Computer Science Technical Report

Topology-Specific Synthesis of Self-Stabilizing Parameterized Systems With Constant-Space Processes

Ali Ebnenasir and Alex Klinkhamer

Michigan Technological University
Computer Science Technical Report
CS-TR-18-03
November 2018

Department of Computer Science
Houghton, MI 49931-1295
www.cs.mtu.edu

Topology-Specific Synthesis of Self-Stabilizing Parameterized Systems With Constant-Space Processes

Ali Ebnenasir and Alex Klinkhamer

November 2018

Abstract

This paper investigates the problem of synthesizing parameterized systems that are self-stabilizing by construction. To this end, we present several significant results. First, we show a counterintuitive result that despite the undecidability of verifying self-stabilization for parameterized unidirectional rings, synthesizing self-stabilizing unidirectional rings is decidable! This is surprising because it is known that, in general, the synthesis of distributed systems is harder than their verification. Second, we present a topology-specific synthesis method (derived from our proof of decidability) that generates the state transition system of template processes of parameterized self-stabilizing systems with elementary unidirectional topologies (e.g., rings, chains, trees). We also provide a software tool that implements our synthesis algorithms and generates interesting self-stabilizing parameterized unidirectional rings in less than 50 microseconds on a regular laptop. We validate the proposed synthesis algorithms for decidable cases in the context of several interesting distributed protocols. Third, we show that synthesis of self-stabilizing bidirectional rings remains undecidable.

Contents

1 Introduction

Developing parameterized Self-Stabilizing (SS) distributed systems is an important and challenging problem since a parameterized SS system must be self-stabilizing regardless of the number of processes. An SS system must have two properties, namely convergence and closure. *Closure* stipulates that, starting from any legitimate state, system executions remain in the set of legitimate states (a.k.a. *invariant*) – that captures the desired behaviors of a system[1]. *Convergence* requires that from *any* configuration/state, every system execution recovers to some invariant state in a finite amount of time. Such a global recovery must be achieved solely by the local actions of processes (without any central point of coordination). Designing self-stabilization becomes even more challenging for *parameterized* systems that include families of unbounded number of symmetric processes. Two processes are *symmetric* if the code of one can be obtained from another by a simple variable renaming. Each *family* may include an unbounded (but finite) number of symmetric processes that can be represented by a *template* process (a.k.a. *representative* process) from which the code of each process is instantiated. As the verification of SS parameterized unidirectional rings (a.k.a. uni-rings) is known to be undecidable [26], the common understanding has been that synthesizing such systems should also be undecidable. In this paper, we prove otherwise! We show that synthesizing self-stabilization is actually decidable for parameterized uni-rings.

Numerous approaches exist for the synthesis of parameterized systems, most of which focus on synthesis from temporal logic specifications while assuming some sort of fairness. For example, Finkbeiner and Schewe [16] present *bounded synthesis* where they formulate the synthesis of fixed-size systems as a constraint solving problem, and use Satisfiability Modulo Theory (SMT) solvers [8] to search for a program that is accepted by a Universal Co-Buchi Tree (UCT) automaton generated from temporal logic specifications. Such a search is conducted up to a specific bound on the size of the state spaces of processes (and/or their automata-theoretic product). Jacobs and Bloem [23] extend the approach in [16] to parameterized systems by reducing the synthesis of parameterized systems (a.k.a. *parameterized synthesis*) to bounded synthesis of a small network of symmetric processes (under the assumption of fair token passing). They enable a semi-decision procedure that will eventually find a solution if one exists. Additionally, some researchers have investigated the synthesis of parameterized SS systems in a problem-specific context. For instance, Dolev *et al.* [11] present a method for generating synchronous and constant-space counting algorithms where processes should implement a distributed finite counter. They provide an SS and Byzantine-tolerant parameterized protocol for clique topologies where all processes increment their value synchronously. Bloem *et al.* [5] use bounded synthesis and parameterized synthesis to extend Dolev *et al.*'s approach for other problems. Lenzen and Rybicki [31] provide an SS and Byzantine-tolerant solution for the counting problem with near-optimal stabilization time and message sizes. What the aforementioned methods have in common is that they are based on bounded/parameterized synthesis from temporal logic specifications (using SMT solvers), and they make assumptions about synchrony, fairness and complete knowledge of the network for each process. Moreover, bounded and parameterized synthesis suffer from the following drawbacks: (i) formulation of constraints and the generation of the UCT from temporal logic specifications are computationally expensive tasks; (ii) SMT solvers are sensitive to small changes in their inputs and take a major chunk of time/resources needed for synthesis, and (iii) the iterative nature of bounded synthesis makes it costly since every time the constraints are deemed unsatisfiable for a specific bound, the bound is increased and the entire process of constraint generation and SMT solving must be repeated.

In this paper, we take a *topology and property-specific* approach where we focus on self-stabilization, and start with a first-order logic formula representing the invariant to which self-stabilization should be synthesized (instead of synthesis from temporal logic specifications). For simplicity and practical reasons, we consider formulas that are conjunctive; i.e., the global invariant is specified as the conjunction of local invariants of processes. While our assumption about the form of the invariant may seem restrictive, there are important applications for such systems [41, 20]. In our previous work [26], we have shown that verifying self-stabilization to conjunctive invariants for symmetric uni-rings is undecidable. That is, given the parameterized code of a fully symmetric uni-ring and a conjunctive invariant $\mathcal{I}$, it is undecidable to verify

---

[1] In this paper, we use the terms invariant and legitimate states interchangeably.

whether the instantiation of the parameterized code for individual processes would result in a system that is SS to $\mathcal{I}$ for arbitrary ring sizes. By contrast, the synthesis problem takes $\mathcal{I}$ and generates the parameterized code of a symmetric uni-ring such that the instantiation of that code for any ring size will provide a system that is SS to $\mathcal{I}$ by construction.

We show that synthesizing SS symmetric uni-rings of constant-space processes is actually decidable. This is surprising because it is known [34] that, in general, the synthesis of distributed systems is harder than their verification. We first present a *necessary and sufficient* condition for the existence of a symmetric SS uni-ring. Our necessary and sufficient condition states that an SS symmetric uni-ring exists if and only if (iff) there is a value to which both a process and its predecessor can recover. Intuitively, we show that, the existence of a simple solution where global convergence is achieved by just setting the local variables of processes to a specific value is *necessary and sufficient* for the existence of an SS solution for an invariant. By contrast, in the case of verification of self-stabilization for uni-rings, one has to investigate an intractable number of scenarios to ensure the correctness of stabilization for all ring sizes.

Using our proof of decidability, we devise a sound and complete algorithm for the synthesis of symmetric SS uni-rings. The input to our algorithm includes a conjunctive invariant and the size of the state space of processes. The output of the proposed algorithm is the parameterized code of the template process so that the entire ring becomes SS for an arbitrary (but finite) number of processes. We extend our results on uni-rings to parameterized chains and trees. Specifically, we perform the synthesis in a bottom-up fashion by systematically constructing a directed graph, called the *legitimacy graph*, that captures the local invariant that a process and its neighbors can have. Each vertex of the legitimacy graph captures a specific value in the state space of each process, and each arc denotes whether the source and the target vertices/values meet the constraints of the local invariant. This makes the legitimacy graph different from a state machine as the arcs are not transitions. The proposed synthesis algorithm then transforms the legitimacy graph into a finite state automaton representing the local actions of the template process. In this sense, our proposed synthesis method is graph-theoretic. We also investigate the synthesis of SS bidirectional rings, and show that this problem remains undecidable.

We have implemented and integrated the proposed algorithms in the Protocon framework [25]. Using Protocon, we have automatically synthesized several SS uni-rings in less than a 50 micro seconds on a regular MacBook Air laptop. More importantly, this work is the first step in the context of a broader *synthesize-and-compose* initiative, where (in our future work) we will develop rules for composing parameterized systems with elementary topologies to generate more sophisticated topologies while preserving self-stabilization.

Contributions. This paper

- presents a surprising result that synthesizing symmetric SS uni-rings under the interleaving semantics and no fairness assumption is decidable (even though verifying self-stabilization of uni-rings is undecidable);
- puts forward a novel synthesis method, where instead of synthesis from temporal logic specifications we characterize local invariants as legitimacy graphs and automatically transform them to the state transition system of template processes;
- provides synthesis algorithms for elementary unidirectional topologies such as chains and trees (in addition to rings), and
- proves that synthesizing SS *bidirectional* rings is undecidable.

Organization. Section 2 presents basic concepts. Section 3 shows that synthesizing SS uni-rings is decidable. Section 4 investigates the synthesis of parameterized SS top-down trees, and Section 5 studies SS bottom-up trees. Section 6 investigates the synthesis of SS bidirectional rings and proves that this problem is undecidable. Section 7 presents our experimental results. Section 8 examines related work. Finally, Section 9 makes concluding remarks and discusses future extensions of this work.

2 Basic Concepts

This section presents the definition of parameterized systems, their representation as action graphs, and self-stabilization. Wlog, we use the term *protocol* to refer to finite-state parameterized systems as we conduct our investigation in the context of network coordination protocols.

Definition 2.1 (Template Process)**.** Intuitively, a template process captures the functionalities of each individual process in a set of $N \geq 1$ symmetric processes parameterized by $i \in \mathbb{Z}_N$, i.e., $0 \leq i \leq N-1$. Formally, a *template* process $\mathcal{P}_i$ is a tuple $\langle R_i, x_i, M_i, \delta_i \rangle$, where R_i represents the set of variables that $\mathcal{P}_i$ can read, x_i is the variable $\mathcal{P}_i$ can write (which is an abstraction of all writable variables), M_i is the domain size of x_i (i.e., $x_i \in \mathbb{Z}_{M_i}$), and δ_i denotes $\mathcal{P}_i$'s transition function. We assume $x_i \subseteq R_i$; i.e., no variable can be written blindly. The variables in R_i define the *locality/neighborhood* of $\mathcal{P}_i$ which includes the processes whose state $\mathcal{P}_i$ can read.

Definition 2.2 (State Space and State Predicate)**.** A unique valuation of all variables in R_i is a *local state* of $\mathcal{P}_i$. We use $v(s)$ to denote the value of a variable v in a state s. The local state space of $\mathcal{P}_i$, denoted Σ_i, includes all possible local states of $\mathcal{P}_i$. A *local state predicate* is a set of local states.

Definition 2.3 (Instantiation of Template Processes)**.** An *instantiation* of a template process $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$ is a process $\langle R_j, x_j, M_j, \delta_j \rangle$, where j is a fixed integer and R_j, x_j, M_j and δ_j are obtained from $\mathcal{P}_i$ by substituting i with j everywhere; i.e., state space and transition function are obtained from those of $\mathcal{P}_i$ by a simple variable re-indexing. (Note that, $M_i = M_j$.) Each template process can be instantiated for an arbitrary number of times $N \geq 1$ in a network. For example, in a fully symmetric uni-ring consisting of $N \geq 1$ processes, we have only one template process since all processes are symmetric, and each instantiated process P_j (where $j \in \mathbb{Z}_N$, i.e., $0 \leq j \leq N-1$) has a predecessor neighbor P_{j-1}, where subtraction and addition are done in modulo N. In this case, $R_j = \{x_{j-1}, x_j\}$.

Definition 2.4 (Parameterized Protocol)**.** A parameterized protocol $p = \langle \mathcal{P}, \mathcal{T}_p \rangle$ for a computer network includes $k \geq 1$ template processes $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_k\}$, and a topology $\mathcal{T}_p$ that defines the underlying communication graph of p through variables each process can read/write. A *global state* of p is a unique valuation to all instantiated processes from any template process. The *projection of a global state s on a process P_j* is the value of x_j in state s; i.e., $x_j(s)$. The global state space of p, denoted Σ_p, includes all possible states.

Definition 2.5 (Transition Function)**.** Let $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$ be a template process. A local transition is an ordered pair (s, s') from a local state s to another local state s' as a result of an atomic update on x_i. Formally, $\delta_i : \Sigma_i \to \Sigma_i$ is a partial function from Σ_i to Σ_i. Since in each valid transition, $\mathcal{P}_i$ updates x_i, we can rewrite δ_i as a partial function from Σ_i to $\mathbb{Z}_{M_i}$. That is, in a transition (s, s'), we have $\forall v : v \in R_i \wedge v \neq x_i : v(s) = v(s')$. Notice that, the transition function of $\mathcal{P}_i$ is deterministic; i.e., from any state $s \in \Sigma_i$, a transition can change the state of $\mathcal{P}_i$ to at most one other state s'. The function $\mathsf{Pre}(\delta_i) : \delta_i \to \Sigma_i$ returns the set of states from where δ_i has some transition, called the *pre-image* of δ_i. Likewise, we define the function $\mathsf{Post}(\delta_i) : \delta_i \to \Sigma_i$ that returns the set of states to which δ_i has some transition, called the *post-image* of δ_i. We assume that $\mathsf{Pre}(\delta_i) \cap \mathsf{Post}(\delta_i) = \emptyset$. That is, when a process executes, it disables itself; i.e., the processes are *self-disabling*[2]. To simplify reasoning in terms of process behaviors, we rewrite δ_i in the form of a parametric action:

$$R_i \in \mathsf{Pre}(\delta_i) \longrightarrow x_i := \delta_i(R_i);$$

where $(R_i \in \mathsf{Pre}(\delta_i))$ checks to see if the current values of variables in R_i are in the preimage of δ_i. An action (a.k.a. *guarded command*) is an atomic "if-then" statement; i.e., if the condition on the lefthand side of $\longrightarrow$ holds (i.e., action is *enabled*) then the statements on the righthand side of $\longrightarrow$ are executed atomically.

---

[2] We have shown [28] that a self-stabilizing solution exists for a problem if and only if there is a self-stabilizing solution for that problem with deterministic and self-disabling processes.

Example 2.6 (Transition Function of Symmetric Uni-Rings)**.** *Let* $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$ *be the template process of a fully symmetric uni-ring, and* $\mathcal{P}_i$ *is instantiated* $N \geq 1$ *times, forming a ring of size* N*. Notice that, in this case, there is only one template process (*$k = 1$*) since the ring is fully symmetric. Each instantiated process* P_j *(*$1 \leq j \leq N$*) has a predecessor, where* $R_j = \{x_{j-1}, x_j\}$*. Let* a, b *and* c *be three values in* $\mathbb{Z}_{M_j}$*. Then, there is a parametric action* $(x_{i-1} = a \wedge x_i = b \longrightarrow x_i := c;)$ *corresponding to the triple* (a, b, c) *iff* $(a, b) \in \mathsf{Pre}(\delta_i)$*, the transition* $\langle (a, b) \to (a, c) \rangle \in \delta_i$*, and* $(a, c) \notin \mathsf{Pre}(\delta_i)$*. Thus, actions can also be represented as triples* (a, b, c) *in uni-rings.*

For other topologies, the same definition of transition function holds except that the preimage of δ might be specified differently depending on the locality of each process.

Definition 2.7 (Computation and Closure)**.** We assume an *interleaving* execution semantics for protocols, where processes act one at a time non-deterministically. That is, if there are some enabled actions (potentially belonging to different processes), then one will be executed non-deterministically. Thus, each *global transition* (s_0, s_1) is actually a local transition of some process P_j starting at the projection of s_0 on P_j. An *execution/computation* of a protocol is a sequence of states $C_0, C_1, \ldots, C_k$ where there is a transition from C_i to C_{i+1} for every $i \in \mathbb{Z}_k$. A state predicate $\mathcal{I}$ is *closed* under/in p iff any computation of p that starts in $\mathcal{I}$ remains in $\mathcal{I}$, in the absence of faults.

Definition 2.8 (Fairness)**.** *Weak* (respectively, *strong*) fairness policy ensures that any action that is continuously (respectively, infinitely often) enabled, will be executed infinitely often. We have shown [27] that synthesizing self-stabilization under weak fairness or no fairness assumptions is an NP-hard problem, whereas it is polynomially solvable under strong fairness [19] (because a strongly fair scheduler ensures recovery from lovelocks). In this paper, we make no assumption on fairness. Since actions are self-disabling, once an action executes it will be disabled until it is enabled again by either its predecessor (in a unidirectional network) or the occurrence of faults. An enabled action may then be selected for execution non-deterministically.

Definition 2.9 (Legitimate States/Invariant)**.** Intuitively, a *set of legitimate states* (a.k.a. *Invariant*) represents the states from where a protocol behaves normally and remains in that set. Formally, an invariant is a state predicate $\mathcal{I}$ that is closed in a protocol p to which convergence is required. Our definition of an invariant is more relaxed in comparison to other researchers [2, 29] as in the synthesis of SS protocols we are mainly concerned with ensuring the closure of the invariant without adding new computations in it while designing convergence. We focus on conjunctive invariants in the form of $\forall i : i \in \mathbb{Z}_N : L_i(R_i)$, where $L_i(R_i)$ denotes a local state predicate that must hold in the locality of each process. Varghese [41, 42] presents methods for specifying some global state predicates as conjunctive predicates.

In the rest of this paper up to Section 4, we shall focus on symmetric uni-rings only. Let $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$ be the template process of a symmetric uni-ring. To ease the presentation, we define the notion of action graphs.

Definition 2.10 (Action Graph of Uni-Rings)**.** An *action graph* is a labeled directed multigraph $G = (V, A)$, where each vertex $v \in V$ represents a value in $\mathbb{Z}_{M_i}$, and each arc $(a, c) \in A$ with a label b captures an action $x_{i-1} = a \wedge x_i = b \longrightarrow x_i := c$.

For example, consider the self-stabilizing Sum-Not-2 protocol given in [14]. The template process $\mathcal{P}_i = \langle R_i, x_i, 3, \delta_i \rangle$ has a variable $x_i \in \mathbb{Z}_3$ and actions $(x_{i-1} = 0 \wedge x_i = 2 \longrightarrow x_i := 1)$, $(x_{i-1} = 1 \wedge x_i = 1 \longrightarrow x_i := 2)$, and $(x_{i-1} = 2 \wedge x_i = 0 \longrightarrow x_i := 1)$. This protocol converges to a state where the sum of each two consecutive x values does not equal 2. The set of such states is formally specified as the state predicate $\forall i : (x_{i-1} + x_i \neq 2)$. We represent this protocol with a graph containing arcs $(0, 2, 1)$, $(1, 1, 2)$, and $(2, 0, 1)$ as shown in Figure 1.

Since protocols consist of *self-disabling* processes, an action (a, b, c) cannot coexist with action (a, c, d) for any d. Moreover, a deterministic process cannot have two actions (a, b, c) and (a, b, d) where $d \neq c$.

Livelock, deadlock, and closure. A *livelock* of p is an infinite execution $\langle s_i, s_{i+1}, \cdots, s_k, s_i \rangle$ that never reaches $\mathcal{I}$. When no invariant is specified, we assume a livelock is any infinite execution. A *deadlock* of p is a state in $\neg\mathcal{I}$ that has no outgoing transition; i.e., no process is enabled to act.

Definition 2.11 (Transient Faults)**.** Let p be a parameterized protocol. We model transient faults as a set of transitions in $\Sigma_p \times \Sigma_p$. Such transition can occur non-deterministically for a finite amount of time. Thus, transient faults may perturb the state of a protocol to *any* state in its state space.

In practice, transient faults may occur due to a variety of reasons (e.g., loss of coordination, bad initialization, soft errors) and manifest themselves as state perturbations, but they do not cause permanent damage.

Definition 2.12 (Self-Stabilization)**.** A protocol p is *self-stabilizing* [10] to an invariant $\mathcal{I}$ *iff* from each illegitimate state in $\neg I$, all executions reach a state in $\mathcal{I}$ (i.e., *convergence*) and remain in $\mathcal{I}$ (i.e., *closure*). That is, p is livelock-free and deadlock-free in $\neg\mathcal{I}$, and $\mathcal{I}$ is closed under p.

Definition 2.13 (Weak Stabilization)**.** A protocol p is *weakly* stabilizing to an invariant $\mathcal{I}$ *iff* from each state in $\neg\mathcal{I}$, there is some execution that reaches a state in $\mathcal{I}$ (i.e., *reachability*) and remains in $\mathcal{I}$.

Notice that, any SS protocol is also weakly stabilizing but the reverse is not true.

Definition 2.14 (Silent Stabilization)**.** A protocol p is *silent stabilizing* to $\mathcal{I}$ *iff* p is self-stabilizing to $\mathcal{I}$ but executes no actions from any state in $\mathcal{I}$.

Definition 2.15 (Legitimacy Graphs)**.** Consider an invariant $\mathcal{I} = \forall i : L_i(x_{i-1}, x_i)$ for a uni-ring. The local state predicate L_i can be represented as a digraph $G = (V, A)$, called the *legitimacy graph*, such that each vertex $v \in V$ represents a value in $\mathbb{Z}_M$, and each arc (a, b) is in A *iff* $L_i(a, b)$ is true.

Next, we represent some of our previous result (from [14, 26]) that we shall use in this paper.

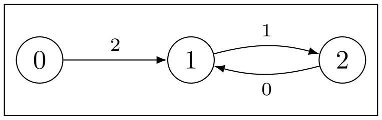


Figure 1: Graph representing Sum-Not-2 protocol.

Propagations and Collisions. When a process acts and enables its successor in a uni-ring, it propagates its ability to act. The successor may enable its own successor by acting, and the pattern may continue indefinitely. Such behaviors can be represented as sequences of actions that are propagated in a ring, called *propagations*. A propagation is a walk through the action graph. For example, the Sum-Not-2 protocol has a propagation $\langle(0, 2, 1), (1, 1, 2), (2, 0, 1), (1, 1, 2)\rangle$ whose actions can be executed in order by processes P_i, P_{i+1}, P_{i+2}, and P_{i+3} from a state $(x_{i-1}, x_i, x_{i+1}, x_{i+2}, x_{i+3}) = (0, 2, 1, 0, 1)$. A propagation is *periodic* with period n *iff* its j-th action and $(j + n)$-th action are the same for every index j. A propagation with period $n > 1$ corresponds to a *closed walk* of length n in the graph. The Sum-Not-2 protocol has such a propagation of period 2: $\langle(1, 1, 2), (2, 0, 1)\rangle$. A *collision* occurs when two consecutive processes, say P_i and P_{i+1}, have enabled actions; e.g., (a, b, c) and (b, e, f), where $b \neq c$. In such a scenario, $x_{i-1}{=}a, x_i{=}b, x_{i+1}{=}e$. A collision occurs when P_i executes and assigns c to x_i. If that occurs, P_i will be disabled (because processes are self-disabling), and P_{i+1} becomes disabled too because x_i is no longer equal to b. As a result, two enabled processes become disabled by one action.

"Leads" Relation. Consider two actions A_1 and A_2 in a process P_i. We say the action A_1 *leads* A_2 *iff* the value of the variable x_i after executing A_1 is the same as the value required for P_i to execute A_2. Formally, this means an action (a, b, c) leads (d, e, f) *iff* $e = c$. Similarly, a propagation leads another *iff* for every index j, its j-th action leads the j-th action of the other propagation. In the action graph, this corresponds to two walks where the label of the destination node of the j-th arc in the first walk matches the arc label of the j-th arc in the second walk (for each index j). In [26], we prove the following theorem:

Theorem 2.16. *A uni-ring protocol of symmetric, deterministic and self-disabling processes has a livelock for some ring size* iff *there exist some m propagations with some period n, where the $(i-1)$-th propagation leads the i-th propagation for each index i modulo m; i.e., the propagations successively lead each other modulo m.*

Undecidability of Verification. We have shown [14] that verifying deadlock-freedom in uni-rings is decidable. However, checking livelock-freedom is an undecidable problem (specifically Π^0_1-complete) for uni-ring protocols (with self-disabling and deterministic processes) [26]. The following results hold for cases where the invariant $\mathcal{I}$ is a conjunctive predicate; i.e., $\mathcal{I} = \forall i : L_i(x_{i-1}, x_i)$.

Theorem 2.17. *Verifying livelock-freedom in a parameterized uni-ring protocol (with self-disabling and deterministic processes) is undecidable [26].*

We have also shown that verifying livelock-freedom remains undecidable even for a special type of livelocks, where exactly one process is enabled in every state of the livelocked computation; i.e., *deterministic livelocks* [26].

Theorem 2.18. *Verifying livelock-freedom in a parameterized uni-ring protocol (with self-disabling and deterministic processes) remains undecidable even for deterministic livelocks [26].*

The above results imply the undecidability of verifying self-stabilization for parameterized uni-rings.

Theorem 2.19. *Verifying self-stabilization for a parameterized uni-ring protocol (with self-disabling and deterministic processes) is undecidable [26].*

3 Decidability of Synthesizing Unidirectional Rings

In this section, we show that synthesizing SS uni-rings of deterministic, self-disabling and constant-space processes is decidable. First, we formulate the synthesis problem. Let $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$ be the template process of a fully symmetric uni-ring p, and $\mathcal{P}_i$ is instantiated $N > 1$ times, forming a uni-ring of size N, where N is an unbounded (but finite) positive integer. Moreover, let $\mathcal{I} = \forall j : 1 \leq j \leq N : L_j(x_{j-1}, x_j)$ represent an invariant of the ring.

Problem 3.1 (Synthesis of Unidirectional Rings)**.** We state the synthesis problem as follows:

- **Input**: $L_i(x_{i-1}, x_i)$, R_i, x_i, M_i and an integer $k > 2$. Note that, R_i defines the topology of the protocol modulo ring size N; i.e., when $i = 0$, $L_0(x_{N-1}, x_0)$.

- **Output**: The transition function δ_i (represented as an action graph) such that the entire ring is SS to $\mathcal{I} = \forall j : 1 \leq j \leq N : L_j(x_{j-1}, x_j)$ for any ring size $N \geq k$.

Remark 1. Considering $L_i(x_{i-1}, x_i)$ as an input would suffice for synthesis since if L_i holds for all processes, then a global state in $\mathcal{I}$ is reached. Moreover, δ_i can be represented as an action graph whose every arc can be specified as a parametric action of the template process $\mathcal{P}_i$.

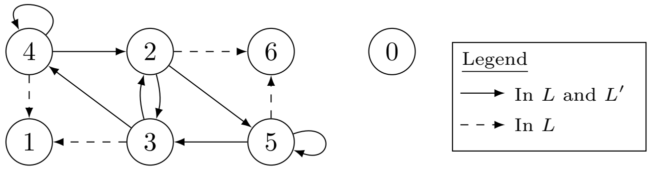
Remark 2. A straightforward solution of Problem 3.1 may seem like a simple parametric action $\neg L_i(x_{i-1}, x_i) \rightarrow x_i := c$, where $c \in \mathbb{Z}_{M_i}$ and $L_i(x_{i-1}, c)$ holds. This simply means that every process updates its x value such that L_i holds. However, such updates on x_i may further perturb the state of the successor of each process and destabilize the entire ring. That is, the resulting parameterized protocol may include livelocks; hence *weakly* stabilizing. This means that we need a systematic approach for local recovery to $L_i(x_{i-1}, x_i)$ such that the correction of the locality of one process will not negatively impact its successor.

Now, we represent a result due to Bernard *et al.* [3] on the impossibility of solving graph coloring on uni-rings as we refer to their results in our proofs. A valid coloring of the ring assigns colors to processes such that no two neighboring processes have similar colors.

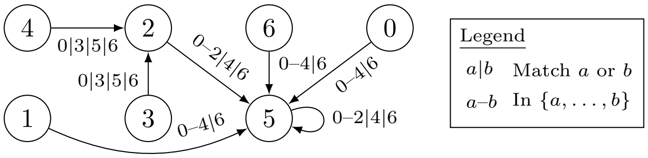
Lemma 3.2. *Let $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$ be the template process of a symmetric uni-ring. It is impossible to have a self-stabilizing graph coloring protocol p for rings of size $N > M_i$.*

Proof. Bernard *et al.* [3] show that if the ring has at most M_i processes, then assigning unique values to processes modulo M_i will provide an acceptable coloring. Otherwise, there is no valid coloring of the rings of sizes $N > M_i$ (as there would always be two neighbors with similar colors). ☐

We also represent one of our previous results as the following lemma since we shall use it in subsequent proofs.



(a) The legitimacy graph G for predicate L and its sub-graph G'.



(b) Stabilizing Protocol p as a digraph, called Action Graph.

Figure 2: Synthesis of stabilization to $\forall i : L(x_{i-1}, x_i)$, where
$L(x_{i-1}, x_i) \overset{\text{def}}{=} \big((x_{i-1}^2 + x_i^3) \bmod 7 = 3\big)$ and $x_i \in \mathbb{Z}_7$.

Lemma 3.3. *A closed walk of length $l > 1$ in the legitimacy graph of a symmetric uni-ring characterizes the global states of uni-rings of sizes $k \times l$, where $k \geq 1$. (Proof in [14].)*

Theorem 3.4. *Let $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$ be the template process of a parameterized symmetric uni-ring, and the state predicate $\mathcal{I} \overset{\text{def}}{=} (\forall i : L_i(x_{i-1}, x_i))$ capture its invariant. There exists a parameterized protocol that stabilizes to $\mathcal{I}$ if and only if $L_i(\gamma, \gamma)$ is true for some $\gamma \in M_i$.*

Proof. Assume that no γ exists in M_i such that $L_i(\gamma, \gamma)$ is true. This implies that $\forall i : x_{i-1} \neq x_i$ in $\mathcal{I}$. In this case, a stabilizing protocol would be a coloring protocol, which is impossible by Lemma 3.2 for ring sizes greater than M_i. This means if we check the entire domain $\mathbb{Z}_{M_i}$ and find no value that makes L_i true, then using Lemma 3.2, we can *decide* that no solution exists for ring sizes greater than M_i. That is, the problem is decidable when $L_i(\gamma, \gamma)$ is false for all $\gamma \in \mathbb{Z}_{M_i}$. We are left to show how to construct a stabilizing protocol p when some γ can make $L_i(\gamma, \gamma)$ true.
$\Rightarrow$ **Find a γ such that $L_i(\gamma, \gamma)$ is true** Assuming such a γ exists, it is trivial to find it by trying each value in $\mathbb{Z}_{M_i}$. Intuitively, we will make the stabilizing protocol p converge to $(\forall i : x_i = \gamma)$ unless it reaches some other state that satisfies $\mathcal{I}$. To illustrate the proof strategy and ease its understanding, Figure 2 provides an example where $L_i(x_{i-1}, x_i) \overset{\text{def}}{=} ((x_{i-1}^2 + x_i^3) \bmod 7 = 3)$ and variables have domain size $M_i = 7$. We arbitrarily choose $\gamma = 5$ to satisfy $L_i(\gamma, \gamma)$; i.e., the solution is not unique.
Construct relation L'_i from arcs that form cycles in the legitimacy graph of L_i. Let G be the legitimacy graph of L_i (e.g., the graph formed by both solid and dashed lines in Figure 2a). By Lemma 3.3, closed walks in G characterize states in $(\forall i : L_i(x_{i-1}, x_i))$. Derive a sub-graph G' (and corresponding relation L'_i) from G by removing all arcs that are not part of a cycle (e.g., arcs $(4, 1)$, $(3, 1)$, $(2, 6)$, and $(5, 6)$ in Figure 2a). We know that for every arc (a, b) in G that is not part of a cycle, no legitimate state contains $x_{i-1}{=}a \wedge x_i{=}b$ at any index i. All closed walks of G are retained by G', which means $\mathcal{I} \overset{\text{def}}{=} (\forall i : L'_i(x_{i-1}, x_i))$.
Construct a bottom-up spanning tree τ with γ at the root. To ensure that no global livelocks will occur in any instance of the protocol, we must guarantee that no periodic propagations exist that lead each other successively (see Theorem 2.16). To this end, we construct a spanning tree of G' with the root of γ. Let τ be a function that returns the parent of a node in a tree; i.e, $\tau(a) = c$ means that c is the parent of a. First, let $\tau(\gamma) \overset{\text{def}}{=} \gamma$ represent the root of the tree. Next, create a tree by backward reachability from γ in G', and assign $\tau(a) \overset{\text{def}}{=} c$ for each a that has a path $a, c, \ldots, \gamma$ in G'. Finally, let $\tau(a) \overset{\text{def}}{=} \gamma$ for each node a that has no path to γ in G'. These extra arcs of τ create no cycles. Thus, $(\forall i : (L'_i(x_{i-1}, x_i) \vee \tau(x_{i-1}){=}x_i))$ is yet another equivalent way to write $\mathcal{I}$.
Construct each action (a, b, c) of p by labeling each arc (a, c) of τ with all b values where $(\neg L'_i(a, b) \wedge \tau(a){\neq}b)$ holds. In this way, τ defines how a process P_i in p will assign x_i when it detects an

illegitimate state. Figure 2b illustrates the solution protocol for our example, as well as τ if we ignore the arc labels. The protocol p is written succinctly by the following action for each process P_i.

$$\neg L'_i(x_{i-1}, x_i) \wedge \tau(x_{i-1}) \neq x_i \longrightarrow x_i := \tau(x_{i-1});$$

This protocol p stabilizes to $\mathcal{I}$. Deadlock-freedom in $\neg\mathcal{I}$ and closure of $\mathcal{I}$ hold because each process P_i is enabled to act *iff* $(\neg L'_i(x_{i-1}, x_i) \wedge \tau(x_{i-1}) \neq x_i)$ holds. Livelock-freedom holds because all periodic propagations of p consist of actions of the form (γ, b, γ) where $L_i(\gamma, b)$ is false (e.g., the self-loops of Node 5 in Figure 2b). Obviously none of these (γ, b, γ) actions lead each other since $b \neq \gamma$; i.e., no periodic propagations exist. Thus, based on Theorem 2.16, no livelocks exist in $\neg\mathcal{I}$ for any ring size greater than M_i. Therefore, the parameterized protocol p stabilizes to $\mathcal{I}$.
Proof $\Leftarrow$: Let p be a parameterized protocol p that stabilizes to $\mathcal{I}$ on a uni-ring. Thus, closure of $\mathcal{I}$ in p, deadlock-freedom and livelock-freedom of p in $\neg\mathcal{I}$ must hold. Since processes are deterministic and self-disabling, each process P_i contains some actions that are enabled in $\neg L_i(x_{i-1}, x_i)$. After the execution of a sequence of such actions $L_i(x_{i-1}, x_i)$ holds by setting x_i to some value $\lambda \in M_i$, and P_i becomes disabled. Due to livelock-freedom of p and Theorem 2.16, no periodic propagations should exists in p. That is, there cannot be any closed walks in the action graph of p other than self-loops over λ. The existence of such self-loops means $L_i(\lambda, \lambda)$ holds. ☐

Using the proof of Theorem 3.4, we present Algorithm 1. Since this algorithm is self-explanatory, we just prove its soundness and completeness.

Theorem 3.5 (Soundness)**.** *Algorithm 1 is sound; i.e., every parameterized protocol generated by Algorithm 1 for an invariant $\mathcal{I}$, upholds closure of $\mathcal{I}$ and converges to $\mathcal{I}$ from any state.*

Proof. The proof of soundness includes two parts, namely proof of closure of $\mathcal{I}$ and convergence to $\mathcal{I}$, where $\mathcal{I} = \forall i : L_i(x_{i-1}, x_i)$. Step 7 of the algorithm guarantees closure because once the protocol reaches a global state where all x_i are equal to γ no more actions will be taken; i.e., silent stabilization. Steps 4 to 7 ensure that the legitimacy graph does not include any periodic propagations (i.e., closed walks) that lead each other in a circular fashion (Theorem 2.16). As a result, the resulting protocol will be livelock-free. Moreover, each process eventually sets the value of x_i to γ by taking the actions in a path of the spanning tree towards its root; hence evaluating $L_i(x_{i-1}, x_i)$ to true. Further, starting from any state where $L_i(x_{i-1}, x_i)$ does not hold (i.e., states in $\neg\mathcal{I}$), there is at least one action that each process P_i can execute because its local state is in a state other than the root of the spanning tree. Thus, there are no deadlock states in $\neg\mathcal{I}$. Deadlock-freedom and livelock-freedom guarantee convergence to $\mathcal{I}$. ☐

Theorem 3.6 (Completeness)**.** *Algorithm 1 is complete; i.e., Algorithm 1 finds a self-stabilizing protocol if one exists.*

Proof. This algorithm declares failure only in Step 2, where no value γ exists that can satisfy $L_i(x_{i-1}, x_i)$, implying that no process can recover to its local invariant. ☐

Theorem 3.7. *The asymptotic time complexity of Algorithm 1 is polynomial (specifically quadratic) in the domain size M_i (proof straightforward; hence omitted).*

3.1 Case Studies

We now present some case studies for the synthesis of parameterized symmetric uni-rings using Algorithm 1.
Sum-Not-2 protocol. The Sum-Not-2 protocol (taken from [13]) is a simple but interesting protocol that illustrates the complexities of designing self-stabilizing systems. This is again a protocol on parameterized uni-rings with a domain size $M = 3$; i.e., values $\{0, 1, 2\}$. The invariant of Sum-Not-2 contains states where $\forall i : (x_{i-1} + x_i) \neq 2$ holds, where addition and subtraction are in modulo 3. Thus, for each process P_i, we have $L_i(x_{i-1}, x_i) \overset{\text{def}}{=} (x_{i-1} + x_i) \neq 2$. Figure 3a illustrates the legitimacy graph representing L_i in the locality

Algorithm 1 Synthesizing self-stabilizing uni-rings.

```
SynUniRing($L_i(x_{i-1}, x_i)$: state predicate, $M_i$: domain size)
Check if a value $\gamma \in \mathbb{Z}_{M_i}$ exists such that $L_i(\gamma, \gamma) = \textbf{true}$.
If no such $\gamma$ exists, then return $\emptyset$ and declare that no solution exists.
Construct the legitimacy graph $G = (V, A)$ of $L_i(x_{i-1}, x_i)$.
Induce a subgraph $G' = (V', E')$ that contains all arcs of $G$ that participate in cycles involving $\gamma$.
Compute a spanning tree of $G'$ rooted at $\gamma$.
For each node $v \in G$ that is absent from $G'$, include an arc from $v$ to the root of the spanning tree of
  $G'$. The resulting graph would still be a tree, denoted $T$.
Include a self-loop $(\gamma, \gamma)$ at the root of $T$.
Transform $T$ into an action graph of a protocol by the following step:
    For each arc $(a, c)$ in $T$, where $a, c \in \mathbb{Z}_{M_i}$, label $(a, c)$ with every value $b$ for which $L_i(a, b) =$
    false and $b \neq c$.
Return the actions represented by the arcs of $T$.
```

(a) Legitimacy graph representing predicate $L_i(x_{i-1}, x_i) = ((x_{i-1} + x_i) \neq 2)$ where each $x_i \in \mathbb{Z}_3$

(b) Action graph of the self-stabilizing protocol.

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

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

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

(c) Actions of each process P_i.

Figure 3: Synthesis of parameterized Sum-Not-2 on uni-rings.

of a process. In this case, there are two candidate values for γ, where $L(\gamma, \gamma)$ holds; i.e., values of 0 and 2. Wlog, we choose $\gamma = 0$ and form the spanning tree of the graph G with the root of 0. Stripping the graph in Figure 3b from the labels on its arcs would give us the spanning tree of G, and the graph with the labels is the action graph of the synthesized self-stabilizing protocol (in Figure 3c).

Parity. The Parity protocol specifies the local invariant of each process P_i as $L_i(x_{i-1}, x_i) \overset{\text{def}}{=} ((x_{i-1} - x_i) \bmod 2) = 0$, where $M_i = 4$. Thus, the invariant is $\forall i : ((x_{i-1} - x_i) \bmod 2) = 0$. Notice that if there is an even (respectively, odd) value in the ring, then all values will be even (respectively, odd) in a legitimate state. Thus, from any state, Parity will converge to either an all-odd or an all-even state. This protocol has applications in choosing a common parity policy in a distributed system, where from an arbitrary state all nodes will agree on a common parity policy. Figure 4a represents the legitimacy graph corresponding to the predicate L_i. All four values in the domain M_i are candidate values for γ. We choose $\gamma = 1$, and generate the action graph of Figure 4b. Figure 4c illustrates the actions of the self-stabilizing protocol. Please notice that this protocol would recover to global states where all values are odd. Symmetrically, one could generate a protocol that would stabilize to states where all values are even. This could be achieved by strengthening $L_i(x_{i-1}, x_i)$ by an additional constraint $(x_i \bmod 2 = 0)$.

(a) Legitimacy graph representing predicate $L_i(x_{i-1}, x_i) = ((x_{i-1} - x_i) \bmod 2 = 0)$ where each $x_i \in \mathbb{Z}_4$.

(b) Action graph of the self-stabilizing protocol.

$$(x_{i-1}{=}1 \lor x_{i-1}{=}3) \land (x_i{=}0 \lor x_i{=}2) \longrightarrow x_i := 1;$$
$$(x_{i-1}{=}0 \lor x_{i-1}{=}2) \land x_i{=}3 \longrightarrow x_i := 1;$$

(c) Actions of each process P_i.

Figure 4: Synthesis of parameterized Parity on uni-rings.

4 Synthesizing Self-Stabilizing Top-Down Trees

In this section, we investigate the synthesis of parameterized self-stabilizing top-down trees, where each node can read its own state and its parent's. First, we make note that top-down trees are not fully symmetric because the root does not have a parent; every other node does. That is, there are two template processes, one for the root and one for non-root processes. We specify the template process of the non-root processes as $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$, where $R_i = \{x_{pi}, x_i\}$, and x_{pi} denotes the parent's x value. The template process of the root is specified as $\mathcal{P}_{root} = \langle R_{root}, x_{root}, M_i, \delta_{root} \rangle$, where $R_{root} = \{x_{root}\}$ because the root does not have a parent node. Notice that, the root process cannot be enabled by any process. Since processes are self-disabling, once the root process takes an action it will be disabled until it is enabled again by the occurrence of transient faults.

Problem 4.1 (Synthesis of Top-Down Trees)**.** We state the synthesis problem as follows:

- **Input**: $L_{root}(x_{root})$ for the root process, $L_i(x_{pi}, x_i)$ for non-root processes of a top-down tree, $R_i = \{x_{pi}, x_i\}, x_i, M_i$ and an integer $k > 2$. Note that, $M_i = M_{root}$.

- **Output**: The transition functions δ_{root} and δ_i respectively for the root process and the template process of non-root processes (represented as action graphs) such that the entire tree is SS to $\mathcal{I} = \forall j : 1 \leq j \leq N : L_j(x_{pj}, x_j)$ for any tree size $N \geq k$.

Lemma 4.2 (Periodic Propagations in Acyclic Unidirectional Topologies)**.** *In any acyclic unidirectional topology of self-disabling and deterministic processes with constant state space, no periodic propagations exist that lead each other successively/circularly.*

Proof. To prove this lemma, we show that the execution of a process cannot enable its predecessors (similar to what may happen in cyclic topologies like rings). Consider a process P_i. The set of immediate predecessors of P_i includes those processes from which P_i can read and the set of immediate successors of P_i consists of processes that read from P_i. Let $Succ_i$ denote the set that includes any process reachable from P_i in the underlying topology graph of the protocol (i.e., transitive closure of the 'successor' relation). Likewise, let $Pred_i$ represent the set that includes any process from which P_i can be reached (i.e., transitive closure of the 'predecessor' relation). Notice that, in unidirectional topologies the intersection of $Succ_i$ and $Pred_i$ is empty because the topology is acyclic. Due to the self-disabling nature of processes, the actions of a process can only enable its successors. This means the actions of a process cannot generate a wave of enablements that come back to itself. Further, no new values can appear in processes because they have constant state spaces. Therefore, periodic propagations cannot lead each other in a circular fashion. ☐

Lemma 4.3 (Livelock-freedom of Acyclic Unidirectional Topologies)**.** *Any acyclic unidirectional topology of self-disabling and deterministic processes with constant state space is livelock-free.*

Proof. Proof follows from Lemma 4.2 and Theorem 2.16.

□

Legitimacy graphs for top-down trees. The notion of legitimacy graph introduced in Section 2 can be directly used for top-down trees as each non-root process can read only the state of its parent/predecessor and its own.

Theorem 4.4 (Decidable Synthesis for Top-Down Unidirectional Trees)**.** *Let $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$ be the template process of non-root nodes in a top-down unidirectional tree, and the state predicate $\mathcal{I} \overset{\text{def}}{=} (\forall i : L_i(x_{pi}, x_i))$ capture the invariant of the tree. A protocol that stabilizes to $\mathcal{I}$ exists iff the legitimacy graph corresponding to L_i is cyclic.*

Proof. $\Leftarrow$ There are two cases depending on the existence of values that make $L_i(x_{pi}, x_i)$ true. For simplicity, we present this proof in the same spirit as that of Theorem 3.4.

- *Case 1*: If there is a single value $\gamma \in \mathbb{Z}_{M_i}$ that makes $L_i(x_{pi}, x_i)$ true, then the legitimacy graph $G = (V, A)$ must have a self-loop on the vertex corresponding to γ. In this case, the stabilizing protocol includes an action for the root that sets its x value to γ and all other processes will have the action $x_{pi} = \gamma \wedge x_i \neq \gamma \rightarrow x_i := \gamma$.
- *Case 2*: G has no self-loop, but includes a cycle. As such, there must exist a finite sequence of distinct values $v_1, v_2, \cdots, v_k$ such that $\{v_1, v_2, \cdots, v_k\} \in \mathbb{Z}_{M_i}$ and $L_i(v_1, v_2)$, $L_i(v_2, v_3), \cdots, L_i(v_{k-1}, v_k), L_i(v_k, v_1)$ hold, where $k \geq 2$. If $k = M_i$, then it is possible to design an M_i-coloring protocol on the top-down tree, where the root sets its value to v_1 and the subsequent levels of the tree respectively choose the colors $v_2, \cdots, v_{M_i}$ (by assigning $x_{pi} \oplus 1$ to x_i where $\oplus$ denotes addition modulo M_i), and the whole pattern gets repeated, thereby meeting the global invariant $\mathcal{I} = \forall i : L_i(x_{pi}, x_i)$. Even if the state of the tree is perturbed to an arbitrary state, the invariant $\mathcal{I}$ will eventually be met since a wave of stabilization will eventually propagate to all levels from the root. If $1 < k < M_i$, then the length of the cycle is k and some vertices of G do not participate in this cycle. In this case, from each vertex v outside of the cycle, we build a path to some vertex u in the cycle. Afterwards, we create the action graph of the self-stabilizing protocol by an arc-labeling method similar to the one we use in the proof of Theorem 3.4. An alternative approach would use only the k values in the cycle to design a parameterized SS protocol. To elaborate on this, let $V_k = \{c_0, \cdots, c_{k-1}\}$ be the values in the cycle of length k in G. We first assign the action $x_{root} \neq c_0 \rightarrow x_{root} := c_0$ to the root.Then, each non-root process P_i will have the action $x_{pi} = c_j \wedge x_i \neq f(c_j) \rightarrow x_i := f(c_j)$, where $j \in \mathbb{Z}_k$ and f is a permutation function that maps c_j to the next value $c_{j \oplus 1}$ and $\oplus$ denotes addition modulo k.

$\Rightarrow$ We prove the contrapositive of this part and assume that the legitimacy graph G is acyclic. Thus, G has a vertex from where no outgoing arcs exist. That is, there is a value $v \in \mathbb{Z}_{M_i}$ for which there is no value $x \in \mathbb{Z}_{M_i}$ that makes $L_i(v, x)$ true. This means that if a process P_i in the top-down tree takes the value v (due to state perturbations), then there is no way for its children to correct their locality. This will cause a global deadlock in $\neg\mathcal{I}$ because children of P_i cannot recover. Therefore, there is no self-stabilizing solution. □

Observe that, given the state predicate $L_i(x_{pi}, x_i)$ and M_i, the proof of Theorem 4.4 provides a graph-theoretic algorithm (Algorithm 2) for deciding the existence and synthesis of a self-stabilizing protocol for the top-down tree that converges to $\mathcal{I} = \forall i : L_i(x_{pi}, x_i)$ from any state.

Theorem 4.5. *Algorithm 2 is sound and complete. (Proof follows from Theorem 4.4.)*

Algorithm 2 Synthesizing self-stabilizing top-down trees.

```
SynTopDownTrees($L_i(x_{pi}, x_i)$: state predicate, $M_i$: domain size)
Construct the legitimacy graph $G = (V, A)$, where each vertex $v \in V$ represents a value $v$ in $\mathbb{Z}_{M_i}$, and each arc $(v, v')$ captures the fact that $L_i(v, v')$ holds.
If $G$ is acyclic, then return and declare that no solution exists.
If $G$ has a self-loop on some vertex $\gamma \in V$, then include the action $x_{root} \neq \gamma \rightarrow x_{root} := \gamma$ for the root, and the action $x_{pi} = \gamma \wedge x_i \neq \gamma \rightarrow x_i := \gamma$ for non-root nodes. exit;
For a cycle in $G$ on vertices $D_k = \{c_0, \cdots, c_{k-1}\}$ (where $2 \leq k \leq M_i$), design a permutation function $f : D_k \rightarrow D_k$, where $f$ includes an ordered pair $(c_i, c_{i \oplus 1})$ iff there is a corresponding arc $(c_i, c_{i \oplus 1})$ in the cycle. ($\oplus$ denotes addition modulo $k$)
Assign the action $x_{root} \neq c_0 \rightarrow x_{root} := c_0$ to the root, and include the following action in each non-root process $P_i$ which is located in $j \times q$ steps from the root, where $1 \leq j \leq k$ and $q$ is a positive integer: $x_{pi} = c_{j-1} \wedge x_i \neq f(c_{j-1}) \rightarrow x_i := f(c_{j-1})$.
```

(a) Legitimacy graph representing predicate $L_i(x_{pi}, x_i) = ((x_{pi} = x_i))$ where each $x_i \in \mathbb{Z}_2$.

(b) Action graph of self-stabilizing broadcast.

$$(x_{pi} \neq x_i) \longrightarrow x_i := x_{pi};$$

(c) Actions of each process P_i.

Figure 5: Synthesis of parameterized Broadcast on top-down trees.

Theorem 4.6. *The asymptotic time complexity of Algorithm 2 is polynomial (specifically quadratic) in the domain size M_i. (Proof straightforward, hence omitted.)*

Corollary 4.7 (Decidability of Synthesis for Unidirectional Chains)**.** *Let $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$ be the template process of the non-root processes of a unidirectional chain of disabling, constant-space and deterministic processes, and the state predicate $\mathcal{I} \overset{\text{def}}{=} (\forall i : L_i(x_{i-1}, x_i))$ capture its invariant. A protocol that stabilizes to $\mathcal{I}$ exists iff the legitimacy graph corresponding to L_i is cyclic.*

Proof. Each unidirectional chain is a special case of a top-down tree. Proof follows by applying Theorem 4.4. ☐

Example: Broadcast. Consider a top-down tree that forms the spanning tree of the nodes in a network and the root is the leader that broadcasts global information. The root simply casts its vote on an issue by setting its binary variable to 0 or 1. Root's decision is required to be propagated throughout the network; i.e., eventually, every node has the same vote as the root's. Nonetheless, transient faults may perturb the vote of some nodes, thereby making their vote inconsistent with root's. The objective is to design a self-stabilizing protocol that ensures every node will eventually receive the vote of the root. The predicate $L_i(x_{pi}, x_i)$ is defined as $x_i = x_{root}$, however, since each node can just read the state of its parent, we can rewrite L_i as $x_{pi} = x_i$, where x_i are binary variables. This specification of L_i implies $x_i = x_{root}$ whenever the tree stabilizes. Figure 5a illustrates the legitimacy graph of L_i, its action graph and the actions of the self-stabilizing protocol. In this case, the value of γ is actually equal to the root's vote (i.e., Case 1 of Theorem 4.4). As such, the action of every non-root node will be $x_{pi} \neq x_i \rightarrow x_i := x_{pi}$.

Example: 2-coloring. The graph coloring problem has applications in scheduling, register allocation, frequency band allocation, etc. The 2-coloring on a top-down tree uses only 2 colors such that no two neighboring nodes have similar colors. As an application, consider the spanning tree of a sensor network where sensor motes are spread in a field in specific distances. The root of the spanning tree determines how frequency bands are allocated such that no two neighboring nodes have the same carrier frequency (hence

(a) Legitimacy graph representing predicate $L_i(x_{pi}, x_i) = ((x_{pi} \neq x_i))$ where each $x_i \in \mathbb{Z}_2$.

(b) Action graph of self-stabilizing 2-coloring.

$$(x_{pi} = x_i) \longrightarrow x_i := \neg x_{pi};$$

(c) Actions of each process P_i.

Figure 6: Synthesis of parameterized 2-coloring on top-down trees.

avoiding potential for overhearing). We consider a binary variable x for each node of the tree. The value of x signifies which frequency band the node should use. Thus, we have $L_i(x_{pi}, x_i) \overset{\text{def}}{=} (x_{pi} \neq x_i)$. Observe that, in this case there is no γ for which $L_i(\gamma, \gamma)$ holds. However, both $L_i(0, 1)$ and $L_i(1, 0)$ hold[3]. Following the algorithm in the proof of Theorem 4.4, the root has an action that sets its x variable to 0 or 1; e.g., $x_{root} \neq 0 \rightarrow x_{root} := 0$. Every other node will have the following action: $x_{pi} = x_i \rightarrow x_i := \neg x_{pi}$.

5 Synthesizing Self-Stabilizing Bottom-UP Trees

In this section, we discuss the synthesis of parameterized self-stabilizing bottom-up trees. Consider a bottom-up tree topology, processes are the nodes of the tree and each process P_i has a variable x_i. Each node can read its children's and its own x values, and can write only its own x value. Note that, in bottom-up trees the locality of non-leaf nodes may include more than two processes. For simplicity, we investigate the synthesis of binary bottom-up trees. As a result, the local invariant of a process P_i, denoted $L_i(x_{li}, x_i, x_{ri})$, should be specified as a state predicate in terms of its variable x_i and the variables of its left and right children, respectively denoted x_{li} and x_{ri}. The global invariant of the tree is specified as $\mathcal{I} = \forall i :: L_i(x_{li}, x_i, x_{ri})$.

A bottom-up tree is not fully symmetric because the leaves have no children. Thus, we specify the template process of non-leaf processes as $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$, where $R_i = \{x_{li}, x_i, x_{ri}\}$. The template process of leaves is specified as $\mathcal{P}_{leaf} = \langle R_{leaf}, x_{leaf}, M_{leaf}, \delta_{leaf} \rangle$, where $R_{leaf} = x_{leaf}$ and $M_i = M_{leaf}$. Wlog, we consider complete bottom-up binary trees. An incomplete tree can have two types of nodes with less than two children; leaves that are not at the lowest level and nodes with one children. For the first type, we can include dummy nodes as children of leaves that copy the actions of their cousins. If a node has just one child, we consider the child as being both the left and right children. We also make two assumptions about the kind of leaves a bottom-up tree can have: (1) leaves have no actions to correct themselves and cannot be perturbed by transient faults, called shielded/hardened leaves, or (2) each leaf process can have a fixed action that sets its x value to a particular value $c_0 \in \mathbb{Z}_M$ if $x \neq c_0$.

Problem 5.1 (Synthesis of Bottom-UP Trees)**.** We state the synthesis problem as follows:

- **Input**: $L_{leaf}(x_{leaf})$ for the leaf processes, $L_i(x_{li}, x_i, x_{ri})$ for non-leaf processes of a bottom-up tree, $R_i = \{x_{li}, x_i, x_{ri}\}, x_i, M_i$ and an integer $k > 2$. Note that, $M_i = M_{leaf}$.

- **Output**: The transition functions δ_{leaf} and δ_i respectively for the leaf processes and the template process of non-leaf processes (represented as action graphs) such that the entire tree is SS to $\mathcal{I} = \forall j : 1 \leq j \leq N : L_j(x_{lj}, x_j, x_{rj})$ for any tree size $N \geq k$.

Definition 5.2. A binary tree has *left-symmetric* (respectively, *right-symmetric*) leaves if all left (respectively, right) leaves have a symmetric action setting their local x variable to a specific value. Two *symmetric actions* can be obtained from each other by a simple variable renaming/re-indexing.

---

[3] Please see Case 2 in the proof of Theorem 4.4.

Theorem 5.3. *There is a parameterized self-stabilizing protocol for a bottom-up binary tree iff there is a parameterized self-stabilizing protocol for a bottom-up binary tree where leaves have left and right-symmetric actions.*

Proof. Proof of right to left is trivial, hence omitted. Let p be a parameterized self-stabilizing protocol for a bottom-up tree $\mathcal{T}$. Thus, p should self-stabilize no matter what values the leaves have. We simply replace the actions of the leaves of $\mathcal{T}$ in a left and right-symmetric fashion such that left (respectively, right) leaves are set to a specific value c_l (respectively, c_r) if their x value is different from c_l (respectively, c_r). Observe that the resulting protocol will also stabilize because it simulates a special scenario under which p stabilizes. $\square$

Consider a process P_i whose $L_i(x_{li}, x_i, x_{ri})$ is false. For P_i to recover, it should set x_i to some value $c \in \mathbb{Z}_{M_i}$ such that $L_i(x_{li}, c, x_{ri})$ holds. Let $Par(P_i)$ denote the parent of P_i and $L_i(x^p_{li}, x^p_i, x^p_{ri})$ represent the local invariant of $Par(P_i)$. Notice that P_i may be a left child or a right child of $Par(P_i)$, but there is no way for P_i to figure out which child of its parent it is. Wlog, assume that P_i is the left child of $Par(P_i)$. Now, if x_i is set to c, there should be some value $c' \in \mathbb{Z}_{M_i}$ that $Par(P_i)$ can assign to x^p_i such that $L_i(c, c', x^p_{ri})$ holds for any value of x^p_{ri}. This means the decision of each process in the value it chooses to correct its locality affects the ability of its parent node to correct itself. Thus, each process P_i should correct its locality by some value c such that $Par(P_i)$ can also correct its locality regardless of the value that the sibling of P_i takes. Such a reasoning percolates up the tree at all levels, which means the ability of correcting locality must be propagated to all levels of the tree in a circular fashion.

Theorem 5.4. *Let $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$ be the template process of the non-leaf processes of a bottom-up binary tree protocol p with left and right symmetric leaves. p is self-stabilizing iff there exists a set of values $V_k = \{c_0, \cdots, c_{k-1}\} \subseteq \mathbb{Z}_{M_i}$ (where $0 < k \leq M_i - 1$) such that these values circularly satisfy L_i in the following fashion $L_i(c_0, c_1, c_0), L_i(c_1, c_2, c_1), \cdots, L_i(c_{k-2}, c_{k-1}, c_{k-2})$, $L_i(c_{k-1}, c_0, c_{k-1})$.*

Proof. $\Leftarrow$: To design a parameterized SS protocol, we assign a symmetric action $x_i \neq c_0 \rightarrow x_i := c_0$ to all leaves. Since all other processes are self-disabling, they will eventually be disabled and any leaf action that is enabled will be forced to execute. Then, we give each non-leaf process of the tree the action $(x_{li} = x_{ri}) \wedge (x_{ri} = c_j) \wedge x_i \neq c_{j\oplus 1} \rightarrow x_i := c_{j\oplus 1}$, where $c_j \in V_k$ and $\oplus$ denotes addition modulo k. Notice that, while transient faults could make the non-leaf processes disabled (due to $x_{li} \neq x_{ri}$), after faults stop occurring, a correction wave will propagate from the leaves up to the root. Since leaves will eventually execute, L_i will eventually hold for each process P_i.

$\Rightarrow$: Let there be a parameterized SS solution that is symmetric on non-leaf processes of the tree, and left and right symmetric on its leaves. Let Level 0 processes include the leaves. We increment the level number as we move upward. As such, for processes in Level 1, there must be some value c_0 such that $L_i(x_0, c_0, y_0)$, where x_0 and y_0 are respectively the values of the left and right leaves. Notice that, in this proof, x_0 may not necessarily be equal to y_0; however, we use the existence of a value c_0 that makes $L_i(x_0, c_0, y_0)$ true to show that the circular dependency starts at some level, which could have been started from the leaves. The non-existence of c_0 would be in contradiction with the assumption of self-stabilization. Then, the siblings of processes in Level 1 all have the value c_0 due to symmetry. At Level 2, there must be some value y that makes $L_i(c_0, y, c_0)$ true. Now, let y be some value $c_1 \in M_i$. As a result, all processes in Level 2 would take value c_1 due to symmetry. A similar reasoning holds for higher level processes. In the worst case, this reasoning can be repeated M_i times. Due to the pigeon hole principle, in Level $M_i + 1$, the y value that would be selected should be one of the previously used values; otherwise, no value can be assigned to processes in level $M_i + 1$, which is a contradiction with the tree being self-stabilizing. Thus, if there is a symmetric SS solution for non-leaf processes of the tree, then there must be a set of values $\{c_0, \cdots, c_{k-1}\}$, where $1 \leq k \leq M_i - 1$, such that these values circularly satisfy L_i; i.e., the following conditions hold $L_i(c_0, c_1, c_0), L_i(c_1, c_2, c_1), \cdots, L_i(c_{k-2}, c_{k-1}, c_{k-2})$, $L_i(c_{k-1}, c_0, c_{k-1})$. $\square$

Corollary 5.5. *Let $\mathcal{P}_i = \langle R_i, x_i, M_i, \delta_i \rangle$ be the template process of the non-leaf processes of a bottom-up binary tree protocol p. p is self-stabilizing iff there exists a set of values $V_k = \{c_0, \cdots, c_{k-1}\} \subseteq \mathbb{Z}_{M_i}$ (where $0 < k \leq M_i - 1$) such that these values circularly satisfy L_i in the following fashion $L_i(c_0, c_1, c_0), L_i(c_1, c_2, c_1), \cdots, L_i(c_{k-2}, c_{k-1}, c_{k-2})$, $L_i(c_{k-1}, c_0, c_{k-1})$.*

Proof. The proof follows from Theorems 5.3 and 5.4. □

Example: LessThan protocol. Consider a bottom-up tree with an invariant $\mathcal{I} = \forall i : L_i(x_{li}, x_i, x_{ri})$, where $L_i(x_{li}, x_i, x_{ri}) \stackrel{\text{def}}{=} ((x_{li} + x_i) > x_{ri})$ and $M_i = 3$. We apply Theorem 5.4 and look for the set of values V_k. Let $c_0 = 2$. Thus, we should find a value for x in $L_i(2, x, 2)$ such that L_i holds. No value in the domain $\mathbb{Z}_3$ can be substituted for x to make $L_i(2, x, 2)$ true. This means that $L_i(2, x, 2)$ cannot participate in any cyclic satisfaction of L_i. If $c_0 = 0$, then to satisfy $L_i(0, x, 0)$, x can be either 1 or 2 (but not 0). If we assign 2 to x in $L_i(0, x, 0)$, then in the next level, we should satisfy $L_i(2, x, 2)$, which we already showed is impossible. Thus, only 1 can be substituted for x in $L_i(0, x, 0)$. The third possible scenario is where $c_0 = 1$; i.e., $L_i(1, x, 1)$ should be satisfied for some value of x. The only possibility in this case is $x = 1$. We can see that $L_i(1, 1, 1)$ holds, and the nodes in the next level of the tree will have their values equal to 1. Thus, $V_k = \{1\}$ and the cyclic satisfaction of L_i occurs just by propagation of $L_i(1, 1, 1)$. A self-stabilizing protocol would be a left and right symmetric protocol that assigns the action $(x_{li} = x_{ri} = 1) \wedge (x_i \neq 1) \rightarrow x_i := 1$ to all non-leaf nodes, and the action $x_i \neq 1 \rightarrow x_i := 1$ to the leaves.

Example: Maximal Independent Set (MIS). A set of vertices $V' \subseteq V$ in a graph $G = (V, E)$ is independent iff there are no edges between any pair of vertices in V'. Such a set is maximal if no vertex can be added to it without breaking its independence property. The Maximal Independent Set problem has applications in several domains (e.g., scheduling, map labeling, largest correcting code, etc.). Consider the problem of finding an MIS of a bottom-up tree. Depending on the domain of application, a bottom-up tree could represent different problems (e.g., a set of prioritized tasks where the leaves hold higher priority tasks). Let each process P_i have a binary variable x_i such that P_i is in an MIS iff x_i is true. Moreover, a process P_i is in an MIS iff neither of its children are in the MIS of their own subtrees. In other words, any one of the children of P_i is in an MIS iff P_i itself is not in that MIS. This in turn means that the local invariant for each process P_i is $L_i(x_{li}, x_i, x_{ri}) \equiv (\neg x_i \Leftrightarrow x_{li} \vee x_{ri})$. Now, applying Theorem 5.4, we can find the following cycle: $L_i(true, false, true)$, $L_i(false, true, false)$, which implies $V_k = \{true, false\}$. The resulting protocol will have the action $x \neq true \rightarrow x := true$ for leaves, and the action $x_i = (x_{li} \vee x_{ri}) \rightarrow x_i := \neg(x_{li} \vee x_{ri})$ for non-leaf processes.

Example: Min/Max protocol. Consider a protocol in a bottom-up tree where the global invariant includes states where the root of the tree includes the minimum of all values in the tree. (A symmetric protocol can be considered for the maximum value.) The objective is to design a self-stabilizing protocol that works for any tree size. Each process has a variable x with a domain of modulo M_i (i.e., $\mathbb{Z}_{M_i}$). The local invariant of each node states that $L_i(x_{li}, x_i, x_{ri}) \stackrel{\text{def}}{=} (x_i = Min(x_{li}, x_i, x_{ri}))$, where Min is a function that returns the minimum of three values. Formally, the global invariant is $\mathcal{I} = \forall i : L_i(x_{li}, x_i, x_{ri})$. Now, using Theorem 5.4, we look for a set $V_k \subseteq \mathbb{Z}_{M_i}$ that includes values that circularly satisfy L_i. For $L_i(0, x, y)$ to hold, x must be set to 0, where $y \in \mathbb{Z}_{M_i}$. This would result in ensuring that $L_i(0, x, y)$ and $L_i(y, x, 0)$ hold at the next level of the tree. Thus, in this case $V_k = \{0\}$, which implies the existence of a left-right symmetric solution. The action $(x_i \neq 0) \rightarrow x_i := 0$ for leaves, and the parameterized action $(x_i \neq Min(x_{li}, x_i, x_{ri})) \rightarrow x_i := Min(x_{li}, x_i, x_{ri})$ for non-leaf processes would provide us a self-stabilizing protocol.

5.1 Synthesis Algorithm

This section presents an algorithm for synthesizing self-stabilizing parameterized protocols in bottom-up binary trees. Specifically, the objective is to determine if for a specific predicate $L_i(x_{li}, x_i, x_{ri})$ a parameterized self-stabilizing protocol exists for a bottom-up tree. If such a protocol exists, we generate its action graph. We first extend the notion of legitimacy graphs for bottom-up trees. Notice that, the semantics of vertices and arcs of the legitimacy graph might differ from one topology to another. Then, we provide a graph-theoretic characterization of Theorem 5.4.

Definition 5.6 (Legitimacy graph for bottom-up trees)**.** Let $G = (V, A)$ be the legitimacy graph corresponding to $L_i(x_{li}, x_i, x_{ri})$ for a parameterized bottom-up tree. A vertex $s \in V$ captures a local legitimate state of the template process of the non-leaf nodes of a bottom-up tree; i.e., L_i holds in s. An arc (s, s')

connects two legitimate states s and s' in G *iff* (1) s' is a state of the parent of a process in state s, and (2) $x'_l(s') = x'_r(s')$.

The second constraint is enforced by Theorem 5.4 as we are looking for a sequence of values that propagate through the tree in a cyclic fashion. Figure 7-(a) illustrates the legitimacy graph of the LessThan protocol presented in this section. If a process P_i in the bottom-up tree is in a legitimate state $s = \langle 2, 1, 0 \rangle$, then we connect s to a legitimate state $s' = \langle x'_l, x', x'_r \rangle$, where $x'_l = x'_r = 1$. The only legitimate states of the LessThan protocol that meet the condition $x'_l = x'_r = 1$ include $\langle 1, 1, 1 \rangle$ and $\langle 1, 2, 1 \rangle$. Thus, we include two arcs from $\langle 2, 1, 0 \rangle$ to $\langle 1, 1, 1 \rangle$ and $\langle 1, 2, 1 \rangle$. The gray states in Figure 7-(a) are the ones that have no outgoing arcs; i.e., deadlock states. Such states represent the locality of a process whose parent cannot correct its own local state under Constraint (2) of Definition 5.6. Figure 7-(b) demonstrates a subgraph of the legitimacy graph G, denoted G', that excludes any arc reaching a deadlock state. Now, the interpretation of Theorem 5.4 in the context of the legitimacy graph is as follows:

Corollary 5.7. *A parameterized protocol exists for a bottom-up binary tree with left and right symmetric leaves, and symmetric non-leaf processes that self-stabilizes to* $\mathcal{I} \stackrel{\text{def}}{=} \forall i :: L_i(x_{li}, x_i, x_{ri})$ *iff the deadlock-free legitimacy graph* G' *of* L_i *has a simple cycle.*

Notice how legitimacy graphs simplify reasoning about global behaviors in comparison with the proof of Theorem 5.4. The following theorem provides a sufficient condition for unsolvability.

Theorem 5.8. *For a parameterized bottom-up binary tree and a predicate* $\mathcal{I} \stackrel{\text{def}}{=} (\forall i :: L_i(x_{li}, x_i, x_{ri}))$*, if* L_i *includes a conjunct that is specified only in terms of* x_{li} *and* x_{ri}*, then no protocol that stabilizes to* $\mathcal{I}$ *exists.*

Proof. Let $L_i \stackrel{\text{def}}{=} X \wedge C_i(x_{li}, x_{ri})$, where $C_i(x_{li}, x_{ri})$ is a predicate specified only in terms of x_{li} and x_{ri}. For a process P_i to correct its locality when L_i is false, P_i should ensure that $C_i(x_{li}, x_{ri})$ holds too. Since P_i can write only x_i, it has no way to update variables x_{li} and x_{ri}. Moreover, the children of P_i cannot read/write each other's state. □

For example, let $L_i \stackrel{\text{def}}{=} X \wedge (x_{li} \neq x_{ri})$. In this case, we have $C_i(x_{li}, x_{ri}) = (x_{li} \neq x_{ri})$. Obviously, if $C_i(x_{li}, x_{ri})$ is false (i.e., $(x_{li} = x_{ri})$), then process P_i can detect it but cannot take any action to ensure $C_i(x_{li}, x_{ri})$ becomes true; nor can any one of P_i's children.

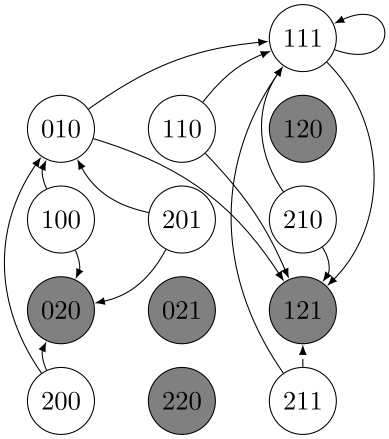
Example: 2-coloring. Consider the case where we design a 2-coloring self-stabilizing protocol on a complete bottom-up tree. The objective of a 2-coloring protocol is to ensure stabilization to a state where the entire tree has been colored by two colors in such a way that the color of each process differs from that of its parent. Formally, we have $L_i \stackrel{\text{def}}{=} ((x_i \neq x_{li}) \wedge (x_i \neq x_{ri}))$. Since x_i are binary variables, the inequality of x_i to x_{li} and x_{ri} implies $x_{li} = x_{ri}$. First, we create the legitimacy graph of this protocol (see Figure 8) to determine if a solution exists at all. Notice that there are only two legitimate states for which L_i holds. Corollary 5.7 implies that a 2-coloring self-stabilizing solution exists. Observe that, for 2-coloring on a bottom-up tree, if the leaves are not symmetric, then no solution exists. For instance, if two sibling leaves take different values, then there is no value that their parent can take towards satisfying the constraint $((x_i \neq x_{li}) \wedge (x_i \neq x_{ri}))$. This means that in the case of 2-coloring on a bottom-up tree, no solution exists if the leaves are not symmetric.

Synthesizing a protocol. In order to synthesize a self-stabilizing protocol on a bottom-up binary tree, we present Algorithm 3. The input to the algorithm includes L_i and the domain size of x_i, and the output contains the parametric actions of the template processes. Due to its simplicity, we present this algorithm in plain English as a stepwise process.

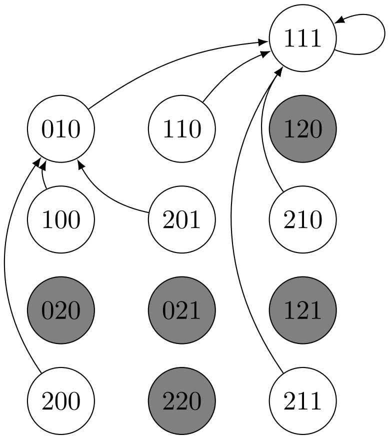
Theorem 5.9. *Algorithm 3 is sound and complete. (Proof follows from Theorem 5.4.)*

Theorem 5.10. *The asymptotic time complexity of Algorithm 3 is polynomial in* M_i^{b+1}*, where* M_i *is the domain size and b denotes the branching factor of the bottom-up tree. (Proof follows from Theorem 5.4.)*

Proof. Every step of Algorithm 3 takes polynomial time in the size of the legitimacy graph of the bottom-up tree. However, the size of the legitimacy graph in this case depends on the maximum number of children; i.e., the branching factor of the tree. The deadlock-free legitimacy graph can have at most M_i^{b+1} vertices. □

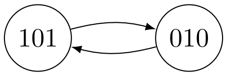


(a) Legitimacy graph for predicate $L_i(x_{li}, x_i, x_{ri}) = ((x_{li} + x_i) > x_{ri})$ where each $x_i \in \mathbb{Z}_3$. Values "*abc*" in states represent a local state where $x_{li} = a, x_i = b, x_{ri} = c$.



(b) Legitimacy graph after eliminating arcs that reach deadlocks.

Figure 7: Legitimacy graph of the LessThan protocol on a bottom-up binary tree.



(a) Legitimacy graph for predicate $L_i(x_{li}, x_i, x_{ri}) \overset{\text{def}}{=} ((x_i \neq x_{li}) \wedge (x_i \neq x_{ri}))$ where each $x_i \in \mathbb{Z}_2$.

Figure 8: Legitimacy graph of the 2-coloring protocol on a bottom-up binary tree.

Algorithm 3 Synthesizing self-stabilizing bottom-up trees.

```
SynBottomUpTrees($L_i(x_{li}, x_i, x_{ri})$: state predicate, $M_i$: domain size)
1: Construct the deadlock-free legitimacy graph $G' = (V, A)$, where each vertex $s \in V$ represents a legitimate state that satisfies $L_i$, and each arc $(s, s')$ captures the possibility of a parent process being in the legitimate state $s'$ while its child is in the legitimate state $s$.
2: If there are no simple cycles in $G'$, then return and declare that no solution exists.
3: Consider one of the simple cycles of $G'$.
4: Select a state $\langle a', c', b' \rangle$ in the cycle.
5: Extract the left and right symmetric actions of the leaves out of $\langle a', c', b' \rangle$ as follows:
       Action assigned to left leaves: $x_i \neq a' \rightarrow x_i := a'$.
       Action assigned to right leaves: $x_i \neq b' \rightarrow x_i := b'$.
6: For each arc to a state $\langle a, c, b \rangle$ in the cycle, consider the action $x_{li} = a \wedge x_{ri} = b \wedge x_i \neq c \rightarrow x_i := c$.
```

Examples. Using Algorithm 3, we synthesize the following parameterized actions for the 2-coloring protocol on the bottom-up tree:

- Use action $x_i \neq 0 \rightarrow x_i := 0$ (respectively, $x_i \neq 1 \rightarrow x_i := 1$) for all leaves.
- The actions of each non-leaf node of the tree are as follows: $(x_{li} = 1) \wedge (x_{ri} = 1) \wedge (x_i \neq 0) \rightarrow x_i := 0$ and $(x_{li} = 0) \wedge (x_{ri} = 0) \wedge (x_i \neq 1) \rightarrow x_i := 1$.

In the case of the LessThan protocol, we have only one simple cycle as a self-loop on the state $\langle 1, 1, 1 \rangle$ in Figure 7-(b). Applying the proposed synthesis algorithm to this cycle, we get the following actions:

- All leaves have the action $x_i \neq 1 \rightarrow x_i := 1$.
- Each non-leaf node of the tree has the action $(x_{li} = 1) \wedge (x_{ri} = 1) \wedge (x_i \neq 1) \rightarrow x_i := 1$.

6 Undecidability of Synthesizing Bidirectional Rings

While synthesizing parameterized self-stabilizing protocols is decidable for uni-rings, we show that synthesis is undecidable for bidirectional rings.

Theorem 6.1. *Let $\mathcal{I} \overset{\text{def}}{=} (\forall i : L(x_{i-1}, x_i, x_{i+1}))$ be an invariant for a bi-directional ring, where each process P_i can read the variables of its left and right neighbors; i.e., $R_i = \{x_{i-1}, x_i, x_{i+1}\}$. It is undecidable whether there is a parameterized symmetric protocol p that is self-stabilizing to $\mathcal{I}$.*

Proof. To show undecidability, we reduce the problem of verifying livelock freedom of a uni-ring protocol p to the problem of synthesizing a bidirectional ring protocol p' that stabilizes to $\mathcal{I}'$, where $\mathcal{I}'$ has some form determined by p. We construct $\mathcal{I}'$ such that exactly one bidirectional ring protocol p' resolves all deadlocks without breaking closure, but it only stabilizes to $\mathcal{I}'$ if p is livelock-free. Thus, p' is the only candidate solution for the synthesis procedure, and the synthesis succeeds *iff* p is livelock-free. Our reduction is broken into two parts: (1) showing that exactly one particular p' resolves all deadlocks without breaking closure, and (2) showing that p' is livelock-free *iff* p is livelock-free.

Assumptions about p. We assume that p (1) has a deterministic livelock that (2) involves all actions and (3) includes all values. These assumptions do not affect the undecidability of verifying livelock freedom in p. First, by Theorem 2.18, deterministic livelock detection is undecidable in uni-rings. Second, deterministic livelock detection remains undecidable when the livelock involves all actions; otherwise, we could detect deterministic livelocks by checking each subset of actions. Third, deterministic livelock detection is undecidable even when the livelock involves all values; otherwise, we could detect deterministic livelocks by checking each subset of values. Thus, verifying livelock-freedom under our assumptions for p remains undecidable.

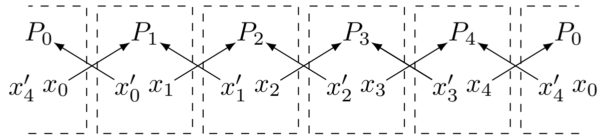


Figure 9: Topology for bidirectional ring protocol p' in Theorem 6.1. Each process P_i owns x'_{i-1} and x_i.

Forming $\mathcal{I}'$ from p. To form $\mathcal{I}'$, we augment each process P_i with a new variable $x'_{i-1} \in \mathbb{Z}_{M_i}$, which is a local copy of x_{i-1}, along with its $x_i \in \mathbb{Z}_{M_i}$, making its effective domain size $M'_i \overset{\text{def}}{=} M_i^2$. Since p' is a bidirectional ring, P_i can read x_{i-1} and x'_{i-2} from P_{i-1} and can read x_{i+1} and x'_i from P_{i+1}. For each action $(a, b, c) \in \delta_i$, we use $x_{i-1} = a$ and $x'_i = b$ to encode the precondition of a P_i action (a, b, c), and $x_i = c$ to encode its assignment. Notice that, δ denotes the transition function of p, and x'_i is from P_{i+1} as depicted in Figure 9 (for an example ring of 5 processes). Thus, we must ensure that x'_i eventually obtains a copy of x_i. The resulting $\mathcal{I}' \overset{\text{def}}{=} (\forall i : L'_i(x_{i-1}, x_i))$ is as follows with instances of x_i replaced with x'_i and a condition that x'_{i-1} is a copy of x_{i-1}.

$$\begin{aligned} L'_i(x_{i-1}, x_i) \overset{\text{def}}{=} \big(& (x_{i-1}, x'_i) \in \mathsf{Pre}(\delta) \\ & \implies x'_{i-1} = x_{i-1} \wedge x_i = \delta(x_{i-1}, x'_i) \big) \end{aligned}$$

Forming p' and δ'_i from $\mathcal{I}'$. We want to show that a particular p' stabilizes to $\mathcal{I}'$ when p is livelock-free, and it is the only bidirectional ring protocol that resolves deadlocks without breaking closure. This p' has the following action for each P_i.

$$\begin{aligned} & (x_{i-1}, x'_i) \in \mathsf{Pre}(\delta) \wedge \big(x'_{i-1} \neq x_{i-1} \vee x_i \neq \delta(x_{i-1}, x'_i) \big) \\ & \longrightarrow \quad x'_{i-1} := x_{i-1};\ \ x_i := \delta(x_{i-1}, x'_i); \end{aligned}$$

Notice that p' is deadlock-free and preserves closure since a process P_i can act *iff* its $L'_i(x_{i-1}, x_i)$ is unsatisfied. We now show that this p' is the only such protocol. That is, each process P_i of p' must have the above action to ensure $x'_{i-1} = x_{i-1}$ and $x_i = \delta(x_{i-1}, x'_i)$ when $(x_{i-1}, x'_i) \in \mathsf{Pre}(\delta)$. To this end, we show that if there is only one process enabled in the entire ring, that process must execute an action as above. Our proof strategy is based on picking values for variables to make the neighboring processes of a specific process disabled. Consider a process P_j in a ring of N processes, and let its readable variables from P_{j-1} and P_{j+1} have arbitrary values. By our earlier assumptions about p, P_j has an action (a, b, c) for any given a or c (not both), and $(a, c) \notin \mathsf{Pre}(\delta)$ because processes of p are self-disabling. Thus, we can choose x_{j-2} of P_{j-2} to make $(x_{j-2}, x'_{j-1}) \notin \mathsf{Pre}(\delta)$ for P_{j-1}, and we can choose x'_{j+1} of P_{j+2} to make $(x_j, x'_{j+1}) \notin \mathsf{Pre}(\delta)$ for P_{j+1}. We have satisfied L'_{j-1} and L'_{j+1}, and we can likewise satisfy L'_{j-2} and L'_{j+2} by choosing values of x_{j+2} and x'_{j+2} respectively. By a similar method, we can ensure that any other process P_k $(k \neq j)$ in the ring has L'_k satisfied. Thus, p' is in a legitimate state *iff* L'_j is satisfied. Therefore, if L'_j is satisfied, then P_j cannot act without adding a transition within I' (i.e., breaking closure). As a consequence, no other process but P_j can act if L'_j is not satisfied. Since processes are symmetric, each P_k of p' must have the above action to ensure $x'_{k-1} = x_{k-1}$ and $x_k = \delta(x_{k-1}, x'_k)$ when $(x_{k-1}, x'_k) \in \mathsf{Pre}(\delta)$.

If p has a livelock, then p' has a livelock. Assume p has a livelock. We show that p' has a livelock too. We prove this by showing that p' can simulate the livelock of p. By assumption, p has a deterministic livelock from some state $C = (c_0, \ldots, c_{N-1})$ on a ring of size N where only the first process is enabled; i.e., $(c_{i-1}, c_i) \in \mathsf{Pre}(\delta)$ only for $i = 0$. Let $C' = (c'_0, \ldots, c'_{N-1})$ be the state of this system after all processes act once. That is, $c'_0 = \delta(c_{N-1}, c_0)$ and $c'_i = \delta(c'_{i-1}, c_i)$ for all other $i > 0$. We can construct a livelock state of p' from the same $x_i = c_i$ values for all i and $x'_i = c_i$ for all $i < N - 1$. The value of x'_{N-1} can be c_{N-1}, but can be anything else such that $(x_{N-2}, x'_{N-1}) \notin \mathsf{Pre}(\delta)$. In this state of p', only P_0 is enabled since we assumed that $(c_{i-1}, c_i) \in \mathsf{Pre}(\delta)$ only holds for $i = 0$. P_0 then performs $x_0 := c'_0$ and $x'_{N-1} := c_{N-1}$. This does not enable P_{N-1}, but does enable P_1 to perform $x_1 := c'_1$ and $x'_0 := c'_0$. The execution continues for

$P_2, \ldots, P_{N-1}$ to assign $x_i := c'_i$ and $x'_{i-1} := c'_{i-1}$ for all $i > 1$. At this point the system is in a state where $x_i = c'_i$ for all i and $x'_i = c'_i$ for all $i < N-1$. The value of x'_{N-1} is c_{N-1}, which leaves it disabled. This state of p' matches the state C' of p using the same constraints as we used to match the initial state C. Therefore, p' can continue to simulate p, showing that it has a livelock.

If p is livelock-free, then p' is livelock-free. Assume p is livelock-free. We show that p' is livelock-free too. First, notice that if P_{i+1} acts immediately after P_i in p', then P_i will not become enabled because $x_i = x'_i$ and self-disabling processes of p ensure that $(a, c) \notin \mathsf{Pre}(\delta)$ for every action (a, b, c). This means that in a livelock, if an action of P_{i+1} enables P_i, then P_{i-1} must have acted since the last action of P_i. As such, an action of P_{i-1} must occur between every two actions of P_i in a livelock of p'. The number of such propagations clearly cannot increase, and thus must remain constant in a livelock. In order to avoid collisions, an action of P_{i+1} must occur between every two actions of P_i. Since P_{i+1} always acts before P_i in a livelock of p', it ensures that $x'_i = x_i$ when P_i acts. By making this substitution, we see that P_i is only enabled when $(x_{i-1}, x_i) \in \mathsf{Pre}(\delta)$, and assigns $x_i := \delta(x_{i-1}, x_i)$, which is equivalent to the behavior of protocol p. Since p is livelock-free, p' must also be livelock-free. Thus, p is livelock-free iff p' is livelock-free. Therefore, synthesizing stabilization on bidirectional rings is undecidable. □

7 Experimental Results

This section presents our experimental results on automatic synthesis of several self-stabilizing parameterized uni-rings. We have integrated Algorithm 1 in a framework for automated synthesis of SS systems available at `http://asd.cs.mtu.edu/projects/protocon/`. The platform of experiments is a regular MacBook Air laptop with an Intel Core i7 2.2 GHz processor, 8 GB RAM and OS X El Capitan 10.11.6. For the examples in this section, we first present $L_i(x_{i-1}, x_i)$ and the domain M_i of x_i as they are the main inputs to our synthesis tool. We also re-run the synthesis for domain sizes in the range of 2 to 11; i.e., $2 \leq M_i \leq 11$ to study the impact of domain size on the time efficiency of synthesis. Figure 10 illustrates how synthesis time grows as we increase the domain size from 2 to 11 (see the horizontal axis). The vertical axis represents the average synthesis time over 1000 runs.

Agreement. Agreement is a fundamental problem in distributed computing where processes in a network should agree on a specific value. Achieving agreement becomes more difficult in the presence of transient faults where the values of processes can be perturbed arbitrarily. For the processes in a uni-ring to agree on the same value, we specify the global invariant as $\forall i : i \in \mathbb{Z}_N : L_i(x_{i-1}, x_i)$, where N denotes the number of processes and $L_i(x_{i-1}, x_i) \stackrel{\text{def}}{=} (x_{i-1} = x_i)$. The synthesized action for the agreement protocol for rings of size $N > 2$ is $(x_{i-1} \neq x_i) \wedge (x_i \neq 0) \rightarrow x_i := 0$. (See Figure 10 for average synthesis time.)

Odd Parity. The Parity protocol in Section 3.1 ensures the adoption of a common parity (odd or even) in uni-rings. We can strengthen its invariant and require odd parity in the ring; i.e., $L_i(x_{i-1}, x_i) \stackrel{\text{def}}{=} (((x_{i-1} - x_i) \ \mathtt{mod}\ 2) = 0) \wedge (x_i \ \mathtt{mod}\ 2 \neq 0)$. The resulting synthesized actions for uni-rings of size $N > 2$ are as follows:

$$\begin{array}{ll}(x_i \ \mathtt{mod}\ 2 = 0) & \rightarrow x_i := 1; \\ (x_{i-1} \ \mathtt{mod}\ 2 = 0) \wedge (x_i \ \mathtt{mod}\ 2 \neq 0) \wedge (x_i \neq 1) & \rightarrow x_i := 1;\end{array}$$

Sorting. Recovery to a global configuration where the values of processes adhere to the constraints of the sorting problem (a.k.a. sorted configuration) has applications in several distributed algorithms such as distributed hashing. On a ring though the first and the last processes are neighbors and this can impact recovery to a sorted configuration. To investigate this, we specify $L_i(x_{i-1}, x_i) \stackrel{\text{def}}{=} (x_{i-1} \leq x_i)$, and automatically synthesize the following action: $(x_{i-1} > x_i) \wedge (x_i \neq 0) \rightarrow x_i := 0$ for ring sizes $N > 2$.

SumNotThree. We extend the SumNotTwo protocol of Section 2 to SumNotThree, where $L_i(x_{i-1}, x_i) \stackrel{\text{def}}{=} ((x_{i-1} + x_i) \ \mathtt{mod}\ M_i) \neq 3$. We synthesize this protocol for $4 \leq M_i \leq 11$ because if $M_i = 3$, then $3 \notin \mathbb{Z}_{M_i}$.

$$\begin{array}{ll}(x_{i-1} = 3) \wedge (x_i = 0) & \rightarrow x_i := 1; \\ ((x_{i-1} + x_i) \ \mathtt{mod}\ M_i = 3) \wedge (x_i \neq 0) & \rightarrow x_i := 0;\end{array}$$

SumNotOdd and SumNotEven. To study the general case of SumNotTwo and SumNotThree protocols, we investigate the cases where the summation of the x values of two neighboring processes must not be odd (respectively, even). That is, $L_i(x_{i-1}, x_i) \overset{\text{def}}{=} (((x_{i-1} + x_i) \texttt{ mod } M_i) \texttt{ mod } 2 = 0)$ (respectively, $L_i(x_{i-1}, x_i) \overset{\text{def}}{=} (((x_{i-1} + x_i) \texttt{ mod } M_i) \texttt{ mod } 2 \neq 0)$). For the SumNotOdd protocol, we synthesize the following action for the case where M_i is odd.

$$(((x_{i-1} + x_i) \texttt{ mod } M_i) \texttt{ mod } 2) \neq 0 \ \rightarrow x_i := (M_i - x_{i-1}) \texttt{ mod } M_i;$$

If M_i is even, we automatically synthesize the following action:

$$((((x_{i-1} + x_i) \texttt{ mod } M_i) \texttt{ mod } 2) \neq 0) \wedge (x_i \neq 0) \ \rightarrow x_i := 0;$$

In the case of the SumNotEven protocol, there are no solutions for cases where M_i is even because there is no $\gamma \in M_i$ for which $L_i(\gamma, \gamma)$ holds.

Summary. First, we would like to emphasize that average synthesis time for SS uni-rings is in the scale of micro seconds, which is the most efficient to the best of our knowledge. Second, while the asymptotic time complexity of Algorithm 1 is quadratic (in domain size), in our case studies, the average synthesis time increases almost linearly.

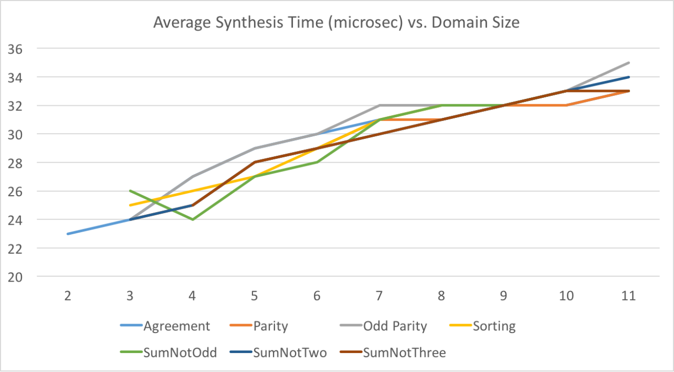


Figure 10: Average synthesis time vs. domain size.

8 Related Work

Most existing approaches [16, 8, 23, 11, 31, 5] for the synthesis of Parameterized Systems (PSs) synthesize from temporal logic specifications and/or make assumptions about synchrony, fairness and complete knowledge of the network for each process. Moreover, most existing methods focus on synthesis for either safety properties or local liveness properties (e.g., progress of a thread); they do not address self-stabilization under asynchronous semantics with no fairness where convergence (i.e., recovery from *any* state) should be achieved through the collaboration of all processes. A different line of work [30] focuses on sketch-based synthesis of fault-tolerant distributed algorithms, where designers provide the control flow/structure of processes as a sketch automaton. The transitions of the sketch automaton are guarded by conditions that contain unknown parameters. Then, they generate the values of threshold parameters using a counterexample-guided refinement method such that specific safety/liveness properties are met. By contrast, the proposed approach in this paper generates the entire control structure of the processes of a parameterized protocol. While the actions of processes in our model lack explicit threshold guards, such kind of guards can be captured as state predicates specified on the locality of each process. The closest work to ours includes r-operators for self-stabilization [9] where the authors present an algebraic method for the design of SS protocols that compute static tasks (e.g., shortest path from a specific process, DFS trees, etc.) and are parameterized in terms of the type of the operators used in each process. Nonetheless, their approach differs from ours in several directions. First, they consider a message passing model of computation on arbitrary topologies. Second, in their proof of correctness they assume global weak fairness. Third, they assume some degree of synchrony implemented by time-out events that trigger each process for execution, whereas our approach is fully asynchronous. Fourth, r-operators define a total order over variable domains. Such total orders along with time-outs ensure livelock-freedom. Finally, their approach is mostly geared towards value-based problems where the legitimate state of each process is determined by the final value it computes (e.g., its distance from source). By contrast, our approach is more general in that a state predicate must hold in the locality/neighborhood of each process.

There is a rich body of work on the verification of PSs whose objective is to take an existing design of a PS and verify if some safety/liveness properties hold for the PS. Such verification methods can hardly be used for our purpose due to the requirements that (1) convergence must be met from *any* state and not just a proper subset of the state space, (2) convergence is a global liveness property rather than local liveness properties, and (3) convergence should be synthesized rather than verified after the fact. Nonetheless, we discuss their relevance to our work as follows. Techniques for the verification of PSs can be classified into several major methods. **Abstraction methods** [4, 22, 35, 12] generate a finite-state model of a PS and then reduce the verification of the PS to the verification of its finite model. **SMT-based verification** [18, 7] is an example of such abstraction methods where SMT solvers are used to verify safety and inclusion properties in a reachability analysis phase. **Parameterized Visual Diagrams** (PVDs) [39] model a PS and its required properties in terms of visual abstractions (e.g., predicate automata); however, they assume weak fairness and generate a large number of verification conditions that should be verified by model checking. **Network invariant** approaches [43, 24, 21] find a process that satisfies the property of interest and is invariant to parallel composition; i.e., composing it with itself for an arbitrary number of times will create a system that still satisfies the property of interest. The network invariant method is mostly used for the verification of safety properties, whereas self-stabilization includes a global liveness property, namely convergence. Methods for **compositional model checking** of PSs (e.g., cache coherence [33]) use abstract interpretation to reduce the verification of unbounded systems to finite-state model checking of a set of local temporal properties. Such abstractions are too coarse for synthesizing self-stabilization because an SS system must guarantee convergence from each concrete state. **Logic program transformations** and inductive verification methods [36, 37, 38, 17] encode the verification of a PS as a constraint logic program and verify the equivalence of goals in the logic program. In **regular model checking** [6, 40, 1], system states are represented by grammars over strings of arbitrary length, and a protocol is represented by a transducer. **Proof spaces** [15] enable a novel method for automated extraction of Hoare triples for unbounded multi-threaded programs, where these verification conditions are used in a deductive reasoning system. **Neo** [32] uses network invariants to identify architectures [32] with special topologies (e.g., trees) for which safety properties are verifiable. Neo's topology-specific verification has similarities to our topology-specific synthesis method; nonetheless, the focus of this project is on synthesis rather then verification.

9 Conclusions and Future Work

In this paper, we investigated the problem of synthesizing parameterized systems that have the property of self-stabilization. The system components/processes are deterministic and have constant state space. Moreover, we consider self-disabling processes, where a process disables itself after executing an action until it is enabled again by the actions of other processes (or by the occurrence of faults). While it is known that verifying self-stabilization of unidirectional rings is undecidable [26], in this paper, we present a surprising result that synthesizing self-stabilizing unidirectional rings is actually decidable. The intuition behind this counterintuitive result is that, during synthesis, the existence of a simple solution (which can be found algorithmically) is necessary and sufficient for the existence of self-stabilizing solutions, in general. However, in the case of verification of self-stabilization, the verifier must examine an intractable number of scenarios. We introduce the notion of legitimacy graphs and action graphs that greatly simplify local reasoning about global properties of parameterized systems. We also present a family of sound and complete algorithms for the synthesis of self-stabilizing parameterized protocols in unidirectional topologies (e.g., uni-rings, chains, top-down and bottom-up trees), and apply our algorithms to a few case studies. We have integrated our algorithm for the synthesis of symmetric uni-rings in Protocon (`http://asd.cs.mtu.edu/projects/protocon/`), and our experimental results demonstrate the extraordinary time efficiency of our method (in the scale of a few tens of microseconds). Further, we show that the synthesis of parameterized rings becomes undecidable if we assume bidirectional rings. Our results hold for the interleaving execution semantics and under no fairness. As an extension to this work, we are investigating rules of composition where one can compose two or more self-stabilizing parameterized systems with elementary topologies (e.g., uni-rings, chains and trees) to generate more complicated topologies while preserving stabilization.

References

[1] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *CONCUR*, pages 35–48, 2004.

[2] A. Arora and S. S. Kulkarni. Detectors and correctors: A theory of fault-tolerance components. *International Conference on Distributed Computing Systems*, pages 436–443, May 1998.

[3] S. Bernard, S. Devismes, M. G. Potop-Butucaru, and S. Tixeuil. Optimal deterministic self-stabilizing vertex coloring in unidirectional anonymous networks. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*, pages 1–8. IEEE, 2009.

[4] G. Bhat, R. Cleaveland, and O. Grumberg. Efficient on-the-fly model checking for ctl*. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 388–397, 1995.

[5] R. Bloem, N. Braud-Santoni, and S. Jacobs. Synthesis of self-stabilising and Byzantine-resilient distributed systems. In *International Conference on Computer Aided Verification*, pages 157–176. Springer, 2016.

[6] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV*, pages 403–418, 2000.

[7] S. Conchon, A. Goel, S. Krstic, A. Mebsout, and F. Zaıdi. Cubicle: A parallel SMT-based model checker for parameterized systems. In *CAV*, pages 718–724. Springer, 2012.

[8] L. De Moura and N. Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.

[9] S. Delaët, B. Ducourthial, and S. Tixeuil. Self-stabilization with r-operators revisited. *Journal of Aerospace Computing, Information, and Communication*, 3(10):498–514, 2006.

[10] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[11] D. Dolev, J. H. Korhonen, C. Lenzen, J. Rybicki, and J. Suomela. Synchronous counting and computational algorithm design. In *Symposium on Self-Stabilizing Systems*, pages 237–250. Springer, 2013.

[12] E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *International Journal of Foundations of Computer Science*, 14(4):527–550, 2003.

[13] A. Farahat. *Automated Design of Self-Stabilization.* PhD thesis, Michigan Technological University, July 2012.

[14] A. Farahat and A. Ebnenasir. Local reasoning for global convergence of parameterized rings. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 496–505, 2012.

[15] A. Farzan, Z. Kincaid, and A. Podelski. Proving liveness of parameterized programs. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 185–196, 2016.

[16] B. Finkbeiner and S. Schewe. Bounded synthesis. *International Journal on Software Tools for Technology Transfer*, 15(5-6):519–539, 2013.

[17] F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni. Generalization strategies for the verification of infinite state systems. *TPLP*, 13(2):175–199, 2013.

[18] S. Ghilardi and S. Ranise. *MCMT: A Model Checker Modulo Theories*, pages 22–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[19] M. Gouda. The theory of weak stabilization. In *5th International Workshop on Self-Stabilizing Systems*, volume 2194 of *Lecture Notes in Computer Science*, pages 114–123, 2001.

[20] M. G. Gouda and F. F. Haddix. The stabilizing token ring in three bits. *Journal of Parallel and Distributed Computing*, 35(1):43–48, May 1996.

[21] O. Grinchtein, M. Leucker, and N. Piterman. Inferring network invariants automatically. In *Automated Reasoning*, pages 483–497. 2006.

[22] C. N. Ip and D. L. Dill. Verifying systems with replicated components in murphi. *Formal Methods in System Design*, 14(3):273–310, 1999.

[23] S. Jacobs and R. Bloem. Parameterized synthesis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 362–376. Springer, 2012.

[24] Y. Kesten, A. Pnueli, E. Shahar, and L. Zuck. Network invariants in action. In *Concurrency Theory*, pages 101–115. Springer, 2002.

[25] A. Klinkhamer and A. Ebnenasir. A software tool for swarm synthesis of self-stabilization. `http://asd.cs.mtu.edu/projects/protocon/`.

[26] A. Klinkhamer and A. Ebnenasir. Verifying livelock freedom on parameterized rings and chains. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 163–177, 2013.

[27] A. Klinkhamer and A. Ebnenasir. On the hardness of adding nonmasking fault tolerance. *IEEE Transactions on Dependable and Secure Computing*, 12(3):338–350, May-June 2015.

[28] 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, Feb. 2016.

[29] S. S. Kulkarni and A. Arora. Automating the addition of fault-tolerance. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 82–93, London, UK, 2000. Springer-Verlag.

[30] M. Lazić, I. Konnov, J. Widder, and R. Bloem. Synthesis of distributed algorithms with parameterized threshold guards. In *21st International Conference on Principles of Distributed Systems*, 2017.

[31] C. Lenzen and J. Rybicki. Near-optimal self-stabilising counting and firing squads. *arXiv preprint arXiv:1608.00214*, 2016.

[32] O. Matthews, J. Bingham, and D. J. Sorin. Verifiable hierarchical protocols with network invariants on parametric systems. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 101–108, 2016.

[33] K. L. McMillan. Parameterized verification of the flash cache coherence protocol by compositional model checking. In *Correct hardware design and verification methods*, pages 179–195. 2001.

[34] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesis. In *Proceedings of 31st IEEE Symposium on Foundation of Computer Science*, pages 746–757, Washington, DC, USA, 1990. IEEE Computer Society.

[35] A. Pnueli, J. Xu, and L. D. Zuck. Liveness with (0, 1, infty)-counter abstraction. In *International Conference on Computer Aided Verification (CAV)*, pages 107–122, 2002.

[36] A. Roychoudhury, K. N. Kumar, C. Ramakrishnan, I. Ramakrishnan, and S. A. Smolka. Verification of parameterized systems using logic program transformations. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 172–187. 2000.

[37] A. Roychoudhury and I. Ramakrishnan. Automated inductive verification of parameterized protocols? In *Computer Aided Verification*, pages 25–37. Springer, 2001.

[38] A. Roychoudhury and I. Ramakrishnan. Inductively verifying invariant properties of parameterized systems. *Automated Software Engineering*, 11(2):101–139, 2004.

[39] A. Sánchez and C. Sánchez. Parametrized verification diagrams. In *Temporal Representation and Reasoning (TIME), 2014 21st International Symposium on*, pages 132–141, 2014.

[40] T. Touili. Regular model checking using widening techniques. *Electronic Notes in Theoretical Computer Science*, 50(4):342–356, 2001.

[41] G. Varghese. *Self-stabilization by local checking and correction.* PhD thesis, MIT, 1993.

[42] G. Varghese. Self-stabilization by counter flushing. In *The 13th Annual ACM Symposium on Principles of Distributed Computing*, pages 244–253, 1994.

[43] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *International Workshop on Automatic Verification Methods for Finite State Systems*, pages 68–80, 1989.