Noname manuscript No.
(will be inserted by the editor)

Parameterized Synthesis of Self-Stabilizing Protocols in Symmetric Networks

Nahal Mirzaie · Fathiyeh Faghih · Swen Jacobs · Borzoo Bonakdarpour

Received: date / Accepted: date

Abstract *Self-stabilization* in distributed systems is a technique to guarantee *convergence* to a set of *legitimate states* without external intervention when a transient fault or bad initialization occurs. Recently, there has been a surge of efforts in designing techniques for automated synthesis of self-stabilizing algorithms that are correct by construction. Most of these techniques, however, are not parameterized, meaning that they can only synthesize a solution for a fixed and predetermined number of processes. In this paper, we report a breakthrough in parameterized synthesis of self-stabilizing algorithms in symmetric networks, including ring, line, mesh, and torus. First, we develop cutoffs that guarantee (1) closure in legitimate states, and (2) deadlock-freedom outside the legitimate states. We also develop a sufficient condition for convergence in self-stabilizing systems. Since some of our cutoffs grow with the size of the local state space of processes, scalability of the synthesis procedure is still a problem. We address this problem by introducing a novel SMT-based technique for *counterexample-guided synthesis* of self-stabilizing algorithms in symmetric networks. We have fully implemented our technique and successfully synthesized solutions to maximal matching, three coloring, and maximal independent set problems for ring and line topologies.

Keywords Parameterized synthesis · Self-stabilization · Formal methods

College of Engineering, University of Tehran
North Kargar st., Tehran, Iran
E-mail: mirzaienahal@ut.ac.ir

College of Engineering, University of Tehran
North Kargar st., Tehran, Iran
E-mail: f.faghih@ut.ac.ir

CISPA Helmholtz Center for Information Security
Saarland Informatics Campus, 66123 Saarbrücken, Germany
E-mail: jacobs@cispa.saarland

Department of Computer Science, Iowa State University,
207 Atanasoff Hall, Ames, IA 50011, USA
E-mail: borzoo@iastate.edu

1 Introduction

Program *synthesis* (often called the "*holy grail*" of computer science) is the problem of automated generation of a computer program from a formally specified set of properties. The program generated in this fashion is guaranteed to be *correct by construction*. Program synthesis is known to be computationally intractable and, thus, is usually used to deal with small but intricate components of a system. An example of such components is concurrent/distributed algorithms that may exhibit obscure corner cases, where reasoning about their correctness is not straightforward.

Dijkstra [13] introduced the notion of *self-stabilization* in distributed systems, where the system always converges to a good behavior even if it is arbitrarily initialized or is subject to transient faults. Proof of self-stabilization is, however, 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 [14]. This means that program synthesis can play a prime role in designing and reasoning about the correctness of self-stabilizing algorithms.

In previous work [20–24], we introduced a set of algorithms and tools for synthesizing self-stabilizing protocols. Our techniques take as input the network topology, timing model (asynchronous or synchronous), the good behavior of the protocol (either explicitly as a set of *legitimate states* or implicitly as a set of temporal logic formulas), type of symmetry, and type of stabilization (e.g., strong, weak, monotonic, ideal) and generate a set of first-order modulo theory (SMT) constraints. Then, an SMT-solver solves these constraints and, if satisfiable, produces a model that respects the input specification. Our tool Assess [22] has successfully synthesized complex algorithms such as Raymond's distributed mutual exclusion [46], Dijkstra's token ring [13] (for both three and four state machines), maximal matching [43], weak stabilizing token circulation in anonymous networks [12], and the three coloring problem [30]. Our algorithms are *complete* for a predetermined fixed number of processes; i.e., if they fail to find a solution to the synthesis problem, then there does not exist one. This completeness, however, comes at a big cost which is scalability. That is, for most instances, we could only synthesize solutions for up to 5 processes at best.

In this paper, our goal is to address scalability as well as the shortcoming that the previous work can synthesize only a fixed and predetermined number of processes. To this end, we focus on automated synthesis of self-stabilizing protocols in *symmetric* and *parameterized* networks, including ring, line, mesh, and torus, where an unbounded number of processes exhibit identical behavior. We make two main contributions. First, we show how to solve the *parameterized synthesis problem* based on the notion of *cutoffs* [18] that can guarantee properties of distributed systems of arbitrary size by considering only systems of up to a certain fixed size $c \in \mathbb{N}$, and augmented by a sound but incomplete abstraction-based synthesis approach for properties whose realizability are known to be undecidable [38,41]. In particular, we provide:

- *cutoffs* for the closure and deadlock-freedom properties, under the assumption that the set of legitimate states is defined by a conjunction of predicates on the local state of processes; we show that smaller cutoffs are possible under additional assumptions, and for rings, we additionally show that our cutoffs are *tight* under their respective assumptions;

– an *abstraction-based method* for the synthesis of the convergence property, which is known to be undecidable in general [38,41]; we show how a *sufficient condition* for convergence of the parameterized system can be efficiently checked on a finite system that over-approximates the behavior of systems of arbitrary size.

Note that the cutoffs and the sufficient condition hold for both synthesis and verification of self-stabilizing protocols.

A drawback of our cutoffs is that some of them are quadratic or exponential in the state space of a single process, so even with a tight cutoff, we need synthesis methods that scale to a large number of processes. Thus, as our second contribution, we propose a counterexample-guided synthesis technique that exploits our symmetry assumption. More specifically, our technique consists of four steps (see Fig. 1):

1. First, we *synthesize* a solution for a small network of i processes using existing techniques [20–24];
2. Next, we trivially generalize this solution to a larger network of $i+1$ processes;
3. Then, we *verify* this solution using a model checker, and
4. If verification succeeds, we return to step 2 and attempt a larger network. Otherwise, we obtain a counterexample that is added as a negative constraint to the synthesis algorithm, and we return to step 1 for another round of synthesis with limited search space.

Using this approach and our cutoff results, we successfully synthesized parameterized self-stabilizing protocols for well-known problems including three coloring, maximal matching, and maximal independent set for ring and line topologies in less than 10 minutes. To our knowledge, this is the first instance of such parameterized synthesis.

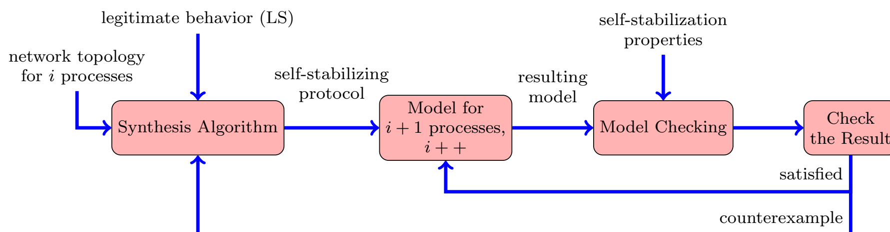

Fig. 1 SMT-based Counterexample-Guided Synthesis Technique

Organization The rest of the paper is organized as follows. Section 2 introduces the preliminary concepts. In Section 3, we present the formal statement of our synthesis problem. The parameterized correctness results are presented in Section 4, while our counterexample-guided synthesis approach is presented in Section 5. Experimental results and case studies are reported in Section 6. Related work is discussed in Section 7, and finally, we make concluding remarks and discuss future work in Section 8.

2 Preliminaries

In this section, we present the background concepts on self-stabilization and our computation model for distributed programs.

2.1 Distributed Programs

Most self-stabilizing algorithms are defined in the shared-memory model. Assume V to be the set of all variables in the system, where each variable $v \in V$ has a finite domain D_v. We define a *state* s as a valuation of each variable in V by a value in its domain. The set of all possible states is called the *state space*, and represented by S. A *transition* is defined as an ordered pair (s_0, s_1), where $s_0, s_1 \in S$. We denote the value of a variable v in state s by $v(s)$.

Definition 1 A *process* π is a tuple $\langle R_\pi, W_\pi, T_\pi \rangle$, where

- $R_\pi \subseteq V$ is the set of variables such that π can read their value and is called the *read-set* of π;
- $W_\pi \subseteq R_\pi$ is the set of variables such that π can change their value and is called the *write-set* of π, and
- T_π is the set of transitions of π, where for each transition $(s_0, s_1) \in T_\pi$ and each variable $v \in V$ with $v(s_0) \neq v(s_1)$, we have $v \in W_\pi$. □

The third condition imposes the constraint that a process can only change the value of a variable in its write-set, and the second condition states that this change cannot be blind. A process $\pi = \langle R_\pi, W_\pi, T_\pi \rangle$ is called *enabled* in state s_0 if there exists a state s_1 such that $(s_0, s_1) \in T_\pi$. The *local state space* of π is the set of all possible valuations of the variables that π can read, i.e., the Cartesian product of the domain of all variables in R_π:

$$S_\pi = \prod_{v \in R_\pi} D_v.$$

Definition 2 A *distributed program* is a tuple $\mathcal{D} = \langle P_\mathcal{D}, T_\mathcal{D} \rangle$, where

- $P_\mathcal{D}$ is a set of processes over a common set V of variables, such that:
  - for any two distinct processes $\pi_1, \pi_2 \in P_\mathcal{D}$, we have $W_{\pi_1} \cap W_{\pi_2} = \emptyset$;
  - for each process $\pi \in P_\mathcal{D}$ and each transition $(s_0, s_1) \in T_\pi$, the following *read restriction* holds:

$$\forall s_0', s_1' : \Big((\forall v \in R_\pi : (v(s_0) = v(s_0') \ \wedge \ v(s_1) = v(s_1'))) \ \wedge \\ (\forall v \notin R_\pi : v(s_0') = v(s_1')) \Big) \implies (s_0', s_1') \in T_\pi \qquad (1)$$

- $T_\mathcal{D}$ is the set of transitions and is the union of transitions of all processes:

$$T_\mathcal{D} = \bigcup_{\pi \in P_\mathcal{D}} T_\pi.$$

□

Intuitively, the read restriction in Definition 2 imposes the constraint that for each process π, each transition in T_π depends only on the variables in the read-set of π. Thus, each transition defines an equivalence class in $T_\mathcal{D}$, which we call a *group* of transitions. The key consequence of read restriction is that during synthesis, if a transition is included (respectively, excluded) in $T_\mathcal{D}$, then its corresponding group must also be included (respectively, excluded) in $T_\mathcal{D}$.

Example. We use the problem of distributed self-stabilizing *one-bit maximal matching* as a running example to describe the concepts throughout the paper. In a graph, a maximal matching is a maximal set of edges, in which no two edges share a common vertex. Consider a ring of 4 processes (see Fig. 2), and let $V = \{x_0, x_1, x_2, x_3\}$ be the set of variables, where each x_i, $i \in [0, 3]$, is a Boolean variable with domain $\{\mathtt{F}, \mathtt{T}\}$. Let $\mathcal{D} = \langle P_\mathcal{D}, T_\mathcal{D}\rangle$ be a distributed program, where $P_\mathcal{D} = \{\pi_0, \pi_1, \pi_2, \pi_3\}$. Each process π_i $(0 \le i \le 3)$ can write to variable x_i (i.e., $W_{\pi_i} = \{x_i\}$), and read the variables of its own and its neighbors ($R_{\pi_i} = \{x_i, x_{(i+1) \mod 4}, x_{(i-1) \mod 4}\}$). Notice that following Definition 2 and read/write restrictions of π_0, (arbitrary) transitions such as:

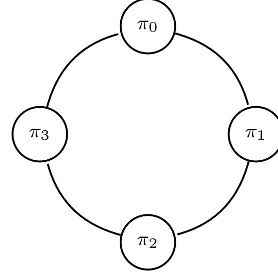

Fig. 2 One-bit maximal matching example.

$$t_1 = \Big([x_0 = \mathtt{F}, x_1 = \mathtt{F}, x_2 = \mathtt{F}, x_3 = \mathtt{F}], [x_0 = \mathtt{T}, x_1 = \mathtt{F}, x_2 = \mathtt{F}, x_3 = \mathtt{F}]\Big)$$
$$t_2 = \Big([x_0 = \mathtt{F}, x_1 = \mathtt{F}, x_2 = \mathtt{T}, x_3 = \mathtt{F}], [x_0 = \mathtt{T}, x_1 = \mathtt{F}, x_2 = \mathtt{T}, x_3 = \mathtt{F}]\Big)$$

are in the same group. The reason is that π_0 cannot read x_2, and if, for example, t_1 is included in the set of transitions, while t_2 is not, it implies that the execution in process π_0 depends on the value of x_2, which is not possible.

Definition 3 An *uninterpreted local function* for a process maps the *local state space* of a process to a domain D_{lf}. The interpretation of an uninterpreted local function for a process π is a function:

$$lf : S_\pi \to D_{lf}$$

where S_π is the local state space of π. □

In the sequel, we use "uninterpreted functions" to refer to uninterpreted local functions.

Example. To formulate the requirements in the one-bit maximal matching example, we assume each process π_i, where $i \in [0, 3]$, is associated with an uninterpreted local function, called $match_i$, with the domain $D_{match_i} = \{l, r, n\}$, where l, r, and n correspond to the cases where the process is matched to its left, right, and no neighbor (self-matched), respectively. The interpretation of $match_i$ is a function:

$$(match_i)_\mathrm{I} : \{\mathtt{F}, \mathtt{T}\} \times \{\mathtt{F}, \mathtt{T}\} \times \{\mathtt{F}, \mathtt{T}\} \to \{l, r, n\}$$

In other words, the value of $match_i$ depends on the value of the process and its neighbors' Boolean variables.

Definition 4 A *local predicate* X of a process maps the *local state space* of a process to a Boolean:

$$X : S_\pi \to \{\mathtt{F}, \mathtt{T}\}$$

Likewise, a *(global) state predicate* Y maps the *state space* of a distributed program to a Boolean:

$$X : S \to \{\mathtt{F}, \mathtt{T}\}$$

□

We say that a local predicate X *does not depend* on a variable $v \in R_\pi$ if the following holds:

$$\forall s_0, s_1 \in S_\pi : \ \big(\forall v' \in R_\pi \setminus \{v\} : v'(s_0) = v'(s_1)\big) \implies (X(s_0) \Leftrightarrow X(s_1)) \qquad (2)$$

We use these definitions to define *locally defined* legitimate states in Section 2.3, and later to define special cases where legitimate states do not depend on a subset of the read-set R_π.

2.1.1 Network Topology

A topology specifies the communication model of a distributed program.

Definition 5 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 $\{0, \ldots, |P_\mathcal{T}| - 1\} \to 2^V$ from a process index to its read-set, and
- $W_\mathcal{T}$ is a mapping $\{0, \ldots, |P_\mathcal{T}| - 1\} \to 2^V$ from a process index to its write-set, such that $W_\mathcal{T}(i) \subseteq R_\mathcal{T}(i)$, for all i $(0 \leq i \leq |P_\mathcal{T}| - 1)$. □

Example. The topology of our maximal matching problem is a tuple $\langle V, |P_\mathcal{T}|, R_\mathcal{T}, W_\mathcal{T} \rangle$:

- $V = \{x_0, x_1, x_2, x_3\}$, with domains $D_{x_0} = D_{x_1} = D_{x_2} = D_{x_3} = \{\mathtt{T}, \mathtt{F}\}$;
- $|P_\mathcal{T}| \ = 4$;
- $R_\mathcal{T}(0) = \{x_0, x_1, x_3\}$; $R_\mathcal{T}(1) = \{x_1, x_2, x_0\}$; $R_\mathcal{T}(2) = \{x_2, x_3, x_1\}$, $R_\mathcal{T}(3) = \{x_3, x_0, x_2\}$;
- $W_\mathcal{T}(0) = \{x_0\}$; $W_\mathcal{T}(1) = \{x_1\}$; $W_\mathcal{T}(2) = \{x_2\}$, and $W_\mathcal{T}(3) = \{x_3\}$.

Definition 6 A distributed program $\mathcal{D} = \langle P_\mathcal{D}, T_\mathcal{D} \rangle$ *has* topology $\mathcal{T} = \langle V, |P_\mathcal{T}|, R_\mathcal{T}, W_\mathcal{T} \rangle$ iff

- each process $\pi \in P_\mathcal{D}$ is defined over V;
- $|P_\mathcal{D}| = |P_\mathcal{T}|$;
- there is a mapping $g : \{0, \ldots, |P_\mathcal{T}| - 1\} \to P_\mathcal{D}$, such that

$$\forall i \in \{0, \ldots, |P_\mathcal{T}| - 1\} : (R_\mathcal{T}(i) = R_{g(i)}) \ \wedge \ (W_\mathcal{T}(i) = W_{g(i)}).$$

□

To simplify the usage of topology in this paper, we refer to different topologies based on the read-set of each process. For example, a bidirectional ring topology is the one, where each process can read the variables of its own, as well as the processes on its left and right, while in a unidirectional ring, a process can only read the variable of one of its neighbors, and not both.

2.2 Symmetric Networks

Roughly speaking, a topology is symmetric, if the read-set and write-set of any two distinct processes can be swapped (i.e., there is a bijection that maps read/write variables of a process to another).

Definition 7 A topology $\mathcal{T} = \langle V, |P_{\mathcal{T}}|, R_{\mathcal{T}}, W_{\mathcal{T}} \rangle$ is *symmetric*, iff for any distinct $i, j \in \{0 \dots |P_{\mathcal{T}}| - 1\}$, 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
- a bijection $g : W_{\mathcal{T}}(i) \rightarrow W_{\mathcal{T}}(j)$, such that $\forall v \in W_{\mathcal{T}}(i) : D_v = D_{g(v)}$. □

For example, we call a symmetric topology a (bi-directional) *ring* (of size $k = |P_{\mathcal{T}}|$) if for every $i \in \{0 \dots |P_{\mathcal{T}}| - 1\}$, we have:

$$R_{\mathcal{T}}(i) = W_{\mathcal{T}}(i - 1 \mod k) \cup W_{\mathcal{T}}(i) \cup W_{\mathcal{T}}(i + 1 \mod k).$$

Example. The topology of our one-bit maximal matching example is a symmetric ring of size 4 (Fig. 2). For any two $i, j \in [0, 3]$, function g is the mapping from x_i to a x_j, and function f maps $x_i \mapsto x_j$, $x_{(i+1) \mod 4} \mapsto x_{(j+1) \mod 4}$, and $x_{(i-1) \mod 4} \mapsto x_{(j-1) \mod 4}$.

Definition 8 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:

$$\begin{aligned} &\forall (s_0, s_1) \in T_\pi : \exists (s'_0, s'_1) \in T_{\pi'} : \\ &\Big(\forall v \in R_\pi : \big(v(s_0) = f(v)(s'_0)\big) \ \wedge \ \big(\forall v \in W_\pi : (v(s_1) = g(v)(s'_1)\big) \Big) \end{aligned} \tag{3}$$

where f and g are the functions defined in Definition 7. □

In other words, in a symmetric distributed program the read- and write-sets of all processes are identical up to renaming, and so are their transitions. Therefore, we also write $\mathcal{T}^\pi$ for a symmetric distributed program that has topology $\mathcal{T}$ and where all processes are identical up to renaming to π.

2.3 Self-Stabilization

Given a state predicate, called the set of *legitimate states* (denoted by LS), a *self-stabilizing* [13] program always recovers to a state in LS from any arbitrary state (e.g., due to bad initialization or occurrence of transient faults) in a finite number of steps, and stays in LS thereafter.

Definition 9 A *computation* of $\mathcal{D} = \langle P_{\mathcal{D}}, T_{\mathcal{D}} \rangle$ is an infinite sequence of states $\overline{s} = s_0 s_1 \cdots$, such that: (1) for all $i \geq 0$, we have $(s_i, s_{i+1}) \in T_{\mathcal{D}}$, and (2) if a computation reaches a state s_i, from where there is no state $\mathfrak{s} \neq s_i$, such that $(s_i, \mathfrak{s}) \in T_{\mathcal{D}}$, then the computation stutters at s_i indefinitely. Such a computation is called a *terminating computation.* □

Definition 10 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 $s_j \in LS$. [1]
2. *(Closure)* For any transition $(s_i, s_{i+1}) \in T_\mathcal{D}$, if $s_i \in LS$, then $s_{i+1} \in LS$. □

Definition 11 A set of legitimate states is *locally defined* if it can be defined by the set

$$\Big\{ s \mid \forall i \in \{0 \ldots |P_\mathcal{T}| - 1\} : LS_i(s) \Big\},$$

where LS_i is a local predicate of process π_i. □

Example. In our maximal matching example in a ring topology, each process can be matched to one of its two adjacent processes. To formulate this requirement, we assume each process π_i is associated with a local uninterpreted function, called $match_i$, with the domain $D_{match_i} = \{l, r, n\}$. LS can be locally defined with following assignment: (see Fig 3)

$$\begin{aligned}
&LS_i = \big\{ s \mid \\
&(match_{i-1}(\Pi_{R_{\pi_{i-1}}}(s)) = r \wedge match_i(\Pi_{R_{\pi_i}}(s)) = l \wedge match_{i+1}(\Pi_{R_{\pi_{i+1}}}(s)) = n) \vee \\
&(match_{i-1}(\Pi_{R_{\pi_{i-1}}}(s)) = n \wedge match_i(\Pi_{R_{\pi_i}}(s)) = r \wedge match_{i+1}(\Pi_{R_{\pi_{i+1}}}(s)) = l) \vee \\
&(match_{i-1}(\Pi_{R_{\pi_{i-1}}}(s)) = l \wedge match_i(\Pi_{R_{\pi_i}}(s)) = n \wedge match_{i+1}(\Pi_{R_{\pi_{i+1}}}(s)) = r) \vee \\
&(match_{i-1}(\Pi_{R_{\pi_{i-1}}}(s)) = l \wedge match_i(\Pi_{R_{\pi_i}}(s)) = r \wedge match_{i+1}(\Pi_{R_{\pi_{i+1}}}(s)) = l) \vee \\
&(match_{i-1}(\Pi_{R_{\pi_{i-1}}}(s)) = r \wedge match_i(\Pi_{R_{\pi_i}}(s)) = l \wedge match_{i+1}(\Pi_{R_{\pi_{i+1}}}(s)) = r) \big\}
\end{aligned}$$

The system is in a legitimate state if and only if all processes are in a local legitimate state. For example, in a ring of size three with the set of processes $P = \{\pi_0, \pi_1, \pi_2\}$, the set of legitimate states can be formulated as the following:

$$\{s \mid LS_0(s) \wedge LS_1(s) \wedge LS_2(s)\}$$

Note how uninterpreted functions can be used to easily express LS. Without $match_i$, the user has to explicitly specify the cases where a process is matched to its left, right or itself, using the Boolean variables of its own and its adjacent processes (its read-set).

3 Problem Statement

Our goal is to propose an automated method for *parameterized* synthesis of self-stabilizing protocols in symmetric networks. That is, we consider a problem where the size of the topology is a parameter, and we want to automatically synthesize the set of transitions and the interpretation of the uninterpreted function of each process, such that the resulting distributed program is self-stabilizing for any value of the parameter.

[1] Note that a computation may be terminating in a state in LS.

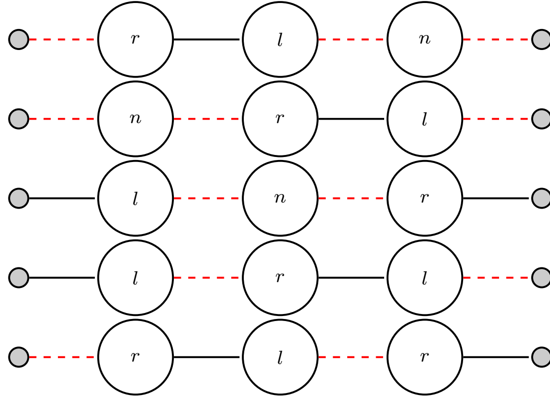

Fig. 3 Definition of local legitimate states for maximal matching.

Formally, 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 read-sets and write-sets bijections, as required in Definition 7 also exist between process indices from different elements of the sequence. 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 *parameterized synthesis problem* takes as input:

– a parameterized topology, and
– a set of locally defined legitimate states LS,

and generates as output:

– a process π, such that for every element $\mathcal{T}_n$ of the topology, the program $\mathcal{D}_n = \mathcal{T}_n^\pi$ is self-stabilizing to LS.

Definition 12 For a given parameterized topology and a property under consideration, 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 $i \leq c$. □

That is, to prove that a property holds for programs of arbitrary size, it is enough to prove it for programs up to the cutoff size. Note that this implies that if c is a cutoff for a given parameterized topology and a property, then any $c' > c$ is also a cutoff. We say that a cutoff c is *tight* if no smaller cutoff (for the given parameterized topology and property) exists, i.e., if $c - 1$ is not a cutoff. We will also consider cutoffs that are restricted to a certain class of processes, or only hold if LS is defined in a certain way. Furthermore, note that cutoffs can be used for both parameterized verification and synthesis.

In Section 4, we will present cutoffs for two properties: (i) closure, and (ii) the absence of deadlocks outside of LS. Our idea for proving the cutoffs is to show that if a system instance with a size greater than the cutoff violates the property under

study, then the property is also violated for some $n \leq c$. Then, we prove tightness by giving an example demonstrating that $c-1$ is not a cutoff. Moreover, we will introduce an abstraction-based method that can be combined with the cutoffs to solve the parameterized synthesis problem.

4 Parameterized Synthesis of Self-Stabilization

In this section, we show how to reduce reasoning about parameterized programs to reasoning about a finite number of finite programs. To prove self-stabilization, we need to prove that the algorithm has the two properties of closure and convergence from Definition 10. We split the latter into two properties: (1) the absence of deadlocks outside of LS, and (2) the absence of cycles outside of LS. In the following, we provide cutoffs for closure and deadlock-freedom outside of LS, as well as a sound abstraction to prove the absence of cycles outside of LS. Finally, we provide our main theorem that combines these results into a method for parameterized synthesis of self-stabilizing algorithms.

4.1 Parameterized Synthesis of Self-Stabilization in Symmetric Rings

In this section, we present our cutoff results for parameterized synthesis of self-stabilization in symmetric rings.

4.1.1 Cutoffs for Closure

Assume that the write-set of each process has l valuations. In other words, if process π_i has $W_{\mathcal{T}}(i) = \{v_1, \ldots, v_k\}$, then $l = |D_{v_1}| \times \cdots \times |D_{v_k}|$. WLOG, we assume that $W_{\mathcal{T}}(i) = \{v_i\}$ for all i, with possible values $v_i(s) \in \{0, \ldots, l-1\}$. In the following, addition and subtraction are always modulo the size of the respective domain, e.g., addition of process indices is modulo n if we consider a program with n processes. We first state our cutoff results, and then give examples that witness the tightness of our cutoffs.

Lemma 1 *For symmetric distributed algorithms on a ring topology, the following are cutoffs for the closure property:*

- $c = l^2$, *if* LS *is locally defined;*
- $c = l+1$, *if* LS *is locally defined and* LS_i *does not depend on* $W_{\mathcal{T}}(i-1)$, *and*
- $c = 3$, *if* LS *is locally defined and* LS_i *depends on neither* $W_{\mathcal{T}}(i-1)$ *nor* $W_{\mathcal{T}}(i+1)$.

All of the cutoffs are tight under their respective assumptions.

Proof Note that to prove that a given c is a cutoff, we only need to prove the "if" direction of Definition 12 — the other direction is trivial. Thus, for each case of the lemma, it is sufficient to prove the following boundedness property for counterexamples to closure: for any process and any LS that satisfies the given assumption, if we have a violation of the closure property in a ring of arbitrary size n, then we also have a violation of the closure property in a ring of size smaller or equal to the given c. Since this is trivial if $n \leq c$, we will show it only for $n > c$.

For the first item, we first show the boundedness property for $c = l^2 + 1$, and in a second step show that it also holds for $c = l^2$. Consider a ring of size $n > l^2 + 1$, and assume there exist states $s \in LS$ and $s' \notin LS$, such that there is a transition from s to s'. WLOG, assume that $(s, s') \in T_{\pi_0}$, i.e., this is a transition of process π_0. Now, consider the $n-1$ pairs of consecutive processes (π_i, π_{i+1}), where $i \in [0, n-2]$. Note that π_0 appears in just one tuple. Based on the pigeonhole principle, at least two of these pairs of processes have the same valuation of their write-sets in s, since we have $n - 1 \geq l^2 + 1$ tuples and only l^2 possible pairs of the write-sets of a tuple. Assume that (π_i, π_{i+1}) and (π_j, π_{j+1}) have the same valuation of their write-sets. Then, consider a smaller ring composed of $\pi_0, \ldots, \pi_i, \pi_{j+1}, \ldots, \pi_{n-1}$ that is in a state s_1 with $v_p(s_1) = v_p(s)$ for all $p \in \{0, \ldots, i, j+1, \ldots, n-1\}$ (see Fig. 4). Note that for every p, this construction ensures that the read-set of π_p in s_1 is the same as in s. Therefore, we have $s_1 \in LS$ and π_0 can still take a transition that leads to a state outside of LS. If $|\{0, \ldots, i, j+1, \ldots, n-1\}| > c$, we can repeat the removal of processes by the same argument until we arrive at a ring of size at most $c = l^2 + 1$.

To see that the boundedness property also holds with $c = l^2$, note that in the construction above we excluded the pair (π_{n-1}, π_0) from the sequence in which we look for the same valuation of the write-sets. The reason is that the construction above does not work if we find that (π_0, π_1) and (π_{n-1}, π_0) have the same valuation. However, if that is the case then we must have $v_0(s) = v_1(s) = v_{n-1}(s)$, and we can reduce the example to a ring that only consists of 3 processes with this valuation of their write-set. Thus, we can assume that (π_0, π_1) and (π_{n-1}, π_0) have different valuations, and in that case the pigeonhole principle works in any ring of size $n > l^2$, since we can now include the pair (π_{n-1}, π_0). This proves the first item.

For the second item (i.e., $c = l + 1$), a similar construction can be used, where we only consider single valuations of write-sets instead of pairs. Therefore, the l^2 in the cutoff value can be replaced by l. However, the reduction to $c = l$ is not possible: it may happen that the valuation of π_0 is the same as one of its neighbors. In that case we cannot reduce to a smaller ring, since that may change the valuation of a neighbor of π_0, which might make the transition to a state outside of LS impossible. Therefore, the cutoff is not l, but $l + 1$.

Finally, for the third item (i.e., $c = 3$), we only need to ensure that π_0 can still take the transition to a state outside of LS, i.e., the valuations of its neighbors remain the same, and that every remaining process keeps the valuation of its write-set. This is enough since LS_i only depends on its own valuation. Thus, $c = 3$ is sufficient.

Tightness for the three cases is witnessed by Examples 1, 2 and 3, respectively. □

Example 1 Consider programs composed of an arbitrary number of processes, where the transition relation T_{π_i} of process π_i is such that $(s, s') \in T_{\pi_i}$ if $v_{i-1}(s) = v_{i+1}(s)$ and $v_i(s') = v_{i-1}(s)$, and otherwise no transition is possible.

Let $(d_0, d_1), (d_1, d_2), \ldots, (d_{l^2-1}, d_0)$ be a sequence of pairs from $\{0, \ldots, l-1\} \times \{0, \ldots, l-1\}$, such that each pair in $\{0, \ldots, l-1\} \times \{0, \ldots, l-1\}$ appears exactly once in the sequence.[2]

[2] One can see that it is possible to find such a sequence of length l^2 by an induction on l.

Let LS be locally defined in the following way:

$$LS_i(s) \Leftrightarrow \bigvee_{j \in \{0,\ldots,l^2-1\}} (v_{i-1}(s) = d_{j-1} \wedge v_i(s) = d_j \wedge v_{i+1}(s) = d_{j+1})$$

Note that for the program to be in LS, there must be at least l^2 processes. Also note that by definition of the sequence of pairs above, in a state $s \in LS$ there must be at least one process with $v_{i-1}(s) = v_{i+1}(s)$. This process can take a transition to a state s' with $v_i(s') = v_{i-1}(s)$, and therefore $v_{i-1}(s') = v_i(s') = v_{i+1}(s')$. Finally, note that $s' \notin LS_i$, since a repetition of pairs was excluded in our definition of the sequence of pairs above.

Since a violation of closure is possible in a ring of size l^2, but not in any smaller ring, $l^2 - 1$ cannot be a cutoff for the first case of Lemma 1.

Example 2 Consider programs where the transition relation T_{π_i} is such that:

- no transition is possible from any state s with $v_{i+1}(s) = v_i(s)+1$ and $v_{i-1}(s) = v_i(s) - 1$,
- no transition is possible if $v_{i-1}(s) = v_i(s) = v_{i+1}(s)$, and
- in all other cases, the process can increment the value v_i. In particular, there is a transition from any state s with $v_{i-1}(s) = l - 1$ and $v_i(s) = v_{i+1}(s) = 0$ to a state s' with $v_i(s') = 1$.

Let LS be locally defined in the following way, where LS_i does not depend on $W_{\mathcal{T}}(i-1)$:

$$LS_i(s) \Leftrightarrow v_{i+1}(s) = v_i(s) + 1 \vee v_{i+1}(s) = v_i(s) = 0$$

Note that for the program to be in LS, either $v_i(s) = 0$ must hold for all i, or there must be at least l processes. Now, note that for a program composed of c processes, a violation of closure is impossible if $c \leq l$ (since for a state in LS either all values are 0, or the values are exactly $0, \ldots, l-1$ in this order; in both cases no transition is possible). However, a violation of closure is possible if $c = l + 1$: let s be such that the values $v_i(s)$ are $0, 0, 1, \ldots, l-1$, in this order. Then s is in LS, and the first process has a transition to a state s' with $s' \notin LS$.

Since a violation of closure is possible in a ring of size $l + 1$, but not in any smaller ring, l cannot be a cutoff for the second case of Lemma 1.

Example 3 Consider programs composed of processes with possible valuations $v_i(s) \in \{0, 1, 2\}$ of their write-set, and a transition relation such that $(s, s') \in T_{\pi_i}$ if $v_{i-1}(s) = 0 \wedge v_{i+1}(s) = 1 \wedge v_i(s) \leq 1$ and $v_i(s') = 2$, and no other transitions are possible.

Let LS be locally defined in the following way, where LS_i depends on neither $W_{\mathcal{T}}(i-1)$ nor on $W_{\mathcal{T}}(i-1)$:

$$LS_i(s) \Leftrightarrow v_i(s) \in \{0, 1\}$$

Since a violation of closure is possible in a ring of size 3, but not in any smaller ring, 2 cannot be a cutoff for the third case of Lemma 1.

4.1.2 Cutoffs for Deadlock Detection

Again, we first state our cutoff results and then provide examples that witness the tightness of our cutoffs.

Lemma 2 *For self-stabilizing algorithms on a ring topology, the following are cutoffs for the detection of deadlocks outside of LS:*

- $c = l^2$*, if transitions of a process* π_i *can depend on its whole read-set (i.e.,* $W_{\mathcal{T}}(i)$*,* $W_{\mathcal{T}}(i+1)$*, and* $W_{\mathcal{T}}(i-1)$*);*
- $c = l + 1$*, if transitions of a process* π_i *do not depend on* $W_{\mathcal{T}}(i-1)$ *(i.e., the ring is uni-directional), and*
- $c = 3$*, if transitions of a process* π_i *depend on neither* $W_{\mathcal{T}}(i-1)$ *nor* $W_{\mathcal{T}}(i+1)$ *(i.e., processes are completely independent).*

All of the cutoffs are tight under their respective assumptions.

Proof For the first case, the proof is based on the same idea as the first case of Lemma 1. That is, given a ring of size $n > l^2 + 1$ where the given program has a deadlock state $s \notin LS$, we can construct a smaller ring with a deadlock state $s_1 \notin LS$, where the definition of LS is based on the same sequence of valuation-pairs as in Lemma 1.

The second and third cases are again similar to the second and third cases of Lemma 1, except that we need to consider restricted transition relations instead of restricted definitions of LS.

Tightness of the cutoffs is witnessed by Examples 4, 5 and 6. □

Example 4 Let $(d_0, d_1), (d_1, d_2), \ldots, (d_{l^2-1}, d_0)$ be a sequence of pairs of values, such that each pair in $\{0, \ldots, l-1\} \times \{0, \ldots, l-1\}$ appears exactly once in the sequence, like in Example 1.

Consider programs composed of processes with a transition relation T_{π_i} such that in state s no transition is possible for process π_i if $(v_{i-1}(s), v_i(s)), (v_i(s), v_{i+1}(s))$ are consecutive pairs in the sequence, and otherwise v_i can be incremented. Thus, a deadlock is only possible if we have at least l^2 processes.

We can ensure that the state s in which we deadlock is not in LS by letting

$$LS_i(s) \Leftrightarrow v_{i-1}(s) = v_i(s) = v_{i+1}(s)$$

Since a deadlock outside of LS is possible in a ring of size l^2, but not in any smaller ring, $l^2 - 1$ cannot be a cutoff for the first case of Lemma 2.

Example 5 Consider programs composed of processes with a transition relation T_{π_i} such that in state s no transition is possible for process π_i if $v_i(s) + 1 = v_{i+1}(s)$ or $v_i(s) = v_{i+1}(s) = 0$, and otherwise v_i can be incremented. Note that this transition relation does not depend on $W_{\mathcal{T}}(i-1)$. Then, a deadlock is possible if all $v_i(s) = 0$ for all processes π_i, or if we have at least l processes.

Let LS be defined by

$$LS_i(s) \Leftrightarrow v_{i-1}(s) = v_i(s) = v_{i+1}(s) \vee v_i(s) + 1 = v_{i+1}(s)$$

Note that a deadlock outside of LS is not possible in rings of size up to l, but it is possible in a ring of size $l + 1$: if s has valuations $0, 0, 1, ..., l-1$, in this order, then s is deadlocked and $s \notin LS_0$. Thus, l cannot be a cutoff for the second case of Lemma 2.

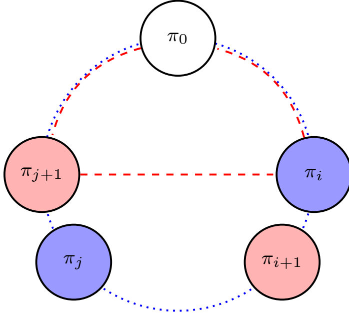

Fig. 4 Reducing a ring (blue dotted) to a smaller ring (red dashed).

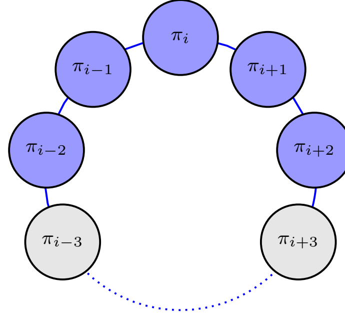

Fig. 5 Blue processes act based on the synthesized algorithm and grey processes act randomly.

Example 6 Consider a process π_i with possible valuations $v_i(s) \in \{0,1,2,3\}$ of its write-set, and a transition relation such that $(s,s') \in T_{\pi_i}$ if $v_i(s) = 3$ and $v_i(s') \in \{0,1,2\}$, and no other transitions are possible. Note that this transition relation depends neither on $W_{\mathcal{T}}(i-1)$ nor on $W_{\mathcal{T}}(i+1)$.

Furthermore, let LS_i be specified in terms of its read-set $R_{\mathcal{T}}(i)$:

$$LS_i(s) \Leftrightarrow (v_{i-1}(s) \neq v_i(s) \wedge v_i(s) \neq v_{i+1}(s) \wedge v_{i-1}(s) \neq v_{i+1}(s))$$

Then $LS_i(s)$ always holds if we are in a program with 1 or 2 processes, i.e., there cannot be deadlocks outside of LS. However, there can be deadlocks outside of LS in a program with 3 processes. Therefore, 2 cannot be a cutoff for the third case of Lemma 2.

4.1.3 Process Abstraction for Convergence

As mentioned before, to prove self-stabilization of a parameterized program, we need to prove closure and convergence. Closure can be proved based on Lemma 1, and Lemma 2 shows how to deal with deadlocks outside of LS. Thus, the missing part is a method to prove that there are no cycles outside of LS that prevents a computation to eventually reach LS. In contrast to the two previous problems, we now consider infinite behaviors of the system. Since parameterized verification and synthesis of symmetric self-stabilization in rings is known to be undecidable [38,41], we cannot obtain cutoffs for this property. Therefore, we resort to proving the absence of cycles based on a sound abstraction of the system behavior.

The basic idea is the following: we check whether there is a loop that starts and ends in the same *local* state outside LS_i, for an arbitrary process. We define the following convergence property:

$$\forall s \mid \neg LS_i(s) \Rightarrow \neg\Box\Diamond\, s$$

, where s is a local state of π_i (i.e., the valuation of its read-set), $\Diamond$ is the 'eventually' operator, and $\Box$ is the 'always' operator in temporal logic. That is, given a local state s outside LS_i, s does not become true infinitely often. [3]

We attempt to prove the property in a ring of size 3, where one process (π_i) behaves according to the synthesized protocol. The other two processes (π_{i-1} and π_{i+1}) have the same write-set, but can execute arbitrary transitions. The idea is that these two processes over-approximate the possible behaviors of all other processes. If we can prove the property above in this abstraction of the system, then this implies that no loops are possible in a concrete system in a ring of size ≥ 1. Otherwise, we add more processes to the left and right of π_i that act according to the protocol, and check the property again [4]. In other words, the abstraction approach is flexible and its precision can be refined by increasing the number of processes that behave according to the protocol. For the problems we considered in our experiments (see Sect. 6), the property is satisfied by refining the abstraction to at most $5+2$ processes (5 processes act according to the protocol and 2 processes at the two ends execute random actions.). Fig. 5 shows the abstraction with $5+2$ processes.

4.1.4 Parameterized Self-Stabilization for Rings

Based on Lemmas 1 and 2, and the approach in Section 4.1.3, we obtain our main result for rings.

Theorem 1 *Let $\mathcal{T}_1, \mathcal{T}_2, \ldots$ be a parameterized ring topology, π a process, and LS be locally defined by LS_i. Let c_1 and c_2 be cutoffs for closure and deadlock-freedom wrt. LS, respectively. If (1) closure holds in rings of size up to c_1, (2) deadlocks outside of LS are impossible in rings of size up to c_2, and (3) the absence of cycles can be proven in an abstract system as above, then every instance of the parameterized program is self-stabilizing to LS.* □

4.2 Parameterized Synthesis of Self-Stabilization in Symmetric Lines

In this section, we present our cutoff results for parameterized synthesis of self-stabilization in lines. Note that in a line, we don't have a completely symmetric topology, as the two processes at the two ends of the line do not have the same read-set (number of neighbors) as the other processes. Therefore, by symmetry, we refer to a protocol that is similar for all processes except for the processes at the two ends of the line.

4.2.1 Cutoffs for Closure

Assume that the write-set of each process has l valuations. In other words, if process π_i has $W_{\mathcal{T}}(i) = \{v_1, \ldots, v_k\}$, then $l = |D_{v_1}| \times \cdots \times |D_{v_k}|$. Again, we assume that $W_{\mathcal{T}}(i) = \{v_i\}$ for all i, with possible values $v_i(s) \in \{0, \ldots, l-1\}$.

[3] Note that this property can be easily transformed to a property without the universal quantifier by introducing new variables that can take arbitrary values at the initialization and then keep their values.

[4] This approach is inspired by similar abstraction-based methods for the verification and synthesis of systems with many processes [3, 10].

Lemma 3 *For self-stabilizing algorithms on a line topology, the following are cutoffs for the closure property:*

- $c = 2l^2 + 2$, *if LS is locally defined,*
- $c = 2l + 2$, *if LS is locally defined and* LS_i *does not depend on* $W_{\mathcal{T}}(i-1)$, *and*
- $c = 3$, *if LS is locally defined and* LS_i *depends on neither* $W_{\mathcal{T}}(i-1)$ *nor* $W_{\mathcal{T}}(i+1)$.

Proof The proof has the same structure as the proof of Lemma 1. Again, it is sufficient to show the boundedness property for counterexamples to closure.

We provide the proof idea for the first case. Consider a line of size $n > 2l^2 + 2$, and assume there exist states $s \in LS$ and $s' \notin LS$ such that there is a transition from s to s'. WLOG, assume that $(s, s') \in T_{\pi_k}$, i.e., this is a transition of π_k. Now, at least one of the lines created by π_k ($[\pi_0 \cdots \pi_k]$ and $[\pi_k \cdots \pi_{n-1}]$) has at least $l^2 + 1$ pairs of processes. WLOG, assume that $\mid \{(\pi_0, \pi_1), \cdots, (\pi_{k-1}, \pi_k)\} \mid \geq l^2 + 1$. Based on the pigeonhole principle, at least two of these pairs of processes have the same valuations of their write-sets in s, since we have only l^2 possible valuations of the write-sets of a tuple. Assume that $(v_i(s), v_{i+1}(s)) = (v_j(s), v_{j+1}(s))$. Then, consider the smaller line composed of $\pi_0, \ldots, \pi_i, \pi_{j+1}, \ldots, \pi_k, \ldots, \pi_{n-1}$ that is in a state s_1 with $v_p(s_1) = v_p(s)$ for all $p \in \{0, \ldots, i, j+1, \ldots, n-1\}$ (see Fig. 6). Note that this construction does not change the valuations of the read-sets of the remaining processes, and hence we have $s_1 \in LS$, and π_k can still take the transition that leads to a state outside of LS.

For the second case (i.e., $c = 2l + 2$), we use a similar idea. Consider a line of size $n > 2l + 2$. Assume there exist states $s \in LS$ and $s' \notin LS$ such that there is a transition from s to s'. WLOG, assume that $(s, s') \in T_{\pi_k}$, i.e., this is a transition of π_k. Now, at least one of the lines created by π_k ($[\pi_0 \cdots \pi_{k-1}]$ and $[\pi_{k+1} \cdots \pi_{n-1}]$) has a size greater than or equal to $l + 1$. WLOG, assume that $\mid [\pi_0 \cdots \pi_{k-1}] \mid \geq l + 1$. Based on the pigeonhole principle, at least two of these processes have the same valuation of their write-sets in s. Assume that $v_i(s) = v_j(s)$. Then consider a smaller line composed of $\pi_0, \ldots, \pi_i, \pi_{j+1}, \ldots, \pi_k, \ldots, \pi_{n-1}$ that is in a state s_1 with $v_p(s_1) = v_p(s)$ for all $p \in \{0, \ldots, i, j+1, \ldots, n-1\}$ (Fig. 7). Note that this construction does not change the valuations of the read-sets of the remaining processes, and hence we have $s_1 \in LS$, and π_k can still take a transition that leads to a state outside of LS.

Finally, in the third case (i.e., $c = 3$), we only need to have the two processes at the two ends and one middle process, as the transition of each process only depends on its own local variables. □

We conjecture that tightness of these cutoffs (possibly up to a small constant) can be observed by an extension of the tightness argument for rings, where instead of one sequence for the ring we define two sequences for the line: one from the left end of the line to a distinguished element in the middle, and one from the middle to the right end, resulting in a lower bound in the order of $2l^2$.

4.2.2 Cutoffs for Deadlock Detection

Lemma 4 *For self-stabilizing algorithms on a line topology, the following are cutoffs for the detection of deadlocks outside of LS:*

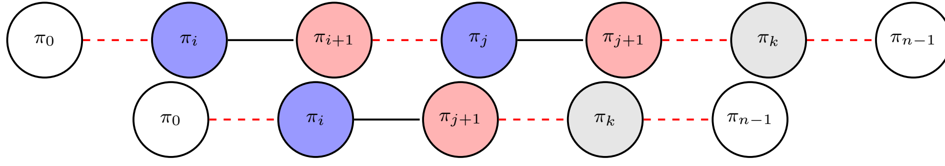

Fig. 6 Reducing a line to a smaller line for the first case

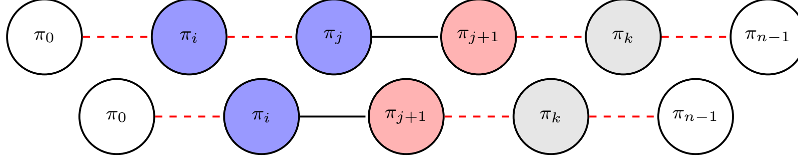

Fig. 7 Reducing a line to a smaller line for the second case

– $c = 2l^2 + 2$, *if transitions of a process* π_i *can depend on its whole read-set (i.e.,* $W_{\mathcal{T}}(i)$, $W_{\mathcal{T}}(i+1)$, *and* $W_{\mathcal{T}}(i-1)$*),*
– $c = 2l + 2$, *if transitions do not depend on* $W_{\mathcal{T}}(i-1)$, *and*
– $c = 3$, *if transitions of a process* π_i *depend on neither* $W_{\mathcal{T}}(i-1)$ *nor* $W_{\mathcal{T}}(i-1)$ *(i.e., processes are completely independent).*

Proof Again, we have to prove the boundedness property for counterexamples.

For the first case, consider a line of size $n > 2l^2 + 2$, and assume that with the given program we have a deadlock state $s \notin LS$. That is, $s \notin LS_i$ for at least one of the processes (assume π_k), and there is no state s' and no process π_i such that $(s, s') \in T_{\pi_i}$. Note that at least one of the two lines created by π_k ($[\pi_0 \cdots \pi_k]$ and $[\pi_k \cdots \pi_{n-1}]$) has at least $l^2 + 1$ processes. Hence, there are at least two pairs of consecutive processes in the same local states. Assume that we have $(v_i(s), v_{i+1}(s)) = (v_j(s), v_{j+1}(s))$. We consider the smaller line $[\pi_0, \ldots, \pi_i, \pi_{j+1}, \ldots, \pi n - 1]$ in a state s_1 with $v_p(s_1) = v_p(s)$ for all $p \in \{0, \ldots, i, j+1, \ldots, n-1\}$. Then $s_1 \notin LS$, and the program is deadlocked in s_1. As before, we can repeat the construction if necessary.

For the second case (i.e., $c = 2l + 2$), consider a line of size $n > 2l + 2$, and assume that with the given program we have a deadlock state $s \notin LS$. Assume $s \notin LS_k$. Note that at least one of the two lines created by π_k ($[\pi_0 \cdots \pi_k]$ and $[\pi_k \cdots \pi_{N-1}]$) has at least $l + 1$ processes. Hence, there are at least two processes with the same valuation of their write-set. Assume that $v_i(s) = v_j(s)$. We consider the smaller line $[\pi_0, \ldots, \pi_i, \pi_{j+1}, \ldots, \pi n - 1]$ in a state s_1 with $v_p(s_1) = v_p(s)$ for all $p \in \{0, \ldots, i, j+1, \ldots, n-1\}$. Then $s_1 \notin LS$, and the program is deadlocked in s_1. As before, we can repeat the construction if necessary.

For the last case (i.e., $c = 3$), since transitions of the process only depend on its own local state, it suffices to check a line with two end processes, and one middle process. □

4.2.3 Process Abstraction for Convergence

Similar to the proof in Sect. 4.1.3, we check whether there is a loop that starts and ends in the same *local* state for an arbitrary process.

$$\forall s \mid \neg LS_i(s) \Rightarrow \neg \Box \Diamond s$$

If we can show that this is not possible, then certainly, no global loop is possible outside LS (see Fig. 8). To show that the protocol satisfies convergence, we should prove that every single process in the line does not visit a local state outside LS infinitely often. We divide the processes in a line into five categories. Consider a line of N processes with the set of processes $\{\pi_0, \pi_1, ..., \pi_{N-2}, \pi_{N-1}\}$, where π_0 and π_{N-1} are left and right process respectively that have different transitions and definitions of local LS than the other processes. The five categories of processes in this topology are as follows:

- **Left process ($\{\pi_0\}$):** To assure that π_0 does not violate the convergence property, we abstract the system behavior for this process. We start with $\{\pi_0, \pi_{rr}\}$ set of processes where π_{rr} is a **r**andom **r**ight process with an arbitrary set of transitions. Then in this topology, we check the convergence property for π_0. If it is not satisfied, we refine the abstraction by adding one more process, and change the topology to $\{\pi_0, \pi_1, \pi_{rr}\}$, where π_1 acts according to the topology of middle processes. We check the convergence property again for π_0 in this new topology. We continue till the property is satisfied. Assume the property is satisfied for π_0 in the $\{\pi_0, \pi_1, ..., \pi_{n_l}, \pi_{rr}\}$ topology. This means that we need at least n_l processes at the right side of π_0 to guarantee that π_0 does not violate the convergence property.

- **Right process ($\{\pi_{N-1}\}$) :** Verification of the convergence property for this process is similar to the left process, but on the other side. Assume $\{\pi_{rl}, \pi_{n_r}, ..., \pi_{N-1}\}$ is the smallest topology, where the property is satisfied for π_{N-1}, and π_{rl} is a random left process with an arbitrary set of transitions.

- **Middle processes** Verification of the convergence property for middle processes is similar to the process abstraction in rings. In other words, we consider one process π_m that acts according to the protocol of the symmetric processes, with two random processes at the two ends, and check the property. If it is not satisfied, we add more processes (that act according to the protocol of the symmetric processes) to the left and right of π_m, until the property is satisfied for π_m. Assume that i processes are added to the left and j processes are added to right of π_m.

- **Left-side processes** As mentioned in the previous item, assume that i processes are added to the left of a middle process, so that the convergence property is satisfied for π_m. The convergence property should be checked for all these processes, when they are in interaction with the left process ($\{\pi_0\}$). If π_{ls} is a left-side process ($1 \le ls \le i$), then we start with $\{\pi_0, \pi_1, ..., \pi_{ls}, \pi_{rr}\}$ topology and check the convergence property for π_{ls}. If the property is not satisfied, we refine the abstraction by adding processes to the right side of π_{ls}, until the property is satisfied for π_{ls}. For each process in this category, we find the smallest required size for the property to be satisfied, and report the maximum of them.

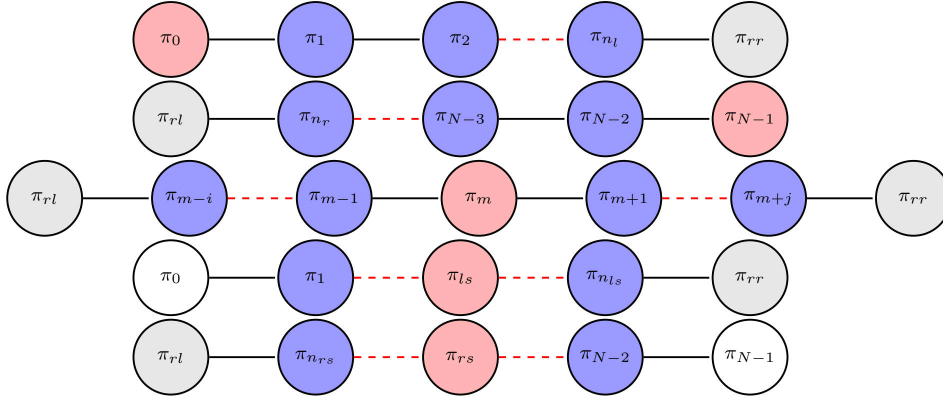

Fig. 8 Process abstraction in line for each category. We check convergence property on the red processe while gray processes act randomly.

– **Right-side processes** The procedure is similar to the left-side processes, but on the other side.

After calculating all minimum sizes, their maximum (let's call it M) is reported as the smallest topology size, where it is guaranteed that convergence is satisfied for every topology with a size $\geq M$.

4.2.4 Parameterized Self-Stabilization for Lines

Based on Lemmas 3 and 4, and the approach in Section 4.2.3, we obtain our main result.

Theorem 2 *Let $\mathcal{T}_1, \mathcal{T}_2, \ldots$ be a parameterized line 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 (1) closure holds in lines of size up to c_1, (2) deadlocks outside of LS are impossible in lines of size up to c_2, and (3) the absence of cycles can be proven in lines of up to size c_3 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.* □

4.3 Parameterized Synthesis of Self-Stabilization in Symmetric Mesh

In this section, we present our cutoff results for parameterized synthesis of self-stabilization in the mesh topology. We study a common type of mesh, namely, the square grid topology, and consider $|c|$ and $|r|$ as the number of columns and rows respectively. Note that we assume the number of rows is constant, and the number of columns can change in a parameterized mesh topology.

4.3.1 Cutoffs for Closure

Assume that the write-set of each process has l valuations. In other words, if process $\pi_{i,j}$ (ith process in jth column) has $W_{\mathcal{T}}(i,j) = \{v_1, \ldots, v_k\}$, then $l = |D_{v_1}| \times \cdots \times |D_{v_k}|$. Again, we assume that $W_{\mathcal{T}}(i,j) = \{v_{(i,j)}\}$ for all (i,j), with possible values $v_{(i,j)}(s) \in \{0, \ldots, l-1\}$.

Lemma 5 *For self-stabilizing algorithms on a mesh topology, the following are cutoffs for the closure property:*

- $c = |r|(2l^{2|r|} + 2)$*, if LS is locally defined,*
- $c = |r|(2l^{|r|} + 2)$*, if LS is locally defined and* $LS_{(i,j)}$ *does not depend on* $W_{\mathcal{T}}(i,j-1)$*, and*
- $c = 3|r|$*, if LS is locally defined and* $LS_{(i,j)}$ *depends on neither* $W_{\mathcal{T}}(i,j-1)$ *nor* $W_{\mathcal{T}}(i,j+1)$*.*

Proof As before, we prove a boundedness property for counterexamples.

Similar to our proof approach for lines, consider a mesh with size of $n > |r|(2l^{2|r|} + 2)$. Since in each column there are exactly $|r|$ processes, we have $k > (2l^{2|r|} + 2)$ columns. Assume there exists a transition from $s \in LS$ to $s' \notin LS$, with $(s,s') \in T_{\pi_{(i,j)}}$. Now consider two sequences of pairs of columns $(c_0, c_1), ..., (c_{j-1}, c_j)$ and $(c_j, c_{j+1}), \ldots , (c_{|c|-2}, c_{|c|-1})$. One of these two sequences has at least $l^{2|r|}+1$ pairs and only $l^{2|r|}$ possible valuations of the write-sets of a pair. So based on pigeonhole principle there exist at least 2 pairs of columns, say (c_t, c_{t+1}) and (c_u, c_{u+1}), such that have for each $i \in \{0, ..., |r|-1\}$, $(v_{(i,t)}(s), v_{(i+1,t)}(s)) = (v_{(i,u)}(s), v_{(i+1,u)}(s))$. So if we consider a smaller mesh with columns of $c_0, ..., c_t$,c_{u+1}, ..., $c_{|c|-1}$ in a state s_1 with local valuations as in state s, then there exists a transition $(s_1, s_1') \in T_{\pi_{(i,j)}}$ with $s_1 \in LS$ to some state $s_1' \notin LS$. Again, this construction can be repeated if necessary.

The proof for the second item works in a similar way. Note that instead of sets of tuples of columns, we have two sequences of columns $c_0, ..., c_{j-1}$ and $c_{j+1}, ..., c_{|c|-1}$, and one of them has at least $l^{|r|} + 1$ columns.

For the third item, we only need to have the two columns at the two ends and one middle column, as the transition of each process only depends on its own local variables. ◻

4.3.2 Cutoffs for Deadlock Freedom

Lemma 6 *For self-stabilizing algorithms on a mesh topology, the following are cutoffs for the detection of deadlocks outside of LS:*

- $c = |r|(2l^{2|r|} + 2)$*, if transitions of a process* π_i *can depend on its whole read-set (i.e.,*$W_{\mathcal{T}}(i,j)$*,* $W_{\mathcal{T}}(i-1,j)$*,* $W_{\mathcal{T}}(i+1,j)$*,* $W_{\mathcal{T}}(i,j-1)$ *and* $W_{\mathcal{T}}(i,j+1)$*),*
- $c = |r|(2l^{|r|}+2)$*, if transitions of a process* π_i *do not depend on* $W_{\mathcal{T}}(i,j-1)$*, and*
- $c = 3|r|$*, if transitions of a process* π_i *depend on neither* $W_{\mathcal{T}}(i,j-1)$ *nor* $W_{\mathcal{T}}(i,j+1)$*.*

Proof The proof idea is analogous to the proof of Lemma 5. □

Note that a line protocol is also a mesh, where $|r| = 1$, and our results for mesh and line are consistent, considering this fact.

4.3.3 Process Abstraction for Convergence

For convergence, again like our approach for lines we develop a sound abstraction of system behavior, but use lines with $|n|$ processes that act randomly. We use two dummy lines and add sufficient lines between them until the following property is satisfied for the three consecutive processes in the middle of mesh. Similar to the other topologies, we verify the following formula:

$$\forall s \mid \neg LS_i(s) \Rightarrow \neg \Box \Diamond s$$

4.3.4 Parameterized Self-Stabilization for Mesh

Based on Lemmas 5 and 6, and the approach in Section 4.3.3, we obtain our main result.

Theorem 3 *Let $\mathcal{T}_1, \mathcal{T}_2, \ldots$ be a parameterized mesh 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 (1) closure holds in meshes of size up to c_1, (2) deadlocks outside of LS are impossible in meshes of size up to c_2, and (3) the absence of cycles can be proven in meshes of up to size c_3 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.* □

4.4 Parameterized Synthesis of Self-Stabilization in Symmetric Torus

We study 2-D torus protocols with k rings and m processes in each ring with expansion by increasing the number of rings (k), not the number of processes in each ring.

4.4.1 Cutoffs for Closure

Assume that the write-set of each process has l valuations. In other words, if process $\pi_{i,j}$ (ith process in jth ring) has $W_\mathcal{T}(i,j) = \{v_1, \ldots, v_k\}$, then $l = |D_{v_1}| \times \cdots \times |D_{v_k}|$. Again, we assume that $W_\mathcal{T}(i,j) = \{v_i\}$ for all (i,j), with possible values $v_{(i,j)}(s) \in \{0, \ldots, l-1\}$.

Lemma 7 *For self-stabilizing algorithms on a torus topology, the following are cutoffs for the closure property:*

- $c = |m|(l^{2|m|})$, *if LS is locally defined,*
- $c = |m|(l^{|m|}+1)$, *if LS is locally defined and LS_i does not depend on $W_\mathcal{T}(i, j-1)$.*

– *$c = 3|m|$, if LS is locally defined and LS_i depends on neither $W_{\mathcal{T}}(i, j-1)$ nor $W_{\mathcal{T}}(i, j+1)$.*

Proof Like our proof approach for rings, we have a torus protocol with size of $n > |m|(l^{2|m|})$. Since in each ring there are exactly $|m|$ processes, we have $k > l^{2|m|}$ rings. Assume there exists a transition from $s \in LS$ to $s' \notin LS$, with $(s, s') \in T_{\pi_{(0,0)}}$. Now consider the sequence of pairs of rings $(r_0, r_1), ..., (r_{k-2}, r_{k-1}), (r_{k-1}, r_0)$. We have at least $l^{2|m|+1}$ pairs and only $l^{2|m|}$ possible valuations of the write-sets of a pair. So based on the pigeonhole principle there exist at least 2 pairs of rings, say (r_t, r_{t+1}) and (r_u, r_{u+1}), such that for each $i \in \{0, ..., |r|\}$, $(v_{(i,t)}(s), v_{(i,t+1)}(s)) = (v_(i,u)(s), v_{(i,u+1)}(s))$. So in a smaller torus with rings $r_0, ..., r_s, r_{t+1}, ..., r_{k-1}$ with local valuations as in state s there exist a transition from $s \in LS$ to $s' \notin LS$ taken by $\pi_{(0,0)}$.

The second and third part work as expected. □

4.4.2 Cutoffs for Deadlock Freedom

Lemma 8 *For self-stabilizing algorithms on a torus topology, the following are cutoffs for the detection of deadlocks outside of LS:*

– *$c = |m|(l^{2|m|})$, if transitions of a process π_i can depend on its whole read-set (i.e., $W_{\mathcal{T}}(i, j)$, $W_{\mathcal{T}}(i-1, j)$, $W_{\mathcal{T}}(i+1, j)$, $W_{\mathcal{T}}(i, j-1)$ and $W_{\mathcal{T}}(i, j+1)$),*

– *$c = |m|(l^{|m|}+1)$, if transitions of a process π_i do not depend on $W_{\mathcal{T}}(i, j-1)$, and*

– *$c = 3|m|$, if transitions of a process π_i depend on neither $W_{\mathcal{T}}(i, j-1)$ nor $W_{\mathcal{T}}(i, j+1)$.*

Proof The proof idea is analogous to the proof of Lemma 7. □

Note that a ring protocol is also a special torus with $|m| = 1$, and hence, our cutoff results for ring is consistent with the cutoffs we propose for torus.

4.4.3 Process Abstraction for Convergence

For convergence, we develop a sound abstraction of the system behavior based on our approach for rings, where single processes are now replaced by fixed-size rings. A *dummy ring* is a ring with $|m|$ processes that act randomly. We use two dummy rings and add sufficient rings between them like a cylinder until the following property is satisfied for the three consecutive processes in the middle of cylinder:

$$\forall s \mid \neg LS_i(s) \Rightarrow \neg \Box \Diamond s$$

4.4.4 Parameterized Self-Stabilization for Torus

Based on Lemmas 7 and 8, and the approach in Section 4.4.3, we obtain our main result.

Theorem 4 *Let $\mathcal{T}_1, \mathcal{T}_2, \dots$ be a parameterized torus 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 (1) closure holds in torus protocols of size up to c_1, (2) deadlocks outside of LS are impossible in torus protocols of size up to c_2, and (3) the absence of cycles can be proven in torus protocols of up to size c_3 and in an abstract system as above, then every instance of the parameterized program $\mathcal{D}_1 = \mathcal{T}_1^\pi, \mathcal{D}_2 = \mathcal{T}_2^\pi, \dots$ is self-stabilizing to LS.* □

5 SMT-based Counterexample-Guided Synthesis

5.1 General Idea

In [20, 21, 23], we introduced SMT-based methods to solve the synthesis problem for self-stabilizing systems. In a nutshell, our techniques generate a set of SMT constraints from the input synthesis instance. Each SMT constraint corresponds to one of the requirements of the desired protocol. They are all given to an SMT solver, and the generated model represents a self-stabilizing protocol. In order to scale up these technique to synthesize solutions up to the cutoff point efficiently, in this section we propose a method that synthesizes solutions in small topologies and tries to generalize them to bigger topologies. Let us first present a naïve idea, where we first synthesize a protocol for a small topology and then simply use this solution for larger topologies with the hope that since the protocol is symmetric, a small solution works in a larger network as well. We now show that this approach is not conceivable even for very simple protocols.

Example. When applying our latest algorithm [23] to the one-bit maximal matching example, the first synthesized solution for 4 processes is the following transition relation encoded by guarded commands for each process π_i:

$$\begin{array}{lrcl} \pi_i: & (x_i = \mathtt{F}) \wedge (x_{(i+1)} = \mathtt{F}) \wedge (x_{(i-1)} = \mathtt{F}) & \rightarrow & x_i := \mathtt{T} \\ & (x_i = \mathtt{T}) \wedge (x_{(i+1)} = \mathtt{T}) & \rightarrow & x_i := \mathtt{F} \end{array}$$

and the following interpretation for uninterpreted function $match_i$:

$$\begin{array}{lrcl} match_i: & (x_i = \mathtt{T}) \wedge \big((x_{(i+1)} = \mathtt{T}) \vee (x_{(i-1)} = \mathtt{F})\big) & \mapsto & l \\ & (x_{(i+1)} = \mathtt{T}) \wedge (x_{(i-1)} = \mathtt{F}) & \mapsto & l \\ & (x_i = \mathtt{T}) \wedge (x_{(i+1)} = \mathtt{F}) \wedge (x_{(i-1)} = \mathtt{T}) & \mapsto & r \\ & (x_i = \mathtt{F}) \wedge (x_{(i+1)} = \mathtt{T}) \wedge (x_{(i-1)} = \mathtt{T}) & \mapsto & r \\ & (x_i = \mathtt{F}) \wedge (x_{(i+1)} = \mathtt{F}) & \mapsto & n \end{array}$$

Now, if we trivially use the synthesized protocol on a topology with 5 processes, the resulting protocol is incorrect. In particular, the following is a counterexample (i.e., a finite computation that violates the specification) in terms of predicate *match*:

$$\begin{array}{l} \big([match_0 = n, match_1 = n, match_2 = n, match_3 = n, match_4 = n], \\ [match_0 = l, match_1 = n, match_2 = n, match_3 = n, match_4 = l], \\ [match_0 = l, match_1 = r, match_2 = l, match_3 = n, match_4 = l]\big) \end{array}$$

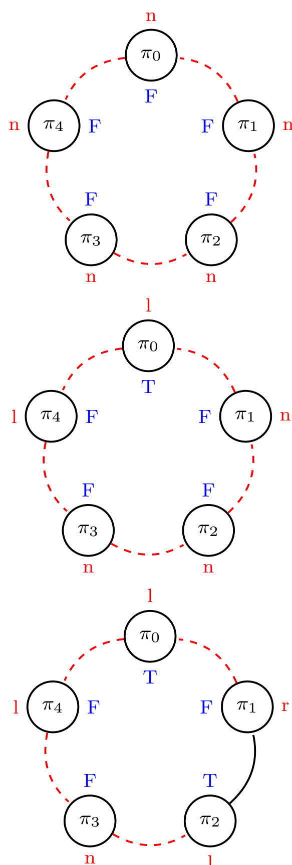

Fig. 9 Example of three counterexamples for one-bit maximal matching. Blue labels are valuation and red labels indicate the match interpretation.

This computation violates convergence, as it reaches a deadlock state in $\neg LS$. The computation is depicted in Fig 9, where the black solid line shows a matching between the two end processes. As you can see in the figure, the last state is not a maximal matching. This example shows that a synthesized symmetric solution cannot be trivially extended to larger topologies.

5.2 The Counterexample-Guided Synthesis Algorithm

In order to limit the search space of SMT-solvers for a solution, we incorporate a synthesis-verification loop guided by counterexamples. Our approach consists of the following steps:

1. Given a topology with i processes and a set of legitimate states, we use our existing approach [20, 21, 23] to formulate the synthesis problem as an SMT instance.
2. We use an SMT solver to find a solution for the SMT instance, as a transition relation and an interpretation for each uninterpreted function. Note that due to symmetry, the transition relations and the interpretation functions are identical for all processes.
3. Next, we generalize the solution for a topology with $i + 1$ processes and verify this solution using a model checker.
4. If the result of verification is positive, then we go back to step 3 to check the properties for a topology with $i + 2$ processes. Otherwise, we transform the generated counterexample into an SMT constraint and add it to the initial SMT instance (step 1) and return to step 2.

We do not include the details of our SMT encoding of the synthesis problem, since it has been described in length in previous publications [20, 21, 23] and we use it as a black box. We now analyze the nature of counterexamples. In the context of closure and convergence, a model checker may generate a counterexample of the form $\overline{s} = s_0 s_1 \cdots s_n$. Observe that $\overline{s}$ is one of the following three types of counterexamples:

- If closure is violated, then $\overline{s} = (s_0, s_1)$, where $s_0 \in LS$ and $s_1 \notin LS$.
- If convergence is violated by $\overline{s} = s_0 s_1 \cdots s_n$, where for all $i \in [0, n]$, we have $s_i \notin LS$ and either
  - $s_0 = s_n$; i.e., a loop exists outside the set of legitimate states, or
  - there does not exist a state $\mathfrak{s}$, where $(s_n, \mathfrak{s})$ is a valid transition; i.e., s_n is a *deadlock* state outside the set of legitimate states.

For example, the counterexample presented in Section 5.1 is of the third type.

Dealing with the first type of counterexamples is pretty straightforward: we only add a constraint to the SMT instance that disallows transition (s_0, s_1) in the transition relation. To address deadlocks, we need to add a constraint to the SMT instance to enforce a change in the resulting synthesized model, so that s_n is not a deadlock state. To this end, we propose two sets of heuristics to change either the transition relation or the interpretation of uninterpreted functions in Section 5.3. Dealing with loops is a bit more complicated. For example, one can remove a transition from the loop to break it, but the choice of transition may involve a combinatorial enumeration to find the right transition. Therefore, reasoning about loop counterexamples could be a difficult problem, which we postpone to future work. Interestingly, all of our case studies in Section 6 do not involve loop counterexamples.

5.3 Heuristics Considering Transition Relations

The simplest method to resolve a deadlock is to formulate a constraint imposing the existence of an outgoing transition from s_n. Since in this paper, our focus is on asynchronous systems, a transition is the execution of one of the processes. We propose two strategies for selecting a process to have an outgoing transition from a deadlock state.

Progress Heuristic. In this approach, we add a constraint stating that at least one of the processes should have an outgoing transition from s_n. More formally, assume that the current topology includes i processes, where the read-set of each process has r variables, with domains $D_0, \ldots, D_{r-1}$, and the write-set of each process includes w variables, with domains $D'_0, \cdots, D'_{w-1}$. Note that since the goal is to synthesize a symmetric program, all processes execute similarly according to the function T_π:

$$T_\pi : \Big(\prod_{j \in [0, r-1]} D_j\Big) \to \Big(\prod_{j \in [0, w-1]} D'_j\Big)$$

and function f is of type:

$$f : [0, i) \to \Big(S \to \big(\prod_{j \in [0, r-1]} D_j\big)\Big)$$

Then, the constraint to be added to the SMT instance can be written as:

$$\exists val_0 \in D'_0, \cdots . \exists val_{w-1} \in D'_{w-1} : \bigvee_{k \in [0, i)} \Big(\big(f(k)(s_n), [val_0, val_1, \cdots, val_{w-1}]\big) \in T_\pi\Big)$$

Note that the function f takes the process id and the global state, and projects out the local state of the process.

Example. Consider the counterexample mentioned in Section 5.1. Each process can read three Boolean variables and write to one Boolean variable and, hence, T_π is defined as follows:

$$T_\pi \ : \ \{\mathtt{F}, \mathtt{T}\} \times \{\mathtt{F}, \mathtt{T}\} \times \{\mathtt{F}, \mathtt{T}\} \to \{\mathtt{F}, \mathtt{T}\}$$

Note that for each process π_j, $f(j)$ returns $[x_{(j-1)}, x_j, x_{(j+1)}]$. In the counterexample we presented in the previous example, the last state where the deadlock happens is:

$$[x_0 = \mathtt{T}, x_1 = \mathtt{F}, x_2 = \mathtt{T}, x_3 = \mathtt{F}, x_4 = \mathtt{F}]$$

Thus, we add the following constraint to the SMT instance:

$$\exists val \in \{\mathtt{F}, \mathtt{T}\} \ : \ \Big(([\mathtt{F}, \mathtt{T}, \mathtt{F}], val) \in T_\pi \ \vee \ ([\mathtt{T}, \mathtt{F}, \mathtt{T}], val) \in T_\pi \ \vee \ ([\mathtt{F}, \mathtt{T}, \mathtt{F}], val) \in T_\pi \ \vee \ ([\mathtt{T}, \mathtt{F}, \mathtt{F}], val) \in T_\pi \ \vee \ ([\mathtt{F}, \mathtt{F}, \mathtt{T}], val) \in T_\pi\Big)$$

In the above constraint, the jth clause imposes a constraint on T_π to have an outgoing transition considering the local state of the jth process. (Note that the first and third clauses are the same, and we just put them for clarity.)

Local LS Heuristic. As mentioned in Section 2, we focus on sets LS that can be locally defined, i.e., the set of legitimate states can be described as a conjunction over local legitimate states of processes. In this case, a deadlock can be resolved by imposing a constraint to have an outgoing transition for at least one of the processes that are not in their local legitimate states.

Example. For the counterexample of one-bit maximal matching with 4 processes, the local set of legitimate states is the following:

$$\begin{aligned}
&(match_{(i-1)} = l \wedge match_i = n \wedge match_{(i+1)} = r) \vee \\
&(match_{(i-1)} = r \wedge match_i = l \wedge match_{(i+1)} = n) \vee \\
&(match_{(i-1)} = r \wedge match_i = l \wedge match_{(i+1)} = r) \vee \\
&(match_{(i-1)} = l \wedge match_i = r \wedge match_{(i+1)} = l) \vee \\
&(match_{(i-1)} = n \wedge match_i = r \wedge match_{(i+1)} = l)
\end{aligned}$$

and the deadlock state is:

$$[x_0 = \mathtt{T}, x_1 = \mathtt{F}, x_2 = \mathtt{T}, x_3 = \mathtt{F}, x_4 = \mathtt{F}]$$

For checking the local state of each process, we should first note the values of uninterpreted functions $match_i$ in this state:

$$[match_0 = l, match_1 = r, match_2 = l, match_3 = n, match_4 = l]$$

Processes π_0, π_3, and π_4 are not in a local legitimate state, and hence, the added constraint to the original SMT model will be as follows:

$$\exists val \in \{\mathtt{F}, \mathtt{T}\} \;:\; \Big(([\mathtt{F}, \mathtt{T}, \mathtt{F}], val) \in T_\pi \;\vee\; ([\mathtt{T}, \mathtt{F}, \mathtt{F}], val) \in T_\pi \;\vee\; ([\mathtt{F}, \mathtt{F}, \mathtt{T}], val) \in T_\pi\Big)$$

Note that although this method seems more efficient than the progress approach in terms of having shorter constraints, it has the drawback of missing some solutions that the previous approach can find. More specifically, for a process being in a legitimate local state in a deadlock state, it may be the case that taking a transition by this process leads to a state, from which its neighbors can take other transitions that finally leads to a legitimate state.

5.4 Heuristics Considering Uninterpreted Functions

Our second class of heuristics focus on uninterpreted functions. That is, we impose a constraint to change the interpretation function of at least one uninterpreted function in the deadlock state. Similar to the heuristics introduced for transition relations, we introduce two approaches for selecting at least one process to change the interpretation of its uninterpreted function. Because of the similarity to the previous heuristics, we skip the details of this heuristic.

6 Case Studies and Experimental Results

We used the model finder Alloy [32] and model checker NuSMV [19] to implement our counterexample-guided synthesis approach. Our experimental platform is an 2.9 GHz Intel Core i7 processor, with 16 GB of RAM.

We present case studies for well-known problems in ring and line topologies. Since the cutoffs for mesh and torus are much bigger [5], we believe that parameterized synthesis in such networks goes beyond any existing synthesis approach, and practical solutions to these problems are left for future work.

[5] For example, for a topology with $l = 3$ in a torus with $|m| = 3$, the cutoff for the first case will be 2187.

6.1 Parameterized Synthesis in Rings

We used our proposed algorithm to synthesize parameterized self-stabilizing algorithms for four problems that are well known in the distributed computing community. Our synthesis results are reported in Table 1. Looking at Table 1, we notice that in some case studies the progress heuristic has better efficiency, while in some others, the local LS finds the solution faster. Since the SMT solvers have many heuristics for finding the solutions, we cannot exactly say why this happens. But one reason we can mention for the progress heuristic having better efficiency compared to the local LS is due to the fact that the constraints added in the local LS heuristic are too restrictive, and hence, Alloy needs to search more in order to find a solution. In the cases that the local LS heuristic works faster, it is probably due to the smaller constraints added in this case.

6.1.1 Three Coloring

We consider the *three coloring problem* [30] on a ring, where each process π_i is associated with a variable c_i with domain $\{0, 1, 2\}$. Each value of the variable c_i represents a distinct color. A process can read and write its own variable. It can also read the variables of its neighbors. LS includes all states, where each process has a color different from its both neighbors. Thus, for a ring of 4 processes, LS is defined by the following predicate:

$$c_0(s) \neq c_1(s) \wedge c_1(s) \neq c_2(s) \ \wedge \ c_2(s) \neq c_3(s) \wedge c_3(s) \neq c_0(s)$$

Observe that the closure/deadlock-freedom cutoff point for this case study is $3^2 = 9$ and, hence, we need to synthesize a solution for 9 processes. The synthesis time reported in Table 1 is a bit smaller in the case of local LS heuristic, which is probably due to the smaller constraints added in this case. The resulting protocols for the two heuristics are different. The following is one synthesized protocol for the case of local LS, which is obtained by 5 iterations of synthesis-verification in our CEGIS approach:

$$\begin{array}{llll}
\pi_i : & (c_i = 2) \wedge (c_{(i+1)} = 2) \wedge (c_{(i-1)} \neq 0) & \to & c_i := 0 \\
& (c_i \neq 0) \wedge (c_{(i+1)} = 1) \wedge (c_{(i-1)} = 2) & \to & c_i := 0 \\
& (c_i = 1) \wedge (c_{(i+1)} = 1) \wedge (c_{(i-1)} \neq 2) & \to & c_i := 2 \\
& (c_i = 0) \wedge (c_{(i+1)} \neq 2) \wedge (c_{(i-1)} = 0) & \to & c_i := 2 \\
& (c_i \neq 1) \wedge (c_{(i+1)} = 2) \wedge (c_{(i-1)} = 0) & \to & c_i := 1
\end{array}$$

6.1.2 One-Bit Maximal Matching

This case study is the running example in this paper with cutoff point of $2^2 = 4$ processes. Note that using the heuristics considering transition relations, we could not synthesize a protocol for this problem (Alloy reports unsatisfiablity after adding the counterexample constraints). The interesting point about this case study is that the progress heuristic has better efficiency compared to the local LS. The reason may be due to the fact that the constraints added in the local LS heuristic

Problem	cutoff #	Heuristic	Synthesis Time	Model Checking Time
Three Coloring	9	Local *LS*	6m 17sec	56 msec
Three Coloring	9	Progress	9m 5sec	51 msec
One-Bit MM*	4	Local *LS*	1m 43sec	47 msec
One-Bit MM	4	Progress	1m 30sec	47 msec
Maximal Matching	9	Local *LS*	7m 59sec	63 msec
Maximal Matching	9	Progress	4m 57sec	68 msec
Maximal Independent Set*	4	Local *LS*	10sec	61 msec
Maximal Independent Set	4	Progress	10sec	61 msec

Table 1 Results for parameterized synthesis in ring topologies. (* In these cases, synthesized protocols for both progress and local LS heuristics are the same.)

are too restrictive, and hence, Alloy needs to search more in order to find a solution. The synthesized solutions using both heuristics are the same for this case study, which are obtained by 6 iterations of synthesis-verification in our CEGIS approach. The synthesized transition relation is the following:

$$\begin{array}{lrcl} \pi_i: & (x_i = \mathtt{F}) \wedge (x_{(i+1)} = \mathtt{F}) \wedge (x_{(i-1)} = \mathtt{F}) & \rightarrow & x_i := \mathtt{T} \\ & (x_i = \mathtt{T}) \wedge (x_{(i+1)} = \mathtt{T}) & \rightarrow & x_i := \mathtt{F} \end{array}$$

and the interpretation function for $match_i$ is the following:

$$\begin{array}{lrcl} match_i: & (x_i = \mathtt{T}) \wedge (x_{(i+1)} = \mathtt{T}) \wedge (x_{(i-1)} = \mathtt{T}) & \mapsto & l \\ & (x_{(i+1)} = \mathtt{F}) \wedge (x_{(i-1)} = \mathtt{F}) & \mapsto & l \\ & (x_i = \mathtt{T}) \wedge (x_{(i+1)} = \mathtt{F}) \wedge (x_{(i-1)} = \mathtt{T}) & \mapsto & r \\ & (x_i = \mathtt{F}) \wedge (x_{(i+1)} = \mathtt{T}) & \mapsto & r \\ & (x_i = \mathtt{T}) \wedge (x_{(i+1)} = \mathtt{T}) \wedge (x_{(i-1)} = \mathtt{F}) & \mapsto & n \\ & (x_i = \mathtt{F}) \wedge (x_{(i+1)} = \mathtt{F}) \wedge (x_{(i-1)} = \mathtt{T}) & \mapsto & n \end{array}$$

6.1.3 Maximal Matching

In this case study, we used the same problem as in Section 6.1.2, but instead of using one Boolean variable for each process, we use a variable with three values $\{l, r, n\}$ and, hence, we do not need the uninterpreted functions anymore. The resulting protocols for the two heuristics are different. As an example, the synthesized protocol for the case of local LS is the following, which is obtained by 3 iterations of synthesis-verification in our CEGIS approach:

$$\begin{array}{lrcl} \pi_i: & (x_i = n) \wedge (x_{(i+1)} = n) \wedge (x_{(i-1)} = n) & \rightarrow & x_i := r \\ & (x_i \neq r) \wedge (x_{(i+1)} \neq r) \wedge (x_{(i-1)} = l) & \rightarrow & x_i := r \\ & (x_i \neq n) \wedge (x_{(i+1)} = r) \wedge (x_{(i-1)} = l) & \rightarrow & x_i := n \\ & (x_i = n) \wedge (x_{(i-1)} = r) & \rightarrow & x_i := l \\ & (x_i = r) \wedge (x_{(i+1)} = r) \wedge (x_{(i-1)} \neq l) & \rightarrow & x_i := l \end{array}$$

6.1.4 Maximal Independent Set

An *independent set* in a graph is a subset of vertices in which no pair of vertices are adjacent. To synthesize a protocol that finds a maximal independent set, we consider a set of processes connected in a ring topology, where each process has a Boolean variable, the value of which shows whether or not it is included in the maximal independent set. The set of legitimate states include those states, where the processes whose variables have the true value form a maximal independent set. As an example, if c_i is the variable of the process π_i, then the set of legitimate states for the case of four processes is formulated by the following predicate:

$$\begin{array}{l}(c_0(s) = \texttt{T} \wedge c_1(s) = \texttt{F} \wedge c_2(s) = \texttt{T} \wedge c_3(s) = \texttt{F}) \\ \vee \\ (c_0(s) = \texttt{F} \wedge c_1(s) = \texttt{T} \wedge c_2(s) = \texttt{F} \wedge c_3(s) = \texttt{T})\end{array}$$

The resulting protocol is the following:

$$\begin{array}{lrcl}\pi_i : & (x_i = \texttt{F}) \wedge (x_{(i+1)} = \texttt{F}) \wedge (x_{(i-1)} = \texttt{F}) & \rightarrow & x_i := \texttt{T} \\ & (x_i = \texttt{T}) \wedge (x_{(i+1)} = \texttt{T}) & \rightarrow & x_i := \texttt{F}\end{array}$$

6.2 Parameterized Synthesis in Lines

Our case studies for synthesis in line topologies are similar to those we presented for rings. Our synthesis results are reported in Table 2.

6.2.1 Three Coloring

The resulting protocols for the two heuristics are the same for the three coloring problem. Note that in the case of line topologies, we synthesize three protocols; two for the two processes at the ends of the line, and one for all the middle processes. The synthesized protocol for the middle processes is the following:

$$\begin{array}{llcl}\pi_i : & (c_i = 2) \wedge (c_{(i+1)} = 2) \wedge (c_{(i-1)} \neq 0) & \rightarrow & c_i := 0 \\ & (c_i \neq 0) \wedge (c_{(i+1)} = 1) \wedge (c_{(i-1)} = 2) & \rightarrow & c_i := 0 \\ & (c_i = 2) \wedge (c_{(i+1)} = 2) \wedge (c_{(i-1)} = 0) & \rightarrow & c_i := 1 \\ & (c_i = 0) \wedge (c_{(i+1)} = 2) \wedge (c_{(i-1)} = 0) & \rightarrow & c_i := 1 \\ & (c_i = 1) \wedge (c_{(i+1)} = 1) \wedge (c_{(i-1)} \neq 2) & \rightarrow & c_i := 2 \\ & (c_i = 0) \wedge (c_{(i+1)} \neq 2) \wedge (c_{(i-1)} = 0) & \rightarrow & c_i := 2\end{array}$$

The synthesized protocol for the left end of the line is:

$$\begin{array}{llcl}\pi_0 : & (c_0 = 1) \wedge (c_1 = 1) & \rightarrow & c_0 := 2 \\ & (c_0 = 0) \wedge (c_1 = 0) & \rightarrow & c_0 := 2 \\ & (c_0 = 2) \wedge (c_1 = 2) & \rightarrow & c_0 := 1\end{array}$$

Problem	cutoff #	Heuristic	Synthesis Time	Overall Model Checking Time*
Three Coloring	20	Local *LS*	2m 9sec	130 msec
Three Coloring	20	Progress	1m 44sec	121 msec
Maximal Independent Set	10	Local *LS*	4sec	151 msec
Maximal Independent Set	10	Progress	4sec	151 msec

Table 2 Results for parameterized synthesis on line.(Overall model checking time is the sum of the times needed for left process, left-side processes, middle processes, right-side processes, and right process.)

Finally, the synthesized protocol for the right end of the line is:

$$\begin{array}{lllll}\pi_n : & (c_n = 0) \wedge (c_{(n-1)} = 0) & \rightarrow & c_n := 1 \\ & (c_n = 2) \wedge (c_{(n-1)} = 2) & \rightarrow & c_0 := 1 \\ & (c_n = 1) \wedge (c_{(n-1)} = 1) & \rightarrow & c_0 := 2\end{array}$$

6.2.2 Maximal Independent Set

The synthesized protocols for this problem are the same for both heuristics. The synthesized protocol for the middle processes is:

$$\begin{array}{llll}\pi_i : & (c_i = \mathtt{F}) \wedge (c_{(i+1)} = \mathtt{F}) \wedge (c_{(i-1)} = \mathtt{F}) & \rightarrow & c_i := \mathtt{T} \\ & (c_i = \mathtt{T}) \wedge (c_{(i+1)} = \mathtt{T}) & \rightarrow & c_i := \mathtt{F}\end{array}$$

The synthesized protocol for the left end of the line is:

$$\begin{array}{llll}\pi_0 : & (c_0 = \mathtt{T}) \wedge (c_{(1)} = \mathtt{T}) & \rightarrow & c_i := \mathtt{F} \\ & (c_0 = \mathtt{F}) \wedge (c_{(1)} = \mathtt{F}) & \rightarrow & c_i := \mathtt{T}\end{array}$$

Finally, the synthesized protocol for the right end of the line is:

$$\pi_n : \quad (c_n = \mathtt{T}) \wedge (c_{(n-1)} = \mathtt{T}) \quad \rightarrow \quad c_n := \mathtt{F}$$

7 Related Work

Synthesis of distributed systems: There are different techniques for automated synthesis of distributed systems, from which we can mention the genetic-based approach [49], and bounded synthesis [28]. There are also techniques for automated completion of distributed protocols based on program sketching in the literature, including [1, 2, 29].

Synthesis of self-stabilization: In the context of self-stabilization, in [16], a lightweight method is proposed to generate initial designs of self-stabilizing protocols, and then a heuristic is introduced to add convergence to a non-stabilizing

protocol for a specific number of processes. Other techniques for synthesizing self-stabilization for fixed-size topologies is introduced in [39, 40]. Klinkhammer and Ebnenasir show that adding strong convergence is NP-complete in the size of the state space, which itself is exponential in the number of variables of the protocol [37]. Faghih and Bonakdarpour introduced an SMT-based synthesis technique for automatically synthesizing self-stabilizing systems [20, 21] that is complete and not based on existing non-stabilizing algorithms. An extension of this work [23] allows us to symbolically specify the legitimate states as a set of requirements, and supports the synthesis of ideal-stabilizing systems. While these approaches are promising and can automatically synthesize a number of well-known self-stabilizing systems, they suffer from the problem of scalability, as the complexity of the problem increases exponentially in the number of processes. For example, all results reported by Faghih and Bonakdarpour [20, 21, 23] correspond to automated synthesis of self-stabilizing systems with at most 5 processes. One way to address this scalability issue in synthesis is to use a counterexample-guided inductive synthesis (CEGIS) method, as it has been proposed for the completion of program sketches [48], for the lazy synthesis of reactive systems [27,34], and for the synthesis of Byzantine-resilient systems [7]. The latter approach also supports the synthesis of self-stabilizing systems, but counterexamples are only used to guide the encoding of Byzantine-resilience, and the approach is limited to synchronous systems. The difference between these methods and our CEGIS approach is that we have used counterexamples to guide synthesis for an increasing size of the topology, which allows us to scale the SMT-based synthesis of self-stabilizing algorithms to systems with up to 200 processes.

Parameterized synthesis of distributed systems: The problem of scalability in the number of processes can be solved once and for all by using a parameterized synthesis approach, as introduced by Jacobs and Bloem [33] for (non-stabilizing) reactive systems, and later applied to synthesize a parameterized controller for the AMBA bus protocol [8]. The approach relies on cutoff results, similar to the ones we introduced in this work for closure and deadlock detection. Parameterized model checking and synthesis of guarded protocols for LTL properties based on cutoff results is studied in [4, 35]. The main difference between our work and theirs is that first of all we are focusing on self-stabilization, and secondly, they study processes with global knowledge (i.e., a process "sees" all other processes, not only its neighbors), while we study systems of processes with partial view of the global state. Modular application of cutoff results in synthesis is introduced in [36]. An extension of the approach [7] also supports the parameterized synthesis of self-stabilizing systems, but only for synchronous systems, and not in all cases resulting in a completely symmetric system. There are also papers on the synthesis of parameterized self-stabilizing protocols for a specific problem. For example, the authors in [15] provide a synchronous self-stabilizing and Byzantine-tolerant parameterized protocol for counting problem in clique topologies. Lazic et al. [42] propose a method for synthesizing parameterized fault-tolerant distributed algorithms. In contrast to our approach, synthesis is based on a sketch of an asynchronous threshold-based fault-tolerant distributed algorithm, and the goal is to find the right values for coefficients that may be missing in the guards.

In [25] and [26], the authors present a method to synthesize parameterized self-stabilizing protocols in rings by proposing necessary and sufficient conditions specified in the local state space of each process for deadlock-freedom. They also

propose sufficient conditions that guarantee the absence of cycles (livelocks) in parameterized unidirectional rings. Our work is different in that first of all, there is no discussion on closure in these references, and secondly, the results are only presented on rings (deadlock-freedom on bidirectional rings and livelock freedom in unidirectional rings). It is not obvious how these results can be extended to reason about these properties in other topologies, or to guarantee closure. The sufficient idea introduced in these references is similar to our process abstraction approach, as they are both only sufficient, and also they both identify livelock-freedom from the local state space. However, in our approach, the abstraction can be refined, if our specified property is not satisfied, while the approach in [26] is only on the local states of one process, and does not rely on the changes in other processes of the ring, and hence, if the proposed condition is not satisfied on the local state space of one process, there is no way to add more information about other processes, and check if it can be satisfied. It is shown that parameterized verification of self-stabilization is undecidable in uni-directional rings [38], while the parameterized synthesis problem is undecidable in bi-directional rings, but surprisingly remains decidable in uni-directional rings [41]. Authors in [41] proposed a sound and complete algorithm for synthesizing self-stabilizing protocols for uni-directional rings. This work is extended in [17] to other unidirectional topologies, including chains and trees. Our paper is different in that although our proposed CEGIS based method is not complete, but it is not specific to any particular topology. Also, our cutoff results cover more topologies, using which researchers can synthesize parameterized solutions to different self-stabilizing problems. In other words, thanks to the cutoffs, any synthesis method that can synthesize a solution up to the identified cutoff, is guaranteed to be a parameterized solution.

Parameterized verification of distributed systems: There are several papers on parameterized verification of distributed systems. Some of these papers use the idea of invariants [9, 50], where invariant is a safety property that is satisfied for a process, and any system that is created by composing the process with itself for arbitrary number of times. In [11], a new model checker for verifying safety properties of parameterized systems is introduced. Parameterized verification of deadlock freedom for a class of symmetric systems based on abstraction is studied in [6]. Parameterized verification of safety properties by determining cut-offs is studied in [31, 47], as well. Note that in self-stabilization, we have safety as well as liveness properties. There is also a class of works on parameterized verification using compositional model checking [5, 44]. In this work, we focus on parameterized synthesis of a class of distributed systems. Although there may be ways for parameterized synthesis using parameterized verification of distributed systems, our idea is based on introducing synthesis cutoffs for different topologies.

8 Conclusion

In this paper, we proposed a new method for parameterized synthesis of self-stabilizing algorithms in symmetric rings using cutoff points. We presented new cutoff results for closure properties and deadlock detection in ring, line, mesh and torus topologies. We proved tightness of the cutoff results for rings and conjecture that those for lines are also tight, while tightness of the cutoffs for the other two topologies remains an open problem.

Furthermore, in order to scale the existing synthesis solutions [20–24] up to the cutoff point, we introduced an iterative loop of synthesis and verification guided by counterexamples. We demonstrated the effectiveness of our approach by synthesizing parameterized self-stabilizing protocols for well-known problems including self-stabilizing three coloring, maximal matching, and maximal independent set. For future, we plan to work on asymmetric and synchronous networks.

References

1. R. Alur, M. Raghothaman, C. Stergiou, S. Tripakis, and A. Udupa. Automatic completion of distributed protocols with symmetry. In *International Conference on Computer Aided Verification*, pages 395–412. Springer, 2015.
2. R. Alur and S. Tripakis. Automatic synthesis of distributed protocols. *SIGACT News*, 48(1):55–90, 2017.
3. Paul C. Attie and E. Allen Emerson. Synthesis of concurrent systems with many similar processes. *ACM Trans. Program. Lang. Syst.*, 20(1):51–115, 1998.
4. S. Außerlechner, S. Jacobs, and A. Khalimov. Tight cutoffs for guarded protocols with fairness. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 476–494, 2016.
5. S. Basu and C. Ramakrishnan. Compositional analysis for verification of parameterized systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 315–330, 2003.
6. B. Bingham, M. Greenstreet, and J. Bingham. Parameterized verification of deadlock freedom in symmetric cache coherence protocols. In *International Conference on Formal Methods in Computer-Aided Design*, pages 186–195, 2011.
7. R. Bloem, N. Braud-Santoni, and S. Jacobs. Synthesis of self-stabilising and byzantine-resilient distributed systems. In *CAV*, pages 157–176, 2016.
8. Roderick Bloem, Swen Jacobs, and Ayrat Khalimov. Parameterized synthesis case study: AMBA AHB. In *Workshop on Synthesis*, volume 157 of *EPTCS*, pages 68–83, 2014.
9. E. M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(5):726–750, 1997.
10. Edmund M. Clarke, Muralidhar Talupur, and Helmut Veith. Environment abstraction for parameterized verification. In *VMCAI*, volume 3855 of *Lecture Notes in Computer Science*, pages 126–141. Springer, 2006.
11. S. Conchon, A. Goel, S. Krstić, A. Mebsout, and F. Zaïdi. Cubicle: A parallel smt-based model checker for parameterized systems. In *International Conference on Computer Aided Verification*, pages 718–724, 2012.
12. S. Devismes, S. Tixeuil, and M. Yamashita. Weak vs. self vs. probabilistic stabilization. In *ICDCS*, pages 681–688, 2008.
13. E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.
14. E. W. Dijkstra. A belated proof of self-stabilization. *Distributed Computing*, 1(1):5–6, 1986.
15. D. Dolev, K. Heljanko, M. Järvisalo, J. Korhonen, Ch. Lenzen, J. Rybicki, J. Suomela, and S. Wieringa. Synchronous counting and computational algorithm design. *Journal of Computer and System Sciences*, 82(2):310–332, 2016.
16. A. Ebnenasir and A. Farahat. A lightweight method for automated design of convergence. In *IPDPS*, pages 219–230, 2011.
17. A. Ebnenasir and A. Klinkhamer. Topology-specific synthesis of self-stabilizing parameterized systems with constant-space processes. *IEEE Transactions on Software Engineering*, 2019.
18. E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *International Journal on Foundations of Computer Science.*, 14(4):527–550, 2003.
19. A. Cimatti et. al. Nusmv 2: An opensource tool for symbolic model checking. In *CAV*, pages 359–364, 2002.
20. F. Faghih and B. Bonakdarpour. SMT-based synthesis of distributed self-stabilizing systems. In *SSS*, pages 165–179, 2014.

21. F. Faghih and B. Bonakdarpour. SMT-based synthesis of distributed self-stabilizing systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(3):21, 2015.
22. F. Faghih and B. Bonakdarpour. ASSESS: A tool for automated synthesis of distributed self-stabilizing algorithms. In *SSS*, pages 219–233, 2017.
23. F. Faghih, B. Bonakdarpour, S. Tixeuil, and S. Kulkarni. Specification-based synthesis of distributed self-stabilizing protocols. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE)*, pages 124–141, 2016.
24. F. Faghih, B. Bonakdarpour, S. Tixeuil, and S. Kulkarni. Specification-based synthesis of distributed self-stabilizing protocols. *Logical Methods in Computer Science*, To appear.
25. A. Farahat. *Automated design of self-stabilization*. PhD thesis, Michigan Technological University, 2012.
26. A. Farahat and A. Ebnenasir. Local reasoning for global convergence of parameterized rings. In *International Conference on Distributed Computing Systems*, pages 496–505, 2012.
27. B. Finkbeiner and S. Jacobs. Lazy synthesis. In *VMCAI*, 2012.
28. B. Finkbeiner and S. Schewe. Bounded synthesis. *International Journal on Software Tools for Technology Transfer (STTT)*, 15(5-6):519–539, 2013.
29. A. Gascón and A. Tiwari. Synthesis of a simple self-stabilizing system. *arXiv preprint arXiv:1407.5392*, 2014.
30. M. G. Gouda and H. B. Acharya. Nash equilibria in stabilizing systems. In *SSS*, pages 311–324, 2009.
31. Y. Hanna, D. Samuelson, S. Basu, and H. Rajan. Automating cut-off for multi-parameterized systems. In *International Conference on Formal Engineering Methods*, pages 338–354, 2010.
32. D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press Cambridge, 2012.
33. S. Jacobs and R. Bloem. Parameterized synthesis. *Logical Methods in Computer Science*, 10(1), 2014.
34. S. Jacobs and M. Sakr. A symbolic algorithm for lazy synthesis of eager strategies. In *Automated Technology for Verification and Analysis (ATVA)*, 2018.
35. Swen Jacobs and Mouhammad Sakr. Analyzing guarded protocols: Better cutoffs, more systems, more expressivity. In *VMCAI*, volume 10747 of *Lecture Notes in Computer Science*, pages 247–268. Springer, 2018.
36. A. Khalimov, S. Jacobs, and R. Bloem. Towards efficient parameterized synthesis. In *VMCAI*, volume 7737 of *Lecture Notes in Computer Science*, pages 108–127. Springer, 2013.
37. A. Klinkhamer and A. Ebnenasir. On the complexity of adding convergence. In *Fundamentals of Software Engineering (FSEN)*, pages 17–33, 2013.
38. A. Klinkhamer and A. Ebnenasir. Verifying livelock freedom on parameterized rings and chains. In *SSS*, pages 163–177, 2013.
39. A. Klinkhamer and A. Ebnenasir. Synthesizing self-stabilization through superposition and backtracking. In *SSS*, pages 252–267, 2014.
40. A. Klinkhamer and A. Ebnenasir. Shadow/puppet synthesis: A stepwise method for the design of self-stabilization. *IEEE Transactions on Parallel and Distributed Systems*, 27(11):3338–3350, 2016.
41. A. Klinkhamer and A. Ebnenasir. Synthesizing parameterized self-stabilizing rings with constant-space processes. In *Fundamentals of Software Engineering (FSEN)*, pages 100–115, 2017.
42. M. Lazic, I. Konnov, J. Widder, and R. Bloem. Synthesis of distributed algorithms with parameterized threshold guards. In *On Principles of Distributed Systems (OPODIS)*, 2017.
43. F. Manne, M. Mjelde, L. Pilard, and S. Tixeuil. A new self-stabilizing maximal matching algorithm. *Theoretical Computer Science*, 410(14):1336–1345, 2009.
44. K. McMillan. Parameterized verification of the flash cache coherence protocol by compositional model checking. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 179–195, 2001.
45. N. Mirzaie, F. Faghih, S. Jacobs, and B. Bonakdarpour. Parameterized synthesis of self-stabilizing protocols in symmetric rings. In *On Principles of Distributed Systems (OPODIS)*, 2018.
46. Kerry Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989.

47. A. Siirtola and K. Heljanko. Dynamic cut-off algorithm for parameterised refinement checking. In *International Conference on Formal Aspects of Component Software*, pages 256–276, 2018.
48. Armando Solar-Lezama. Program sketching. *STTT*, 15(5-6):475–495, 2013.
49. T. Weise and K. Tang. Evolving distributed algorithms with genetic programming. *IEEE Transactions on Evolutionary Computation*, 16(2):242–265, 2011.
50. P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *International Conference on Computer Aided Verification*, pages 68–80. Springer, 1989.