Computer Science Technical Report

Local Reasoning for Global Convergence of Parameterized Rings

Aly Farahat and Ali Ebnenasir

Michigan Technological University

Computer Science Technical Report
CS-TR-11-04
November 2011

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

Local Reasoning for Global Convergence of Parameterized Rings

Aly Farahat and Ali Ebnenasir

November 2011

Abstract

This report demonstrates a method that can generate Self-Stabilizing (SS) *parameterized* protocols that are generalizable; i.e., correct for arbitrary number of finite-state processes. Specifically, we present *necessary and sufficient* conditions for deadlock-freedom specified in the local state space of the representative process of parameterized rings. Moreover, we introduce sufficient conditions that guarantee livelock-freedom in arbitrary-sized unidirectional rings. More importantly, we sketch a methodology for automated design of global convergence in the local state space of the representative process. We illustrate our method in the context of several examples including maximal matching, agreement, two-coloring, three-coloring and sum-not-two protocols.

Contents

1 Introduction **3**

2 Preliminaries **4**
2.1 Parameterized Protocols 4
2.2 Computations and Execution Semantics 4
2.3 Closure, Convergence and Self-Stabilization 5

3 Problem Statement **5**

4 Deadlock-Freedom **6**

5 Livelock-Freedom **9**

6 Application in Automated Addition of Convergence to Non-Stabilizing Protocols **15**
6.1 Synthesis Methodology 16
6.2 Further Examples 17

7 Discussion and Related Work **19**

8 Conclusion and Future Work **21**

1 Introduction

Self-Stabilizing (SS) network protocols have increasingly become important as today's complex distributed systems are subject to different kinds of transient faults such as soft errors, loss of coordination and bad initialization. A SS protocol *converges* from any network configuration/state to a set of legitimate states when transient faults occur [1–3], i.e., *convergence*. Once converged, a SS protocol remains in legitimate states as long as no faults occur; i.e., *closure*. Self-stabilization is also an important property in the design of self-adaptive distributed systems. Nonetheless, the design and verification of SS systems are difficult tasks in part due to the fact that global convergence should be achieved while each process is aware of only its *locality*. (The locality of a process P includes any neighboring process whose state is readable for P.) Another factor that complicates global convergence is *interference*; i.e., the actions of one process may cancel out the effect of the actions of another process towards achieving global recovery. To facilitate the design and verification of SS protocols, this report presents a method for the design of parameterized SS protocols that are correct-by-construction.

There are numerous methods for the manual design and after-the-fact verification of SS protocols [4–8] most of which provide little guidance for designers as to how a protocol should be redesigned if it fails to meet both closure and convergence. For example, layering and modularization techniques [9–12] define a strictly decreasing *ranking function* over a hierarchically-partitioned state space in order to ensure that the local actions of processes can only decrease the ranking function, thereby converging layer by layer. Several researchers present local checking and correction for global recovery [7,13,14], where correcting the local state of a process does not necessarily corrupt the state of its neighbors. Constraint satisfaction methods [6] verify the non-interference of convergence actions by checking a set of sufficient conditions on a graph representing the dependencies of the local constraints of processes. Methods for compositional design [8] use ready-to-use detector/corrector components along with a set of correction and corruption relations defined in the locality of components. Distributed reset [11] propagates a wave of reset throughout the network, which is a computationally expensive method. The aforementioned approaches require developers' ingenuity for the design of convergence actions and lack systematic mechanisms for redesign when a protocol fails to ensure convergence. Moreover, most existing automated techniques [13, 15–17] are based on systematic exploration of the global state space of protocols, and the generated solutions are not provably generalizable; i.e., there are no guarantees that if the number of processes is increased, then self-stabilization will be preserved.

This report presents a local reasoning method for the design of global convergence in *parameterized* protocols with the ring topology. In a parameterized protocol, the code of each process is instantiated from the code of a *representative process* by variable substitution.[1] The entire reasoning in the proposed method is performed in the local state space of the representative process. To ensure convergence to a set of legitimate states I (specified as the conjunction of a set of local constraints), starting from any state $s \in \neg I$, every execution of the protocol from s should eventually reach a state in I. Thus, a protocol must ensure that it is deadlock-free in $\neg I$. Moreover, there must be no cycles formed by processes' actions such that all states of the cycle belong to $\neg I$; i.e., *livelock*-freedom. For a parameterized protocol, deadlock/livelock-freedom properties must hold for any number of processes in the ring. To address this problem, we present *necessary and sufficient* conditions specified in the local state space of the representative process for deadlock-freedom in the global state space of the ring (with an arbitrary number of processes). Moreover, we introduce sufficient conditions that guarantee livelock-freedom in arbitrary-sized unidirectional rings. Our sufficient conditions are weaker than what is proposed in existing methods. For instance, as demonstrated in Section 6, it is unclear how existing methods [6, 8] can be used to design convergence for an agreement protocol. We demonstrate our preliminary results on how the proposed approach can enable automated design of convergence in the local state space of the representative process. We apply our necessary and sufficient conditions throughout a methodology for the design of several parameterized SS protocols on a ring including maximal matching, agreement, coloring and a sum-not-two protocols.

Organization. Section 2 presents preliminary concepts and definitions. Section 3 formally states the problem of designing convergence. We present a necessary and sufficient condition for deadlock-freedom in parameterized rings in Section 4. In Section 5, we introduce the notion of a local transition graph and

[1] In other words, the code of one process can be obtained from the code of another process by a simple variable re-naming/re-indexing.

illustrate how we use it to reason about non-terminating computations in unidirectional rings. We sketch a methodology, supported by examples, for design of convergence in unidirectional rings in Section 6. Section 7 discusses the related work. We make concluding remarks and outline future work in Section 8.

2 Preliminaries

In this section, we present definitions of parameterized protocols, convergence and self-stabilization. The definitions of convergence and self-stabilization are adapted from [1, 4, 18, 19].

2.1 Parameterized Protocols

A parameterized protocol $p(K)$ is a triplet $\langle \Phi_p(K), \Pi_p(K), \Delta_p(K)\rangle$ where K is an integer parameter, and $\Phi_p(K) = \{v_0, \cdots, v_{M(K)-1}\}$ is a set of $M(K)$ variables where M depends on K. Each variable v_i in $\Phi_p(K)$ has a finite domain D_i ($0 \leq i \leq M(K)$). $\Pi_p(K) = \{P_0, \cdots, P_{K-1}\}$ is a set of K similar processes. We represent the set of *similar* processes by a template/representative process P_r ($0 \leq r \leq K-1$), where $P_r = \langle R_r, W_r, \delta_r\rangle$ is a triplet such that $R_r \subset \Phi_p(K)$ is a subset of variables (each indexed by a function of r) that process P_r can read. The cardinality of R_r is a constant independent of K. The *locality* of P_r is the set of variables in R_r. $W_r \subset \Phi_p(K)$ is a subset of variables that process P_r can write. We assume that $W_r \subset R_r$; i.e., P_r can only write variables that it can read.

A *global state* of $p(K)$ is a valuation of every variable in $\Phi_p(K)$. The *global state space* $S_p(K)$ is the set of all possible global states of $p(K)$. A *global state predicate* is any subset of $S_p(K)$ specified as a Boolean expression over variables of Φ_p. We say a global state predicate X *holds in a global state* s, denoted $s \in X$, *if and only if* (*iff*) X evaluates to true at s. The value of variable $v \in \Phi_p(K)$ at global state s is denoted $v(s)$. A *global transition* t of $p(K)$ is a pair of global states (s, s'): s is the source state of t and s' is the target state of t. Likewise, a *local state* s^l_r of P_r is a valuation of the variables in R_r ($0 \leq r \leq K-1$). The *local state space* S^l_r is the set of all possible local states P_r. A *local state predicate* is any subset of S^l_r specified as a Boolean expression over variables of R_r. We say a local state predicate X_r *holds in a local state* s^l_r denoted, $s^l_r \in X_r$ iff X_r evaluates to true at s^l_r. The value of a variable $v \in R_r$ at local state s^l_r is denoted $v(s^l_r)$. A *local transition* t^l of P_r is a pair of local states $(s^l_r, {s^l_r}')$ of P_r such that, $\forall v \in (R_r - W_r) : v(s^l_r) = v({s^l_r}')$. δ_r denotes the set of local transitions of P_r ($0 \leq r \leq K-1$).

The *projection* $s \downarrow Var$ of a global state $s \in S_p(K)$ on a set of variables $Var \in \Phi_p(K)$ is a valuation of every variable $v \in Var$ such that $v(s) = v(s \downarrow Var)$. Likewise, we define the *projection* of a global transition $(s, s') \downarrow Var$ as the pair $((s \downarrow Var), (s' \downarrow Var))$. Every local state s^l_r of P_r is mapped to a set of global states $g^K(s^l_r) = \{s \in S_p(K) : \forall v \in R_r : v(s) = v(s^l_r)\}$. Likewise, every local transition t^l_r corresponds to a *group* of global transitions $g^K(s^l_r, {s^l_r}') = \{(s, s') \in S_p(K) \times S_p(K) : (\forall v \in R_r : v(s) = v(s^l_r) \wedge v(s') = v({s^l_r}')) \wedge (\forall v \notin W_r : v(s) = v(s'))\}$. Thus, $g^K(\delta_r)$ represents the set of global transitions of P_r in $p(K)$. The set of global transitions of $p(K)$ is the union of the set of global transitions of each process P_r, i.e.; $\Delta_p(K) = \cup_{r=0}^{K-1} g^K(\delta_r)$.
Notational convention. For abbreviation, we denote universally quantified statements over K by omitting K. For instance, we denote $\forall K : p(K)$ converges to $I(K)$ by p converges to I, and $\forall K : s \in g^K(s^l_r)$ by $s \in g(s^l_r)$.
Protocol Representation. We use Dijkstra's guarded commands language [20] as a shorthand for representing the set of local transitions of P_r (i.e., δ_r). A guarded command (i.e., *action*) is of the form $L : grd_r \rightarrow stmt_r$, where L is an optional label, grd_r is a Boolean expression in terms of variables in R_r; i.e., a local predicate of P_r, and $stmt_r$ is a statement that updates variables of W_r *atomically*. Formally, an action $grd_r \rightarrow stmt_r$ includes a set of local transitions $(s^l_r, {s^l_r}')$ such that grd_r holds in every local state s^l_r and the atomic execution of $stmt_r$ results in a local state ${s^l_r}'$ of P_r. An action $grd_r \rightarrow stmt_r$ is *enabled* in a global (local) state s (respectively, s^l_r) iff grd_r holds at s (respectively, s^l_r). The process P_r is *enabled* in s (respectively, s^l_r) iff there exists an action of P_r that is enabled at s (respectively, s^l_r).

2.2 Computations and Execution Semantics

A *computation* of a protocol p is a sequence $\sigma = \ll s_0, s_1, \cdots \gg$ of global states that satisfies the following conditions: (1) for each global transition (s_i, s_{i+1}) ($i \geq 0$) in σ, there exists an action $grd_r \rightarrow stmt_r$ in some process P_r ($0 \leq r \leq K-1$) such that grd_r holds at s_i and the execution of $stmt_r$ at s_i yields s_{i+1}, and (2)

σ is *maximal* in that either σ is infinite or if it is finite, then σ reaches a global state s_f where no action is enabled. In other words, a computation is generated by a nondeterministic interleaving of actions and can be extended wherever possible. A *computation prefix* of a protocol p is a *finite* sequence $\sigma = \ll s_0, s_1, \cdots, s_m \gg$ of global states, where $m \geq 0$, such that each transition (s_i, s_{i+1}) in σ $(0 \leq i < m)$ belongs to some action $grd_r \rightarrow stmt_r$ in P_r for some $0 \leq r \leq K-1$. The *projection of a protocol* p on a non-empty state predicate X, denoted as $\Delta_p|X$, is a protocol with the set of global transitions $\{(s_0, s_1) : (s_0, s_1) \in g(\Delta_p) \wedge s_0, s_1 \in X\}$.

2.3 Closure, Convergence and Self-Stabilization

A state predicate X is *closed in an action* $grd_r \rightarrow stmt_r$ iff executing $stmt_r$ from any state $s \in (X \wedge grd_r)$ results in a state in X. We say a state predicate X is *closed in a protocol* p iff X is closed in every action of p. In other words, *closure* [19] requires that every computation that starts in X remains in X.

Let I be a state predicate. We say that a protocol p *strongly converges to* I iff from any state, *every* computation of p reaches a state in I. A protocol p *weakly converges* to I iff from any state, *there exists* a computation of p that reaches a state in I. A protocol p is *strongly (respectively, weakly) self-stabilizing* to a state predicate I iff (1) I is closed in p and (2) p strongly (respectively, weakly) converges to I.

I is *locally conjunctive* iff for every K, $I(K)$ is a conjunction of K local state predicates LC_r, where LC_r specifies a local state predicate of P_r; i.e., $I(K) = \bigwedge_{r=0}^{K-1} LC_r$. In this report, we assume that I is locally conjunctive.

An *enablement* of P_r is a local state where P_r is enabled. A *corruption* (*non-corruption*) with respect to I is an enablement s_r^l of P_r such that $s_r^l \notin LC_r$ (respectively, $s_r^l \in LC_r$). Let I be a locally conjunctive closed predicate for p, a process P_r in a non-corrupt local state will never corrupt its own local state.

Deadlocks and Livelocks. A *global deadlock* state s_d has no outgoing global transitions (i.e., no process is enabled), and no action of P_r is enabled in a *local deadlock* state s_d^l of P_r. A global deadlock state s_d (respectively, s_d^l) is legitimate iff $s_d \in I$ (respectively, $s_d^l \in LC_r$), otherwise s_d (respectively, s_d^l) is illegitimate. Notice that a parameterized protocol $p(K)$ is in a global deadlock state iff every process $P_r \in \Pi_p(K)$ $(0 \leq i \leq K-1)$ is in a local deadlock state. A global deadlock is illegitimate iff there exists a process P_r whose local deadlock is illegitimate.

In a finite-state parameterized protocol $p(K)$, a livelock for a state predicate $I(K)$ is a computation $\ll sc_0, sc_1, \cdots, sc_{m-1}, \cdots \gg$ where $\forall i : i \in \mathbb{N} : sc_{i+m} = sc_i$ and $\forall i : 0 \leq i \leq m-1 : sc_i \notin I(K)$; i.e., an infinite repetition of a sequence of global states outside $I(K)$.

Proposition 2.1. A protocol p *strongly converges to* I iff there are no global deadlock states in $\neg I$ and no livelocks in $\Delta_p \mid \neg I$.

When it is clear from the context, we shall omit the set of legitimate states I; e.g., instead of saying 'a livelock for $I(K)$', we say 'a livelock'. Furthermore, we assume an interleaving semantics, every global transition of L belongs to only one local transition. The sequence of local transitions of L can be obtained by projecting every global transition of some P_i in L over R_i.

3 Problem Statement

Consider a non-stabilizing parameterized protocol $p(K)$ with a set of global transitions $\Delta_p(K)$ and a locally conjunctive state predicate $I(K)$ closed in $p(K)$, for all K. Our objective is to design a revised version of $p(K)$, denoted $p_{ss}(K)$, that is strongly converging to $I(K)$, for all K. We require that the behaviors of $p_{ss}(K)$ from any state in $I(K)$ remain the same as $p(K)$, for all K. Thus, during the addition of convergence to $p(K)$, no states (respectively, transitions) should be added to or removed from $I(K)$ (respectively, $\Delta_p(K)|I(K)$). This way, the behaviors of $p_{ss}(K)$ are exactly the same as $p(K)$'s starting from any state inside $I(K)$, for all K. Moreover, for all K, if $p_{ss}(K)$ starts in a state outside $I(K)$, $p_{ss}(K)$ will provide strong convergence to $I(K)$.

Problem 3.1: Designing Convergence

- **Input**: (1) A protocol $p(K)$ with the set of transitions $\Delta_p(K)$, and (2) A *non-empty* locally conjunctive state predicate $I(K)$ such that $I(K)$ is closed in $p(K)$, for all K.

- **Output**: A protocol $p_{ss}(K)$ with the set of transitions $\Delta_{p_{ss}}(K)$ such that the following constraints are met for all K:

  (1) $I(K)$ is unchanged, (2) $(\Delta_{p_{ss}}(K) \neq \emptyset) \wedge (\Delta_{p_{ss}}(K)|I(K) = \Delta_p(K)|I(K))$, and (3) $p_{ss}(K)$ is strongly self-stabilizing to $I(K)$. □

We investigate problem 3.1 for parameterized rings. Convergence of parameterized rings is especially challenging as cyclic corruption of processes may hinder recovery, which is why some researchers consider acyclic topologies for compositional design of self-stabilization [21].

4 Deadlock-Freedom

In this section, we address the following problem: *For a parameterized protocol p with a ring topology and a conjunctive predicate I, determine whether p(K) is deadlock-free outside I(K) for all K without exploring the global state space of p(K).* To address this problem, we define a relation between the local states of the representative process P_r capturing the way the local states of a process are related with the local states of its neighboring processes. Using this relation, we present a necessary and sufficient condition defined in the local state space of P_r for global deadlock-freedom of p.

The Right Continuation Relation for Rings. Due to the locality of each process P_r, a local state s_i^l of process P_i restricts the allowable set of local states for each successor P_j of P_i. P_j *is a successor* of P_i (P_i is a predecessor of P_j) iff $W_i \cap R_j \neq \emptyset$. A local state of ${s_r^l}'$ is a *continuation* of a local state s_r^l iff there exists two processes P_i and P_j such that ${s_r^l}'$ is a local state of P_j, s_r^l is a local state of P_i and P_j is a successor of P_i.

In a bidirectional ring of size K, $P_{(i+1)\mathsf{mod}K}$ and $P_{(i-1)\mathsf{mod}K}$ are right and left successors of P_i, respectively. As such, the *right (left) continuation* of a local state s_i^l of P_i is a local state s_{i+1}^l of P_{i+1} (s_{i-1}^l of P_{i-1}) such that for every $x \in R_i \cap R_{i+1}, x(s_i^l) = x(s_{i+1}^l)$ (respectively $x \in R_i \cap R_{i-1}, x(s_i^l) = x(s_{i-1}^l)$)[2]. Since all processes are similar, s_i^l and s_{i+1}^l are local states of the representative process P_r. Notice that for a unidirectional ring, we can only define a right continuation relation.

Definition 4.1. *A directed Right Continuation Graph (RCG_p) of a ring is a pair (V_r, S_R) such that:*

1. V_r is a set of vertices representing local states of the representative process P_r.
2. $S_R = \{(s_1^l, s_2^l) \in V_r \times V_r : \forall x \in R_r \cap R_{r+1} : x(s_1^l) = x(s_2^l)$ and P_{r+1} is a successor of $P_r\}$.
3. In bidirectional rings, we choose to include arcs in S_R connecting to local states of only the right successor. In fact, S_R is *sufficient* for determining how the ring is constructed even if it is bidirectional.

Our definition of continuation relation naturally extends to network topologies other than rings. For instance, we construct RCG of a tree from the locality of a non-root process that includes the writable variables of its *parent*, itself and its *children*.

Example 4.1. In maximal matching over a bidirectional ring, all processes are similar. P_r has P_{r+1} as a successor and P_{r-1} as a predecessor. $R_r = \{m_{r-1}, m_r, m_{r+1}\}$, $W_r = \{m_r\}$. $W_r \cap R_{r+1} = W_r \cap R_{r-1} = \{m_r\}$. $R_r \cap R_{r+1} = \{m_r, m_{r+1}\}$. D_r = {left, right, self} are values of m_r meaning that P_r matches with its predecessor, successor or none of them, respectively. We represent the right continuation relation over the local state space of P_r in Figure 1. The set of local legitimate states LC_r is defined by the Boolean expression $(m_r = \text{right} \wedge m_{r+1} = \text{left}) \vee (m_{r-1} = \text{right} \wedge m_r = \text{left}) \vee (m_{r-1} = \text{left} \wedge m_r = \text{self} \wedge m_{r+1} = \text{right})$.

RCG_p captures the relation with all possible local states of a successor of P_r. Let $(s_1, s_2) \in S_R$, then P_r is in a local state s_1 when its successor P_{r+1} is in a local state s_2. Due to symmetry, local state spaces of all similar processes are captured by P_r's local state space ${S_r}^l$. We observe that any directed cycle of length L in Figure 1 represents a possible valuation of local states to a ring of processes of size $k \times L$ (k a positive integer). For instance, ⟨rss, ssl, sls, lsl, sll, lll, llr, lrr, rrr, rrs, rsr, srs⟩ represents a ring of 12.k processes in

[2] Addition and subtractions of indices are modulo K.

the global state ⟨self, self, left, self, left, left, left, right, right, right, self, right⟩^{k}. ⟨sss⟩ represents a ring of arbitrary size in a global state where m_r =self ($0 \leq r \leq n-1$) and n is an arbitrary positive integer. An example of a global state in a ring whose size is a multiple of two is ⟨rsr,srs⟩ corresponding to a ring having $2 \times k$ processes in a state ⟨ r,s ⟩^{k}; i.e., repeatedly concatenated k times. ◁

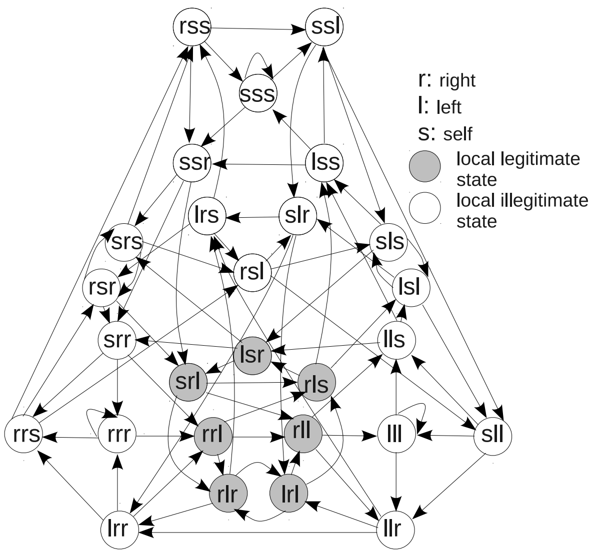

Figure 1: Continuation relation over all local states of Maximal Matching

Theorem 4.2 (Deadlock-Freedom in Parametrized Rings)**.** *A parameterized protocol $p(K)$ over a ring topology is deadlock-free outside $I(K)$ for every K iff the induced subgraph[3] of RCG_p over local deadlocks has no directed cycles containing a local state/vertex in $\neg LC_r$.*

Proof. ⇒: Let $p(K)$ be a parameterized protocol that is deadlock-free outside $I(K)$ for every K. By contradiction, assume that RCG_p has a directed cycle over its local deadlocks $C = \{s^l_0, s^l_1, \cdots, s^l_{n-1}\}$ and for some $0 \leq j \leq n-1$, $s^l_j \notin LC_r$. By definition of RCG_p, s^l_{i+1} is a right continuation of s^l_i for every $0 \leq i < n-1$ and s^l_0 is a right continuation of s^l_{n-1}. By assigning to P_i the local state s^l_i for $0 \leq i \leq n-1$, we construct a ring R of size $k \times n$ (k is a positive integer) in which every P_i is locally deadlocked. Moreover, for some j, P_j is in a local state $s^l_j \notin LC_r$. Because $I(K)$ is locally conjunctive, the corresponding global state of R is a global deadlock outside $I(K)$. This contradicts our premise.

⇐: Let the induced subgraph over local deadlocks in RCG_p have no directed cycles with a local state $s^l_j \notin LC_r$. By contradiction, assume that there exists a protocol $p(K)$ over a ring of size K that is globally deadlocked outside $I(K)$. It follows that every process P_i of $p(K)$ is in a local deadlock s^l_{di} ($0 \leq i \leq K-1$) among which there exists a local deadlock $s^l_{dj} \notin LC_r$. By definition of the continuation relation, RCG_p captures *every possible* right continuation of every local state of P_r. Hence, for every $0 \leq i \leq K-1$, $(s^l_{di}, s^l_{di+1}) \in \text{RCG}_p$. Since $p(K)$ is a ring of local deadlocks, RCG_p's induced subgraph over local deadlocks should have a directed cycle containing s^l_{dj}, which is a contradiction. □

We illustrate the application of Theorem 4.2 by the following examples.

Example 4.2 (Deadlock-Free Generalizable Maximal Matching)**.**

We consider the following parameterized protocol for maximal-matching on a bidirectional ring. We automatically synthesized this protocol for $K = 6$ using the STabilization Synthesizer tool (STSyn) [17].

[3]An induced subgraph $G' = (V', E')$ of a directed graph $G = (V, E)$ is such that $V' \subset V$, E' is the maximum subset of E such that the source and target vertices of every arc in E' are in V'.

A_1: m_{r-1} =left $\wedge m_r \neq$ self $\wedge m_{r+1}$ =right $\rightarrow$ m_r := self

A_2: m_{r-1} =self $\wedge m_r$ =self $\wedge m_{r+1}$ =self $\rightarrow$ m_r := right | left

A_3: m_{r-1} =right $\wedge m_r$ =self $\rightarrow$ m_r :=left
m_r =self $\wedge m_{r+1}$ =left $\rightarrow$ m_r :=right

A_4: m_{r-1} =right $\wedge m_r$ =right $\wedge m_{r+1} \neq$ left$\rightarrow$ m_r :=left
$m_{r-1} \neq$right $\wedge m_r$ =left $\wedge m_{r+1}$ =left $\rightarrow$ m_r :=right

A_5: m_{r-1} =self $\wedge m_r \neq$left $\wedge m_{r+1}$ =right $\rightarrow$ m_r :=left
m_{r-1} =left $\wedge m_r \neq$right $\wedge m_{r+1}$ =self $\rightarrow$ m_r :=right

We model-checked this protocol for different sizes of ring (5,6,7 and 8 processes) and demonstrated its deadlock freedom. We illustrate how the continuation relation over the protocols local deadlocks implies its deadlock freedom for any number of processes. $\triangleleft$

Figure 2 illustrates RCG$_p$ of Example 4.2 induced over its local deadlocks. As we can see, there are no directed cycles that include local illegitimate states. This proves the deadlock freedom of the parametrized maximal matching protocol in Example 4.2.

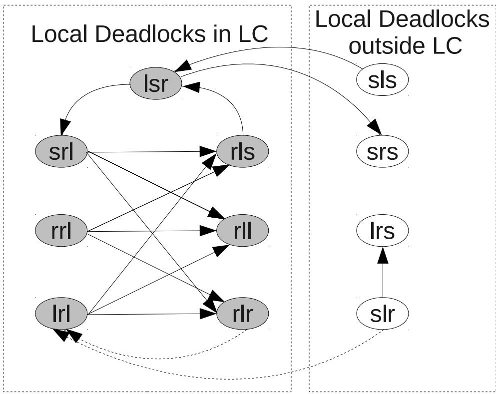

Figure 2: Continuation Relation over local deadlocks of Example 4.2

Example 4.3 (Non-generalizable Maximal Matching)**.**

We automatically synthesized the following protocol that stabilizes only for 5 processes and has deadlocks for rings of sizes 6. We illustrate how the right continuation relation helps us reason about global deadlocks.

B_1: m_{r-1} =left $\wedge m_r \neq$ self $\wedge m_{r+1}$ =right $\rightarrow$ m_r := self

B_2: m_{r-1} =right $\wedge m_r$ =self $\wedge m_{r+1}$ =left $\rightarrow$ m_r := right
m_{r-1} =self $\wedge m_r$ =self $\wedge m_{r+1}$ =self $\rightarrow$ m_r := right

B_3: m_{r-1} =right $\wedge m_r$ =right $\wedge m_{r+1} = left \rightarrow$ m_r :=left
m_{r-1} =self $\wedge m_r$ =self $\wedge m_{r+1}$ =right $\rightarrow$ m_r :=left

B_4: m_{r-1} =right $\wedge m_r \neq$left $\wedge m_{r+1} \neq$ left $\rightarrow$ m_r :=left
$m_{r-1} \neq$right $\wedge m_r \neq$right $\wedge m_{r+1}$ =left $\rightarrow$ m_r :=right

Figure 3 illustrates a subgraph of the RCG in Figure 1 that is induced over the local deadlocks of the maximal matching protocol presented in Example 4.3. There are only two directed cycles having local illegitimate deadlocks in Figure 3. Both cycles include the local state ⟨*left,left,self*⟩. The first directed cycle has length 4:⟨lls, lsr, srl, rll⟩ and represents global deadlocks ⟨left,self,right,left ⟩^{k} in rings whose size is a multiple of 4. The second directed cycle has length 6:⟨lls,lsr,srl,rlr,lrl,rll⟩ and represents global deadlocks ⟨*left,self,right,left,right,left* ⟩^{k} in rings whose size is a multiple of 6. We deduce that Example 4.3 is deadlock free for ring sizes that are not multiples of 4 or 6; i.e, two-thirds of the family of rings. Moreover, *resolving* the local deadlock ⟨*left,left,self*⟩ renders RCG$_p$ without cycles including local states in $\neg LC_r$; i.e., $p(K)$ becomes deadlock free for any ring size K. Including a local transition whose source state is a local deadlock s^l_d *resolves* the local deadlock; i.e., s^l_d is not anymore a local deadlock. ◁

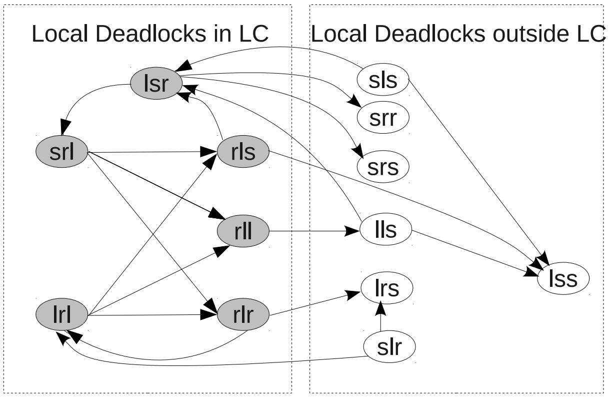

Figure 3: Continuation Relation over local deadlocks of Example 4.3

5 Livelock-Freedom

In this section, we focus on the following problem: *For a protocol $p(K)$ with a ring topology and a conjunctive predicate $I(K)$, determine whether $p(K)$ is livelock-free outside $I(K)$ for all K without exploring the global state space of $p(K)$.*

A necessary condition for strong convergence of a protocol p to I is the absence of livelocks in $\neg I$. A sufficient condition for livelock-freedom is to guarantee that convergence actions are *non-corrupting.* However, non-corruption is not necessary for livelock-freedom. For instance, Dijkstra's token-ring [1] converges to a global state where only one token is in the ring despite its corrupting convergence actions. Moreover, sometimes it is impossible to find a strongly stabilizing protocol with no corrupting convergence actions; i.e, for some protocols, every non-corrupting livelock-free protocol has deadlocks in $\neg I$.

Therefore, it is necessary to weaken the sufficient condition of non-corruption while maintaining livelock-freedom. To this end, we study livelocks in rings of arbitrary sizes. Rings are attractive in this regard because of their susceptibility to circulating corruptions, unlike acyclic topologies. A circulating corruption occurs when an action of a process P_i corrupts the local state of its successor P_{i+1} for every process in the ring. For simplicity, we investigate this problem for unidirectional rings under the following assumptions:

1. Every process P_i is *self- terminating.* As such, every sequence of local transitions of P_i terminates in a local deadlock.

2. No process P_i has *self-enabling actions.* An action A: $\mathsf{guard}_A \rightarrow \mathsf{statement}_A$ is self-enabling if there exists a global transition $(s_g, s'_g) \in A$ such that $s_g \in \mathsf{guard}_A$ and $s'_g \in \mathsf{guard}_A$. Intuitively, an action B is *self-disabling* iff B disables guard_B after executing $\mathsf{statement}_B$.

Assumption 2 is at no loss of protocol's generality because self-enabling actions can be transformed into self-disabling without adding neither deadlocks nor livelocks in $\neg I$. If P_i is self enabling, then for some local state s^l_{i1} of P_i, there exists a sequence of local states $\langle s^l_{i1}, s^l_{i2}, \cdots, s^l_{ik}\rangle$ of P_i such that $(s^l_{ij}, s^l_{i(j+1)})$ is a local transition of P_i $(1 \leq j \leq k-1)$ and s^l_{ik} is a local deadlock (Item 1 prohibits local non-terminating computations). We substitute, every local transition $(s^l_{ij}, s^l_{i(j+1)})$, where $1 \leq j \leq k-1$, with (s^l_{ij}, s^l_{ik}). This substitution renders P_i self disabling and preserves reachability to s^l_{ik} from every local state s^l_{ij}. Moreover, it does not introduce new local deadlock states.

Definition 5.1 (Network Topology Graph)**.** *Topology*$_p(K) = (\Pi_p(K),$ *Links*$_p)$ *where Links*$_p = \{(P_i, P_j) : ((P_i, P_j) \in \Pi_p(K) \times \Pi_p(K))$ *and* $((P_i, P_j) : R_j \cap W_i \neq \emptyset)\}$.

Topology$_p(K)$ is a directed graph for a protocol of definite size K whose vertex set is $\Pi_p(K)$ and arc set illustrating the neighborhood relation.

Lemma 5.2 (Enablement Propagation)**.** *Let* $C = \ll c_1, \cdots, c_k, \cdots \gg$ *be a computation of a protocol* p. $\forall k > 1$: **If** $(\exists j : P_j$ *is* enabled *in* c_k **and** P_j *is* disabled *in* $c_{k-1})$ **then** $\exists i : (c_{k-1}, c_k) \in g(\delta_i)$ **and** P_j *is the successor of* P_i.

Proof. P_j is not enabled in c_{k-1} and enabled in c_k means that (c_{k-1}, c_k) writes a variable $x \in R_j$. Then $x \in W_i$ of some process such that $(c_{k-1}, c_k) \in g(\delta_i)$. It follows that $\{x\} \subset W_i \cap R_j$, hence P_j is a successor of P_i. □

The significance of Lemma 5.2 is to illustrate that in the course of a program computation, a disabled process is enabled only by the action of its predecessor. In other words, a process can only pass *enablement* to its successor. To represent the propagation of enablement in the local state space of the representative process P_r, we augment the RCG with the local transitions of P_r, called t-arcs. Thus, the augmented RCG has two types of arcs: s-arcs that represent the continuation relation and t-arcs representing local transitions of P_r. We call the new RCG, the Local Transition Graph (LTG).

Under interleaving semantics, we locally capture global computations by interleaving the local transitions of enabled processes; i.e., t-arcs, with the transfer of control to possible local states of successor processes represented by a sequence of s-arcs.

Definition 5.3. The Local Transition Graph (LTG) of p is a triplet LTG$_p = (V, T, S)$. V is a set of vertices representing local states of the representative process P_r. T is the set of directed arcs representing the local transitions/t-arcs of the representative process P_r. S captures the continuation relation; i.e., s-arcs representing the set of *right* (*left*) s-arcs.

We construct LTG$_p$ as follows:

1. For the representative process P_r, assign a vertex in V corresponding to each local state of P_r. V represents ${S_r}^l$.
2. In V, add a t-arc (v_r, v'_r) to T to represent a local transition of P_r.
3. For every local state/vertex in V add an s-arc (v_r, v'_r) to S if v_r represents a local state of P_r and v'_r represents a possible local state of a successor of P_r.

Notice that LTG$_p$ does not depend on K since every process P_r is captured by V regardless of the number of similar processes. Moreover, for bidirectional rings, an s-arc $(v_r, v'_r) \in S$ iff $(v'_r, v_r) \in S$.

Example 5.1. We illustrate LTG$_p$ of Example 4.2 in Figure 4. We omitted left s-arcs for readability.

Definition 5.4 (Collision)**.** *Let* $p(K)$ *be a parameterized protocol with a unidirectional ring topology and* P_j *be the successor of* P_i. *Let* s^l_i *and* s^l_j *be local states where* P_i *and* P_j *are both enabled, respectively. A* collision *is an execution of any local transition of* P_i *enabled at* s^l_i.

Lemma 5.5 (Enablement Conservation in a Unidirectional Ring)**.** *Let* $p(K)$ *be a protocol on a unidirectional ring of size* K. *If* L *is a livelock of* $p(K)$, *then in every global state of* L, *the number of enabled processes is the same.*

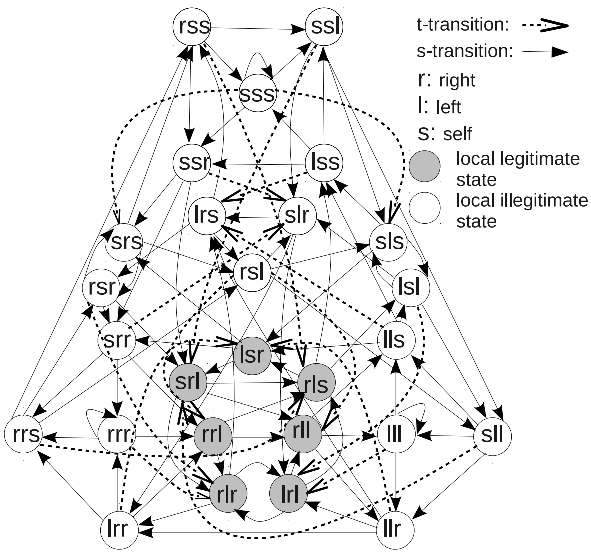

Figure 4: LTG$_p$ of Example 4.2

Proof. Let s be some global state of L. Assume the number of enabled processes at s is $|E|$. From Assumption 2, every local transition of any process P_i disables P_i. Since every process in a unidirectional ring has only one successor, a local transition of any enabled process will not increase $|E|$. It follows that $|E|$ can either stay constant or decrease. However, if an execution of a transition at s decreased $|E|$ to $|E|-1$, thus since $|E|-1$ cannot increase in subsequent transitions, s cannot be re-encountered in the computation of L following s. Therefore, s cannot be in a livelock L. Consequently, $|E|$ is constant in any livelock on a unidirectional ring. □

Corollary 5.6 (Absence of Collisions in Livelocks in Unidirectional Rings)**.** *If L is a livelock on a unidirectional ring then for every global transition t in L, there is no collision t^l such that $t \in g(t^l)$.*

Proof. In a unidirectional ring, a collision decreases the number of enabled processes by 1. This is in contradiction with Lemma 5.5. □

Corollary 5.7 (Insensitivity to Weak Fairness)**.** *Let $p(K)$ be a parameterized protocol on a unidirectional ring of size K. If L is a livelock of $p(K)$ then there is no continuously enabled process in L.*

Proof. Let s_g be a global state of L where every process of $p(K)$ is enabled. Hence, any execution of any enabled process will cause a collision. From Corollary 5.6, s_g cannot be in L. It follows that in every global state of L, there exists a disabled process. According to Lemma 5.2 and 5.5, a constant number of enablements propagate along the arcs of the unidirectional ring. Hence, disabled local states propagate in the opposite direction. Thus, every process in the ring will eventually be disabled. □

Corollary 5.7 implies that the assumption of the existence of a weakly fair scheduler[4] does not simplify the design of livelock-freedom in unidirectional rings because no process is continuously enabled in a livelock on a unidirectional ring.

Lemma 5.8 (Local Illegitimacy)**.** *Let $p(K)$ be a parametrized protocol on a unidirectional ring, If $p(K)$ has a livelock L for some K, then for every global state of L there exists a process P_i in an illegitimate local state.*

[4]A weakly fair scheduler infinitely often executes any action that is continuously enabled.

Proof. Every global state of L is in $\neg I$. Since I is locally conjunctive, for every global state of L, there exists LC_i that evaluated to false by the local state of P_i. In other words, there exists a process P_i whose local state is in $\neg LC_i$. $\square$

In every global state of a livelock L, there exists an enabled process P_i and some process P_j in an illegitimate local state: notice that we do not rule out the possibility of $i = j$, in this case P_i's local state is a corruption.

Lemma 5.8 establishes the existence of a process P_i in a local state $s_i^l \notin LC_i$ in every global state of L.

Lemma 5.9 (Local Corruptions). *Let p be a parameterized protocol on a unidirectional ring. If $p(K)$ has a livelock L for some K, then for some global state of L there exists a process P_i having a corruption.*

Proof. From Lemma 5.8, every global state of L has a process P_i in an illegitimate local state. Lemmas 5.2 and 5.5 establish that enabled local states propagate along a unidirectional ring without collisions. By contradiction, assume that at every global state of L, all enabled processes are in non-corruptions. Due to closure of $I(K)$ in $p(K)$, a propagation of a non-corruption in any process P_i should leave P_i in a local legitimate deadlock. As such, eventually every process P_i will be in a legitimate state. This contradicts Lemma 5.8. Therefore, there exists a global state of L where some P_i is in a corruption. $\square$

Lemma 5.9 helps us identify sufficient conditions for livelock freedom by studying LTG_p.

To understand how livelocks represent themselves in LTG, we observe that each sequence *Sch* of local transitions representing a livelock belongs to an equivalence class of sequences whose local transitions preserve some precedence relation. Lemma 5.11 establishes our observation for a reduction based on an irreflexive partial order. Godefroid [22] originally introduced partial order reduction to simplify automatic verification.

Definition 5.10 (Livelock Induced Precedence Relation $\prec$). *Let the local transitions of a livelock L be represented by a sequence of local transitions Sch=$\ll {t_0}^l, {t_1}^l, \cdots, {t_{n-1}}^l \gg$. We say ${t_i}^l$ precedes ${t_j}^l$, denoted ${t_i}^l \prec {t_j}^l$ iff*

1. *the execution of ${t_i}^l$ enables ${t_j}^l$, or,*
2. *if ${t_j}^l$ executes, then it collides with ${t_i}^l$ enabled in P_i, or,*
3. *if 1 and 2 are false, then there exists ${t_k}^l$ in Sch such that ${t_i}^l \prec {t_k}^l$ and ${t_k}^l \prec {t_j}^l$.*

Example 5.2 (Binary Agreement). A binary agreement protocol on a unidirectional ring has the representative process P_r such that $M(K) = K$, $R_r = \{x_{r-1}, x_r\}$, $W_r = \{x_r\}$, $D_r = \{0, 1\}$ and has the following local transitions.

$$
\begin{aligned}
{t_{10}}^r &: x_{r-1} = 0 \wedge x_r = 1 \rightarrow x_r := 0 \\
{t_{01}}^r &: x_{r-1} = 1 \wedge x_r = 0 \rightarrow x_r := 1
\end{aligned}
$$

Intuitively, P_r local transitions set $x_r = x_{r-1}$ whenever $x_r \neq x_{r-1}$. For $K = 4$, we examine a livelock L such that $L = \ll 1000, 1100, 0100, 0110, 0111, 0011, 1011, 1001 \gg^k$. We represent L by the sequence of local transitions *Sch*=$\ll {t_{01}}^1, {t_{10}}^0, {t_{01}}^2, {t_{01}}^3, {t_{10}}^1, {t_{01}}^0, {t_{10}}^2, {t_{10}}^3 \gg$. Figure 5 illustrates the dependencies imposed by *Sch* between local transitions of L. Since we have only three pairs of independent local transitions, the precedence relation allows $8 = 2^3$ possible precedence-preserving permutations of *Sch*. Two local transitions ${t_i}^l$ and ${t_j}^l$ are independent if and only if ${t_i}^l \nprec {t_j}^l$ and ${t_j}^l \nprec {t_i}^l$. Figure 6 depicts L and another livelock generated by a permutation of *Sch* preserving the same precedence relation in Figure 5. Observe that *Sch* is defined up to cyclic permutations, thus we have to fix the "starting" local transition of all sequences in order to decide their membership in a given precedence-preserving class of sequences.

Lemma 5.11 (Precedence Relation Reduction). *Let $p(K)$ be a protocol on a unidirectional ring of size K. If $p(K)$ has a livelock L, for some K, whose local transitions are represented by a sequence Sch=$\ll {t_0}^l, {t_1}^l, \cdots, {t_{n-1}}^l \gg$ then every precedence-preserving permutation of Sch represents a livelock of $p(K)$.*

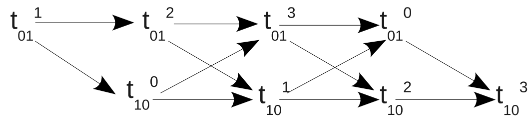

Figure 5: Precedence relation for local transitions in Example 5.2

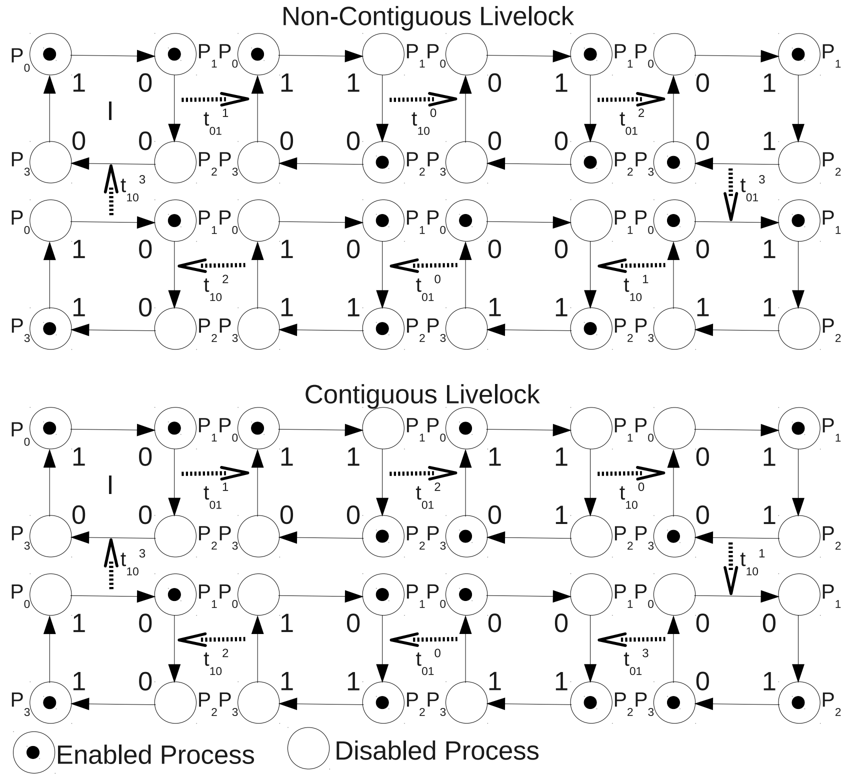

Figure 6: Two precedence-preserving livelocks for Example 5.2. The starting global state is marked by "I"

Proof. Let *Sch'* be a precedence preserving permutation of *Sch* obtained by swapping two arbitrary independent local transitions ${t_i}^l$ and ${t_j}^l$ where $i < j$. Now consider the subsequence *Middle*$=\ll {t_{i+1}}^l, \cdots, {t_{j-1}}^l \gg$ of *Sch*, since swapping of ${t_i}^l$ and ${t_j}^l$ in *Sch'* is precedence preserving, then each of ${t_i}^l$ and ${t_j}^l$ form independent pairs with every local transition in *Middle*. If it is not the case, a swap of ${t_i}^l$ and ${t_j}^l$ would have violated the precedence relation. Consequently, if ${t_k}^l \prec {t_j}^l$ then $k < i$, and if ${t_i}^l \prec {t_k}^l$ then $j < k$. The execution of *Sch'* proceeds as follows. Every transition ${t_k}^l$ for $k < i$ executes exactly as in *Sch*. Now ${t_j}^l$ is enabled since all local transitions preceding it already executed, then ${t_j}^l$ executes as in *Sch*. None of the transitions in *Middle* depends on ${t_i}^l$ nor ${t_j}^l$ and they execute as in *Sch*. ${t_k}^l$ $(k \geq j)$ execute as in *Sch* since all their preceding transitions already executed. Since no local transition has been disabled due to the precedence preserving swap, *Sch'* represents a new livelock L'. □

Using Lemma 5.11, we can reduce our search for livelocks in unidirectional rings to a search for a representative livelock. We choose as representative livelock that we call a *contiguous livelock*. Let L be a livelock on a unidirectional ring having $|E|$ enablements. A *contiguous livelock* C_L has a global state where $|E|$ adjacent processes are enabled as illustrated in Figure 7 (I). The subsequent global states of C_L are such that only the rightmost enablement in the segment of adjacent processes propagates while the remaining $|E|-1$ enablements do not propagate. After $K-|E|$ propagations of the rightmost enablement, a new global state with $|E|$ adjacent enablements is reached. Figure 7 (II, III, and IV) illustrates this scenario for $K = 6$ and $|E| = 3$. Notice that a K times repetition of the scenario in Figure 7 results in a full rotation of the segment of adjacent enablements in an opposite direction to that of the rightmost enablement propagation. It follows from Lemma 5.11 that $p(K)$ has a livelock if and only if it has a contiguous livelock.

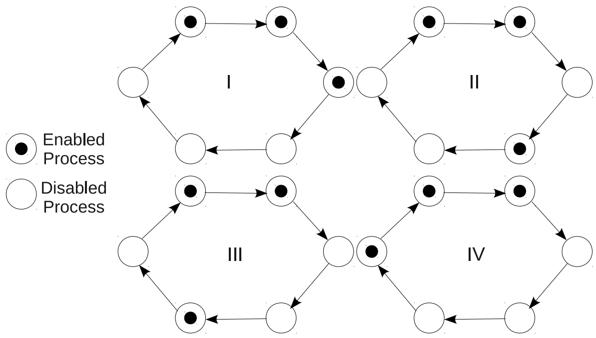

Figure 7: Sequence of enabled processes in a contiguous livelock

Lemma 5.12 demonstrates the kind of structure LTG_p has when $p(K)$ has a contiguous livelock. We call this structure a *contiguous trail* of LTG_p.

Lemma 5.12 (Representation of a Contiguous Livelock in LTG_p)**.** *Let p be a parameterized protocol on a unidirectional ring. If for some K, $p(K)$ has a contiguous livelock C_L with $|E|$ enabled processes, then LTG_p has an alternating trail T_R of the following format.*

1. *if $|E| = 1$, then T_R is an alternating trail of a t-arc followed by an s-arc and vice versa.*
2. *if $|E| > 1$, then T_R is an alternation of two types of walks: w_1 and w_2. w_1 consists of $|E|$ consecutive s-arcs such that every vertex/local state in w_1 has an outgoing t-arc in w_2. w_2 has $2(K - |E|)$ arcs of an alternating walk of t-arcs and s-arcs.*

We call T_R a contiguous trail of LTG_p.

Proof. If $|E| = 1$, then there exists only one enablement in the ring. An enablement propagation at a process P_i corresponds to a t-arc $({s_i}^l, {s'_i}^l)$. Now, P_{i+1}, the successor of P_i, is in an enabled local state ${s_{i+1}}^l$ that is a right continuation of ${s'_i}^l$. Therefore, there exists an s-arc from ${s'_i}^l$ to ${s_{i+1}}^l$. Following a similar reasoning for every process P_i that propagates a single enablement along C_L, we conclude that T_R is a trail of alternating s-arcs and t-arcs when $|E| = 1$.

If $|E| > 1$, C_L consists of two types of computations. The first type of computation is such that $p(K)$ is in a global state s^c where $|E|$ enabled processes are adjacent. The first type of computation implies a walk of type w_1 of $|E|$ consecutive s-arcs in T_R. Moreover, every local state in w_1 is an enablement that will eventually propagate. Thus, every local state in w_1 should have an outgoing t-arc participating in T_R but not in w_1. The second type of computation is the rightmost enablement propagation through the execution of $K - |E|$ local transitions. Using a similar reasoning as in the case where $|E| = 1$, the second type of computation is represented by a walk of type w_2 in T_R consisting of an alternating t-arc followed by an s-arc and vice versa. As such, the length of the alternating walk w_2 is $2(K - |E|)$. Since C_L is an alternation of both types of computations, T_R is an alternation of both types of walks: w_1 and w_2. Moreover, every s-arc in a walk of type w_1 should reach a target local state that is a source of a t-arc in a walk of type w_2 in T_R. $\square$

In a global livelock, a finite sequence of global states indefinitely repeats. A pseudo-livelock is a partial observation of the writable variables of a process that manifests itself during a global livelock and does not necessarily imply the existence of a livelock. For example, a local transition $t_{02} : y = 0 \wedge x = 0 \rightarrow x := 2$ and a local transition $t_{20} : y = 1 \wedge x = 2 \rightarrow x := 0$ form a psuedo-livelock; if we project each local transition on x, we obtain the local transitions $t'_{02} : x = 0 \rightarrow x := 2$ and $t'_{20} : x = 2 \rightarrow x := 0$, respectively. $t'02$ and t'_{20} form the repeating sequence of values $\ll 0, 2 \gg^k$ for x. However, neither of $\{t_{02}, t_{20}\}$ enables the other because of different values of the unwritable variable y.

Definition 5.13. *A* pseudo-livelock *of process P_i is a subset $pl \subset \delta_i$ of local transitions of P_i whose projection on W_i forms a repetitive sequence of values for variables in W_i.*

Theorem 5.14 establishes a sufficient condition for livelock freedom in unidirectional rings.

Theorem 5.14 (Sufficient Conditions for Livelock Freedom)**.** *For some K, if L is a livelock in a parameterized protocol $p(K)$ on a unidirectional ring, then LTG_p has a contiguous directed trail T_R in LTG_p such that:*

1. *There exists an illegitimate local state in T_R, and,*
2. *All t-arcs of T_R form pseudo-livelocks.*

Proof. From Lemma 5.11, $p(K)$ has a livelock L iff $p(K)$ has a contiguous livelock C_L. Lemma 5.12 implies that LTG$_p$ has a contiguous trail T_R representing C_L.

According to Lemma 5.9, there exists a global state in L such that some process is corrupted. Since T_R is a representation of C_L on a ring, we conclude that some vertex in T_R represents a local illegitimate state. This proves Item 1.

Since L is a livelock, for every P_i, the projection of every global transition t_i in L on the writable variables of P_i; a.k.a., $t_i \downarrow W_i$, induces a repetitive sequence of values for variables in W_i . Therefore, t-arcs in T_R form a pseudo-livelock. This proves Item 2. ☐

Note that we use the contrapositive of Theorem 5.14 to prove livelock freedom. Observe that bidirectional rings may also include contiguous livelocks. Therefore, we can apply Theorem 5.14 on bidirectional rings to prove contiguous livelock freedom. However, other types of livelocks may occur in bidirectional rings that are beyond the scope of Theorem 5.14.

We demonstrate how a contiguous livelock forms a contiguous trail in LTG$_p$. Figure 8 illustrates t-arcs of a solution to maximal matching on a bidirectional ring due to Gouda and Acharya [23]. For readability, we only include t-arcs participating in a livelock.

$$t_{ls} : m_i = \text{left} \wedge m_{i-1} = \text{left} \rightarrow m_i := \text{self}$$
$$t_{sl} : m_i = \text{self} \wedge m_{i-1} \neq \text{left} \rightarrow m_i := \text{left}$$

We include s-arcs only where necessary. This protocol has a global livelock L when $K = 5$ represented by $L = \ll$ lslsl, sslsl, sllsl, slssl, slsll, slsls, llsls, lssls, lslls, lslss $\gg^k$. This livelock represents a single enablement that circulates twice from P_0 to P_1, $\cdots$ to P_4; i.e., $|E| = 1$. L is represented in Figure 8 by the alternating trail $T_R = \ll$ lls, t_{ls}, lss, s-arc, ssl, t_{sl}, sll, s-arc $\gg$. Moreover, t_{ls} and t_{sl} form a pseudo-livelock; once projected on m_i, they represent the local transitions $(t_{ls} \downarrow W_i) : m_i = \text{left} \rightarrow m_i := \text{self}$ and $t_{sl} \downarrow W_i : m_i = \text{self} \rightarrow m_i := \text{left}$, respectively. The corresponding global transitions of the projections form a livelock.

In Section 6, we demonstrate an agreement protocol with a livelock circulating more than one enablement ($|E| > 1$).

6 Application in Automated Addition of Convergence to Non-Stabilizing Protocols

This section presents an outline for a method that synthesizes global convergence for parameterized protocols in the local state space of the representative process (without exploring the global state). Previous work on automated design of convergence [16,17,24] mainly explores the global state space of a protocol to synthesize recovery from any illegitimate state. Moreover, existing work addresses the synthesis of convergence for protocols with a fix number of processes; i.e., synthesized solutions are not generalizable. Thus, the proposed method in this section enables a significant improvement in the time/space complexity of automated design of convergence.

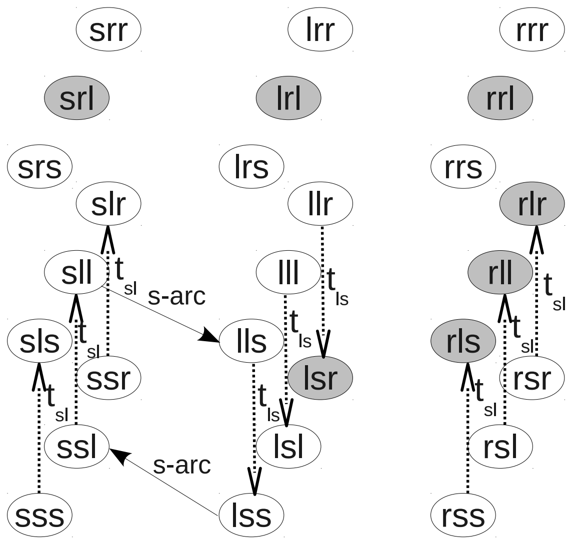

Figure 8: LTG_p of a matching solution adapted from Gouda and Acharya [23].

6.1 Synthesis Methodology

Given a parameterized protocol p over a ring whose representative process is P_r and whose set of legitimate states is defined by LC_r, we construct LTG_p as in Section 5.

1. Identify the subset ${D_L}^l \subset {S_r}^l$ of local deadlocks of P_r. Form the induced subgraph of RCG_p over ${D_L}^l$.

   3-coloring example. Since the input protocol p for 3-coloring is empty, we have ${D_L}^l = {S_r}^l$ (Figure 9)$\lhd$.

2. Identify a subset *Resolve* $\subset \neg LC_r \cap {D_L}^l$ of local deadlocks that should be resolved by local t-arcs in the revised protocol p_{ss}. As such, $\text{RCG}_{p_{ss}}$ is the induced subgraph of RCG_p over ${D_L}^l -$ *Resolve* should represent a deadlock free protocol for every K. By Theorem 4.2, $\text{RCG}_{p_{ss}}$ has no directed cycles through any local deadlock in LC_r iff $p_{ss}(K)$ has no deadlocks for every K. As such, *Resolve* captures a minimal subset of local deadlocks of p that should be resolved in p_{ss}. One way to compute *Resolve* is as a *minimal feedback subset*[5] of RCG_p restricted to be a subset of $\neg LC_i$. Therefore, all minimal feedback subsets that are subsets of $\neg LC_r$ are possible candidates for *Resolve*.

   3-coloring example. A parameterized 3-coloring protocol over a unidirectional ring is defined by a process P_r, a set of variables $\Phi_p(K) = \{c_0, \cdots, c_{K-1}\}$ such that c_r takes values from a domain $D_r = \{0, 1, 2\}$. A local legitimate state of P_r is such that P_r's color is different from its predecessor's; i.e., $LC_r = (c_r \neq c_{r-1})$. In Figure 9, the set of illegitimate local states identified by uncolored vertices is $\{00, 11, 22\}$. Since every illegitimate local state has a self-loop, *Resolve*$= \{00, 11, 22\}$. We denote a possible local transition of P_r by t_{ij} where $i, j \in D_r$, such that $t_{ij} : c_{r-1} = c_r = i \rightarrow c_r := j$. $\lhd$

3. Identify *Candidates$_r$* as the set of all possible candidate local transitions ${t_r}^l$ of P_r that resolve every local deadlock in *Resolve*. ${t_r}^l = ({s_0}^l, {s'_0}^l) \in$ *Candidates$_r$* is a local transition of P_r such that ${s_0}^l \in$ *Resolve* and ${s'_0}^l \notin$ *Resolve*. As such, we guarantee that all actions are self-disabling as in Assumption 2 of Section 5.

   3-coloring example. The set of candidate local transitions in Figure 9 that resolve all local deadlocks in *Resolve* is $\{t_{01}, t_{02}, t_{10}, t_{12}, t_{20}, t_{21}\}$. $\lhd$

4. Identify a subset of *Non-Pseudo-Livelocks* (*NPL*) of *Candidates$_r$* such that:

   (a) Local transitions in *NPL* do not form pseudo-livelocks.

[5] A feedback subset *FS* of a directed graph G is a subset of vertices of G such that, when omitted from the G, induces a subgraph of G with no directed cycles. *FS* is minimal when it has no subset that is a feedback set.

(b) Local transitions in *NPL* resolve every local deadlock in *Resolve*.

If such *NPL* exists, declare success (Theorem 5.14).

3-coloring example It is sufficient to include only one local transition originating at every local deadlock to resolve it. For example, it is sufficient to include either t_{01} or t_{02}, but not both, to resolve the local deadlock 00. Every local deadlock in *Resolve* is the source state of two possible local transitions in $Candidates_r$. As such, 2^3 possible subsets of $Candidates_r$ render 3-coloring deadlock free for any K. These subsets are $\{\{t_{01}, t_{12}, t_{20}\}, \{t_{01}, t_{12}, t_{21}\}, \{t_{01}, t_{10}, t_{20}\}, \{t_{01}, t_{10}, t_{21}\}, \{t_{02}, t_{12}, t_{20}\}, \{t_{02}, t_{12}, t_{21}\}, \{t_{02}, t_{10}, t_{20}\}, \{t_{02}, t_{10}, t_{21}\}\}$. However, every subset has a pseudo-livelock. For example, local transitions $\{t_{01}, t_{12}, t_{20}\}$, when projected on W_r, form the pseudo-livelock $\ll 0, 1, 2 \gg^k$. Likewise, any two local transitions t_{ij}, t_{ji} form a pseudo-livelock. $\lhd$

5. Identify a subset of *Pseudo-Livelocks* (*PL*) of $Candidates_r$ such that:

   (a) Local transitions in *PL* resolve every local deadlock in *Resolve*.

   (b) Local transitions in *PL* have subsets forming pseudo-livelocks. Otherwise, local transitions in *PL* would have been in *NPL* and we should not have reached the current step.

   (c) Each pseudo-livelock in *PL* is not forming a contiguous trail T_R in LTG$_p$ as in Lemma 5.12.

If such *PL* exists, there are no pseudo-livelocks in *PL* whose t-arcs form contiguous trails. Consequently, we can conclude from Theorem 5.14 that p_{ss} is livelock free for every size of the ring. Otherwise, declare failure.

3-coloring example. Every subset of t-arcs forming a pseudo-livelock corresponds to a contiguous livelock. For example, in Figure 9, $\{t_{01}, t_{12}, t_{20}\}$ forms a pseudo-livelock and creates the contiguous trail $T_R = \{00, 01, 11, 12, 22, 20\}$ that includes illegitimate local states. The sufficient conditions for livelock freedom in the contrapositive of Theorem 5.14 are not satisfied. Therefore, we declare failure. $\lhd$

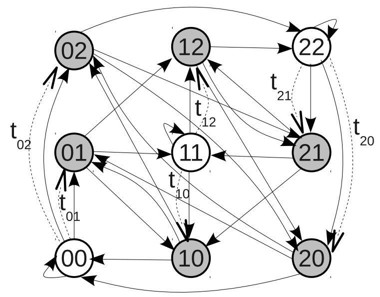

Figure 9: LTG$_p$ of 3-coloring example

6.2 Further Examples

In this subsection, we apply our proposed methodology to design three protocols: binary agreement, two-coloring and sum-not-two protocols. In the latter example, we illustrate how the conditions of Theorem 5.14 are sufficient but unnecessary, however, they are weak enough to provide a converging solution on a symmetric unidirectional ring.

Agreement example. We investigate a parameterized binary agreement protocol as in Example 5.2. A local legitimate state is such that $x_r = x_{r-1}$; i.e., the protocol stabilizes when all variable values are equal.

Figure 10 represents LTG$_p$ of the parametrized agreement protocol. t_{01} and t_{10} are local transitions resolving illegitimate local states. $t_{01}: \ (x_r < x_{r-1}) \rightarrow x_r := x_{r-1}$ or $t_{10}: \ (x_{r-1} < x_r) \rightarrow x_r := x_{r-1})$.

In Figure 10, the local illegitimate states are ${D_L}^l = \{10, 01\}$, however, it is sufficient to resolve either of them to obtain a continuation relation that has no directed cycles passing by illegitimate deadlocks. Therefore, *Resolve*= $\{01\}$ or *Resolve*= $\{10\}$. As such, including either t_{01} or t_{10} (but not both!) renders the protocol deadlock free. Since including just one of the candidate local transitions does not form pseudo-livelocks, both solutions are livelock free. Hence follows convergence.

If we unnecessarily include both t_{01} and t_{10} that form a pseudo-livelock, we observe $T_R = \ll 01, t_{10}, 00, s, 01, s, 10, t_{01}, 11, s, 10, s, 01 \gg$ as an alternating trail satisfying the implications of Lemma 5.12. Moreover, t_{01} and t_{10} form a pseudo-livelock. Hence, including both t_{01} and t_{10} does not satisfy the sufficient conditions of the contrapositive of Theorem 5.14.

Notice that if we apply constraint satisfaction for cyclic constraint graphs as described in reference [6], there is no way to differentiate between the case where only one of the convergence actions $\{t_{01}, t_{10}\}$ is included in p_{ss}, and the case where we include both convergence actions in p_{ss}. In fact, both constraint graphs are the same since the set of legitimate states does not change. Moreover, our methodology computes a possibly strict subset of local deadlocks outside LC_r and still guarantees deadlock freedom for every K. $\lhd$

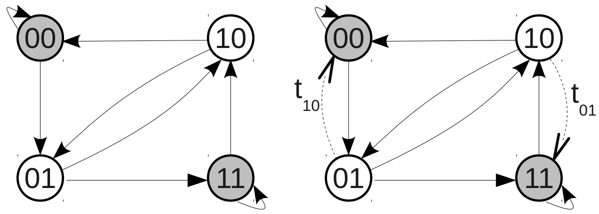

Figure 10: RCG$_p$ and LTG$_p$ of Agreement Example

Two-coloring example. For a 2-coloring protocol whose RCG and LTG are represented in Figure 11, $R_r = \{c_{r-1}, c_r\}$ and $W_r = \{c_r\}$. $D_r = \{0, 1\}$ and $LC_r = c_r \neq c_{r-1}$. A legitimate local state is such that a process and its predecessor should have different colors.

Unlike deadlock states in agreement, 2-coloring requires the resolution of both illegitimate local deadlocks ${D_L}^l$ =*Resolve*= $\{00, 11\}$ because they have self-loops of s-arcs[6]. However, the resolution of both local deadlocks results in a directed trail T_R as in Lemma 5.12 $\ll 00, t_{01}, 01, s, 11, t_{10}, 10, s, 00 \gg$ and not satisfying the sufficient conditions in the contrapositive of Theorem 5.14. As such, we cannot conclude livelock freedom of 2-coloring for arbitrary K. In fact, 2-coloring self-stabilizing protocols are impossible in unidirectional rings [25], however our lack of necessary conditions for livelock freedom prevents us from deducing any impossibility results. $\lhd$

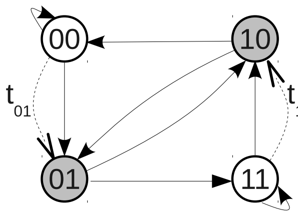

Figure 11: LTG$_p$ of the Two Coloring Example

Sum-not-two example. We present a hypothetical example to illustrate the interplay between having a trail, having pseudo-livelocks and having both. The Sum-Not-Two protocol on a unidirectional ring is such that P_r reads x_{r-1} and x_r and writes x_r. For simplicity of presentation, we restrict our example such that x_r takes values in $\{0, 1, 2\}$. A local legitimate state is such that $x_r + x_{r-1} \neq 2$. The input protocol p is empty.

Since p is empty, the set of local deadlocks outside LC_r is $\neg LC_r = \{20, 11, 02\}$. For a deadlock free protocol, no proper subset of $\neg LC_r$ can be resolved to render p_{ss} deadlock free for every K. Thus, *Resolve*= $\{20, 11, 02\}$.

Figure 12 illustrates LTG$_p$ of Sum-Not-Two protocol with all candidate t-arcs included. Every local deadlock has two possible t-arcs that resolve it and hence, we have 2^3 possibilities for $Candidates_r$. The

[6]Recall that for deadlocks-freedom, we make sure that there are no directed cycles over local deadlocks in RCG that include illegitimate local states.

following two possibilities form pseudo-livelocks and each of them participate in a trail: $\{\{t_{21}, t_{10}, t_{02}\}, \{t_{01}, t_{12}, t_{20}\}\}$. For example, the first possibility participates in the trail: $T_R = \ll 02, t_{21}, 01, s, 11, s, 11, t_{10}, 10, s, 02, s, 20, t_{20}, 22, s, 20, s \gg$. This possibility forms a pseudo-livelock and participates in a trail T_R as implied by Lemma 5.12. Hence, sufficient conditions of the contrapositive of Theorem 5.14 are not satisfied by the first possible set of candidates and we cannot include this set.

In fact, if we examine T_R, it should represent a contiguous livelock L having $|E| = 2$ and only one propagation of enablement; i.e., $K - |E| = 1$. Hence, T_R is possibly representing a livelock in a ring where $K = 3$. However, if we try to reconstruct the global livelock of a ring of three processes using T_R, we fail! In other words, T_R does not represent a real livelock and due to the lack of necessity, we could not include $\{t_{21}, t_{10}, t_{02}\}$ in p_{ss}.

None of the remaining candidate subsets of t-arcs forms a trail whose t-arcs are pseudo-livelocks. For example, let $Candidates_r = \{t_{21}, t_{12}, t_{01}\}$. Here, t_{21} and t_{12} form a pseudo-livelock, however, there is no trail where they solely participate and has the properties implied in Lemma 5.12. Moreover, there is a trail that includes all the three t-arcs together, but since, together, they do not form a pseudo-livelock, conditions of the contrapositive of Theorem 5.14 remain satisfied. As such, including $\{t_{21}, t_{12}, t_{01}\}$ in p_{ss} renders Sum-Not-Two converging. The following action captures $Candidates_r$: $(x_r + x_{r-1} = 2) \wedge (x_r \neq 2) \rightarrow x_r := (x_r + 1)$ mod 3, $(x_r + x_{r-1} = 2) \wedge (x_r = 2) \rightarrow x_r := (x_r - 1)$ mod 3.

To prove convergence of our proposed solution using constraint satisfaction, we must ingenuously identify a partitioning of the protocols actions. We argue that our methodology bypasses constraint satisfaction in this respect as we directly design/verify convergence through local state space exploration. $\lhd$

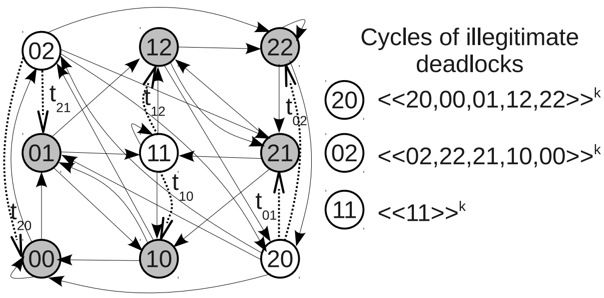

Figure 12: LTG$_p$ of the Sum-Not-Two example including every candidate t-arc. To the right, we demonstrate a s-cycle for each individual local deadlock that we resolved.

7 Discussion and Related Work

In this section, we discuss related work regarding reasoning in local vs. global state space, support for revision when verification fails and automated design of convergence.

Design of convergence. While there are several methods for compositional and local reasoning about self-stabilization [5–7,9,21], the proposed approach in this report enables a systematic method for designing parameterized SS rings that are correct by construction. For example, Varghese [7] presents a method for proving the correctness of convergence in systems that are composed of components that converge independently. Dolev and Herman [21] introduce a technique for the composition of synchronous processes in acyclic networks towards generating scalable systems that converge fast. Arora *et al.* [6] present a set of sufficient conditions for the verification of convergence in parameterized systems. The method of [8] provides a scalable approach for compositional design of SS protocols using a correction and a corruption relation that define how components could corrupt each other and how the correction of a component depends upon the correction of other components. By contrast, the proposed approach of this report is more fine-grained in that it generates the required transitions for detection and correction.

Most existing automated methods [16, 17, 26, 27] explore the global state space of protocols with a fixed number of processes in order to synthesize recovery functionalities. For example, Kulkarni and Arora [26] present algorithms for adding recovery from a set of states reachable in the presence of faults, called a *fault-span*, to the set of legitimate states. Bonakdarpour and Kulkarni [27] demonstrate the hardness of algorithmic design of progress properties for distributed systems and propose a heuristic for automated design. Abujarad and Kulkarni [16] present heuristics for automated exploration of the global state space of acyclic networks towards synthesizing convergence. None of the aforementioned methods addresses the design of convergence for parameterized rings in the local state space.

Verification using cutoff sizes. In the area of automatic verification of parameterized systems, Emerson and Kahlon [28, 29] derive cutoff sizes for parameterized rings; they reduce parameterized verification of temporal logic properties defined over pairs of processes to finite model checking. They extend their cutoff theorems to arbitrary protocols whose guards are conjunctive/disjunctive to verify properties including pairs of processes [30]. Emerson and Namjoshi [31] extend their approach to rings whose computations eventually terminate in a finite number of steps. Their cutoff bound depends on the length of the terminating computation and the size of the local state spaces. Moreover, they can only verify conjunctive properties defined at most over pairs of components.

Our methodology emphasizes automatic verification of convergence in local state space which is less computationally intensive than verification for every K smaller than or equal to the cutoff. Moreover, our contribution is not restricted to terminating computations. We could have applied Emerson's approach to verify strong convergence in unidirectional rings of terminating protocols, however, we sought a method where we can simultaneously design and verify convergence in a local state space; it is unclear how a protocol should be revised if its verification for a property fails.

Verification by abstraction. A considerable amount of work adopt abstraction to handle the infinite number of states in parameterized verification. *Network invariants* are introduced by Wolper and Lavinfosse [32] to capture all possible behaviors of an arbitrary number of processes in the network. A property satisfied by a network invariant is satisfied by any instance of the network but not necessarily the converse; abstraction is hence necessarily incomplete. Kurshan and McMillan [33] demonstrate a general abstraction rule based on composition and induction over a sequence of processes. The generality of their approach is due to the abstract properties of their composition operators and partial order relations on processes. Kesten *et al.* [34] present yet another induction method using network invariants with a proof rule based on an abstraction relation and composition of processes.

The main drawback of abstraction methods with respect to convergence synthesis is their dependence on human ingenuity for generating abstractions; every protocol requires a different abstract network invariant that, in general, cannot be automatically computed. To overcome this drawback, Pnueli *et al.* [35] demonstrate a method where conjunctive sets of reachable states can be automatically deduced. They project the set of reachable states, for a specific network size, over a subset of "variables of interest" in some conjunct. Their method generalizes the projected conjunct for every process in the network. They provide a cutoff theorem, thereby reducing verification of an arbitrary-sized network to a finite number of protocol instances. Despite the inherent incompleteness of this method, it has proved that it is of practical values in automated verification of safety properties. A similar approach for verifying response properties by Fang *et al.* [36] abstracts out decreasing ranking functions for an arbitrary protocol instance. They generalize the convergence stairs likewise while using a cutoff theorems proper to response properties.

Namjoshi [37] illustrates that the cutoff method for verification of parameterized systems is complete for safety properties. That is, there always exists a maximum size for the number of symmetric processes that captures all the "behaviors of interest" in the network with respect to a given safety property. Furthermore, he provides a modification to the method by Pnueli *et al.* [35] to accommodate his completeness result.

Network grammars. Shtadler and Grumberg [38] introduce network grammars as a means to representing global states of arbitrary-sized networks of linear or ring topologies, as words generated by network grammars. For verification purposes, they compute an equivalent network invariant to the network grammar and apply finite state verification on the equivalent model/abstraction. As an extension, Clarke *et al.* [39] relax the equivalence relation between the model and its network invariant to a pre-order relation such that the network invariant abstracts out the grammar; this relaxation increases the possibility of finding an invariant at the cost of completeness. Kesten *et al.* [40] restrict network grammars to regular languages; however their approach extends verification to tree-like topologies by capturing their global states as accepted trees by a tree-automaton. Moreover, they represent reachable sets of states by finite automata, thereby reducing the verification of safety properties to automata-theoretic product and emptiness problems.

A follow-up of the aforementioned approaches generated a plethora of publications in what is now called *regular model checking*. Jonnson and Nilsson [41] describe how to derive a finite state transducer representing the transitive closure of the network's transition relation. A finite state transducer is a finite state automaton augmented with a function that maps the set of input alphabet to the set of output symbols. Subsequently, they illustrate how to verify safety properties using their derived transitive closure automaton. Bouajjani *et al.* [42] demonstrate different techniques to compute finite state transducers representing the set of reachable states and the transitive closure relation of a parameterized protocol, respectively. They illustrate how to make use of the transitive closure relation to verify liveness properties. Abdulla *et al.* [43] introduce an abstraction on regular model checking by assuming a preorder relation between words representing states. This relation eliminates transducers in verification of safety properties, thereby simplifying the computationally demanding automata-theoretic operations required by regular model checking. Due to the extensive literature on regular model checking, we direct the reader to a survey by Abdulla *et al.* [44].

In contrast to the above approaches, our methodology reasons about a variety of possible solutions for a given conjunctive set of legitimate states closed in an input protocol. We investigate generalization in local state spaces, thereby enabling a method that combines design and verification instead of conceiving them as separate tasks. Thus, our approach differs from automated abstraction techniques like Fang *et al.*'s decreasing ranking functions [36], or any of the aforementioned regular model checking techniques.

8 Conclusion and Future Work

This report proposed a method for local reasoning about global convergence of parameterized network protocols with the ring topology. In such protocols, the code of each process is instantiated from the parameterized code of a representative/template process by variable substitution. Parameterized ring protocols have important applications as they can be used to construct more complicated topologies where multiple rings are intertwined (e.g., multi-ring token passing [17], scalable group communication [45]). Global convergence to a set of legitimate states I requires both deadlock-freedom and livelock-freedom in $\neg I$. While most existing design methods enable the design of convergence by reasoning in the global state space of a protocol, this report takes a different approach of reasoning in the local state space of the representative process to ensure global convergence. Specifically, we presented necessary and sufficient conditions for deadlock-freedom, and sufficient conditions for livelock-freedom in parameterized unidirectional rings. We illustrated our approach in the context of a maximal matching protocol. We sketched a methodology for design of convergence in local state space and applied it on several examples including agreement, 2-coloring, 3-coloring and sum-not-two examples.

We would like to extend this work in several directions. First, we plan to investigate local reasoning for global convergence of parameterized protocols with topologies other than rings (e.g., tree, mesh, etc.). Second, we are currently investigating sufficient conditions for bidirectional rings. Third, another interesting problem is automation. According to our proposed design methodology, we will design synthesis algorithms that can automate the generation of the LTG graphs and can revise the graphs so they meet our conditions for deadlock/livelock-freedom. Such a synthesis in local state space is a significant paradigm shift with respect to previous work on automated design of convergence in global state [16, 17, 26], which could result in producing software tools that are substantially more efficient in automated design of parameterized self-stabilizing protocols.

References

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

[2] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, 1993.

[3] S. Dolev. *Self-Stabilization*. MIT Press, 2000.

[4] M. Gouda. The triumph and tribulation of system stabilization. In Jean-Michel Helary and Michel Raynal, editors, *Distributed Algorithms, (9th WDAG'95)*, volume 972 of *Lecture Notes in Computer Science (LNCS)*, pages 1–18. Springer-Verlag, Le Mont-Saint-Michel, France, September 1995.

[5] Ted Herman. *Adaptivity Through Distributed Convergence*. PhD thesis, University of Texas - Austin, TX, USA, 1991.

[6] A. Arora, M. Gouda, and G. Varghese. Constraint satisfaction as a basis for designing nonmasking fault-tolerant systems. *Journal of High Speed Networks*, 5(3):293–306, 1996. A preliminary version appeared at ICDCS'94.

[7] G. Varghese. Compositional proofs of self-stabilizing protocols. In *Proceedings of the Third Workshop on Self-Stabilizing Systems*, pages 80–94, 1997.

[8] W. Leal and A. Arora. Scalable self-stabilization via composition. In *IEEE International Conference on Distributed Computing Systems*, pages 12–21, 2004.

[9] Mohamed G. Gouda and Ted Herman. Adaptive programming. *IEEE Transactions on Software Engineering*, 17(9):911–921, 1991.

[10] F. Stomp. Structured design of self-stabilizing programs. In *Proceedings of the 2nd Israel Symposium on Theory and Computing Systems*, pages 167–176, 1993.

[11] Anish Arora and Mohamed G. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43(9):1026–1038, 1994.

[12] M. Gouda. Multiphase stabilization. *IEEE Transactions on Software Engineering*, 28(2):201–208, 2002.

[13] G. Varghese. *Self-stabilization by local checking and correction*. PhD thesis, MIT/LCS/TR-583, 1993.

[14] Yehuda Afek, Shay Kutten, and Moti Yung. The local detection paradigm and its application to self-stabilization. *Theoretical Computer Science*, 186(1-2):199–229, 1997.

[15] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 268–277, 1991.

[16] Fuad Abujarad and Sandeep S. Kulkarni. Automated constraint-based addition of nonmasking and stabilizing fault-tolerance. *Journal of Theoretical Computer Science*, 258(2):3–15, 2011. In Press.

[17] Ali Ebnenasir and Aly Farahat. A lightweight method for automated design of convergence. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 219–230, 2011.

[18] A. Arora and M. G. Gouda. Closure and convergence: A foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering*, 19(11):1015–1027, 1993.

[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] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1990.

[21] S. Dolev and T. Herman. Parallel composition of stabilizing algorithms. In *19th IEEE International Conference on Distributed Computing Systems (ICDCS), Workshop on Self-Stabilizing Systems*, pages 25–32, June 1999.

[22] P. Godefroid. Using partial orders to improve automatic verification methods. In *Computer-Aided Verification*, pages 176–185. Springer, 1991.

[23] Mohamed G. Gouda and Hrishikesh B. Acharya. Nash equilibria in stabilizing systems. In *11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 311–324, 2009.

[24] B. Bonakdarpour and S. S. Kulkarni. Exploiting symbolic techniques in automated synthesis of distributed programs with large state space. In *Proceedings of the 27th International Conference on Distributed Computing Systems*, pages 3–10, Washington, DC, USA, June 2007. IEEE Computer Society.

[25] S. Shukla, D. Rosenkrantz, and S. Ravi. Developing self-stabilizing coloring algorithms via systematic randomization. In *Proceedings of the International Workshop on Parallel Processing*, pages 668–673. Citeseer, 1994.

[26] 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.

[27] Borzoo Bonakdarpour and Sandeep S. Kulkarni. Revising distributed UNITY programs is NP-complete. In *12th International Conference on Principles of Distributed Systems (OPODIS)*, pages 408–427, 2008.

[28] E. Emerson and V. Kahlon. Model checking large-scale and parameterized resource allocation systems. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 55–69, 2002.

[29] E.A. Emerson and V. Kahlon. Parameterized model checking of ring-based message passing systems. In *Computer Science Logic*, pages 325–339. Springer, 2004.

[30] E. Emerson and V. Kahlon. Reducing model checking of the many to the few. *Automated Deduction-CADE-17*, pages 236–254, 2000.

[31] E.A. Emerson and K.S. Namjoshi. Reasoning about rings. In *conference record of the ACM symposium on principles of programming languages*, volume 22, pages 85–94. Association of Computer Machinery, 1995.

[32] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems*, pages 68–80. Springer, 1990.

[33] R.P. Kurshan and K. McMillan. A structural induction theorem for processes. In *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 239–247. ACM, 1989.

[34] Y. Kesten, A. Pnueli, E. Shahar, and L. Zuck. Network invariants in action*. *CONCUR 2002Concurrency Theory*, pages 217–264, 2002.

[35] A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 82–97, 2001.

[36] Y. Fang, N. Piterman, A. Pnueli, and L. Zuck. Liveness with invisible ranking. In *Verification, Model Checking, and Abstract Interpretation*, pages 109–132. Springer, 2004.

[37] K. Namjoshi. Symmetry and completeness in the analysis of parameterized systems. In *Verification, Model Checking, and Abstract Interpretation*, pages 299–313. Springer, 2007.

[38] Z. Shtadler and O. Grumberg. Network grammars, communication behaviors and automatic verification. In *Automatic Verification Methods for Finite State Systems*, pages 151–165. Springer, 1990.

[39] E. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. *CONCUR'95: Concurrency Theory*, pages 395–407, 1995.

[40] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *Computer Aided Verification*, pages 424–435. Springer, 1997.

[41] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 220–235, 2000.

[42] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *Computer Aided Verification*, pages 403–418. Springer, 2000.

[43] P. Abdulla, G. Delzanno, N. Henda, and A. Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). *Tools and Algorithms for the Construction and Analysis of Systems*, pages 721–736, 2007.

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

[45] William Yurcik. Survivable ATM group communications using disjoint meshes, trees, and rings. In *Networked Group Communication*, volume 1736 of *Lecture Notes in Computer Science*, pages 235–243. Springer, 1999.