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\(^3\)
Motivation

2 Preliminaries

3 Problem Statement

4 Cutoff Results

5 Scalable Synthesis

6 Conclusion
Motivation

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

Formal Specification
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:

Formal Specification → Synthesis Algorithm
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!

Formal Specification \rightarrow Synthesis Algorithm \rightarrow Program
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; 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;
correct by construction!

Formal Specification → Synthesis Algorithm → Program

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; 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 \textit{computationally intractable} and in many cases \textit{undecidable}.

- It is particularly useful to deal with small but \textit{intricate components} of a system, e.g., concurrent/distributed algorithms.
Self-stabilization is a versatile technique for forward fault recovery that has two key features:
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.
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.

- **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?

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.
Why Synthesizing Self-stabilizing Algorithms?

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


Related Work


Related Work


Related Work


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 \textit{symmetric} and \textit{parameterized} rings.

Our Approach

A \textit{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:
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.
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

1. Motivation
2. Preliminaries
3. Problem Statement
4. Cutoff Results
5. Scalable Synthesis
6. Conclusion
Model of Computation – Distributed Programs

Variables: \[ V = \{ x_0, x_1, x_2, x_3 \}, \text{ each } x_i \text{ is a Boolean.} \]
Model of Computation – Distributed Programs

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

Distributed program: \( D = \langle P_D, T_D \rangle \).
Model of Computation – Distributed Programs

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

Distributed program: \( D = \langle P_D, T_D \rangle \).

Processes: \( P_D = \{ \pi_0, \pi_1, \pi_2, \pi_3 \} \).
Model of Computation – Distributed Programs

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

Distributed program: $D = \langle P_D, T_D \rangle$.

Processes: $P_D = \{\pi_0, \pi_1, \pi_2, \pi_3\}$.

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: $D = \langle P_D, T_D \rangle$.

Processes: $P_D = \{\pi_0, \pi_1, \pi_2, \pi_3\}$.

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: \( D = \langle P_D, T_D \rangle \).

Processes: \( P_D = \{\pi_0, \pi_1, \pi_2, \pi_3\} \).

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

\[ t_1 = ([x_0 = \text{false}, x_1 = \text{false}, x_2 = \text{false}, x_3 = \text{false}], \]
\[ [x_0 = \text{true}, x_1 = \text{false}, x_2 = \text{false}, x_3 = \text{false}]) \]
Model of Computation – Distributed Programs

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

**Distributed program:** \( D = \langle P_D, T_D \rangle \).

**Processes:** \( P_D = \{\pi_0, \pi_1, \pi_2, \pi_3\} \).

**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**

\[
t_1 = ([x_0 = \text{false}, x_1 = \text{false}, x_2 = \text{false}, x_3 = \text{false}], \\
[ x_0 = \text{true}, x_1 = \text{false}, x_2 = \text{false}, x_3 = \text{false}])
\]
Model of Computation – Distributed Programs

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

Distributed program: \( D = \langle P_D, T_D \rangle \).

Processes: \( P_D = \{\pi_0, \pi_1, \pi_2, \pi_3\} \).

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

\( t_1 = ([x_0 = false, x_1 = false, x_2 = false, x_3 = false], \\
\quad [x_0 = true, x_1 = false, x_2 = false, x_3 = false]) \)

\( t_2 = ([x_0 = false, x_1 = false, x_2 = true, x_3 = false], \\
\quad [x_0 = true, x_1 = false, x_2 = true, x_3 = false]) \)
Model of Computation – Distributed Programs

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

Distributed program: \( D = \langle P_D, T_D \rangle \).

Processes: \( P_D = \{\pi_0, \pi_1, \pi_2, \pi_3\} \).

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: \( \mathcal{D} = \langle P_D, T_D \rangle \).

Processes: \( P_D = \{\pi_0, \pi_1, \pi_2, \pi_3\} \).

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

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

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: \( \mathcal{D} = \langle P_D, T_D \rangle \).

Processes: \( P_D = \{\pi_0, \pi_1, \pi_2, \pi_3\} \).

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

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

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

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

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

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

A topology $T = \langle V, |P_T|, R_T, W_T \rangle$ is symmetric, iff for any distinct $i, j \in \{0, \ldots, |P_T| - 1\}$, there exists a bijection $f: R_T(i) \rightarrow R_T(j)$, such that $\forall v \in R_T(i): D_v = D_{f(v)}$, and a bijection $g: W_T(i) \rightarrow W_T(j)$, such that $\forall v \in W_T(i): D_v = D_{g(v)}$. 
Model of Computation – Symmetric Topology

A topology $\mathcal{T} = \langle V, |P_T|, R_T, W_T \rangle$ is \textit{symmetric}, iff for any distinct $i, j \in \{0 \ldots |P_T| - 1\}$, there exists
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 $\mathcal{T} = \langle V, |P_T|, R_T, W_T \rangle$ is symmetric, iff for any distinct $i, j \in \{0 \ldots |P_T| - 1\}$, there exists

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

- a bijection $g : W_T(i) \rightarrow W_T(j)$, such that $\forall v \in W_T(i) : D_v = D_{g(v)}$. 
Model of Computation – Symmetric Topology

A distributed program \( D = \langle P_D, T_D \rangle \) is called symmetric iff it has a symmetric topology, and for any two distinct processes \( \pi, \pi' \in P_D \), the following condition holds:

\[
\forall (s_0, s_1) \in T_\pi: \exists (s'_0, s'_1) \in T_{\pi'}: \\
(\forall v \in R_\pi: (v(s_0) = f(v)(s'_0))) \land (\forall v \in W_\pi: (v(s_1) = g(v)(s'_1)))
\]

The read- and write-sets of all processes are identical up to renaming, and so are their transitions.
A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is called \textit{symmetric} iff

$$\forall (s_0, s_1) \in T_\pi : \exists (s'_0, s'_1) \in T_{\pi'} :$$

$$\left( \forall v \in R_\pi : (v(s_0) = f(v)(s'_0)) \right) \land \left( \forall v \in W_\pi : (v(s_1) = g(v)(s'_1)) \right)$$

The read- and write-sets of all processes are identical up to renaming, and so are their transitions.
A distributed program $\mathcal{D} = \langle P_D, T_D \rangle$ is called \textit{symmetric} iff

- it has a symmetric topology, and
A distributed program $\mathcal{D} = \langle P_D, T_D \rangle$ is called *symmetric* iff

- it has a symmetric topology, and
- for any two distinct processes $\pi, \pi' \in P_D$, the following condition holds:

$$\forall (s_0, s_1) \in T_\pi : \exists (s'_0, s'_1) \in T_{\pi'} :$$

$$\left( \forall v \in R_\pi : (v(s_0) = f(v)(s'_0)) \land (\forall v \in W_\pi : (v(s_1) = g(v)(s'_1)) \right)$$

The read- and write-sets of all processes are identical up to renaming, and so are their transitions.
A distributed program $\mathcal{D} = \langle P_D, T_D \rangle$ is called symmetric iff

- it has a symmetric topology, and
- for any two distinct processes $\pi, \pi' \in P_D$, the following condition holds:

$$\forall (s_0, s_1) \in T_\pi : \exists (s'_0, s'_1) \in T_{\pi'} : \left( \forall v \in R_\pi : (v(s_0) = f(v)(s'_0)) \land (\forall v \in W_\pi : (v(s_1) = g(v)(s'_1)) \right)$$

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

1. (Convergence) For any computation $s = s_0 s_1 \cdots$, there exists a state $s_j \in s(j \geq 0)$, 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 $s_1 \in LS$. 
A distributed program $\mathcal{D} = \langle P_\mathcal{D}, T_\mathcal{D} \rangle$ is \textit{self-stabilizing} for a set $LS$ of legitimate states iff

1. \textbf{(Convergence)} For any computation $\bar{s} = s_0s_1\cdots$, there exists a state $s_j \in \bar{s}$ ($j \geq 0$), such that $s_j \in LS$. 
A distributed program $D = \langle P_D, T_D \rangle$ is self-stabilizing for a set $LS$ of legitimate states iff

1. (Convergence) For any computation $\bar{s} = s_0s_1 \cdots$, there exists a state $s_j \in \bar{s} (j \geq 0)$, such that $s_j \in LS$. 

"S0"
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 $\bar{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \bar{s}$ ($j \geq 0$), such that $s_j \in LS$. 

$$
\text{Graph:} \quad S_0 \rightarrow S_1
$$
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

- (Convergence) For any computation $\bar{s} = s_0s_1 \cdots$, there exists a state $s_j \in \bar{s}$ ($j \geq 0$), such that $s_j \in LS$. 

Diagram:

```
S0 → S1 → S2
```
A distributed program $\mathcal{D} = \langle P_D, T_D \rangle$ is \textit{self-stabilizing} for a set $LS$ of legitimate states iff

1. \textit{(Convergence)} For any computation $\bar{s} = s_0 s_1 \cdots$, there exists a state $s_j \in \bar{s}$ ($j \geq 0$), such that $s_j \in LS$. 
A distributed program $\mathcal{D} = \langle P_D, T_D \rangle$ is self-stabilizing for a set $LS$ of legitimate states iff

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

\[ \begin{array}{ccccccc}
S_0 & \rightarrow & S_1 & \rightarrow & S_2 & \rightarrow & S_3 & \rightarrow & S_4 \\
\end{array} \]
A distributed program $\mathcal{D} = \langle P_D, T_D \rangle$ is self-stabilizing for a set $LS$ of legitimate states iff

1. (Convergence) For any computation $\bar{s} = s_0s_1 \cdots$, there exists a state $s_j \in \bar{s}$ ($j \geq 0$), such that $s_j \in LS$. 

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \]
A distributed program $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is \textit{self-stabilizing} for a set $LS$ of legitimate states iff

1. \textit{(Convergence)} For any computation $\bar{s} = s_0s_1 \cdots$, there exists a state $s_j \in \bar{s}$ ($j \geq 0$), such that $s_j \in LS$. 

\begin{center}
\begin{tikzpicture}
  \node[shape=circle,draw=black,fill=black] (S0) at (1,0) {$s_0$};
  \node[shape=circle,draw=black,fill=black] (S1) at (2,0) {$s_1$};
  \node[shape=circle,draw=black,fill=black] (S2) at (3,0) {$s_2$};
  \node[shape=circle,draw=black,fill=black] (S3) at (4,0) {$s_3$};
  \node[shape=circle,draw=black,fill=black] (S4) at (5,0) {$s_4$};
  \node[shape=circle,draw=black,fill=black] (S5) at (6,0) {$s_5$};
  \draw[->] (S0) -- (S1);
  \draw[->] (S1) -- (S2);
  \draw[->] (S2) -- (S3);
  \draw[->] (S3) -- (S4);
  \draw[->,blue] (S4) -- (S5);
\end{tikzpicture}
\end{center}
A distributed program $\mathcal{D} = \langle P_D, T_D \rangle$ is self-stabilizing for a set $LS$ of legitimate states iff

1. **(Convergence)**  For any computation $\bar{s} = s_0s_1 \cdots$, there exists a state $s_j \in \bar{s} \ (j \geq 0)$, such that $s_j \in LS$. 

2. **(Closure)**  For any transition $(s_0, s_1) \in T_D$, if $s_0 \in LS$, then $s_1 \in LS$. 

\[ \begin{array}{ccccccc}
S_0 & \rightarrow & S_1 & \rightarrow & S_2 & \rightarrow & S_3 & \rightarrow & S_4 & \rightarrow & S_5 & \rightarrow & S_6 \\
\end{array} \]
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 \( \bar{s} = s_0s_1 \cdots \), there exists a state \( s_j \in \bar{s} \ (j \geq 0) \), 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 \( s_1 \in LS \).
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 $\bar{s} = s_0s_1 \cdots$, there exists a state $s_j \in \bar{s}$ ($j \geq 0$), 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 $s_1 \in LS$. 

![Diagram of states](image)
Presentation outline

1. Motivation
2. Preliminaries
3. Problem Statement
4. Cutoff Results
5. Scalable Synthesis
6. Conclusion
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 \( \pi \).
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 \( \pi \).
The Synthesis Problem

- A *parameterized topology* is a sequence of symmetric topologies
  \[ T_1, T_2, \ldots \]
  where for all \( n \) we have \( |P_{T_n}| = n \) and symmetry conditions hold.

- A *parameterized program* is a sequence of symmetric distributed programs
  \[ D_1, D_2, \ldots \]
  such that \( D_i = T_i^{\pi} \) for a parameterized topology \( T_1, T_2, \ldots \), and some process \( \pi \).
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 \( \pi \).
The Synthesis Problem

- A \textit{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 \textit{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 \( \pi \).
Presentation outline

1. Motivation
2. Preliminaries
3. Problem Statement
4. Cutoff Results
5. Scalable Synthesis
6. Conclusion
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 \( \pi \) and a locally defined \( LS \) the following holds:

\[
\mathcal{D}_n = \mathcal{T}_n^\pi \text{ satisfies the property wrt. } LS \text{ for all } n \in \mathbb{N} \quad \text{iff} \quad \mathcal{D}_i = \mathcal{T}_i^\pi \text{ satisfies the property wrt. } LS \text{ for all } i \in [1 \ldots c].
\]
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 \( s' \in LS \) by \( \pi_0 \).

Now, consider the \( M - 1 \) pairs of consecutive processes. At least two of these pairs of processes \( (\pi_i, \pi_{i+1}) \) and \( (\pi_j, \pi_{j+1}) \) 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

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.
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 \( s' \not\in LS \) by \( \pi_0 \).
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 \( \pi_0 \).
- Now, consider the \( M - 1 \) pairs of consecutive processes.
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 \( \pi_0 \).
- Now, consider the \( M - 1 \) pairs of consecutive processes.
- At least two of these pairs of processes \((\pi_i, \pi_{i+1})\) and \((\pi_j, \pi_{j+1})\) have the same valuation of their write-sets in \( s \).
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 \( \pi_0 \).
- Now, consider the \( M - 1 \) pairs of consecutive processes.
- At least two of these pairs of processes \((\pi_i, \pi_{i+1})\) and \((\pi_j, \pi_{j+1})\) 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 \( s \).
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 \( \pi_0 \).
- Now, consider the \( M - 1 \) pairs of consecutive processes.
- At least two of these pairs of processes \((\pi_i, \pi_{i+1})\) and \((\pi_j, \pi_{j+1})\) 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

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);

\[ c = 3 \], 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 Closure and Deadlock-freedom

Theorem

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

\[ c = l^2 + 1, \text{ if } LS \text{ is locally defined}; \]
\[ c = l + 1, \text{ if } LS \text{ is locally defined and } LS_i \text{ only depends on } W_T(i) \text{ and } W_T(i+1); \]
\[ c = 3, \text{ if } LS \text{ is locally defined and } LS_i \text{ only depends on } W_T(i). \]
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; only depends on $W_T(i)$ and $W_T(i + 1)$, and
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
Challenge: We have to consider *infinite* behaviors of the system.
Challenge: We have to consider infinite behaviors of the system.

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

We fix five processes and define the following property:

\[ S \Rightarrow (S \lor \neg S) \]

where \( S \) is the local state of \( \pi_i \).

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 \( S \).
Cutoff for Convergence

We fix five processes and define the following property:

\[ S \Rightarrow (S \lor \neg S) \]

where \( S \) is the local state of \( \pi_i \).

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 \( S \).
We fix five processes and define the following property:

\[ S \Rightarrow (\lozenge \square S \lor \neg \square \lozenge S) \]

where \( S \) is the local state of \( \pi_i \).
We fix five processes and define the following property:

\[ S \Rightarrow (\Diamond \square S \lor \neg \square \Diamond S), \]

where \( S \) is the local state of \( \pi_i \).

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.
We fix five processes and define the following property:

\[ S \Rightarrow (\Diamond \Box S \lor \neg \Box \Diamond S), \]

where \( S \) is the local state of \( \pi_i \).

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.
We fix five processes and define the following property:

\[ S \Rightarrow (\Diamond \lozenge S \lor \neg \Box \Diamond S), \]

where \( S \) is the local state of \( \pi_i \).

- 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 \( S \).
Theorem

Let $T_1, T_2, \ldots$ be a parameterized ring topology, $\pi$ 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 $D_1 = T_\pi_1, D_2 = T_\pi_2, \ldots$ is self-stabilizing to $LS$. 
Theorem

Let $T_1, T_2, \ldots$ be a parameterized ring topology, $\pi$ 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
Theorem

Let \( T_1, T_2, \ldots \) be a parameterized ring topology, \( \pi \) 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 \( c_1 \),
Let $T_1, T_2, \ldots$ be a parameterized ring topology, $\pi$ 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
Theorem

Let $T_1, T_2, \ldots$ be a parameterized ring topology, $\pi$ 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,
Theorem

Let $T_1, T_2, \ldots$ be a parameterized ring topology, $\pi$ 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 $\mathcal{D}_1 = T_1^\pi, \mathcal{D}_2 = T_2^\pi, \ldots$ is self-stabilizing to $LS$. 

SMT-based Synthesis – The Tool ASSESS [*]

SMT-based Synthesis – The Tool ASSESS [*]

SMT-based Synthesis – The Tool ASSESS [*]

[ ] Fixed Topology

SMT-based Synthesis – The Tool ASSESS [*]

**Fixed Topology**

**Timing model**

SMT-based Synthesis – The Tool ASSESS [*]

- Fixed Topology
- Timing model
- Symmetry Type

SMT-based Synthesis – The Tool ASSESS [*]

- Fixed Topology
- Timing model
- Symmetry Type
- Stabilization Type

SMT-based Synthesis – The Tool ASSESS [*]

- Fixed Topology
- Timing model
- Symmetry Type
- Stabilization Type
- *LS Specification in LTL*

SMT-based Synthesis – The Tool ASSESS [*]

[Fixed Topology]

[Timing model]

[Symmetry Type]

[Stabilization Type]

[LS Specification in LTL]

SMT Instance Generator

SMT-based Synthesis – The Tool ASSESS [*]

SMT-based Synthesis – The Tool ASSESS [*]

SMT-based Synthesis – The Tool ASSESS [*]

Counterexample-guided Synthesis
Counterexample-guided Synthesis

Diagram:
- **network topology for \(i\) processes**
- **legitimate behavior (LS)**
- **Synthesis Algorithm**

Diagram Description:
1. The network topology for \(i\) processes is connected to the Synthesis Algorithm.
2. The Synthesis Algorithm produces legitimate behavior (LS).

Concepts:
- **Network topology**: The structure of the connections between processes.
- **Legitimate behavior**: The desired behavior that the system should maintain.
- **Synthesis Algorithm**: A method for automatically generating a system that meets the specified requirements.
Counterexample-guided Synthesis

network topology for $i$ processes

Synthesis Algorithm

self-stabilizing protocol

Model for $i + 1$ processes, $i + + \leq \text{cutoff}$

legitimate behavior (LS)
Counterexample-guided Synthesis

network topology for $i$ processes

Synthesis Algorithm

legitimate behavior (LS)

self-stabilizing protocol

Model for $i + 1$ processes, $i + + \leq \text{cutoff}$

resulting model

self-stabilization properties

Model Checking
Counterexample-guided Synthesis

- Motivation
- Preliminaries
- Problem Statement
- Cutoff Results
- Scalable Synthesis
- Conclusion

**Network Topology** for \( i \) processes

**Synthesis Algorithm**

**Legitimate Behavior (LS)**

**Self-stabilizing Protocol**

**Model for** \( i + 1 \) processes, \( i + + \leq \text{cutoff} \)

**Model Checking**

**Self-stabilization Properties**

- **Resulting Model**
- **Check the Result**
Counterexample-guided Synthesis

The process begins with the network topology for \( i \) processes, which is input into the synthesis algorithm. The algorithm then generates the legitimate behavior (LS). A self-stabilizing protocol is employed for \( i+1 \) processes, where \( i++ \leq \text{cutoff} \). The resulting model is then checked for self-stabilization properties. If the result is satisfied, the process is complete; otherwise, a counterexample is found and the process repeats.
Counterexample-guided Synthesis

- Synthesis Algorithm
  - network topology for $i$ processes
  - legitimate behavior (LS)
- Model for $i + 1$ processes, $i + 1 \leq \text{cutoff}$
  - self-stabilizing protocol
  - resulting model
- Model Checking
  - self-stabilization properties
- Check the Result
  - satisfied
  - counterexample
### Experimental Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>cutoff #</th>
<th>Heuristic</th>
<th>Synthesis Time</th>
<th>Model Checking Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three Coloring</td>
<td>10</td>
<td>Local LS</td>
<td>7m 3sec</td>
<td>16 msec</td>
</tr>
<tr>
<td>Three Coloring</td>
<td>10</td>
<td>Progress</td>
<td>9m 5sec</td>
<td>16 msec</td>
</tr>
<tr>
<td>One-Bit MM</td>
<td>5</td>
<td>Local LS</td>
<td>1m 48sec</td>
<td>27 msec</td>
</tr>
<tr>
<td>One-Bit MM</td>
<td>5</td>
<td>Progress</td>
<td>1m 44sec</td>
<td>33 msec</td>
</tr>
<tr>
<td>Maximal Matching</td>
<td>10</td>
<td>Local LS</td>
<td>7m 59sec</td>
<td>36 msec</td>
</tr>
<tr>
<td>Maximal Matching</td>
<td>10</td>
<td>Progress</td>
<td>4m 57sec</td>
<td>37 msec</td>
</tr>
<tr>
<td>Maximal Independent Set</td>
<td>5</td>
<td>Local LS</td>
<td>10sec</td>
<td>18 msec</td>
</tr>
</tbody>
</table>
Guarded commands of the 1-bit maximal matching problem:

\[ \pi_i : \quad (x_i = false) \land (x_{i+1} = false) \land (x_{i-1} = false) \quad \rightarrow \quad x_i := true \]

\[ (x_i = true) \land (x_{i+1} = true) \quad \rightarrow \quad x_i := false \]
Guarded commands of the 1-bit maximal matching problem:

\[ \pi_i : (x_i = false) \land (x_{i+1} = false) \land (x_{i-1} = false) \rightarrow x_i := true \]

\[ (x_i = true) \land (x_{i+1} = true) \rightarrow x_i := false \]

The interpretation function for \( \text{match}_i \) is the following:

\[ \text{match}_i : \begin{align*}
(x_i = true) \land (x_{i+1} = true) \land (x_{i-1} = true) & \rightarrow l \\
(x_{i+1} = false) \land (x_{i-1} = false) & \rightarrow l \\
(x_i = true) \land (x_{i+1} = false) \land (x_{i-1} = true) & \rightarrow r \\
(x_i = false) \land (x_{i+1} = true) & \rightarrow r \\
(x_i = true) \land (x_{i+1} = true) \land (x_{i-1} = false) & \rightarrow n \\
(x_i = false) \land (x_{i+1} = false) \land (x_{i-1} = true) & \rightarrow n
\end{align*} \]
Presentation outline

1. Motivation
2. Preliminaries
3. Problem Statement
4. Cutoff Results
5. Scalable Synthesis
6. Conclusion
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 than 10 minutes) parameterized:

- Three coloring
- Maximal matching
- Maximal independent set
We proposed a new method for *parameterized synthesis* of self-stabilizing algorithms in symmetric rings using *cutoff* points.
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

- 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* network

- *Dynamic* networks
Future Work

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

- *Asymmetric* networks

- *Dynamic* networks

- Protocol *live* in the set of legitimate states.
Please download and use ASSESS!

http://web.cs.iastate.edu/~borzoo/assess
I am looking for Ph.D. students and postdocs!
Thank you!