (e.g., tamper evidence, access control, anonymity, etc.)
      - Local characterization of security breaches?
      - Interplay of fault tolerance and security aspects for template processes
    - General LTL properties (e.g., LeadsTo, Dwyer's Specification patterns)

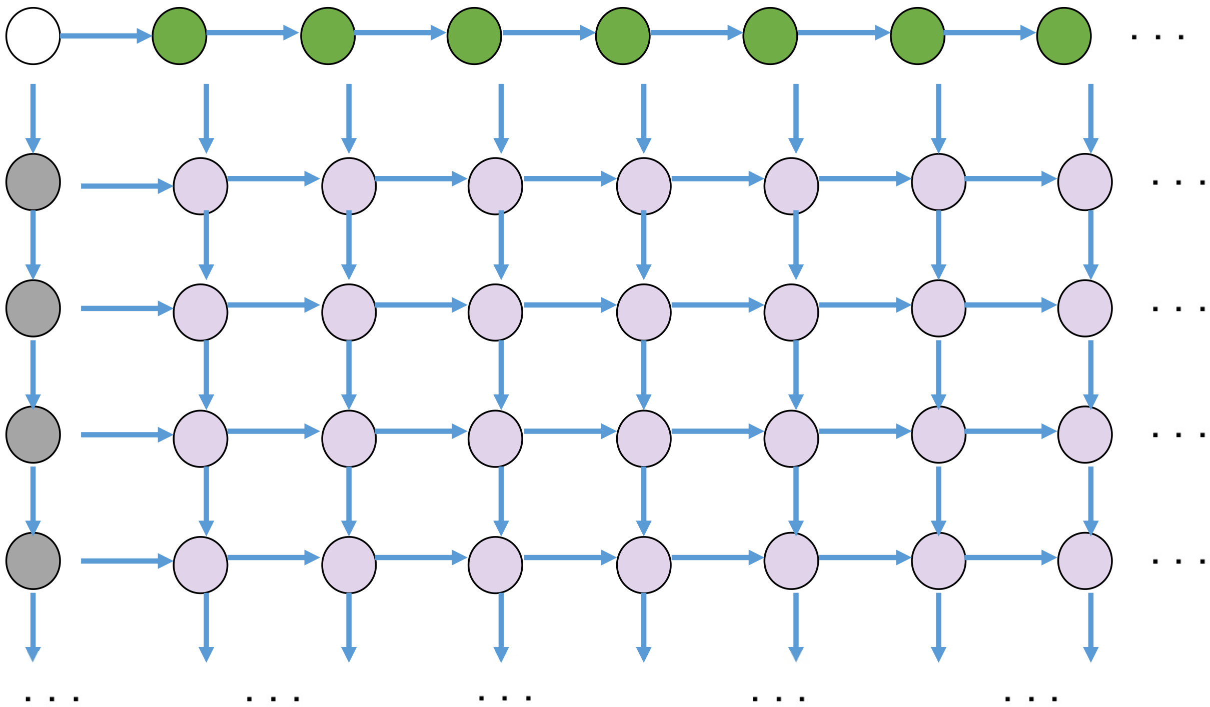
Open Problems

- V&S for
  - Property:
    - General temporal logic properties
      - LeadsTo (FMCAD 2019, IEEE TSE 2021)
      - Dwyer's Specification patterns?
  - Topology:
    - Mesh
    - Katz graph

Property-Preserving Compositions

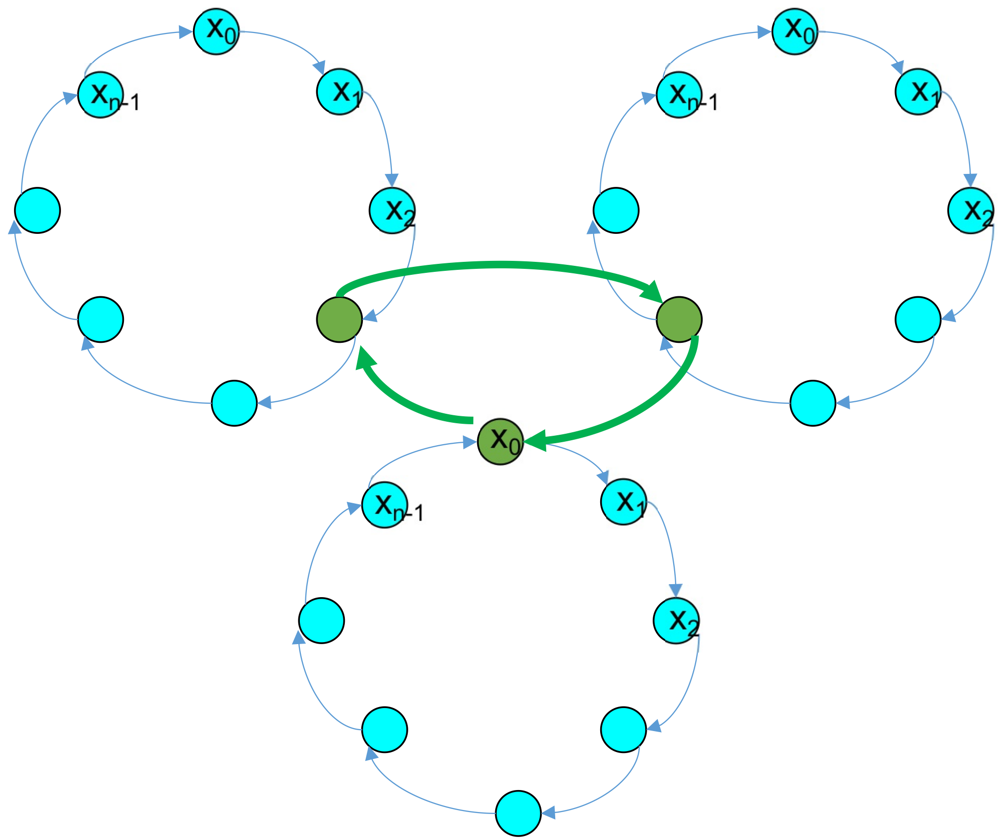
- Problem Statement
  - **<u>Input</u>**: Two PDPs P_1 and P_2 with
    - elementary topologies T_1 and T_2
    - invariants I_1 and I_2
    - <u>assumption</u>: P_1 and P_2 satisfy a global property φ respectively from I_1 and I_2 for any number of processes
  - **<u>Output</u>**: PDP P_c with a topology T_c and an invariant I_c such that
    - T_c is a (hierarchical/sequential/parallel/superposition) compostion of T_1 and T_2 , and
    - P_c is a (synchronous/asynchronous) composition of P_1 and P_2 which satisfies φ from I_c where
      - I_c is a conjunctive invariant $I_c = (I_1 \land I_2)$;
      - I_c is a disjunctive invariant $I_c = (I_1 \lor I_2)$;
      - I_c is a implicative invariant $I_c = (I_1 \Rightarrow I_2)$, or $I_c = (I_2 \Rightarrow I_1)$.

Mesh

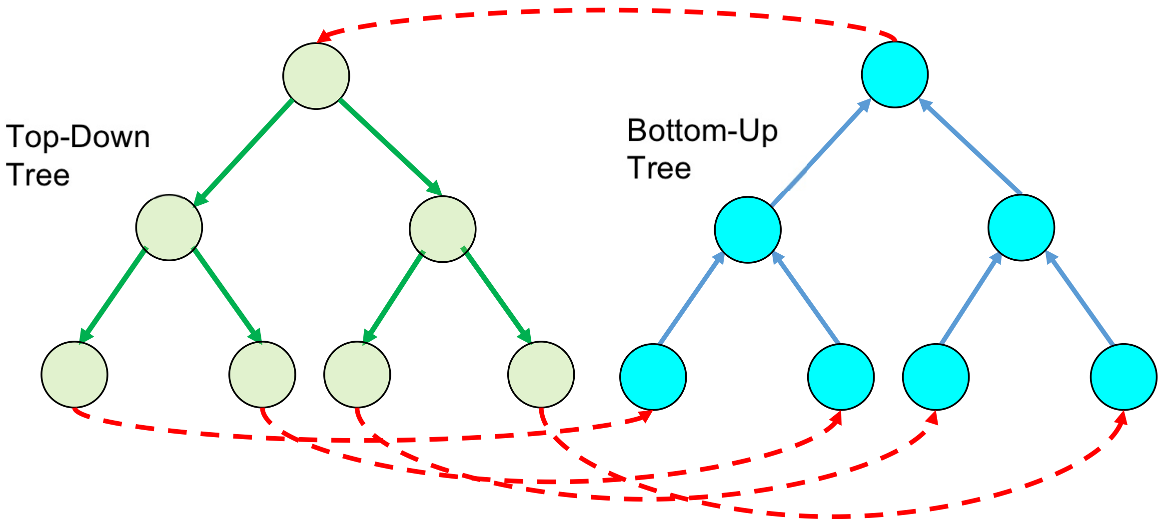


Information flow-based sufficient conditions.

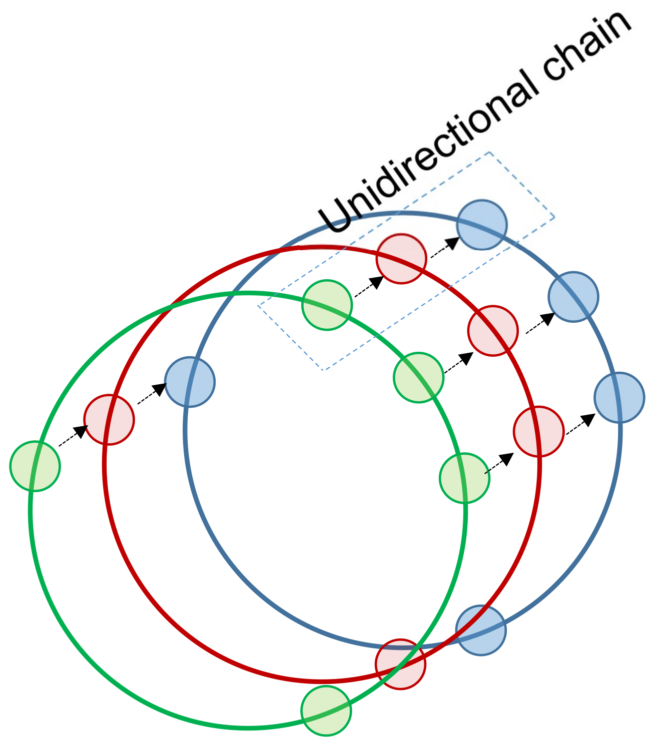
HyperRing



Superposed Trees



Variable-Space Processes



Scalable composition of resilient ring and chain
generating a scalable tube that can grow
in depth and diameter.

Acknowledgement

- Former graduate students:
  - Dr. Alex Klinkhamer
    - Google (Mountain View, CA)
  - Dr. Aly Farahat
    - Intuitive Surgical Inc. (Bay Area, CA)
  - Dr. Amer Tahat
    - Pennsylvania State University
  - Dr. Reza Hajisheykhi (co-advised)
    - Rubrik Inc.
  - Several other M.Sc. students

- NSF grants CCF-1116546 and CCF-0950678
- Michigan Tech's Research Excellence Fund