Verification and Synthesis of Symmetric Uni-Rings for Leads-To Properties

Ali Ebnenasir
Department of Computer Science
Michigan Technological University
Houghton, MI 49931, U.S.A.
Email: aebnenas@mtu.edu

***Abstract*—This paper investigates the verification and synthesis of parameterized protocols that satisfy global leadsto properties $R \rightsquigarrow Q$ on symmetric unidirectional rings (a.k.a. uni-rings) of deterministic and constant-space processes, where R and Q denote global state predicates. First, we show that verifying $R \rightsquigarrow Q$ for parameterized protocols on symmetric uni-rings is undecidable, even for deterministic and constant-space processes, and conjunctive state predicates. Then, we show that surprisingly synthesizing symmetric uni-ring protocols that satisfy $R \rightsquigarrow Q$ is actually decidable. We identify necessary and sufficient conditions for the decidability of synthesis based on which we devise a sound and complete algorithm that takes the predicates R and Q, and automatically generates a parameterized protocol that satisfies $R \rightsquigarrow Q$ for unbounded (but finite) ring sizes. We use our algorithm to synthesize some parameterized protocols, including an agreement protocol.**

I. Introduction

This paper investigates the verification and synthesis of parameterized protocols that satisfy *leadsto* properties $R \rightsquigarrow Q$ on symmetric unidirectional rings (a.k.a. *uni-rings*) of deterministic and constant-space processes under no fairness and interleaving semantics, where R and Q represent global state predicates. The significance of this problem is two-fold. First, ring is a simple, but important topology for distributed systems whose underlying communication graph includes cycles (which is the case in many practical domains). Second, the leadsto property $R \rightsquigarrow Q$ is a critical liveness requirement in numerous contexts where system executions should guarantee eventual response (i.e., reaching the set of states Q) to specific stimuli (i.e., being in the set of states R). In a *symmetric* ring, the code of each process is generated from the code of a *template/representative* process by a simple variable renaming. Moreover, the number of processes in the ring is unbounded (but finite). In this paper, we first extend Suzuki's undecidability results of verifying Linear Temporal Logic (LTL) properties of uni-rings [1] to the special case of verifying leadsto properties for symmetric uni-rings of deterministic and constant-space processes, and show that the verification problem remains undecidable. We then prove that the synthesis of uni-rings that satisfy leadsto properties is actually decidable. This is a counterintuitive result as it is believed [2] that the synthesis of distributed systems is no easier than their verification.

While in our previous work [3], [4] we have addressed the verification and synthesis of self-stabilizing symmetric uni-rings, our previous results cannot be directly used for the verification and synthesis of uni-rings that satisfy leadsto properties $R \rightsquigarrow Q$. First, self-stabilization subsumes *true*$\rightsquigarrow Q$, which is a special case of $R \rightsquigarrow Q$. The property of reaching a state in Q from *any* state, called *convergence*, is a constraint that must be met by self-stabilizing protocols in addition to *closure* in Q, which means once the computations of a protocol reach some state in Q, they stay in Q; the property $R \rightsquigarrow Q$ lacks this constraint. Second, in $R \rightsquigarrow Q$, the state predicate R may be a proper subset of the state space (i.e., $R \subset$ *true*), and reachability of Q must be guaranteed only from states in R. To address this challenge, we should identify local characterization of states that are reachable from R as well as ensuring deadlock-freedom and livelock-freedom only in states reachable from R. Third, a self-stabilizing solution may not be the best solution for $R \rightsquigarrow Q$ in cases where multiple leadsto properties must be satisfied. For example, consider the case of two leadsto properties $R_1 \rightsquigarrow Q_1$ and $R_2 \rightsquigarrow Q_2$, where $R_1 \cap R_2 = \emptyset$. If one synthesizes a self-stabilizing protocol p that converges to Q_1 from any state, then p certainly satisfies $R_1 \rightsquigarrow Q_1$ too. However, all hopes for revising p towards satisfying $R_2 \rightsquigarrow Q_2$ are lost because p instead satisfies $R_2 \rightsquigarrow Q_1$. The core novelty of the proposed approach of this paper lies in identifying local characterizations of global properties (e.g., reachability of livelocks from some initial states) towards enabling synthesis in the local state space of the template process for global correctness. Such local characterizations would enable more efficient automated reasoning methods, where the time/space complexity of synthesis will be in terms of the size of the local state space of template processes instead of semi-decision procedures that conduct backward/forward reachability analysis in an over-approximated space [5].

In this paper, we first show that verifying $R \rightsquigarrow Q$ remains undecidable even for protocols with constant-space and deterministic processes, and for conjunctive state predicates $R = \forall i \in \mathbb{Z}_N : r(x_{i-1}, x_i)$ and $Q = \forall i \in \mathbb{Z}_N : q(x_{i-1}, x_i)$, where N denotes the number of processes in the ring, $\mathbb{Z}_N$ represents values modulo N, and x_i captures an abstraction of all writeable variables of the template process P_i. In a uni-ring, each process P_j can read x_{j-1} and x_j, and is able to write x_j only, where $j \in \mathbb{Z}_N$. (Conjunctive predicates

may seem restricted but they have important applications for many systems [6], [7].) We show some negative results that satisfying $R \rightsquigarrow Q$ by reaching states in Q where $x_{i-1} \neq x_i$ is impossible. Intuitively, this negative result is an outcome of the impossibility of convergence to an L-coloring in symmetric uni-rings from any state (due to Bernard *et al.*'s [8]). Subsequently, we show that synthesizing a parameterized protocol that satisfies $R \rightsquigarrow Q$ is decidable *iff* (if and only if) there is a sequence of steps for every process towards satisfying $q(\gamma, \gamma)$ locally, starting from a state that satisfies R. We then provide a sound and complete algorithm that takes the state predicates R and Q, and generates the parameterized actions of the template process, if a solution exists. The time complexity of the proposed algorithm is polynomial in the state space of the template process, which is often a small value for constant-space processes. We demonstrate our algorithm through a few case studies including a protocol that ensures reaching agreement in uni-rings when processes of the ring disagree on a value. More examples are available in [9].

Organization. Section II presents some basic concepts. Section III shows that verifying leadsto on uni-rings is undecidable. Section IV proves the decidability of synthesizing symmetric uni-rings that satisfy leadsto properties. Section V discusses related work, and Section VI summarizes our contributions and outlines some future work.

II. Preliminaries

This section represents the definition of parameterized protocols and their representation as action graphs (adopted from [10], [11], [3], [4]). Wlog, we use the term *protocol* to refer to finite-state symmetric uni-rings as we conduct our investigation in the context of network protocols. Such rings are parameterized in the number of processes in the ring. A protocol p for a computer network includes $N > 1$ processes (finite-state machines), where each process P_i has a finite set of readable and writeable variables. Any *local state* of a process (a.k.a. *locality/neighborhood*) is determined by a unique valuation of its readable variables. We assume that any writeable variable is also readable. Network topology defines the set of readable variables of a process. For example, in a uni-ring consisting of N processes, each process P_i (where $i \in \mathbb{Z}_N$, i.e., $0 \leq i \leq N-1$) has a predecessor P_{i-1}, where subtraction and addition are in modulo N. The *global state* of a protocol is defined by a snapshot of the local states of all processes. The *state space* of a protocol, denoted by Σ, is the universal set of all global states. A *state predicate* is a subset of Σ. A process *acts* (i.e., *transitions*) when it atomically updates its state based on its locality. We assume that processes act one at a time (i.e., interleaving semantics). Thus, each *global transition* corresponds to the action of a single process from some global state. An *execution/computation* of a protocol is a sequence of states $s_0, s_1, \ldots, s_k$ where there is a transition from s_i to s_{i+1} for every $i \in \mathbb{Z}_k$. We consider *parameterized* protocols that consist of families of symmetric processes. Each family is represented by a *template* process from which the code of all family members is instantiated by a simple variable renaming/re-indexing. For instance, a symmetric uni-ring includes just one family for which we use triples of the form (a, b, c) to denote actions $(x_{i-1} = a \wedge x_i = b \longrightarrow x_i := c;)$ of the template process P_i. An action has two components; a *guard*, which is a Boolean expression in terms of readable variables and a *statement* that atomically updates the state (i.e., writeable variables) of the process once the guard evaluates to *true*; i.e., the action is *enabled*.

Definition II.1 (Transition Function)**.** Let P_i be a process with a variable x_i in a uni-ring protocol p. p's transition function $\delta : \Sigma \times \Sigma \rightarrow \Sigma$ is a partial function such that $\delta(a, b) = c$ if and only if P_i has an action $(x_{i-1} = a \wedge x_i = b \longrightarrow x_i := c;)$. That is, δ can be used to define all actions of P_i in the form of a single parametric action:

$$((x_{i-1}, x_i) \in \mathsf{Pre}(\delta)) \longrightarrow x_i := \delta(x_{i-1}, x_i);$$

where $(x_{i-1}, x_i) \in \mathsf{Pre}(\delta)$ checks to see if the current x_{i-1} and x_i values are in the preimage of δ. For other topologies, the preimage of δ might be specified differently depending on the locality of each process.

Definition II.2 (Action Graph)**.** We depict the set of actions of the template process of a symmetric uni-ring by a labeled directed multigraph $G = (V, A)$, called the *action graph*, where each vertex $v \in V$ represents a value in $\mathbb{Z}_M$, where M denotes the domain size of x_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 Sum-Not-2 protocol given in [12]. Each process P_i 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 ensures that, from any global state, a state is reached where the sum of each two consecutive x values does not equal 2. We formally specify these states as the state predicate $\forall i \in \mathbb{Z}_N : (x_{i-1} + x_i \neq 2)$. We can represent this protocol as a graph containing arcs $(0, 2, 1)$, $(1, 1, 2)$, and $(2, 0, 1)$ as shown in Figure 1.

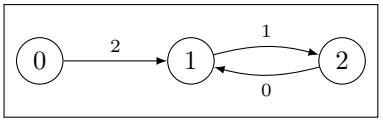


Fig. 1: Graph representing Sum-Not-2 protocol.

For simplicity, we assume that protocols consist of *self-disabling* processes. As such, an action (a, b, c) cannot coexist with action (a, c, d) for any d. Moreover, a *deterministic* process cannot have two actions enabled at the same time; i.e., an action (a, b, c) cannot coexist with an action (a, b, d) where $d \neq c$.

Definition II.3 (Leadsto Properties)**.** The focus of this paper is on *leadsto* properties that are specified as $\Box(R \Rightarrow \Diamond Q)$ in Linear Temporal Logic (LTL), also denoted $R \rightsquigarrow Q$, where $\Box$ and $\Diamond$ respectively represent the universality and eventuality modalities, and $R = \forall i \in \mathbb{Z}_N : r(x_{i-1}, x_i)$ and $Q = \forall i \in \mathbb{Z}_N : q(x_{i-1}, x_i)$ are conjunctive state predicates for a uni-ring of N processes. A computation $\sigma = \langle s_0, s_1, \cdots \rangle$ of a protocol p satisfies $\Box S$ *iff* the state predicate S holds in every state $s_i \in \sigma$, for all $i \geq 0$. A computation $\sigma = \langle s_0, s_1, \cdots \rangle$ of p

satisfies $\Diamond S$ *iff* there is some $i \geq 0$ such that the state predicate S holds in the state $s_i \in \sigma$. A computation $\sigma = \langle s_0, s_1, \cdots \rangle$ of p satisfies $R \rightsquigarrow Q$ *iff* $s_0 \in R$ implies that there is some $i \geq 0$ such that $s_i \in Q$. A uni-ring protocol p satisfies $R \rightsquigarrow Q$ *iff* all computations of p satisfy $R \rightsquigarrow Q$, for unbounded (but finite) ring sizes.

Definition II.4 (Fairness)**.** A *strongly* fair scheduler ensures that any action that is infinitely often enabled will be executed infinitely often (due to Gouda [13]). A *weakly* fair scheduler guarantees that if an action is continuously enabled, then it will be executed infinitely often.

Livelock, deadlock, and closure. A *global livelock* of a protocol p is an infinite computation $l = \langle s_0, s_1, \cdots, s_0 \rangle$, where s_i is a global state, for all $i \geq 0$. A *local livelock* of a process P of protocol p is an infinite execution $l = \langle s_0, s_1, \cdots, s_0 \rangle$, where s_i is a local state of P, for all $i \geq 0$. Unless stated otherwise, we use the terms ''livelock" and ''global livelock" interchangeably. For satisfying a leadsto property $R \rightsquigarrow Q$, a reachable livelock l that includes at least one state in Q is acceptable; otherwise, it is considered as a failure towards satisfying $R \rightsquigarrow Q$. A *deadlock* of p is a state that has no outgoing transition; i.e., no process is enabled to act. A state predicate I is *closed* under p *iff* there is no transition (s, s'), where $s \in I$ and $s' \notin I$.

Definition II.5 (Self-Stabilization and Convergence)**.** A protocol p is *self-stabilizing* [14] to a state predicate I (under no fairness) *iff* from any state in $\neg I$, *every* computation of p reaches a state in I (i.e., *convergence*) and remains in I (i.e., *closure*). That is, p is livelock-free and deadlock-free in $\neg I$, and I is closed under p. A protocol p is *silent-stabilizing* to I *iff* p converges to I and p has no computation starting in I. Notice that, in LTL, convergence to I is specified as $\Box\Diamond I$, which is logically equivalent to *true* $\rightsquigarrow I$.

Definition II.6 (Weak Stabilization)**.** A *weakly* stabilizing protocol to I ensures that from each state in $\neg I$, there is *some* computation that reaches a state in I (a.k.a., *weak convergence*) and remains in I.

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

Definition II.7 (Locality Graphs)**.** Consider a state predicate $Q = \forall i : q(x_{i-1}, x_i)$ for a uni-ring. The relation $q(x_{i-1}, x_i)$ captures a set of local states, representing an acceptable relation between each process P_i and its predecessor. The relation $q(x_{i-1}, x_i)$ must also be *locally correctable* in that, for any value of x_{i-1}, there is always a sequence of steps that P_i can take to establish q by updating x_i only. To simplify reasoning, we represent $q(x_{i-1}, x_i)$ as a digraph $G = (V, A)$, called the *locality graph*, such that each vertex $v \in V$ represents a value in $\mathbb{Z}_M$, and an arc (a, b) is in A *iff* $q(a, b)$ is true.

Figure 2 illustrates the locality graph of the Sum-Not-2 protocol introduced in this section for the state predicate $Q = \forall i \in \mathbb{Z}_N : (x_{i-1} \oplus x_i \neq 2)$ where $M = 4$ and $\oplus$ represents addition modulo 4. Each closed walk in the locality graph characterizes a class of global states in the state predicate Q. For example, the closed walk $(0, 1, 3, 1, 0)$ captures the global states of ring sizes $4 \times k$ where the x_i values of processes follow a repeated pattern of $0, 1, 3, 1$, and k is a positive integer. We now represent one of our previous results [12] on the relation between closed walks in locality graphs and global states of parameterized protocols on uni-rings.

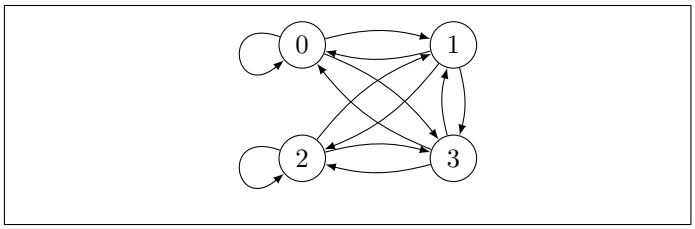


Fig. 2: Locality graph of the Sum-Not-2 protocol.

Theorem II.8. *Any closed walk of length $L \geq 1$ in the locality graph of a conjunctive predicate Q of a uni-ring characterizes global states in Q of ring sizes of $L \times k$, where k is a positive integer. (Proof in [12])*

Corollary II.9. *Any conjunctive state predicate $Z = \forall i : i \in \mathbb{Z}_N : z(x_{i-1}, x_i)$ whose locality graph G_Z is acyclic specifies an empty set of states in a parameterized symmetric uni-ring.*

Next, we represent some of our previous results regarding livelocks (from [12], [10], [4]) that we shall use in this paper.

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 $n \geq 1$ corresponds to a *closed walk* of length n in the action graph. The Sum-Not-2 protocol has such a propagation of period 2: $\langle (1, 1, 2), (2, 0, 1) \rangle$ (see Figure 1). 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; i.e., 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 [10], [4], we prove the following theorem:

Theorem II.10. *A uni-ring protocol of symmetric, deterministic and self-disabling processes has a livelock for some ring size* iff *there exist m propagations with some period n, where the $(i-1)$-th propagation leads the i-th propagation for each index i modulo m, for $m > 1$ and $n \geq 1$; i.e., the propagations successively lead each other modulo m.*

We have shown [12] that verifying deadlock-freedom in uni-rings is decidable. However, checking livelock-freedom is an undecidable problem for uni-ring protocols (with self-disabling and deterministic processes) [10], [4]. 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* [10].

Theorem II.11. *Verifying livelock-freedom in a uni-ring protocol (with self-disabling and deterministic processes) remains undecidable even for deterministic livelocks [10].*

Since in every state of a deterministic livelock there is exactly one enabled process, the choice of fairness policy has no impact on which process will be executed in each state because the scheduler has only one enabled process to select.

Corollary II.12. *Verifying livelock-freedom in a symmetric uni-ring protocol (with self-disabling and deterministic processes) remains undecidable regardless of the fairness assumption (i.e., scheduling policy) [10].*

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

Theorem II.13. *Verifying convergence, silent-stabilization and self-stabilization for a uni-ring protocol (with self-disabling and deterministic processes) is undecidable [10].*

While verifying self-stabilization for uni-rings is undecidable, we have shown that synthesis of self-stabilizing uni-rings is surprisingly decidable.

Theorem II.14. *There is a symmetric uni-ring protocol p that self-stabilizes to a state predicate $Q = \forall i \in \mathbb{Z}_N : q(x_{i-1}, x_i)$ for unbounded (but finite) ring size N iff there is a vertex γ in the locality graph of Q, where γ has a self-loop (i.e., $q(\gamma, \gamma)$ holds) that can be reached from another vertex [11], [3].*

III. Undecidability of Verification

This section presents some impossibility results for the verification and synthesis of symmetric uni-ring protocols that satisfy *leads-to* properties. Throughout the rest of the paper, $R = \forall i \in \mathbb{Z}_N : r(x_{i-1}, x_i)$ and $Q = \forall i \in \mathbb{Z}_N : q(x_{i-1}, x_i)$ represent conjunctive state predicates, and p denotes a symmetric uni-ring protocol of self-disabling, constant-space and deterministic processes.

Problem III.1 (Verification of LeadsTo)**.** Does p satisfy $R \rightsquigarrow Q$ for unbounded (but finite) ring sizes?

Theorem III.2. *Problem III.1 is undecidable even for uni-rings of self-disabling, constant-space and deterministic processes.*

Proof. For a symmetric protocol p to satisfy $R \rightsquigarrow Q$ on a uni-ring, p should ensure that starting in R, its computations will eventually reach some state in Q under no fairness and interleaving semantics. As a special case, let R =*true*; i.e., verifying convergence to Q. Theorem II.13 shows that verifying convergence is undecidable for uni-rings of self-disabling, constant-space and deterministic processes. Therefore, the general case of Problem III.1 is undecidable too. □

Problem III.3 (Synthesis of LeadsTo)**.** Does there exist a symmetric protocol p on the ring that satisfies $R \rightsquigarrow Q$ for unbounded (but finite) ring sizes?

To investigate Problem III.3, we first consider a special case of this problem where R=*true*.

Problem III.4 (Synthesis of Convergence)**.** Does there exist a symmetric protocol p on the ring that satisfies $true \rightsquigarrow Q$ for unbounded (but finite) ring sizes?

Converging to a state where $Q = \forall i \in \mathbb{Z}_N : q(x_{i-1}, x_i)$ holds in a uni-ring can be achieved in two ways. First, there may be some value $c \in \mathbb{Z}_M$ such that $q(c, c)$ holds. Such values are represented as self-loops in the locality graph of $q(x_{i-1}, x_i)$. Second, L distinct values $c_0, c_1, \cdots, c_{L-1} \in \mathbb{Z}_M$ satisfy $q(x_{i-1}, x_i)$ in a circular fashion, where $q(c_0, c_1), q(c_1, c_2), \cdots, q(c_{L-2}, c_{L-1}), q(c_{L-1}, c_0)$ hold and $L > 1$. Such values form a cycle of length L in the locality graph of $q(x_{i-1}, x_i)$. A cycle like that represents a family of rings that include global states where Q holds through an ordered placement of the values that appear in the cycle. The sizes of such rings are multiples of L; i.e., ring sizes of $k \cdot L$, where k is a positive integer (Theorem II.8). To design a symmetric protocol p that satisfies $\Diamond Q$ from some state predicate R, developers should design p in such a way that it ensures convergence to one of the aforementioned scenarios (under no fairness and interleaving semantics). We first show that, there is no protocol that can ensure convergence through the second scenario.

Definition III.5 (Ordered L-coloring)**.** Consider a set of L distinct colors and a permutation function $next(c)$ (i.e., a bijective function from L to L) that takes a color $c \in L$ and returns a color $c' \in L$ where $c \neq c'$. An ordered L-coloring of a uni-ring is an L-coloring where $\forall i \in \mathbb{Z}_N : c_i = next(c_{i \ominus 1})$, N denotes the number of processes in the ring, and $\ominus$ represents subtraction modulo N.

Theorem III.6. *No protocol exists that converges to an ordered L-coloring on symmetric uni-rings for rings of $N > L$ processes.*

Proof. Bernard *et al.*'s [8] show that no converging L-coloring protocol exists on symmetric uni-rings for rings of $N > L$ processes. By contradiction, assume there is a converging protocol p that ensures an *ordered* L-coloring for some uni-ring with size $N > L$. Such an ordered L-coloring is a legitimate L-coloring. Thus, p converges to an L-coloring, which is a contradiction with [8]. □

Corollary III.7. *Theorem III.6 holds under any fairness assumption, including strong fairness. In other words, Theorem III.6 holds for weak convergence too.*

Proof. By contradiction, suppose Theorem III.6 is falsified assuming strong fairness. That is, there is a weakly converging protocol that guarantees L-coloring on uni-rings of $N > L$ processes. This means that from any arbitrary state, there is at least a computation that reaches a valid L-coloring of the uni-ring for any $N > L$. Wlog, consider the case where $N = L+1$; i.e., the number of colors is one unit less than the number of processes, and processes that have a similar color to their predecessor simply choose the next color available by incrementing their value. Bernard *et al.* [8] show that the executions of any deterministic coloring protocol are isomorphic to a coloring protocol where all processes follow the rule of incrementing their value (i.e., choosing the next color in order). Now, consider an example state $\langle 0, 0, 1, 2, 3, \cdots, N-2\rangle$ for the uni-ring of size N. In this state, only the second process is enabled to change its color because it is the only processes that has the same color as that of its predecessor's. The execution of the second process would get the protocol to the state $\langle 0, 1, 1, 2, 3, \cdots, N-2\rangle$, where the third process is now enabled. Following the same pattern, the uni-ring reaches the state $\langle 0, 1, 2, 2, 3, \cdots, N-2\rangle$, where the fourth process is the only process that is enabled. This sequence of states forms the livelock $\langle 0, 0, 1, 2, 3, \cdots, N-2\rangle$, $\langle 0, 1, 1, 2, 3, \cdots, N-2\rangle$, $\langle 0, 1, 2, 2, 3, \cdots, N-2\rangle$, $\cdots$, $\langle 0, 1, 2, 3, 4, \cdots, N-2, N-2\rangle$, $\langle 0, 1, 2, 3, \cdots, N-2, 0\rangle$. Notice that, the last state is actually the same as the initial state of this execution of the symmetric uni-ring. Moreover, this livelock is deterministic (Theorem II.11). Thus, by Corollary II.12, even strong fairness will not resolve this deterministic livelock, which prevents convergence to L-coloring. A similar argument holds for weak fairness. □

Theorem III.8. *No symmetric uni-ring protocol exists that converges to $Q = \forall i \in \mathbb{Z}_N : q(x_{i-1}, x_i)$ through cyclic satisfaction of $q(c_0, c_1), q(c_1, c_2), \cdots, q(c_{L-2}, c_{L-1}), q(c_{L-1}, c_0)$, for rings of $N > L$ processes and $L > 1$. This result holds under any fairness assumption.*

Proof. Let p be a parameterized protocol that converges to states in Q (under any fairness assumption) where $q(c_0, c_1), q(c_1, c_2), \cdots, q(c_{L-2}, c_{L-1}), q(c_{L-1}, c_0)$ hold for some values $c_0, c_1, \cdots, c_{L-1}$, where $L > 1$. Thus, the locality graph of $q(x_{i-1}, x_i)$ must include a cycle whose vertices are labeled with $c_0, c_1, \cdots, c_{L-1}$ in order. As a results, such states of Q represent an ordered L-coloring. That is, p converges to an ordered L-coloring protocol for $N > L$. This is a contradiction with Theorem III.6 and Corollary III.7. □

Theorem III.9. *Suppose $R \cap Q = \emptyset$ and $R \neq true$. No symmetric protocol exists on uni-rings that can satisfy $(R \rightsquigarrow Q)$ by reaching states where Q holds through cyclic satisfaction of $q(c_0, c_1), q(c_1, c_2), \cdots, q(c_{L-2}, c_{L-1}), q(c_{L-1}, c_0)$, for rings of $N > L$ processes.*

Proof. By contradiction, let p be such a protocol whose computations reach states in Q from states in R through cyclic satisfaction of $q(c_0, c_1), q(c_1, c_2), \cdots, q(c_{L-2}, c_{L-1}), q(c_{L-1}, c_0)$. Using the decidability of synthesizing self-stabilizing protocols (Theorem II.14), we design a silent stabilizing protocol p' that converges to R from any state, and once in R, p' becomes disabled. However, the actions of p and p' may interfere by creating livelocks outside $R \vee Q$. Such livelocks include states where p and p' both have enabled actions. Thus, such livelocks are not deterministic livelocks. To ensure recovery from such livelocks, we assume strong fairness. As a result, we guarantee that p' will eventually converge to R, and from R, protocol p can guarantee that we reach states in Q through cyclic satisfaction of $q(c_0, c_1), q(c_1, c_2), \cdots, q(c_{L-2}, c_{L-1}), q(c_{L-1}, c_0)$. Therefore, the net result is a protocol that converges to Q through cyclic satisfaction of $q(c_0, c_1), q(c_1, c_2), \cdots, q(c_{L-2}, c_{L-1}), q(c_{L-1}, c_0)$ under strong fairness, which contradicts with Theorem III.8. □

IV. Decidability of Synthesis

This section proves the decidability of synthesizing symmetric uni-ring protocols that satisfy the leads-to property $(R \rightsquigarrow Q)$, where R and Q denote conjunctive state predicates. We first establish a relation between self-stabilization and leads-to by the following lemma:

Lemma IV.1. *There is a symmetric protocol that satisfies $(R \rightsquigarrow Q)$ for unbounded (but finite) ring sizes iff there is a symmetric protocol that stabilizes to Q for unbounded (but finite) ring sizes.*

Proof. Any protocol p that is self-stabilizing to Q ensures convergence to Q; i.e., $(true \rightsquigarrow Q)$. Since $R \subset true$, it follows that protocol p satisfies $(R \rightsquigarrow Q)$. Now, if there is no protocol that stabilizes to Q, then Theorem II.14 implies that there is no value γ in the domain of x_i for which $q(\gamma, \gamma)$ holds. Thus, for a protocol p' to satisfy $(R \rightsquigarrow Q)$, p' must converge to states where Q holds through cyclic satisfaction of $q(c_0, c_1), q(c_1, c_2), \cdots, q(c_{L-2}, c_{L-1}), q(c_{L-1}, c_0)$, which is impossible due to Theorem III.9. □

Theorem IV.2. *Synthesizing a symmetric protocol p on (unbounded) uni-rings where p satisfies $(R \rightsquigarrow Q)$ is decidable.*

To prove Theorem IV.2, we present a synthesis algorithm that takes two disjoint conjunctive state predicates R and Q, and generates a protocol that satisfies $R \rightsquigarrow Q$ on symmetric uni-rings, for unbounded ring sizes. Wlog, we assume that $R \cap Q = \emptyset$; even if R and Q intersect the synthesis problem is formulated for $(R - X) \rightsquigarrow (Q - X)$, where $X = R \cap Q$.

Algorithm 1. *SynLeadsTo($r(x_{i-1}, x_i), q(x_{i-1}, x_i)$: state predicate, M: domain size of x_i)*

1: Create the locality graphs $G_Q = (V_Q, A_Q)$ and $G_R = (V_R, A_R)$ respectively for both $q(x_{i-1}, x_i)$ and $r(x_{i-1}, x_i)$.
2: Find a $\gamma \in V_Q$ such that $q(\gamma, \gamma)$ holds and γ has not been used before. If no such γ exists, then declare that no solution exists and **exit**.
3: Induce a subgraph $G'_Q = (V'_Q, A'_Q)$ that contains all arcs of A_Q that participate in simple cycles involving γ. If there is no such subgraph, then $V'_Q = \{\gamma\}$ and $A'_Q = \emptyset$.
4: Compute a spanning tree τ of G'_Q rooted at γ.
5: Let V'_R be the subset of V_R that do not participate in any cycle.
6: Let V'_{Rleaf} be the set of vertices $v \in V'_R$ that are leaves in τ. Remove the outgoing arc of each $v \in V'_{Rleaf}$, hence creating a tree τ' (which is no longer a *spanning* tree of G'_Q).
7: For each node $v \in (V_Q - (V'_Q \cup V'_{Rleaf}))$, include an arc from v to the root γ of the spanning tree τ' of G'_Q, unless $r(v, \gamma)$ holds.
8: For each node $v \in V_Q$ where $r(v, \gamma)$ holds, include an arc (v, l), where l is a leaf in τ'. The resulting graph would still be a tree, denoted τ''. Include a self-loop (γ, γ) at the root of τ''. If τ'' has no leaves in common with any cycle in G_R, then go to Step 2.
9: For each leaf vertex a in τ'', label its outgoing arc (a, c) with a value $b \in \mathbb{Z}_M$ iff $b \neq c$ and $r(a, b) \wedge \neg q(a, b)$.
10: For any other arc (a, c) in τ'', label it with a value $b \in \mathbb{Z}_M$ iff $b \neq c$ and $\neg q(a, b)$.
11: For each labelled arc (a, b, c) in τ'' (where b is the label of arc (a, c)), generate a parameterized action $x_{i-1} = a \wedge x_i = b \rightarrow x_i := c$.

end

Fig. 3: Synthesis algorithm for LeadsTo in symmetric uni-rings

(Note that $R \rightsquigarrow Q$ is vacuously true in X.) We later show that Algorithm 1 is sound and complete.

Intuitively, Algorithm 1 identifies if there is a sequence of steps for every process towards satisfying $q(\gamma, \gamma)$ locally, starting from a state that satisfies R. Specifically, Algorithm 1 constructs the state transition system of the template process by discovering if there are paths from local states in $r(x_{i-1}, x_i)$ to a self-loop in a state γ, where $q(\gamma, \gamma)$ holds, without including any cycles and any step (v, γ), where $r(v, \gamma)$ holds. To this end, Algorithm 1 forms a tree with the root γ, and the leaves that have no intersection with states outside $r(x_{i-1}, x_i)$, while excluding any arc $r(v, \gamma)$. For a specific γ, Algorithm 1 performs Steps 3 to 8 in order to determine if there is a solution for that γ. In Steps 3 to 8, Algorithm 1 constructs the underlying structure of an action graph that will form the synthesized protocol p after the labeling steps of 9 and 10. A correct protocol must meet certain constraints that are the minimum requirements for a protocol p that satisfies $R \rightsquigarrow Q$. For instance, p should guarantee $R \rightsquigarrow Q$ only starting in R. That is why Step 5 calculates the set of values in $\mathbb{Z}_M$ that do not appear in any state of R, and Step 6 removes them from the spanning tree of G'_Q. Moreover, since we assume $R \cap Q = \emptyset$, the cycles of G_R and G_Q are arc-disjoint (due to Theorem II.8). Thus, any correct protocol cannot include an arc $(a, c) \in A_R$ for which $r(a, c)$ holds and $c = \gamma$. Otherwise, the processes will include an action $x_{i-1} = a \wedge x_i = b \rightarrow x_i := c$, for some b where $\neg q(a, b)$. Such an action would set the locality of each process to a state (i.e., $x_{i-1} = a \wedge x_i = c$) where R holds, thereby preventing reachability to Q. That is why Step 7 excludes such arcs. After executing Steps 3 to 8, if we have a tree-like structure that has no leaves in common with the cycles in G_R, then we move to Step 2 and repeat the same process for a different γ. Algorithm 1 exits only if no valid action graph can be built for any γ.

Theorem IV.3 (Soundness)**.** *Algorithm 1 is sound.*

Proof. We show that if Algorithm 1 generates a parameterized protocol p for two disjoint predicates R and Q, then p satisfies $R \rightsquigarrow Q$ for unbounded ring sizes. To prove this, we show that any computation of p starting in a state $s \in R$ will be deadlock-free and livelock-free outside Q and will eventually reach a state s_f in Q (regardless of the ring size).

Deadlock-freedom: Since the synthesized action graph, denoted AG, is a tree with a self-loop on its root γ, AG must have some leaves. Steps 8 and 9 of the algorithm ensure that, starting in R, we have some enabled action. The labeling method of Step 9 guarantees that any leaf a of AG has an outgoing arc (a, c) (to some vertex $c \in \mathbb{Z}_M$) with a label b such that $r(a, b)$ holds. Step 11 would translate each labeled arcs (a, b, c) to a parameterized action $x_{i-1} = a \wedge x_i = b \rightarrow x := c$. By Theorem II.8, we also know that cycles in locality graphs characterize global states of uni-rings. Thus, the guard condition of such an action evaluates to *true* in a state in R because $r(a, b)$ holds. Thus, starting in R, there is at least one enabled action; i.e., deadlock-freedom in R. We now show that the computations of the synthesized protocol p remain deadlock-free until reaching a state in Q. The only values in $\mathbb{Z}_M$ that are excluded from the synthesized action graph AG include those values that do not participate in any cycle in G_R; i.e., those values do not appear in states in R. If computations of p reach a state outside $R \vee Q$, the labeling method of Step 10 ensures that there is some enabled action; hence deadlock-freedom.

Livelock-freedom: The only type of propagation included in the synthesized action graph is (γ, b, γ). Thus, there are no propagations that lead each other circularly. Therefore, based on Theorem II.10, the synthesized protocol is livelock-free.

Reachability of Q: By deadlock-freedom and livelock-freedom, each process will eventually satisfy $q(\gamma, \gamma)$, which would result in a global state of the ring in Q. □

Theorem IV.4 (Completeness)**.** *Algorithm 1 is complete.*

Proof. Let p be a parameterized protocol that satisfies $R \rightsquigarrow Q$ but Algorithm 1 fails to generate p. By Theorem III.9, p cannot satisfy $R \rightsquigarrow Q$ through cyclic satisfaction of some values; i.e., the action graph of p, denoted A_p, can include only cycles of length 1. Further, if a node v has a self-loop in A_p, then v cannot have any other outgoing labeled arc; otherwise, A_p will include either cycles of length greater than one, which contradicts Theorem III.9, or ends in nodes without any outgoing arcs; i.e., deadlock. Thus, there must be a value $\gamma \in \mathbb{Z}_M$ for which $q(\gamma, \gamma)$ holds and there is some value $v \in \mathbb{Z}_M$, where $v \neq \gamma$ and $q(v, \gamma)$ holds too. That is, A_p must include γ as well as another labeled arc (v, b, γ) for some $b \in \mathbb{Z}_M$, where $v \neq \gamma$ and $b \neq \gamma$. This means that Algorithm 1 would actually find such γ in Step 2. The action graph A_p must also include some source nodes without any incoming arcs. Otherwise, it would include cycles of length greater than one (which again contradicts Theorem III.9). Such source nodes must intersect with the vertices of some cycles in V_R and have outgoing labeled arcs (a, b, c) such that $r(a, b)$ holds and a is a vertex in a cycle in G_R. Otherwise, p deadlocks in R, which contradicts with p satisfying $R \rightsquigarrow Q$. As such, Algorithm 1 would have included such labeled arcs in Steps 8 and 9, and would have created the action graph A_p sinking towards γ. □

Theorem IV.5. *Algorithm 1 has an asymptotic polynomial time complexity in the domain size M. (Proof in [9].)*

A. Sum-Not-2

The Sum-Not-2 protocol is a simple but non-trivial example that can clearly demonstrate the complexity of synthesizing parameterized uni-rings that satisfy leadsto properties. In this protocol, we specify the set of initial states for even-size uni-rings where the summation of x_{i-1} and x_i is equal to two (modulo $M = 4$) and $x_i \neq 1$. That is, $R = \forall i \in \mathbb{Z}_N : r(x_{i-1}, x_i)$, where $r(x_{i-1}, x_i) \equiv (x_i = 2 \wedge x_{i-1} = 0) \vee (x_i = 0 \wedge x_{i-1} = 2)$, and N denotes the number of processes (i.e., ring size). The objective is to synthesize a protocol that eventually reaches states where the summation of x_{i-1} and x_i is no longer equal to two, and at least one of the values of x_{i-1} or x_i is non-zero; i.e., $Q = \forall i \in \mathbb{Z}_N : q(x_{i-1}, x_i)$, where $q(x_{i-1}, x_i) \equiv ((x_{i-1} \oplus_4 x_i) \neq 2) \wedge ((x_{i-1} \neq 0) \vee (x_i \neq 0))$, and $\oplus_4$ denotes addition modulo $M = 4$. The state predicate Q implies that there is at least some non-zero value in the ring. We require that $R \rightsquigarrow Q$ is satisfied from R for all even values of N. Notice that, for odd-size rings R is empty (see Figure 4-(a) and Theorem II.8).

Step 1: Figure 4 illustrates the locality graphs of R and Q. Each arc (a, b) captures the fact that $r(a, b)$ (respectively, $q(a, b)$) holds. For example, the only arcs in Figure 4-(a) are between 0 and 2 because $r(x_{i-1}, x_i)$ does not hold for any other pairs of values in $\mathbb{Z}_4$. As another example, Figure 4-(b) lacks any arcs between 0 and 2 because their summation adds up to 2, which violates Q. Moreover, the only vertex that has a self-loop is 2 because the arc $(0, 0)$ violates the second conjunct of Q and arcs $(1, 1)$ and $(3, 3)$ violate the first conjunct of Q (i.e., $1 \oplus_4 1 = 2$ and $3 \oplus_4 3 = 2$).

(a) Locality graph G_R representing predicate R.

(b) Locality graph G_Q representing predicate Q.

Fig. 4: Locality graphs for the predicates R and Q.

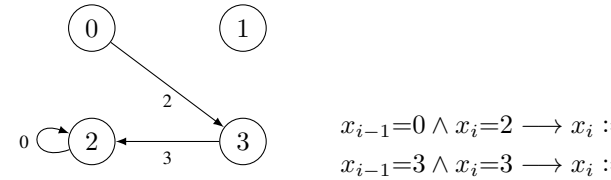
Step 2: $\gamma = 2$ since $q(2, 2)$ holds in G_Q (see Figure 4-(b)).
Step 3: Figure 5-(a) illustrates the induced subgraph G'_Q including simple cycles of G_Q that contain γ (e.g., the simple cycle $(0, 3, 2, 1, 0)$).
Step 4: Figure 5-(b) illustrates a spanning tree τ rooted at $\gamma = 2$, where each arc (a, b) denotes that b is the parent of a in τ. Notice that, there may be several spanning trees; i.e., the solution is not unique.

(a) Induced subgraph G'_Q.

(b) Spanning tree τ of the induced subgraph G'_Q with the root $\gamma = 2$.

Fig. 5: Construction of the spanning tree of Sum-Not-2.

Step 5: Figure 4-(a) illustrates that vertices 1 and 3 do not participate in any cycles in G_R; hence $V'_R = \{1, 3\}$.
Step 6: Since 1 is the only vertex of V'_R that is also a leaf of the spanning tree τ, we eliminate its outgoing arc to 3, illustrated by the dashed arrow in Figure 5-(b).
Step 7: Since $V'_Q = V_Q$, Step 7 does not make any changes.
Step 8: The only leaf of the tree τ' for which $r(0, 2)$ holds is 0. Thus, we include just the self-loop $(2, 2)$ to generate the tree τ'' in Figure 6-(a).



(a) Action graph of the synthesized protocol.

$x_{i-1}{=}0 \wedge x_i{=}2 \longrightarrow x_i := 3;$
$x_{i-1}{=}3 \wedge x_i{=}3 \longrightarrow x_i := 2;$
$x_{i-1}{=}2 \wedge x_i{=}0 \longrightarrow x_i := 2;$

(b) Parameterized actions.

Fig. 6: Synthesized protocol that satisfies $R \rightsquigarrow Q$.

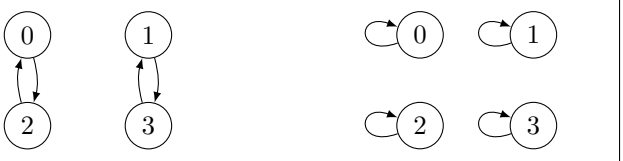
Steps 9, 10 and 11: Consider the arc $(3, 2)$. The candidate labels of this arc include $0, 1$ and 3. We exclude 2 because it is equal to the parent vertex of 3. Since $q(3, 0)$ and $q(3, 1)$ are *true* and $q(3, 3)$ is *false*, the only acceptable label for the arc $(3, 2)$ is 3 (see Figure 6-(a)). The arc $(0, 3)$ has the label

2 because $r(0,2) \wedge \neg q(0,2)$ holds. Likewise, the self-loop on 2 gets 0 as its label. Figure 6-(b) illustrates the synthesized actions for any even-size uni-ring that satisfies $R \rightsquigarrow Q$.

B. Agreement Protocol

Agreement is a fundamental problem in distributed computing. This section demonstrates how Algorithm 1 synthesizes a parameterized protocol on symmetric uni-rings that ensures agreement from a set of initial states $R = \forall i \in \mathbb{Z}_N : (((x_i \ominus x_{i-1}) \bmod 2 = 0) \wedge (x_i \neq x_{i-1}))$ to states $Q = \forall i \in \mathbb{Z}_N : x_{i-1} = x_i$.

Steps 1 and 2: Figure 7 illustrates the locality graphs G_R and G_Q, respectively for predicates R and Q. We have four possible values for γ. Wlog, we let γ be 1.



(a) The locality graph G_R for the state predicate R.

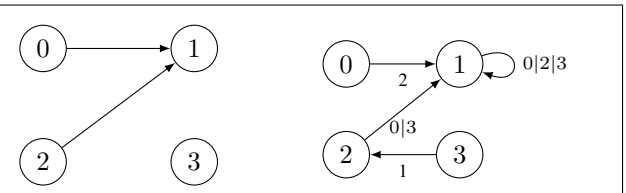
(b) The locality graph G_Q for the state predicate Q.

Fig. 7: Locality graphs of the Agreement protocol.

Steps 3 and 4: Since G_Q has only self-loops on its vertices, G'_Q would include just Vertex 1. Correspondingly, the spanning tree τ includes just a single vertex (i.e., 1).

Steps 5, 6: Since V'_R is empty, V'_{Rleaf} becomes empty too.

Steps 7 and 8: We have $V_Q - (V'_Q \cup V'_{Rleaf}) = \{0,2,3\}$. As a result, τ' would include arcs $(0,1)$ and $(2,1)$, but excludes $(3,1)$ because $r(3,1)$ holds (see Figure 8-(a)). Executing Step 8 would result in including arcs $(3,2)$ and $(1,1)$, culminating in tree τ'' in Figure 8-(b), excluding the labels.



(a) The tree τ'.

(b) The tree τ'' and the action graph.

Fig. 8: Trees computed during synthesis of Agreement.

Steps 9, 10, 11: The remaining steps of the algorithm labels τ'' (Figure 8-(b)), which would result in the action graph in Figure 8-(b). The synthesized parameterized actions of the agreement protocol are as follows:

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

V. Related Work

There are a variety of methods for the synthesis of parameterized programs. For example, Attie and Emerson [15] compose a pair of representative processes (under weak fairness) to reason about the global safety and local leads-to properties of a symmetric parameterized system. Finkbeiner and Schewe [16] formulate the synthesis of fixed-size protocols as a set of constraints, and use Satisfiability Modulo Theory (SMT) solvers [17] to find a protocol that is accepted by a Universal Co-Buchi Tree (UCT) automaton generated from temporal logic specifications. The search for a protocol is conducted up to a certain bound on the state space of processes and/or their automata-theoretic product; i.e., *bounded synthesis*. Jacobs and Bloem [18] extend bounded synthesis to parameterized protocols by identifying *cutoff* bounds, where a solution exists for a protocol with *cutoff* number of processes *iff* a solution exists for the parameterized protocol with unbounded number of processes (a.k.a. *parameterized synthesis*). Then, they apply the SMT-based bounded synthesis for an instance of the problem with at most *cutoff* processes. Verification and synthesis methods based on *threshold automata* [19] take a sketch automaton (whose transitions have incomplete guard conditions capturing the number of received messages), and complete the guards towards satisfying program specifications. QBF-based bounded synthesis [20] takes an incomplete design (a.k.a. *sketch*) of a protocol and uses bounded synthesis towards generating fault-tolerant protocols in a bounded space. Mirzaie *et al.* [21] present (i) cutoffs for the detection of deadlocks and closure violation, and (ii) sufficient conditions for livelock-freedom in parameterized systems.

While methods for verification [22], [23], [24], [25], [5], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35] and synthesis of parameterized programs inspire our work, they (1) are mainly based on cut-offs and bounded/parameterized synthesis from temporal logic specifications; (2) make assumptions about synchrony, weak/strong fairness and complete knowledge of the network for each process; (3) are computationally expensive (in part because of the iterative nature of bounded and parameterized synthesis) and sensitive to their input (due to using SMT solvers), and (4) mostly focus on safety and local liveness properties that are specified in terms of the neighborhood of a proper subset of processes.

VI. Conclusions and Future Work

We investigated the problems of verifying and synthesizing parameterized protocols on symmetric unidirectional rings for leadsto properties $R \rightsquigarrow Q$. We showed that the verification problem remains undecidable even for constant-space and deterministic processes and for global state predicates R and Q that are formed by the conjunction of symmetric local state predicates; i.e., conjunctive predicates. We then proved that synthesizing symmetric protocols that satisfy $R \rightsquigarrow Q$ on uni-rings is actually decidable. We presented a sound and complete synthesis algorithm that takes two state predicates R and Q, and generates the parameterized code of a symmetric protocol (on a uni-ring) that satisfies $R \rightsquigarrow Q$. We illustrated the proposed synthesis algorithm in the context of a Sum-Not-2 and an agreement protocol. We are currently working on the implementation of our synthesis algorithm as well as synthesizing protocols on other topologies (e.g., mesh, torus).

REFERENCES

[1] I. Suzuki, "Proving properties of a ring of finite-state machines," *Information Processing Letters*, vol. 28, no. 4, pp. 213–214, Jul. 1988.

[2] A. Pnueli and R. Rosner, "Distributed reactive systems are hard to synthesize," in *Proceedings of 31st IEEE Symposium on Foundation of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1990, pp. 746–757.

[3] A. Ebnenasir and A. Klinkhamer, "Topology-specific synthesis of self-stabilizing parameterized systems with constant-space processes," *IEEE Transactions on Software Engineering*, 2019, in Press.

[4] A. Klinkhamer and A. Ebnenasir, "On the verification of livelock-freedom and self-stabilization on parameterized rings," *ACM Transactions on Computational Logic*, vol. 20, no. 3, pp. 1–36, 2019.

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

[6] G. Varghese, "Self-stabilization by local checking and correction," Ph.D. dissertation, MIT, 1993.

[7] M. G. Gouda and F. F. Haddix, "The stabilizing token ring in three bits," *Journal of Parallel and Distributed Computing*, vol. 35, no. 1, pp. 43–48, May 1996.

[8] 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, pp. 1–8.

[9] A. Ebnenasir, "Verification and synthesis of symmetric uni-rings for leads-to properties," *arXiv preprint arXiv:1905.09726*, 2019, https://arxiv.org/abs/1905.09726.

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

[11] A. P. Klinkhamer and A. Ebnenasir, "Synthesizing parameterized self-stabilizing rings with constant-space processes," in *International Conference on Fundamentals of Software Engineering*. Springer, 2017, pp. 100–115.

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

[13] M. Gouda, "The theory of weak stabilization," in *5th International Workshop on Self-Stabilizing Systems*, ser. Lecture Notes in Computer Science, vol. 2194, 2001, pp. 114–123.

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

[15] P. C. Attie and E. A. Emerson, "Synthesis of concurrent systems with many similar processes," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 20, no. 1, pp. 51–115, 1998.

[16] B. Finkbeiner and S. Schewe, "Bounded synthesis," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 5-6, pp. 519–539, 2013.

[17] L. De Moura and N. Bjørner, "Satisfiability modulo theories: introduction and applications," *Communications of the ACM*, vol. 54, no. 9, pp. 69–77, 2011.

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

[19] M. Lazic, I. Konnov, J. Widder, and R. Bloem, "Synthesis of distributed algorithms with parameterized threshold guards," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 95. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[20] A. Gascón and A. Tiwari, "Synthesis of a simple self-stabilizing system," *arXiv preprint arXiv:1407.5392*, 2014.

[21] N. Mirzaie, F. Faghih, S. Jacobs, and B. Bonakdarpour, "Parameterized synthesis of self-stabilizing protocols in symmetric rings," in *22nd International Conference on Principles of Distributed Systems, OPODIS*, 2018, pp. 29:1–29:17.

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

[23] C. N. Ip and D. L. Dill, "Verifying systems with replicated components in Murphi," *Formal Methods in System Design*, vol. 14, no. 3, pp. 273–310, 1999.

[24] E. A. Emerson and K. S. Namjoshi, "On reasoning about rings," *International Journal of Foundations of Computer Science*, vol. 14, no. 4, pp. 527–550, 2003.

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

[26] A. Sánchez and C. Sánchez, "Parametrized verification diagrams," in *Temporal Representation and Reasoning (TIME), 2014 21st International Symposium on*. IEEE, 2014, pp. 132–141.

[27] 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*, 1989, pp. 68–80.

[28] O. Grinchtein, M. Leucker, and N. Piterman, "Inferring network invariants automatically," in *Automated Reasoning*. Springer, 2006, pp. 483–497.

[29] K. L. McMillan, "Parameterized verification of the flash cache coherence protocol by compositional model checking," in *Correct hardware design and verification methods*. Springer, 2001, pp. 179–195.

[30] 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*. Springer, 2000, pp. 172–187.

[31] F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni, "Generalization strategies for the verification of infinite state systems." *TPLP*, vol. 13, no. 2, pp. 175–199, 2013.

[32] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili, "Regular model checking," in *CAV*, 2000, pp. 403–418.

[33] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena, "A survey of regular model checking," in *CONCUR*, 2004, pp. 35–48.

[34] 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*. ACM, 2016, pp. 185–196.

[35] 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), 2016*. IEEE, 2016, pp. 101–108.