Parameterized Synthesis of Self-Stabilizing Protocols in Symmetric Rings

Nahal Mirzaei[1] Fathiyeh Faghih[1] Swen Jacobs[2]
Borzoo Bonakdarpour[3]

University of Tehran, Iran[1]

CISPA Helmholtz Center, Germany[2]

Iowa State University, USA[3]

Presentation outline

Motivation

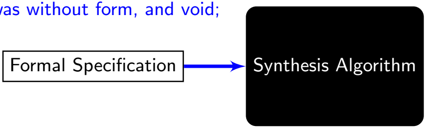
In the beginning God
created the heaven and
the earth. And the earth
was without form, and void;

Formal Specification

Motivation

In the beginning God created the heaven and the earth. And the earth was without form, and void;

And God said, Let there be light:

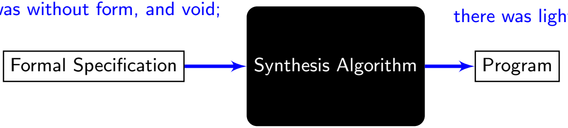


Motivation

In the beginning God created the heaven and the earth. And the earth was without form, and void;

And God said, Let there be light:

and there was light!



Motivation

In the beginning God created the heaven and the earth. And the earth was without form, and void;

And God said, Let there be light:

and there was light!

And God saw the light, and it was good;

Formal Specification → Synthesis Algorithm → Program

correct by construction!

Motivation

In the beginning God created the heaven and the earth. And the earth was without form, and void;

And God said, Let there be light:

and there was light!

And God saw the light, and it was good;

Formal Specification → Synthesis Algorithm → Program

correct by construction!

Program Synthesis

Program *synthesis*, the "holy grail" of computer science, is the problem of automated generation of a computer program from a formally specified set of properties.

Motivation

In the beginning God created the heaven and the earth. And the earth was without form, and void;

And God said, Let there be light:

and there was light!

And God saw the light, and it was good;

Formal Specification → Synthesis Algorithm → Program

correct by construction!

Program Synthesis

Program *synthesis*, the "holy grail" of computer science, is the problem of automated generation of a computer program from a formally specified set of properties.

- Program synthesis is known to be *computationally intractable* and in many cases *undecidable*.

Motivation

In the beginning God created the heaven and the earth. And the earth was without form, and void;

And God said, Let there be light:

and there was light!

And God saw the light, and it was good;

Formal Specification → Synthesis Algorithm → Program

correct by construction!

Program Synthesis

Program *synthesis*, the "holy grail" of computer science, is the problem of automated generation of a computer program from a formally specified set of properties.

- Program synthesis is known to be *computationally intractable* and in many cases *undecidable*.
- It is particularly useful to deal with small but *intricate components* of a system, e.g., concurrent/distributed algorithms.

Self-stabilization

Self-stabilization is a versatile technique for forward fault recovery that has two key features:

Self-stabilization

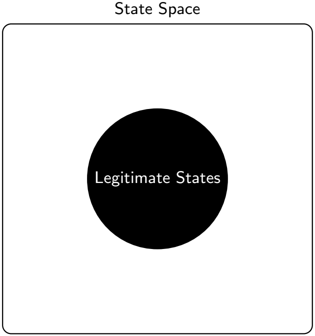
Self-stabilization is a versatile technique for forward fault recovery that has two key features:

- *Convergence.* Starting from any arbitrary state, the system is guaranteed to recover proper behavior (i.e., *legitimate states*) within a finite number of execution steps.

Self-stabilization

Self-stabilization is a versatile technique for forward fault recovery that has two key features:

- *Convergence.* Starting from any arbitrary state, the system is guaranteed to recover proper behavior (i.e., *legitimate states*) within a finite number of execution steps.

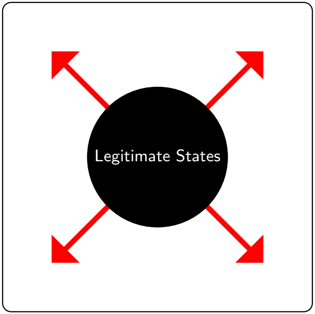


Self-stabilization

Self-stabilization is a versatile technique for forward fault recovery that has two key features:

- *Convergence.* Starting from any arbitrary state, the system is guaranteed to recover proper behavior (i.e., *legitimate states*) within a finite number of execution steps.

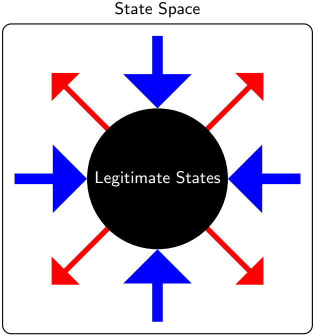
State Space



Self-stabilization

Self-stabilization is a versatile technique for forward fault recovery that has two key features:

- *Convergence.* Starting from any arbitrary state, the system is guaranteed to recover proper behavior (i.e., *legitimate states*) within a finite number of execution steps.

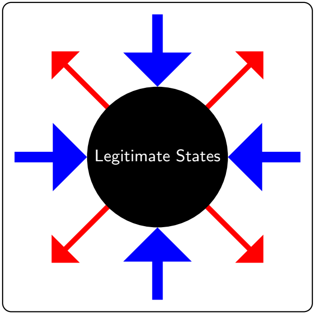


Self-stabilization

Self-stabilization is a versatile technique for forward fault recovery that has two key features:

- *Convergence.* Starting from any arbitrary state, the system is guaranteed to recover proper behavior (i.e., *legitimate states*) within a finite number of execution steps.

State Space



- *Closure.* Once the system reaches a legitimate state, it remains in this set thereafter in the absence of new faults.

Why Synthesizing Self-stabilizing Algorithms?

Why Synthesizing Self-stabilizing Algorithms?

- *Proof* of self-stabilization is often much more complex than what it initially seems like.

Why Synthesizing Self-stabilizing Algorithms?

- *Proof* of self-stabilization is often much more complex than what it initially seems like.
- *Dijkstra* himself published the proof of correctness of his seminal 3-state machine solution *12 years* later.

Why Synthesizing Self-stabilizing Algorithms?

- *Proof* of self-stabilization is often much more complex than what it initially seems like.
- *Dijkstra* himself published the proof of correctness of his seminal 3-state machine solution *12 years* later.
- *Program synthesis* can play a prime role in designing and reasoning about the correctness of *self-stabilizing* algorithms.

Related Work

Related Work

- A. Klinkhamer and A. Ebnenasir: *On the Hardness of Adding Nonmasking Fault Tolerance*. IEEE TDSC 12(3): 338-350, 2015.

Related Work

- A. Klinkhamer and A. Ebnenasir: *On the Hardness of Adding Nonmasking Fault Tolerance*. IEEE TDSC 12(3): 338-350, 2015.
- A. Klinkhamer and A. Ebnenasir: *Synthesizing Self-stabilization through Superposition and Backtracking*. SSS 2014: 252-267

Related Work

- A. Klinkhamer and A. Ebnenasir: *On the Hardness of Adding Nonmasking Fault Tolerance*. IEEE TDSC 12(3): 338-350, 2015.
- A. Klinkhamer and A. Ebnenasir: *Synthesizing Self-stabilization through Superposition and Backtracking*. SSS 2014: 252-267
- F. Faghih and B. Bonakdarpour: *SMT-Based Synthesis of Distributed Self-Stabilizing Systems*. ACM TAAS 10(3): 21:1-21:26, 2015.

Related Work

- A. Klinkhamer and A. Ebnenasir: *On the Hardness of Adding Nonmasking Fault Tolerance*. IEEE TDSC 12(3): 338-350, 2015.
- A. Klinkhamer and A. Ebnenasir: *Synthesizing Self-stabilization through Superposition and Backtracking*. SSS 2014: 252-267
- F. Faghih and B. Bonakdarpour: *SMT-Based Synthesis of Distributed Self-Stabilizing Systems*. ACM TAAS 10(3): 21:1-21:26, 2015.
- F. Faghih, B. Bonakdarpour, S. Tixeuil, S. S. Kulkarni: *Automated Synthesis of Distributed Self-Stabilizing Protocols*. Logical Methods in Computer Science 14(1), 2018.

Related Work

- A. Klinkhamer and A. Ebnenasir: *On the Hardness of Adding Nonmasking Fault Tolerance*. IEEE TDSC 12(3): 338-350, 2015.
- A. Klinkhamer and A. Ebnenasir: *Synthesizing Self-stabilization through Superposition and Backtracking*. SSS 2014: 252-267
- F. Faghih and B. Bonakdarpour: *SMT-Based Synthesis of Distributed Self-Stabilizing Systems*. ACM TAAS 10(3): 21:1-21:26, 2015.
- F. Faghih, B. Bonakdarpour, S. Tixeuil, S. S. Kulkarni: *Automated Synthesis of Distributed Self-Stabilizing Protocols*. Logical Methods in Computer Science 14(1), 2018.

Problem

All these approaches can synthesize only a *fixed* number of processes.
Parameterized synthesis is an *undecidable* problem.

Contributions

Contributions

Automated synthesis of self-stabilizing protocols in *symmetric* and *parameterized* rings.

Contributions

Automated synthesis of self-stabilizing protocols in *symmetric* and *parameterized* rings.

Our Approach

A *cutoff* point guarantees properties of a distributed system of arbitrary size by considering only systems of up to a certain fixed size $c \in \mathbb{N}$.

Contributions

Automated synthesis of self-stabilizing protocols in *symmetric* and *parameterized* rings.

Our Approach

A *cutoff* point guarantees properties of a distributed system of arbitrary size by considering only systems of up to a certain fixed size $c \in \mathbb{N}$.

We provide:

Contributions

Automated synthesis of self-stabilizing protocols in *symmetric* and *parameterized* rings.

Our Approach

A *cutoff* point guarantees properties of a distributed system of arbitrary size by considering only systems of up to a certain fixed size $c \in \mathbb{N}$.

We provide:

- *Tight cutoffs* for the *closure* and *deadlock-freedom* properties.

Contributions

Automated synthesis of self-stabilizing protocols in *symmetric* and *parameterized* rings.

Our Approach

A *cutoff* point guarantees properties of a distributed system of arbitrary size by considering only systems of up to a certain fixed size $c \in \mathbb{N}$.

We provide:

- *Tight cutoffs* for the *closure* and *deadlock-freedom* properties.
- A *sufficient condition* for *convergence* that can be efficiently checked on an over-approximated finite system.

Contributions

Automated synthesis of self-stabilizing protocols in *symmetric* and *parameterized* rings.

Our Approach

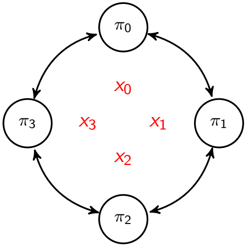
A *cutoff* point guarantees properties of a distributed system of arbitrary size by considering only systems of up to a certain fixed size $c \in \mathbb{N}$.

We provide:

- *Tight cutoffs* for the *closure* and *deadlock-freedom* properties.
- A *sufficient condition* for *convergence* that can be efficiently checked on an over-approximated finite system.
- A scalable *counterexample-guided synthesis* technique.

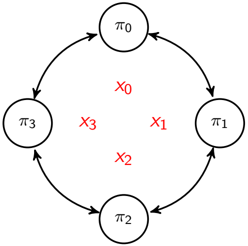
Presentation outline

Model of Computation – Distributed Programs



Variables: $V = \{x_0, x_1, x_2, x_3\}$, each x_i is a Boolean.

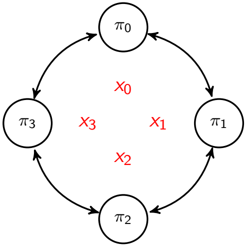
Model of Computation – Distributed Programs



Variables: $V = \{x_0, x_1, x_2, x_3\}$, each x_i is a Boolean.

Distributed program: $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$.

Model of Computation – Distributed Programs

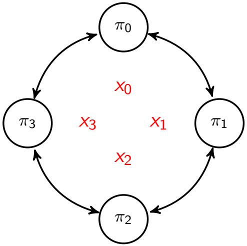


Variables: $V = \{x_0, x_1, x_2, x_3\}$, each x_i is a Boolean.

Distributed program: $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$.

Processes: $P_{\mathcal{D}} = \{\pi_0, \pi_1, \pi_2, \pi_3\}$.

Model of Computation – Distributed Programs



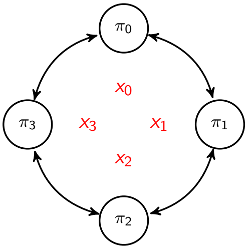
Variables: $V = \{x_0, x_1, x_2, x_3\}$, each x_i is a Boolean.

Distributed program: $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$.

Processes: $P_{\mathcal{D}} = \{\pi_0, \pi_1, \pi_2, \pi_3\}$.

Write-set: $W_{\pi_i} = \{x_i\}$

Model of Computation – Distributed Programs



Variables: $V = \{x_0, x_1, x_2, x_3\}$, each x_i is a Boolean.

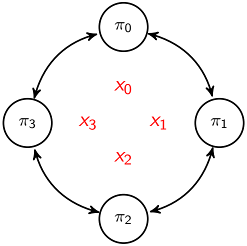
Distributed program: $\mathcal{D} = \langle P_\mathcal{D}, T_\mathcal{D} \rangle$.

Processes: $P_\mathcal{D} = \{\pi_0, \pi_1, \pi_2, \pi_3\}$.

Write-set: $W_{\pi_i} = \{x_i\}$

Read-set: $R_{\pi_i} = \{x_i, x_{(i+1) \mod 4}, x_{(i-1) \mod 4}\}$).

Model of Computation – Distributed Programs



Variables: $V = \{x_0, x_1, x_2, x_3\}$, each x_i is a Boolean.

Distributed program: $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$.

Processes: $P_{\mathcal{D}} = \{\pi_0, \pi_1, \pi_2, \pi_3\}$.

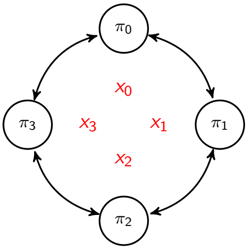
Write-set: $W_{\pi_i} = \{x_i\}$

Read-set: $R_{\pi_i} = \{x_i, x_{(i+1) \bmod 4}, x_{(i-1) \bmod 4}\})$.

Read Restrictions

$$t_1 = ([x_0 = \mathit{false}, x_1 = \mathit{false}, x_2 = \mathit{false}, x_3 = \mathit{false}],$$
$$[x_0 = \mathit{true}, x_1 = \mathit{false}, x_2 = \mathit{false}, x_3 = \mathit{false}])$$

Model of Computation – Distributed Programs



Variables: $V = \{x_0, x_1, x_2, x_3\}$, each x_i is a Boolean.

Distributed program: $\mathcal{D} = \langle P_\mathcal{D}, T_\mathcal{D} \rangle$.

Processes: $P_\mathcal{D} = \{\pi_0, \pi_1, \pi_2, \pi_3\}$.

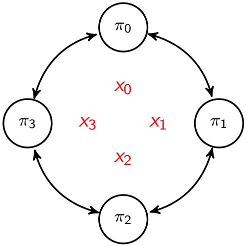
Write-set: $W_{\pi_i} = \{x_i\}$

Read-set: $R_{\pi_i} = \{x_i, x_{(i+1) \mod 4}, x_{(i-1) \mod 4}\})$.

Read Restrictions

$$t_1 = ([x_0 = \mathit{false}, x_1 = \mathit{false}, x_2 = \mathit{false}, x_3 = \mathit{false}],$$
$$[x_0 = \mathit{true}, x_1 = \mathit{false}, x_2 = \mathit{false}, x_3 = \mathit{false}])$$

Model of Computation – Distributed Programs



Variables: $V = \{x_0, x_1, x_2, x_3\}$, each x_i is a Boolean.

Distributed program: $\mathcal{D} = \langle P_\mathcal{D}, T_\mathcal{D} \rangle$.

Processes: $P_\mathcal{D} = \{\pi_0, \pi_1, \pi_2, \pi_3\}$.

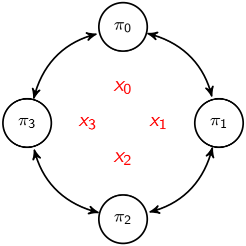
Write-set: $W_{\pi_i} = \{x_i\}$

Read-set: $R_{\pi_i} = \{x_i, x_{(i+1) \mod 4}, x_{(i-1) \mod 4}\}$).

Read Restrictions

$$t_1 = ([x_0 = false, x_1 = false, x_2 = false, x_3 = false],$$
$$[x_0 = true, x_1 = false, x_2 = false, x_3 = false])$$
$$t_2 = ([x_0 = false, x_1 = false, x_2 = true, x_3 = false],$$
$$[x_0 = true, x_1 = false, x_2 = true, x_3 = false])$$

Model of Computation – Distributed Programs



Variables: $V = \{x_0, x_1, x_2, x_3\}$, each x_i is a Boolean.

Distributed program: $\mathcal{D} = \langle P_\mathcal{D}, T_\mathcal{D} \rangle$.

Processes: $P_\mathcal{D} = \{\pi_0, \pi_1, \pi_2, \pi_3\}$.

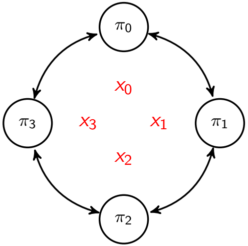
Write-set: $W_{\pi_i} = \{x_i\}$

Read-set: $R_{\pi_i} = \{x_i, x_{(i+1) \mod 4}, x_{(i-1) \mod 4}\})$.

Read Restrictions

$$
\begin{aligned}
t_1 = (&[x_0 = \mathit{false}, x_1 = \mathit{false}, x_2 = \mathit{false}, x_3 = \mathit{false}],\\
&[x_0 = \mathit{true}, x_1 = \mathit{false}, x_2 = \mathit{false}, x_3 = \mathit{false}])\\
t_2 = (&[x_0 = \mathit{false}, x_1 = \mathit{false}, x_2 = \mathit{true}, x_3 = \mathit{false}],\\
&[x_0 = \mathit{true}, x_1 = \mathit{false}, x_2 = \mathit{true}, x_3 = \mathit{false}])
\end{aligned}
$$

Model of Computation – Distributed Programs



Variables: $V = \{x_0, x_1, x_2, x_3\}$, each x_i is a Boolean.

Distributed program: $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$.

Processes: $P_{\mathcal{D}} = \{\pi_0, \pi_1, \pi_2, \pi_3\}$.

Write-set: $W_{\pi_i} = \{x_i\}$

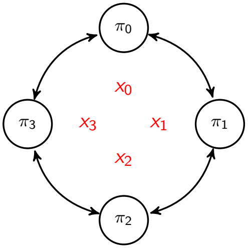
Read-set: $R_{\pi_i} = \{x_i, x_{(i+1) \mod 4}, x_{(i-1) \mod 4}\})$.

Read Restrictions

$$
\begin{aligned}
t_1 = ([&x_0 = false, x_1 = false, x_2 = false, x_3 = false],\\
&[x_0 = true, x_1 = false, x_2 = false, x_3 = false])\\
t_2 = ([&x_0 = false, x_1 = false, x_2 = true, x_3 = false],\\
&[x_0 = true, x_1 = false, x_2 = true, x_3 = false])
\end{aligned}
$$

Such transitions create an equivalence class, called a *group*.

Model of Computation – Distributed Programs



Variables: $V = \{x_0, x_1, x_2, x_3\}$, each x_i is a Boolean.

Distributed program: $\mathcal{D} = \langle P_\mathcal{D}, T_\mathcal{D} \rangle$.

Processes: $P_\mathcal{D} = \{\pi_0, \pi_1, \pi_2, \pi_3\}$.

Write-set: $W_{\pi_i} = \{x_i\}$

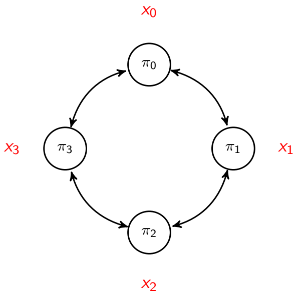
Read-set: $R_{\pi_i} = \{x_i, x_{(i+1) \mod 4}, x_{(i-1) \mod 4}\})$.

Read Restrictions

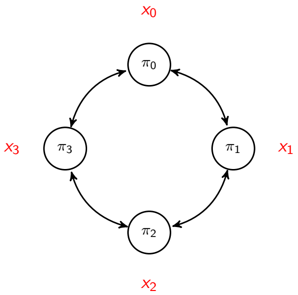
$$
\begin{aligned}
t_1 = (&[x_0 = \mathit{false}, x_1 = \mathit{false}, x_2 = \mathit{false}, x_3 = \mathit{false}],\\
&[x_0 = \mathit{true}, x_1 = \mathit{false}, x_2 = \mathit{false}, x_3 = \mathit{false}])\\
t_2 = (&[x_0 = \mathit{false}, x_1 = \mathit{false}, x_2 = \mathit{true}, x_3 = \mathit{false}],\\
&[x_0 = \mathit{true}, x_1 = \mathit{false}, x_2 = \mathit{true}, x_3 = \mathit{false}])
\end{aligned}
$$

Such transitions create an equivalence class, called a *group*.
Transition t_1 is included if and only if t_2 is.

Model of Computation – Topology

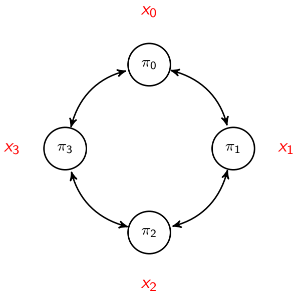


Model of Computation – Topology



A *topology* is a tuple $\mathcal{T} = \langle V, |P_{\mathcal{T}}|, R_{\mathcal{T}}, W_{\mathcal{T}} \rangle$, where

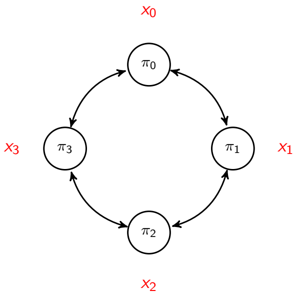
Model of Computation – Topology



A *topology* is a tuple $\mathcal{T} = \langle V, |P_\mathcal{T}|, R_\mathcal{T}, W_\mathcal{T} \rangle$, where

- V is a finite set of finite-domain discrete variables,

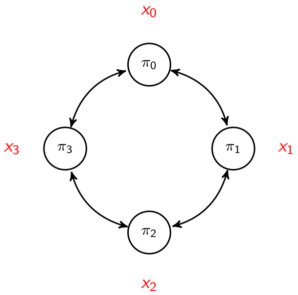
Model of Computation – Topology



A *topology* is a tuple $\mathcal{T} = \langle V, |P_\mathcal{T}|, R_\mathcal{T}, W_\mathcal{T} \rangle$, where

- V is a finite set of finite-domain discrete variables,
- $|P_\mathcal{T}| \in \mathbb{N}_{\geq 1}$ is the number of processes,

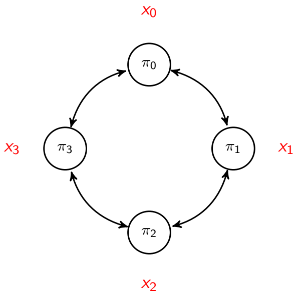
Model of Computation – Topology



A *topology* is a tuple $\mathcal{T} = \langle V, |P_\mathcal{T}|, R_\mathcal{T}, W_\mathcal{T} \rangle$, where

- V is a finite set of finite-domain discrete variables,
- $|P_\mathcal{T}| \in \mathbb{N}_{\geq 1}$ is the number of processes,
- $R_\mathcal{T}$ is a mapping $\{0 \ldots |P_\mathcal{T}| - 1\} \rightarrow 2^V$ from a process index to its read-set,

Model of Computation – Topology



A *topology* is a tuple $\mathcal{T} = \langle V, |P_{\mathcal{T}}|, R_{\mathcal{T}}, W_{\mathcal{T}} \rangle$, where

- V is a finite set of finite-domain discrete variables,
- $|P_{\mathcal{T}}| \in \mathbb{N}_{\geq 1}$ is the number of processes,
- $R_{\mathcal{T}}$ is a mapping $\{0 \ldots |P_{\mathcal{T}}| - 1\} \rightarrow 2^V$ from a process index to its read-set,
- $W_{\mathcal{T}}$ is a mapping $\{0 \ldots |P_{\mathcal{T}}| - 1\} \rightarrow 2^V$ from a process index to its write-set, such that $W_{\mathcal{T}}(i) \subseteq R_{\mathcal{T}}(i)$, for all i $(0 \leq i \leq |P_{\mathcal{T}}| - 1)$.

Model of Computation – Symmetric Topology

Model of Computation – Symmetric Topology

A topology $\mathcal{T} = \langle V, |P_{\mathcal{T}}|, R_{\mathcal{T}}, W_{\mathcal{T}} \rangle$ is *symmetric*, iff for any distinct $i, j \in \{0 \ldots |P_{\mathcal{T}}| - 1\}$, there exists

Model of Computation – Symmetric Topology

A topology $\mathcal{T} = \langle V, |P_{\mathcal{T}}|, R_{\mathcal{T}}, W_{\mathcal{T}} \rangle$ is *symmetric*, iff for any distinct $i, j \in \{0 \ldots |P_{\mathcal{T}}| - 1\}$, there exists

- a *bijection* $f : R_{\mathcal{T}}(i) \rightarrow R_{\mathcal{T}}(j)$, such that $\forall v \in R_{\mathcal{T}}(i) : D_v = D_{f(v)}$, and

Model of Computation – Symmetric Topology

A topology $\mathcal{T} = \langle V, |P_\mathcal{T}|, R_\mathcal{T}, W_\mathcal{T} \rangle$ is *symmetric*, iff for any distinct $i, j \in \{0 \ldots |P_\mathcal{T}| - 1\}$, there exists

- a *bijection* $f : R_\mathcal{T}(i) \to R_\mathcal{T}(j)$, such that $\forall v \in R_\mathcal{T}(i) : D_v = D_{f(v)}$, and
- a *bijection* $g : W_\mathcal{T}(i) \to W_\mathcal{T}(j)$, such that $\forall v \in W_\mathcal{T}(i) : D_v = D_{g(v)}$.

Model of Computation – Symmetric Topology

Model of Computation – Symmetric Topology

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is called *symmetric* iff

Model of Computation – Symmetric Topology

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is called *symmetric* iff

- it has a symmetric topology, and

Model of Computation – Symmetric Topology

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is called *symmetric* iff

- it has a symmetric topology, and
- for any two distinct processes $\pi, \pi' \in P_{\mathcal{D}}$, the following condition holds:

$$\forall (s_0, s_1) \in T_\pi : \exists (s_0', s_1') \in T_{\pi'} : \\ \Big(\forall v \in R_\pi : (v(s_0) = f(v)(s_0')) \wedge (\forall v \in W_\pi : (v(s_1) = g(v)(s_1')) \Big)$$

Model of Computation – Symmetric Topology

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is called *symmetric* iff

- it has a symmetric topology, and
- for any two distinct processes $\pi, \pi' \in P_{\mathcal{D}}$, the following condition holds:

$$\forall (s_0, s_1) \in T_\pi : \exists (s_0', s_1') \in T_{\pi'} :$$
$$\Big(\forall v \in R_\pi : (v(s_0) = f(v)(s_0')) \Big) \land \Big(\forall v \in W_\pi : (v(s_1) = g(v)(s_1')) \Big)$$

The read- and write-sets of all processes are identical up to renaming, and so are their transitions.

Self-stabilization

A distributed program $\mathcal{D} = \langle P_\mathcal{D}, T_\mathcal{D} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

Self-stabilization

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

1. *(Convergence)* For any computation $\overline{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \overline{s}$ ($j \geq 0$), such that $s_j \in LS$.

Self-stabilization

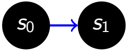
A distributed program $\mathcal{D} = \langle P_\mathcal{D}, T_\mathcal{D} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

1. *(Convergence)* For any computation $\overline{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \overline{s}$ $(j \geq 0)$, such that $s_j \in LS$.

Self-stabilization

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

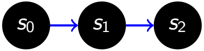
1. *(Convergence)* For any computation $\overline{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \overline{s}$ ($j \geq 0$), such that $s_j \in LS$.



Self-stabilization

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

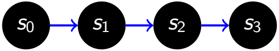
1. *(Convergence)* For any computation $\overline{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \overline{s}$ $(j \geq 0)$, such that $s_j \in LS$.



Self-stabilization

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

1. *(Convergence)* For any computation $\overline{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \overline{s}$ $(j \geq 0)$, such that $s_j \in LS$.



Self-stabilization

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

1. *(Convergence)* For any computation $\overline{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \overline{s}$ $(j \geq 0)$, such that $s_j \in LS$.

Self-stabilization

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

1. *(Convergence)* For any computation $\overline{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \overline{s}$ $(j \geq 0)$, such that $s_j \in LS$.

$$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4$$

Self-stabilization

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

1. *(Convergence)* For any computation $\overline{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \overline{s}$ $(j \geq 0)$, such that $s_j \in LS$.

$$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5$$

Self-stabilization

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

1. *(Convergence)* For any computation $\overline{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \overline{s}$ $(j \geq 0)$, such that $s_j \in LS$.

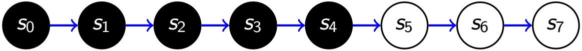
2. *(Closure)* For any transition $(s_0, s_1) \in T_{\mathcal{D}}$, if $s_0 \in LS$, then $s_1 \in LS$.

Self-stabilization

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

1. *(Convergence)* For any computation $\overline{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \overline{s}$ ($j \geq 0$), such that $s_j \in LS$.

2. *(Closure)* For any transition $(s_0, s_1) \in T_{\mathcal{D}}$, if $s_0 \in LS$, then $s_1 \in LS$.



Self-stabilization

A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is *self-stabilizing* for a set LS of legitimate states iff

1. *(Convergence)* For any computation $\overline{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \overline{s}$ $(j \geq 0)$, such that $s_j \in LS$.
2. *(Closure)* For any transition $(s_0, s_1) \in T_{\mathcal{D}}$, if $s_0 \in LS$, then $s_1 \in LS$.

Presentation outline

The Synthesis Problem

- A *parameterized topology* is a sequence of symmetric topologies

$$\mathcal{T}_1, \mathcal{T}_2, \ldots$$

where for all n we have $|P_{\mathcal{T}_n}| = n$ and symmetry conditions hold.

The Synthesis Problem

- A *parameterized topology* is a sequence of symmetric topologies

  $$\mathcal{T}_1, \mathcal{T}_2, \ldots$$

  where for all n we have $|P_{\mathcal{T}_n}| = n$ and symmetry conditions hold.
- A *parameterized program* is a sequence of symmetric distributed programs

  $$\mathcal{D}_1, \mathcal{D}_2, \ldots$$

  such that $\mathcal{D}_i = \mathcal{T}_i^\pi$ for a parameterized topology $\mathcal{T}_1, \mathcal{T}_2, \ldots$, and some process π.

The Synthesis Problem

- A *parameterized topology* is a sequence of symmetric topologies

$$\mathcal{T}_1, \mathcal{T}_2, \ldots$$

where for all n we have $|P_{\mathcal{T}_n}| = n$ and symmetry conditions hold.

- A *parameterized program* is a sequence of symmetric distributed programs

$$\mathcal{D}_1, \mathcal{D}_2, \ldots$$

such that $\mathcal{D}_i = \mathcal{T}_i^\pi$ for a parameterized topology $\mathcal{T}_1, \mathcal{T}_2, \ldots$, and some process π.

Parameterized Topology

The Synthesis Problem

- A *parameterized topology* is a sequence of symmetric topologies

$$\mathcal{T}_1, \mathcal{T}_2, \ldots$$

where for all n we have $|P_{\mathcal{T}_n}| = n$ and symmetry conditions hold.

- A *parameterized program* is a sequence of symmetric distributed programs

$$\mathcal{D}_1, \mathcal{D}_2, \ldots$$

such that $\mathcal{D}_i = \mathcal{T}_i^\pi$ for a parameterized topology $\mathcal{T}_1, \mathcal{T}_2, \ldots$, and some process π.

Parameterized Topology

Conjunctive local uninterpreted predicate LS

The Synthesis Problem

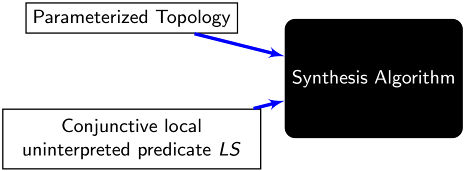
- A *parameterized topology* is a sequence of symmetric topologies

$$\mathcal{T}_1, \mathcal{T}_2, \ldots$$

  where for all n we have $|P_{\mathcal{T}_n}| = n$ and symmetry conditions hold.
- A *parameterized program* is a sequence of symmetric distributed programs

$$\mathcal{D}_1, \mathcal{D}_2, \ldots$$

  such that $\mathcal{D}_i = \mathcal{T}_i^\pi$ for a parameterized topology $\mathcal{T}_1, \mathcal{T}_2, \ldots$, and some process π.



The Synthesis Problem

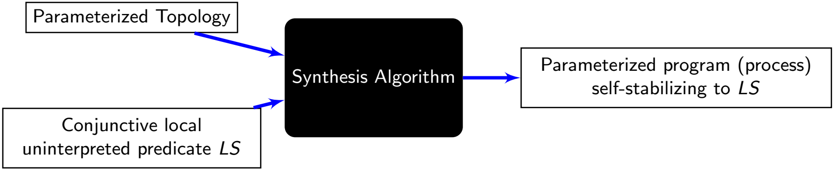
- A *parameterized topology* is a sequence of symmetric topologies

$$\mathcal{T}_1, \mathcal{T}_2, \ldots$$

  where for all n we have $|P_{\mathcal{T}_n}| = n$ and symmetry conditions hold.
- A *parameterized program* is a sequence of symmetric distributed programs

$$\mathcal{D}_1, \mathcal{D}_2, \ldots$$

  such that $\mathcal{D}_i = \mathcal{T}_i^{\pi}$ for a parameterized topology $\mathcal{T}_1, \mathcal{T}_2, \ldots$, and some process π.



Presentation outline

The Notion of Cutoff

For a given parameterized topology and a property, a *cutoff* is a natural number c, such that for any given process π and a locally defined LS the following holds:

> $\mathcal{D}_n = \mathcal{T}_n^\pi$ satisfies the property wrt. LS for all $n \in \mathbb{N}$ iff $\mathcal{D}_i = \mathcal{T}_i^\pi$ satisfies the property wrt. LS for all $i \in [1 \ldots c]$.

Cutoffs for Closure

Cutoffs for Closure

Lemma

For self-stabilizing algorithms on a parameterized symmetric ring the tight cutoffs for closure is $c = l^2 + 1$, where l is the size local state space of each process.

Proof sketch.

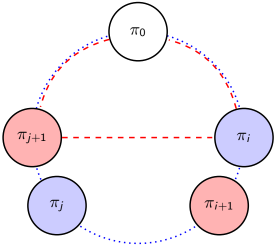
Cutoffs for Closure

Lemma

For self-stabilizing algorithms on a parameterized symmetric ring the tight cutoffs for closure is $c = l^2 + 1$, where l is the size local state space of each process.

Proof sketch.

- Consider a ring of size $M > l^2 + 1$.



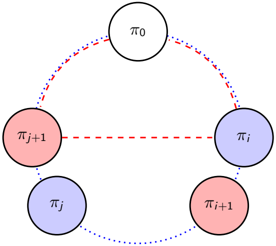
Cutoffs for Closure

Lemma

For self-stabilizing algorithms on a parameterized symmetric ring the tight cutoffs for closure is $c = l^2 + 1$, where l is the size local state space of each process.

Proof sketch.

- Consider a ring of size $M > l^2 + 1$.
- Assume there exists a transition from $s \in LS$ to $s' \notin LS$ by π_0.

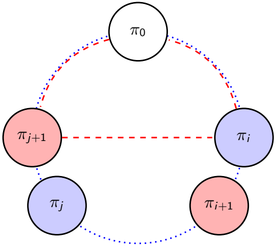


Cutoffs for Closure

Lemma

For self-stabilizing algorithms on a parameterized symmetric ring the tight cutoffs for closure is $c = l^2 + 1$, where l is the size local state space of each process.

Proof sketch.



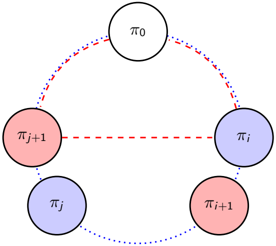
- Consider a ring of size $M > l^2 + 1$.
- Assume there exists a transition from $s \in LS$ to $s' \notin LS$ by π_0.
- Now, consider the $M - 1$ pairs of consecutive processes.

Cutoffs for Closure

Lemma

For self-stabilizing algorithms on a parameterized symmetric ring the tight cutoffs for closure is $c = l^2 + 1$, where l is the size local state space of each process.

Proof sketch.



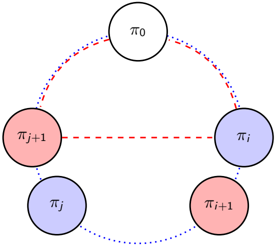
- Consider a ring of size $M > l^2 + 1$.
- Assume there exists a transition from $s \in LS$ to $s' \notin LS$ by π_0.
- Now, consider the $M - 1$ pairs of consecutive processes.
- At least two of these pairs of processes (π_i, π_{i+1}) and (π_j, π_{j+1}) have the same valuation of their write-sets in s.

Cutoffs for Closure

Lemma

For self-stabilizing algorithms on a parameterized symmetric ring the tight cutoffs for closure is $c = l^2 + 1$, where l is the size local state space of each process.

Proof sketch.



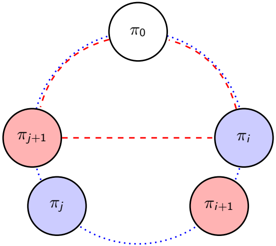
- Consider a ring of size $M > l^2 + 1$.
- Assume there exists a transition from $s \in LS$ to $s' \notin LS$ by π_0.
- Now, consider the $M - 1$ pairs of consecutive processes.
- At least two of these pairs of processes (π_i, π_{i+1}) and (π_j, π_{j+1}) have the same valuation of their write-sets in s.
- Then, we can consider a smaller ring composed of $\pi_0, \dots, \pi_i, \pi_{j+1}, \dots, \pi_M$ with local valuations as in state s

Cutoffs for Closure

Lemma

For self-stabilizing algorithms on a parameterized symmetric ring the tight cutoffs for closure is $c = l^2 + 1$, where l is the size local state space of each process.

Proof sketch.



- Consider a ring of size $M > l^2 + 1$.
- Assume there exists a transition from $s \in LS$ to $s' \notin LS$ by π_0.
- Now, consider the $M - 1$ pairs of consecutive processes.
- At least two of these pairs of processes (π_i, π_{i+1}) and (π_j, π_{j+1}) have the same valuation of their write-sets in s.
- Then, we can consider a smaller ring composed of $\pi_0, \ldots, \pi_i, \pi_{j+1}, \ldots, \pi_M$ with local valuations as in state s
- We can repeat the removal of processes until we arrive at $c = l^2 + 1$.

Cutoffs for Closure and Deadlock-freedom

Theorem

Cutoffs for Closure and Deadlock-freedom

Theorem

For self-stabilizing algorithms on a ring topology, the following are cutoffs for the closure and deadlock-freedom properties:

Cutoffs for Closure and Deadlock-freedom

Theorem

For self-stabilizing algorithms on a ring topology, the following are cutoffs for the closure and deadlock-freedom properties:

- $c = l^2 + 1$, *if LS is locally defined;*

Cutoffs for Closure and Deadlock-freedom

Theorem

For self-stabilizing algorithms on a ring topology, the following are cutoffs for the closure and deadlock-freedom properties:

- $c = l^2 + 1$, *if* LS *is locally defined;*
- $c = l + 1$, *if* LS *is locally defined and* LS_i *only depends on* $W_\mathcal{T}(i)$ *and* $W_\mathcal{T}(i+1)$, *and*

Cutoffs for Closure and Deadlock-freedom

Theorem

For self-stabilizing algorithms on a ring topology, the following are cutoffs for the closure and deadlock-freedom properties:

- $c = l^2 + 1$, *if* LS *is locally defined;*
- $c = l + 1$, *if* LS *is locally defined and* LS_i *only depends on* $W_T(i)$ *and* $W_T(i+1)$, *and*
- $c = 3$, *if* LS *is locally defined and* LS_i *only depends on* $W_T(i)$.

All of the cutoffs are tight under their respective assumptions.

Cutoffs for Convergence

Cutoffs for Convergence

Challenge: We have to consider *infinite* behaviors of the system.

Cutoffs for Convergence

Challenge: We have to consider *infinite* behaviors of the system.

Parameterized verification and synthesis of convergence in symmetric rings is known to be *undecidable*.

Cutoffs for Convergence

Challenge: We have to consider *infinite* behaviors of the system.

Parameterized verification and synthesis of convergence in symmetric rings is known to be *undecidable*.

Idea

We check whether there is a loop that starts and ends in the same local state for an arbitrary process.

Cutoffs for Convergence

Challenge: We have to consider *infinite* behaviors of the system.

Parameterized verification and synthesis of convergence in symmetric rings is known to be *undecidable*.

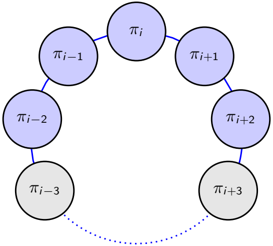
Idea

We check whether there is a loop that starts and ends in the same local state for an arbitrary process.

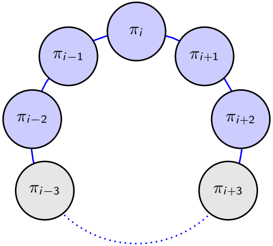
If we can show that this is not possible, then certainly no global loop is possible.

Cutoff for Convergence

Cutoff for Convergence



Cutoff for Convergence

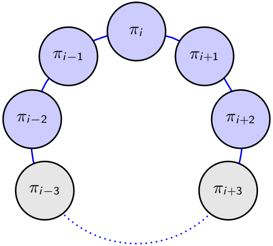


- We fix five processes and define the following property:

$$S \Rightarrow (\Diamond\Box S \vee \neg\Box\Diamond S),$$

where S is the local state of π_i.

Cutoff for Convergence



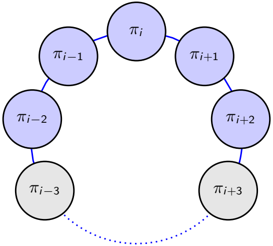
- We fix five processes and define the following property:

$$S \Rightarrow (\Diamond\Box S \vee \neg\Box\Diamond S),$$

where S is the local state of π_i.

- We prove the property in a ring of size 7, assuming that 5 processes behave correctly and the other two processes have the same write-set, but can execute arbitrary transitions.

Cutoff for Convergence

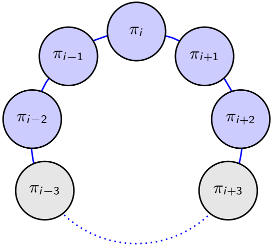


- We fix five processes and define the following property:

$$S \Rightarrow (\Diamond\Box S \ \vee \ \neg\Box\Diamond S),$$

  where S is the local state of π_i.
- We prove the property in a ring of size 7, assuming that 5 processes behave correctly and the other two processes have the same write-set, but can execute arbitrary transitions.
- These two processes *over-approximate* the possible behavior of all other processes.

Cutoff for Convergence



- We fix five processes and define the following property:

$$S \Rightarrow (\Diamond\Box S \vee \neg\Box\Diamond S),$$

where S is the local state of π_i.

- We prove the property in a ring of size 7, assuming that 5 processes behave correctly and the other two processes have the same write-set, but can execute arbitrary transitions.
- These two processes *over-approximate* the possible behavior of all other processes.
- The *precision of the abstraction* can be refined by increasing the number of processes that behave according to the protocol, or by including the local state of additional processes into S.

General Cutoff Result

General Cutoff Result

Theorem

Let $\mathcal{T}_1, \mathcal{T}_2, \ldots$ be a parameterized ring topology, π a process, and let LS be locally defined by LS_i. Let c_1 and c_2 be cutoffs for closure and deadlock detection wrt. LS, respectively. If

General Cutoff Result

Theorem

Let $\mathcal{T}_1, \mathcal{T}_2, \ldots$ be a parameterized ring topology, π a process, and let LS be locally defined by LS_i. Let c_1 and c_2 be cutoffs for closure and deadlock detection wrt. LS, respectively. If

- closure *holds in rings of size up to* c_1,

General Cutoff Result

Theorem

Let $\mathcal{T}_1, \mathcal{T}_2, \ldots$ be a parameterized ring topology, π a process, and let LS be locally defined by LS_i. Let c_1 and c_2 be cutoffs for closure and deadlock detection wrt. LS, respectively. If

- closure *holds in rings of size up to* c_1,
- deadlocks *outside of LS are impossible in rings of size up to* c_2, *and*

General Cutoff Result

Theorem

Let $\mathcal{T}_1, \mathcal{T}_2, \ldots$ be a parameterized ring topology, π a process, and let LS be locally defined by LS_i. Let c_1 and c_2 be cutoffs for closure and deadlock detection wrt. LS, respectively. If

- closure *holds in rings of size up to* c_1,
- deadlocks *outside of LS are impossible in rings of size up to* c_2, *and*
- *the absence of* cycles *can be proven in rings of up to size* 4 *and in an abstract system as above,*

General Cutoff Result

Theorem

Let $\mathcal{T}_1, \mathcal{T}_2, \ldots$ be a parameterized ring topology, π a process, and let LS be locally defined by LS_i. Let c_1 and c_2 be cutoffs for closure and deadlock detection wrt. LS, respectively. If

- closure *holds in rings of size up to* c_1,
- deadlocks *outside of LS are impossible in rings of size up to* c_2, *and*
- *the absence of* cycles *can be proven in rings of up to size* 4 *and in an abstract system as above,*

then every instance *of the parameterized program $\mathcal{D}_1 = \mathcal{T}_1^\pi, \mathcal{D}_2 = \mathcal{T}_2^\pi, \ldots$ is self-stabilizing to LS.*

Presentation outline

SMT-based Synthesis – The Tool ASSESS [*]

[*] F. Faghih and B. Bonakdarpour: *ASSESS: A Tool for Synthesizing Self-Stabilizing Protocols*, SSS 2017.

SMT-based Synthesis – The Tool ASSESS [*]

[*] F. Faghih and B. Bonakdarpour: *ASSESS: A Tool for Synthesizing Self-Stabilizing Protocols*, SSS 2017.

SMT-based Synthesis – The Tool ASSESS [*]

Fixed Topology

[*] F. Faghih and B. Bonakdarpour: *ASSESS: A Tool for Synthesizing Self-Stabilizing Protocols*, SSS 2017.

SMT-based Synthesis – The Tool ASSESS [*]

Fixed Topology

Timing model

[*] F. Faghih and B. Bonakdarpour: *ASSESS: A Tool for Synthesizing Self-Stabilizing Protocols*, SSS 2017.

SMT-based Synthesis – The Tool ASSESS [*]

Fixed Topology

Timing model

Symmetry Type

[*] F. Faghih and B. Bonakdarpour: *ASSESS: A Tool for Synthesizing Self-Stabilizing Protocols*, SSS 2017.

SMT-based Synthesis – The Tool ASSESS [*]

Fixed Topology

Timing model

Symmetry Type

Stabilization Type

[*] F. Faghih and B. Bonakdarpour: *ASSESS: A Tool for Synthesizing Self-Stabilizing Protocols*, SSS 2017.

SMT-based Synthesis – The Tool ASSESS [*]

$\boxed{\text{Fixed Topology}}$

$\boxed{\text{Timing model}}$

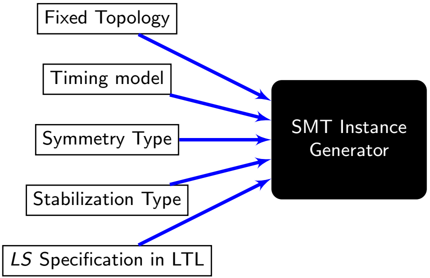
$\boxed{\text{Symmetry Type}}$

$\boxed{\text{Stabilization Type}}$

$\boxed{LS \text{ Specification in LTL}}$

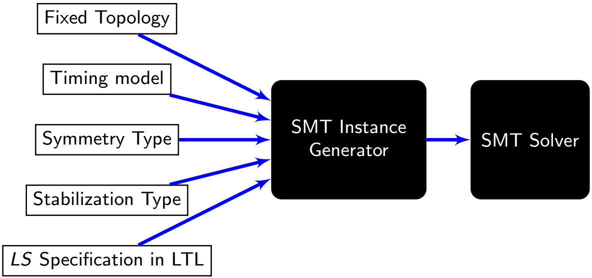
[*] F. Faghih and B. Bonakdarpour: *ASSESS: A Tool for Synthesizing Self-Stabilizing Protocols*, SSS 2017.

SMT-based Synthesis – The Tool ASSESS [*]



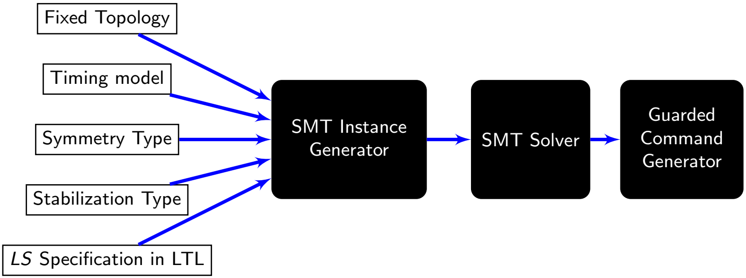
[*] F. Faghih and B. Bonakdarpour: *ASSESS: A Tool for Synthesizing Self-Stabilizing Protocols*, SSS 2017.

SMT-based Synthesis – The Tool ASSESS [*]



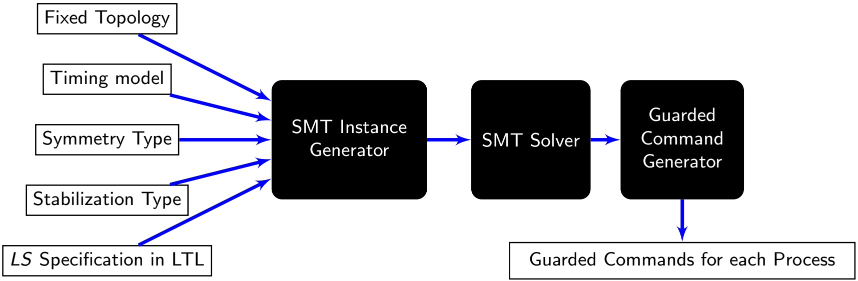
[*] F. Faghih and B. Bonakdarpour: *ASSESS: A Tool for Synthesizing Self-Stabilizing Protocols*, SSS 2017.

SMT-based Synthesis – The Tool ASSESS [*]



[*] F. Faghih and B. Bonakdarpour: *ASSESS: A Tool for Synthesizing Self-Stabilizing Protocols*, SSS 2017.

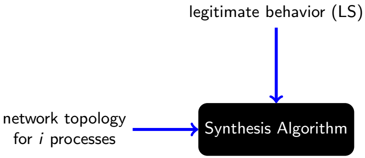
SMT-based Synthesis – The Tool ASSESS [*]



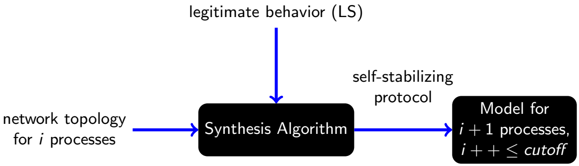
[*] F. Faghih and B. Bonakdarpour: *ASSESS: A Tool for Synthesizing Self-Stabilizing Protocols*, SSS 2017.

Counterexample-guided Synthesis

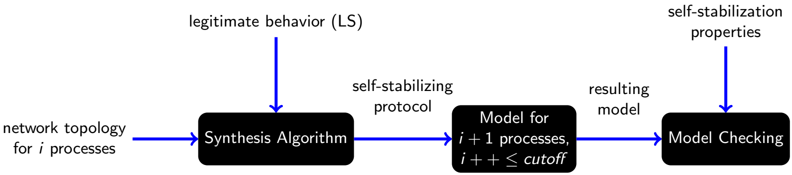
Counterexample-guided Synthesis



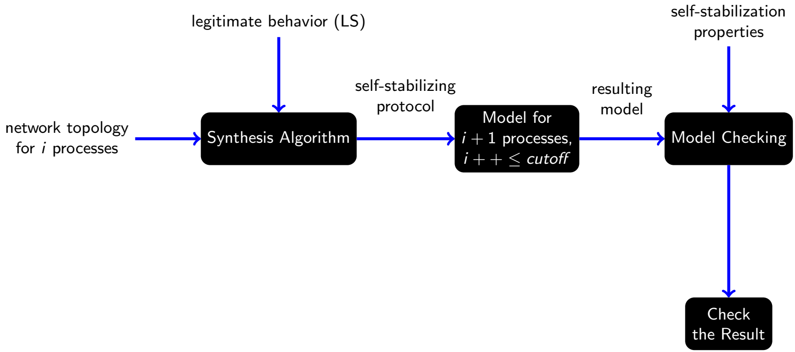
Counterexample-guided Synthesis



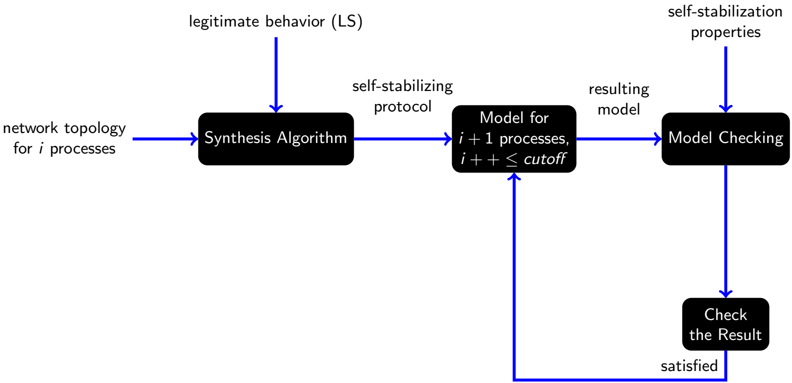
Counterexample-guided Synthesis



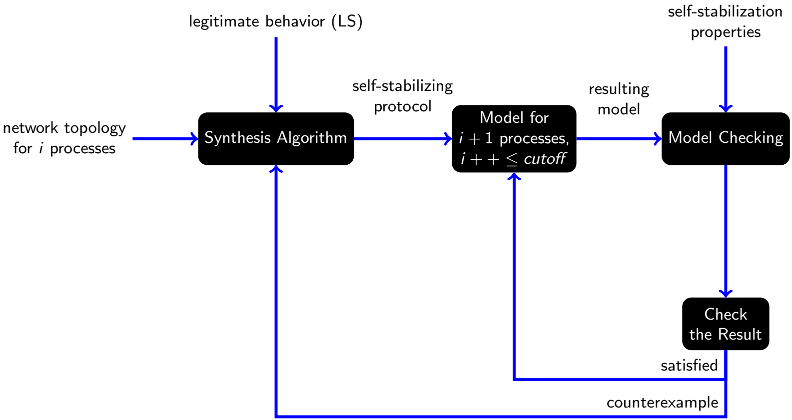
Counterexample-guided Synthesis



Counterexample-guided Synthesis



Counterexample-guided Synthesis



Experimental Results

Problem	**cutoff #**	**Heuristic**	**Synthesis Time**	**Model Checking Time**
Three Coloring	10	Local *LS*	7m 3sec	16 msec
Three Coloring	10	Progress	9m 5sec	16 msec
One-Bit MM	5	Local *LS*	1m 48sec	27 msec
One-Bit MM	5	Progress	1m 44sec	33 msec
Maximal Matching	10	Local *LS*	7m 59sec	36 msec
Maximal Matching	10	Progress	4m 57sec	37 msec
Maximal Independent Set	5	Local *LS*	10sec	18 msec

Experimental Results

Guarded commands of the *1-bit maximal matching* problem:

$$\begin{array}{rrcl} \pi_i : & (x_i = \mathit{false}) \wedge (x_{(i+1)} = \mathit{false}) \wedge (x_{(i-1)} = \mathit{false}) & \longrightarrow & x_i := \mathit{true} \\ & (x_i = \mathit{true}) \wedge (x_{(i+1)} = \mathit{true}) & \longrightarrow & x_i := \mathit{false} \end{array}$$

Experimental Results

Guarded commands of the *1-bit maximal matching* problem:

$$
\begin{aligned}
\pi_i : \quad (x_i = \mathit{false}) \wedge (x_{(i+1)} = \mathit{false}) \wedge (x_{(i-1)} = \mathit{false}) \quad &\longrightarrow \quad x_i := \mathit{true} \\
(x_i = \mathit{true}) \wedge (x_{(i+1)} = \mathit{true}) \quad &\longrightarrow \quad x_i := \mathit{false}
\end{aligned}
$$

The *interpretation function* for match_i is the following:

$$
\begin{aligned}
\mathit{match}_i : \quad (x_i = \mathit{true}) \wedge (x_{(i+1)} = \mathit{true}) \wedge (x_{(i-1)} = \mathit{true}) \quad &\mapsto \quad l \\
(x_{(i+1)} = \mathit{false}) \wedge (x_{(i-1)} = \mathit{false}) \quad &\mapsto \quad l \\
(x_i = \mathit{true}) \wedge (x_{(i+1)} = \mathit{false}) \wedge (x_{(i-1)} = \mathit{true}) \quad &\mapsto \quad r \\
(x_i = \mathit{false}) \wedge (x_{(i+1)} = \mathit{true}) \quad &\mapsto \quad r \\
(x_i = \mathit{true}) \wedge (x_{(i+1)} = \mathit{true}) \wedge (x_{(i-1)} = \mathit{false}) \quad &\mapsto \quad n \\
(x_i = \mathit{false}) \wedge (x_{(i+1)} = \mathit{false}) \wedge (x_{(i-1)} = \mathit{true}) \quad &\mapsto \quad n
\end{aligned}
$$

Presentation outline

Summary

Summary

- We proposed a new method for *parameterized synthesis* of self-stabilizing algorithms in symmetric rings using *cutoff* points

Summary

- We proposed a new method for *parameterized synthesis* of self-stabilizing algorithms in symmetric rings using *cutoff* points
- To scale up to the cutoff point, we introduced an iterative loop of synthesis and verification guided by counterexamples.

Summary

- We proposed a new method for *parameterized synthesis* of self-stabilizing algorithms in symmetric rings using *cutoff* points
- To scale up to the cutoff point, we introduced an iterative loop of synthesis and verification guided by counterexamples.
- We synthesized (in less then *10 minutes*) parameterized:
  - Three coloring
  - Maximal matching
  - Maximal independent set

Future Work

Future Work

- Other *topologies* (grid, tree, torus, line, etc).

Future Work

- Other *topologies* (grid, tree, torus, line, etc).
- *Asymmetric* netowrk

Future Work

- Other *topologies* (grid, tree, torus, line, etc).
- *Asymmetric* netowrk
- *Dynamic* networks

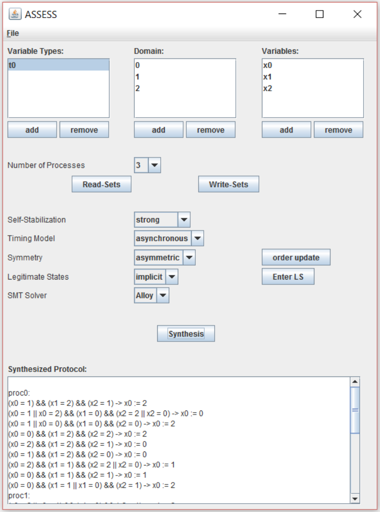
Future Work

- Other *topologies* (grid, tree, torus, line, etc).
- *Asymmetric* netowrk
- *Dynamic* networks
- Protocol *live* in the set of legitimate states.

Commercial 1!

Please download and use ASSESS!

http://web.cs.iastate.edu/~borzoo/assess



Commercial 2!

I am looking for Ph.D. students and postdocs!

Thank you!