Computer Science Technical Report

Verifying Livelock Freedom on Parameterized Rings

Alex P. Klinkhamer and Ali Ebnenasir

Michigan Technological University
Computer Science Technical Report
CS-TR-13-01
July 2013

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

Verifying Livelock Freedom on Parameterized Rings

Alex P. Klinkhamer and Ali Ebnenasir

July 2013

Abstract

This paper investigates the complexity of verifying livelock freedom and weak/strong self-stabilization in parameterized unidirectional ring and bidirectional chain topologies. Specifically, we illustrate that verifying livelock freedom of parameterized rings consisting of self-disabling and deterministic processes is undecidable (specifically, Π^0_1-complete). This result implies that verifying weak/strong self-stabilization for parameterized rings of self-disabling processes is also undecidable. The results of this paper strengthen previous work on the undecidability of verifying temporal logic properties in symmetric rings. The proof of undecidability is based on a reduction from the periodic domino problem.

Contents

1 Introduction

Verifying *strong convergence* is known to be a difficult task [17], where from any state, *every* execution of a distributed system recovers to a set of legitimate states. From any state, a *weakly* converging system has at least one execution that recovers to legitimate states. Designing and verifying convergence are important problems as they have applications in several fields such as network protocols [7,19], multi-agent systems [16], cloud computing [22], and equilibrium in socioeconomic systems [18]. A common feature of such systems is that they comprise a finite but unbounded number of components/processes that communicate based on a specific network topology; i.e., *parameterized systems*. Deadlock freedom and livelock freedom outside legitimate states are necessary and sufficient conditions for strong convergence, whereas a system is weakly converging if and only the system can reach the legitimate states from each illegitimate state via some execution. There are numerous methods [6,11,14,15] for the verification of safety properties of parameterized systems, where safety requires that nothing bad happens in system executions (e.g., no deadlock state is reached). Apt and Kozen [3] illustrate that, in general, verifying Linear Temporal Logic (LTL) [8] properties for parameterized systems is Π_1^0-complete. Suzuki [24] shows that the verification problem remains Π_1^0-complete even for unidirectional rings where all processes have a similar code that is parameterized in the number of nodes.

Contributions. In this paper, we extend this result for the special case where the property of interest is livelock freedom, where every system state is under consideration. We make restrictive assumptions about processes, that they are deterministic, have constant state spaces, and are *self-disabling*, i.e., no actions of a process are enabled immediately after it acts. Specifically, we illustrate that, even when processes are symmetric, deterministic, self-disabling, and have a constant state space, livelock detection is undecidable (Σ_1^0-complete) on unidirectional ring and bidirectional chain topologies. Further, we conclude that verifying strong or weak convergence on these topologies is Π_1^0-complete. The proof of undecidability in our work is based on a reduction from the periodic domino problem.

Organization. Section 2 presents some basic concepts. Section 3 provides a formal characterization of livelocks in unidirectional rings. Then, Section 4 represents a well-known undecidable problem, which we will use to show the undecidability of verifying livelock freedom in rings. Section 5 illustrates that verifying livelock freedom of unidirectional ring and bidirectional chain protocols is undecidable. Section 6 discusses related work, and Section 7 summarizes our contributions and outlines some future work.

2 Basic Concepts

This section presents the definition of protocols and action graphs. A protocol p defines the behavior for a network of $N > 1$ processes (finite-state machines), where each process P_i owns a set of variables whose valuation determines its *state*. The state of the network/system is defined by the current states of all processes. A process *acts* when it atomically changes its state based on its current state and the states of its neighboring processes, where neighbors are defined by the network topology. For example, in a unidirectional ring topology consisting of N processes, each process P_i (where $0 \le i \le N-1$) has a neighbor P_{i-1}, where subtraction is modulo N. An *execution* of a protocol is a sequence of states $C_0, C_1, \ldots, C_k$ where there is a transition from C_i to C_{i+1} for every $i \in \mathbb{N}_k$.

We consider *symmetric* protocols, where each process has identical rules for changing its state. Furthermore, we assume that the state space Σ and rules for each process are independent of the topology (e.g., number of processes).

Definition 2.1 (Transition Function). Let P_i be any process with a state variable x_i in a unidirectional ring protocol p. We define its transition function $\xi : \Sigma \times \Sigma \to \Sigma$ as a partial function such that $\xi(a, b) = c$ if and only if P_i has an action $(x_{i-1} = a \wedge x_i = b \longrightarrow x_i := c;)$. In other words, ξ can be used to define all actions of P_i in the form of a single parametric action:

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

where $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$ checks to see if the current x_{i-1} and x_i values are in the preimage of ξ.

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 any process P_i in a unidirectional ring protocol. To visually represent the structure of a process, we depict a protocol by a labeled directed multigraph where each action (a, b, c) in the protocol appears as an arc from node a to node c labeled b in the graph. For example, consider the self-stabilizing sum-not-2 protocol given in [12]. Each process P_i has a variable $x_i \in \mathbb{N}_3$ and actions $(x_{i-1} = 0 \wedge x_i = 2 \longrightarrow x_i := 1)$, $(x_{i-1} = 1 \wedge x_i = 1 \longrightarrow x_i := 2)$, and $(x_{i-1} = 2 \wedge x_i = 0 \longrightarrow x_i := 1)$. This protocol converges to a state where the sum of each two consecutive x values does not equal 2 (i.e., the state predicate $\forall i : (x_{i-1} + x_i \neq 2)$). We represent this protocol with a graph containing arcs $(0, 2, 1)$, $(1, 1, 2)$, and $(2, 0, 1)$ as shown in Figure 1.

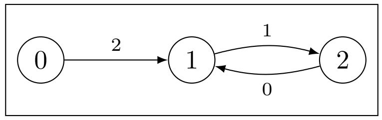


Figure 1: Graph representing sum-not-2 protocol.

Since protocols consist of *self-disabling* processes, an action (a, b, c) cannot coexist with action (a, c, d) for any d. Moreover, when the protocol is deterministic, a 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 $d \neq c$.

Livelock, deadlock, and closure. A *legitimate* state is a state which we want the system to be in. Let I be a predicate representing the legitimate states for some protocol p. A *livelock* of p is an infinite execution which never reaches I. When legitimate states are not specified, we assume a livelock is any infinite execution. A *deadlock* of p is an state in $\neg I$ which has no outgoing transition; i.e., no process is enabled to act. The state predicate I is *closed* under p when no transition exists which brings the system from a state in I to a state in $\neg I$. These concepts allow us to define weak and strong self-stabilization (adapted from [17]).

Definition 2.2 (Strong Stabilization)**.** A protocol p is *(strongly) self-stabilizing* with respect to its legitimate state predicate I if and only if from each illegitimate state, all executions reach, and remain in, the set of legitimate states. That is, p is livelock-free and deadlock-free, and I is closed under p.

Definition 2.3 (Weak Stabilization)**.** A protocol p is *weakly self-stabilizing* with respect to its legitimate state predicate I if and only if from each illegitimate state, an execution exists to a legitimate state, and I is closed under p. Notice that deploying a weakly stabilizing protocol under a strongly fair scheduler guarantees convergence to I, even if there are livelocks in $\neg I$. *Strong fairness* ensures that any process that is infinitely often enabled will act infinitely often.

3 Livelock Characterization

This section presents a formal characterization of livelocks in parameterized rings. This characterization is based on a notion of sequences of actions that are propagated in a ring, called *propagations* and a *leads* relation between the propagations. We shall use propagations and the leads relation to specify necessary and sufficient conditions for the existence of livelocks in symmetric unidirectional ring protocols of self-disabling processes.

Propagations. When a process acts and enables its successor, it propagates its ability to act. The successor may enable its own successor by acting, and the pattern may continue indefinitely. This behavior is called a *propagation* and is represented by a sequence of parameterized actions. Consider a propagation $\langle (a, b, c), (d, e, f) \rangle$ of length 2 which says a state exists which allows some P_i to perform action (a, b, c) which enables P_{i+1} to perform (d, e, f). Since P_i assigns its variable x_i to c and P_{i+1} is then enabled to perform (d, e, f) which relies on $x_i = d$ and $x_{i+1} = e$, we know $c = d$. We therefore write the jth action of a propagation as (a_{j-1}, b_j, a_j). It follows that a propagation is a walk through the protocol's 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 m if its jth action and $(j + m)$th action are the same for every index j. A periodic propagation corresponds to a closed walk of length m in the graph. The sum-not-2 protocol has such a propagation of period 2: $\langle(1,1,2),(2,0,1)\rangle$.

"Leads" relation. An action *leads* another action if and only if the value of a process's variable after executing the first action is the same as the value required for the process to execute the second action. Formally, this means an action (a,b,c) leads (d,e,f) if and only if $e=c$. Similarly, a propagation leads another if and only if, for every index j, its jth action leads the jth action of the other propagation. Therefore if we have a propagation whose jth action is (a_{j-1},b_j,a_j) which leads another propagation whose jth action is (d_{j-1},e_j,d_j), then we know $e_j=a_j$ and write the led action as (d_{j-1},a_j,d_j). In the context of the protocol graph, this corresponds to two walks (representing propagations) where the jth destination node label of the first walk matches the jth arc label of the second walk for each index j. After some first propagation executes through a ring segment $P_q,\ldots,P_{q+m-1}$, a second propagation can execute through the same segment only if the first propagation leads the second. This is true since each process P_{q+j} performs the jth action of the first propagation, assigning its variable x_{q+j} to some value a_j. If the second propagation executes through the segment, each P_{q+j} must perform the jth action of the second propagation from a state where $x_{q+j}=a_j$. As such, each jth action of the first propagation must lead the jth action of the second propagation. Thus, the first propagation itself must lead the second.

We focus on scenarios where for some positive integers m and n, there are m periodic propagations with period n where the ith propagation leads the $(i+1)$th propagation for each i (and the last propagation leads the first). This case can be represented succinctly. Using $\mathtt{X}$ as a wildcard value (i.e., any value, do not assume $\mathtt{X}=\mathtt{X}$), recall that if action is defined to lead another action $(\mathtt{X},a,\mathtt{X})$ when it has the form $(\mathtt{X},\mathtt{X},a)$. Also recall that a propagation of period n has the form $\langle(a_{n-1},\mathtt{X},a_0),(a_0,\mathtt{X},a_1),\ldots,(a_{n-2},\mathtt{X},a_{n-1})\rangle$. Thus, if we write each ith propagation as $\langle(a^i_{n-1},\mathtt{X},a^i_0),(a^i_0,\mathtt{X},a^i_1),\ldots,(a^i_{n-2},\mathtt{X},a^i_{n-1})\rangle$, then we can determine the $\mathtt{X}$ values as $\langle(a^i_{n-1},a^{i-1}_0,a^i_0),(a^i_0,a^{i-1}_1,a^i_1),\ldots,(a^i_{n-2},a^{i-1}_{n-1},a^i_{n-1})\rangle$. This case is succinctly visualized by an $m\times n$ matrix as shown in Remark 3.1 where for each row i and column j, the triple $(a^i_{j-1},a^{i-1}_j,a^i_j)$ is an action in the protocol.

Remark 3.1. Consider the following $m\times n$ matrix M whose element at row i and column j is denoted as $M[i,j]=a^i_j$.

$$M=\begin{bmatrix} a^0_0 & a^0_1 & \cdots & a^0_{n-1} \\ a^1_0 & a^1_1 & \cdots & a^1_{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ a^{m-1}_0 & a^{m-1}_1 & \cdots & a^{m-1}_{n-1} \end{bmatrix}$$

Assuming a unidirectional ring protocol of self-disabling, symmetric processes, the following statements are equivalent:

- The triple $(a^i_{j-1},a^{i-1}_j,a^i_j)$ is an action in the protocol for every row $i\in\mathbb{N}_m$ and column $j\in\mathbb{N}_n$.
- The protocol contains m propagations of period n where each ith propagation leads the $(i+1)$th propagation for each $i\in\mathbb{N}_m$. For each $i\in\mathbb{N}_m$ and $j\in\mathbb{N}_n$, the jth action of the ith propagation is $(a^i_{j-1},a^{i-1}_j,a^i_j)$.

Example 3.2. *Livelock freedom of the sum-not-2 protocol.*

Recall from Figure 1 that the sum-not-2 protocol consists of three parameterized actions $(0,2,1)$, $(1,1,2)$, and $(2,0,1)$. Every periodic propagation in this protocol alternates between actions $(2,0,1)$ and $(1,1,2)$. These propagations require x_i values to alternate between 0 and 1 for each subsequent i. However, these propagations assign x_i values alternating between 1 and 2 for each subsequent i. Clearly no periodic propagation can execute through a ring segment of alternating 1 and 2 values, therefore no propagation leads another in this protocol. For any ring size, an infinite execution requires that actions propagate around the ring. This is not possible since no propagation leads another, therefore the protocol is livelock-free.

We form the same argument in terms of walks in the protocol's graph. Every closed walk in the graph alternates between visiting node 1 and node 2 indefinitely. No closed walk exists which alternates between visiting arcs labeled 1 and 2, therefore no periodic propagation leads another in the this protocol. As such, no livelock exists.

Lemma 3.3. *Assume a ring protocol where processes are symmetric. Let $C = (c_0, \ldots, c_{N-1})$ and $C' = (c'_0, \ldots, c'_{N-1})$ be state of a ring of size N such that $\exists k : \forall i : c'_i = c_{i+k}$. In other words, if C is rotated clockwise by k positions, then it equals C'. If an execution exists from C to C', then an infinite execution exists.*

Proof. Since processes are symmetric, we know a second execution exists from C' to a state $C'' = (c''_0, \ldots, c''_{N-1})$ where $c''_i = c'_{i+k} = c_{i+2k}$ for each i. States C' and C'' meet the same respective conditions as C and C', therefore an infinite execution exists. Emerson and Namjoshi [10] similarly use this notion of rotational symmetry to reason about rings of symmetric processes. □

Example 3.4. *3-coloring unidirectional ring protocol with a livelock.*

Consider a unidirectional ring protocol where each process P_i has a variable $x_i \in \mathbb{N}_3$ whose value represents a color. Our goal is to reach a state where no two consecutive colors are equal. As such, a process P_i must act when $x_{i-1} = x_i$. Give each process an action $(x_{i-1} = x_i \longrightarrow x_i := x_i - 1;)$ where subtraction is modulo 3. In our triple notation, there are 3 actions $(0, 0, 2)$, $(1, 1, 0)$, and $(2, 2, 1)$, one for each possible x value. Figure 2 illustrates the protocol as a graph.

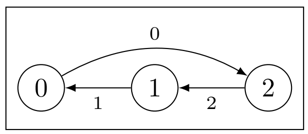


Figure 2: Graph representing the 3-coloring protocol.

Clearly this graph contains a closed walk starting at node 1 and visiting nodes 0, 2, and 1.

1. We can find a closed walk through arcs labeled 0, 2, and 1 by starting at node 0 and visiting nodes 2, 1, and 0. This walk corresponds to the periodic propagation $\langle(0, 0, 2), (2, 2, 1), (1, 1, 0)\rangle$.
2. We now look for a closed walk through arcs labeled 2, 1, and 0 by starting at node 2 and visiting nodes 1, 0, and 2. This corresponds to the periodic propagation $\langle(2, 2, 1), (1, 1, 0), (0, 0, 2)\rangle$.
3. Finally, we find a closed walk through arcs labeled 1, 0, and 2 by starting at node 1 and visiting 0, 2, and 1. We started with this same sequence of nodes, therefore we are done and have found the first periodic propagation to be $\langle(1, 1, 0), (0, 0, 2), (2, 2, 1)\rangle$.

Indeed, we have found three periodic propagations which lead each other in order (first leads second, second leads third, third leads first):

$$\langle(1, 1, 0), (0, 0, 2), (2, 2, 1)\rangle$$
$$\langle(0, 0, 2), (2, 2, 1), (1, 1, 0)\rangle$$
$$\langle(2, 2, 1), (1, 1, 0), (0, 0, 2)\rangle$$

We can use Remark 3.1 to view these compactly as a matrix of elements a^i_j where $(a^i_{j-1}, a^{i-1}_j, a^i_j)$ denotes the jth action of the ith propagation.

$$\begin{bmatrix} a^0_0 & a^0_1 & a^0_2 \\ a^1_0 & a^1_1 & a^1_2 \\ a^2_0 & a^2_1 & a^2_2 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 1 \\ 2 & 1 & 0 \\ 1 & 0 & 2 \end{bmatrix}$$

We can explicitly construct a ring of 9 processes whose initial state admits a livelock. Let the initial state be as follows.

$$(x_0, x_1, \ldots, x_8) = (a^2_0, a^2_1, \ldots, a^0_2) = (1, 0, 2, 2, 1, 0, 0, 2, 1)$$

From this state, processes P_0, P_3, and P_6 can act to reach $(0, 0, 2, 1, 1, 0, 2, 2, 1)$ which has the same values as the initial state but rotated by 4 positions to the right (each new x_i value is equal to the original value of x_{i-4}). By Lemma 3.3, a livelock exists.

Lemma 3.5. *Assume a unidirectional ring protocol of symmetric, self-disabling processes. Given m propagations with period n, where the $(i-1)$th propagation leads the ith propagation for each index i (note: $(n-1)$th leads 0th when $i = 0$), the protocol contains a livelock for some ring size.*

Proof. Write the ith propagation as

$$\left\langle (a^i_{n-1}, a^{i-1}_0, a^i_0), (a^i_0, a^{i-1}_1, a^i_1), \ldots, (a^i_{n-2}, a^{i-1}_{n-1}, a^i_{n-1}) \right\rangle$$

Construct a ring of mn processes with an initial state

$$\left(a^{m-1}_0, a^{m-1}_1, \ldots, a^{m-1}_{n-1}, \ldots, a^1_0, a^1_1, \ldots, a^1_{n-1}, a^0_0, a^0_1, \ldots, a^0_{n-1}\right)$$

In this state, every process whose index is a multiple of n is enabled. If each process executes its enabled action, we obtain the following state.

$$\left(a^0_0, a^{m-1}_1, \ldots, a^{m-1}_{n-1}, \ldots, a^2_0, a^1_1, \ldots, a^1_{n-1}, a^1_0, a^0_1, \ldots, a^0_{n-1}\right)$$

If each propagation executes $n-1$ more times, we reach the following state.

$$\left(a^0_0, a^0_1, \ldots, a^0_{n-1}, \ldots, a^2_0, a^2_1, \ldots, a^2_{n-1}, a^1_0, a^1_1, \ldots, a^1_{n-1}\right)$$

Every x_i now holds the initial value of x_{i-n}, therefore a livelock exists by Lemma 3.3. □

Lemma 3.6. *Assume a unidirectional ring protocol of symmetric, self-disabling processes. The protocol has a livelock if and only if there exist some m propagations with some period n, where the $(i-1)$th propagation leads the ith propagation for each index i.*

Proof. Consider a fixed state C in the livelock where m processes are enabled at indices $i_0, \ldots, i_{m-1}$. Between two visitations of C, the propagation which started at index i_j has shifted to index i_{j+k} for some $k \in \mathbb{N}_m$. Regardless of the value of k, we know that if C is visited m times after the initial visitation, the propagation which started at index i_j will be at $i_{j+mk} = i_j$. Thus, each of the m propagations will repeat at least every mth time the system reaches C. Such a list of propagations is necessary to form a livelock, and Lemma 3.5 shows it is sufficient, thus completing the proof. □

4 Tiling

With our new characterization of livelocks in a unidirectional ring protocol from Lemma 3.6, we can explore the difficulty of livelock detection. We use the protocol graph as an intuitive bridge between problems. To complete the bridge, we introduce a well-studied undecidable problem, the domino problem, and reduce livelock detection to one of its variants.

4.1 Variants of the Domino Problem

Problem 4.1 (The Domino Problem)**.**

- **Input:** A set of square tiles with a color (label) on each edge. All tiles are the same size.
- **Question:** Can copies of these tiles cover an infinite plane by placing them side-by-side, without changing tile orientations, such that edge colors match where tiles meet? In other words, can the following be satisfied for each tile $T[i,j]$ on the plane?

$$(T[i,j].N = T[i-1,j].S) \wedge (T[i,j].W = T[i,j-1].E)$$

where $T[i,j].N$ is the color on the north edge of tile $T[i,j]$. Similarly, the $.S$, $.W$, and $.E$ suffixes refer to south, west, and east edge colors of their respective tiles.

The domino problem was introduced by Wang [26], and the square tiles are commonly referred to as Wang tiles. Berger showed the problem to be undecidable [4]. Specifically, the problem is co-semi-decidable, also written as Π^0_1-complete using the arithmetical hierarchy notation of Rogers [23].

A tile set is *NW-deterministic* when each tile in the set can be identified uniquely by its north and west edge colors. In this case, if a tile meets another at its southwest (resp. northeast) corner, then the tile to its south (resp east) side is uniquely determined. Kari proved that the domino problem remains undecidable for NW-deterministic tile sets [21].

Problem 4.2 (The Periodic Domino Problem)**.** This domino problem asks whether an infinite plane can be covered by placing copies of a fixed rectangular arrangement of tiles side-by-side such that a repeating pattern forms. In other words, can Problem 4.1 be solved such that there exist m and n such that the following be satisfied for each tile $T[i,j]$ on the plane?

$$(T[i,j] = T[i+m,j]) \wedge (T[i,j] = T[i,j+n])$$

Problem 4.2 is equivalent to asking whether a torus can be completely covered using the same tiling rules. Gurevich and Koriakov [20] give a semi-algorithm which terminates if the given tile set can periodically tile the plane or cannot tile the plane, otherwise it does not halt and the plane can be tiled, but not periodically. It follows that this problem is semi-decidable, also written as Σ^0_1-complete using notation of Rogers [23].

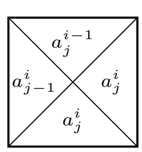
Action tiles. A tile is *SE-identical* when it has identical south and east edge colors. For such sets, we refer to the south and east edge colors as of a tile $T[i,j]$ as $T[i,j].SE$. We write (a,b,c) to denote such a tile with colors a, b, c, and c on its west, north, east, and south edges respectively. A set of SE-identical tiles is *W-disabling* when no two tiles which have the same west color have matching north and south colors respectively. In other words, a SE-identical tile set is W-disabling if and only if for every tile (a,b,c) in the set, no color d exists such that (a,c,d) is also in the set. Due to the following lemma, we use the term *action tile* strictly to denote tiles in a SE-identical W-disabling tile set and *action tile set* to denote the set itself.

The triples that we use to represent tiles in an action tile set are subject to the same constraints as actions in a unidirectional ring protocol of symmetric, self-disabling processes. That is, the W-disabling constraint for tiles is equivalent to the self-disabling constraint for actions. It states that, for every triple (a,b,c) in the set, no d exists such that (a,c,d) is also in the set. As such, we have a bijection between these kinds of tile sets and protocols.

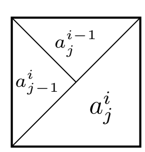
4.2 Equivalence to Livelock Detection

Lemma 4.3. *There is a bijective function which maps a unidirectional ring protocol of self-disabling processes to an action tile set such that the protocol contains a livelock if and only if the tile set admits a periodic tiling. The mapping preserves determinism (resp. NW-determinism) in the protocol (resp. tile set).*

Proof. Recall that a livelock can be characterized by a list of m periodic propagations of length n where each propagation leads the next one in the list (and the last leads the first). From Remark 3.1, we know that this is equivalent to an $m \times n$ matrix where for each row i and column j, the element a^i_j forms an action $(a^i_{j-1}, a^{i-1}_j, a^i_j)$ in the protocol with the help of its west and north neighbors a^i_{j-1} and a^{i-1}_j (with indices computed modulo the matrix bounds). It is straightforward to see that satisfying these constraints is equivalent to solving the periodic domino problem, where each action $(a^i_{j-1}, a^{i-1}_j, a^i_j)$ must exist as a tile in the action tile set as shown in Figure 3.



(a) Wang Tile



(b) Simplified

Figure 3: Tile for action $(a^i_{j-1}, a^{i-1}_j, a^i_j)$.

We already know that there is a bijection between action tile sets and unidirectional ring protocols of symmetric, self-disabling processes since the tiles and actions respectively are triples conforming to the same conditions. Therefore, the periodic tiling problem for action tile sets is equivalent to the livelock detection problem for unidirectional ring protocols of symmetric, self-disabling processes.

Similarly, an action tile set is NW-deterministic if and only if its corresponding protocol is deterministic. That is, for each triple (a, b, c) in the set, no triple (a, b, d) exists in the set where $c \neq d$. Thus, our bijection function (the identity) preserves determinism. ☐

Example 4.4. *Fictional "a-b-c" protocol with a livelock.*

Figure 4 shows the graph of our example unidirectional ring protocol where each arc corresponds to an action. Note that the labels $a_0, \ldots, c_4$ are constants which could equivalently be changed to numbers $0, \ldots, 13$. The protocol does not attempt to function in any meaningful way, but it does provide an interesting livelock. First, propagations which characterize the livelock do not correspond to simple cycles in the protocol's graph. Secondly, 3 of these 6 propagations correspond to walks which are unique regardless of the starting node. The 3-coloring protocol of Example 3.4 has a rather boring livelock by comparison since all propagations form walks which are simple cycles in its graph. Further, these closed walks are identical if we disregard the starting node.

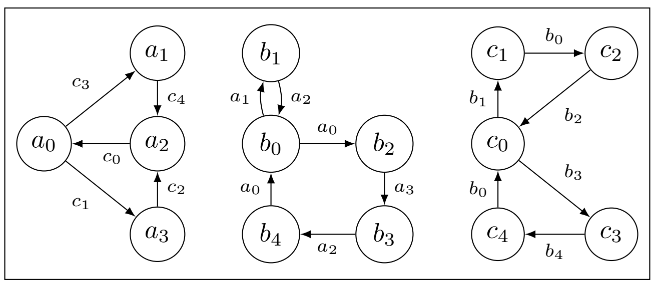


Figure 4: "*A*-b-c" protocol graph.

A livelock can be found by taking a walk through the graph. We start by choosing a closed walk starting with node a_2 and visiting nodes a_0, a_1, a_2, a_0, a_3, and a_2 without considering which arcs were taken.

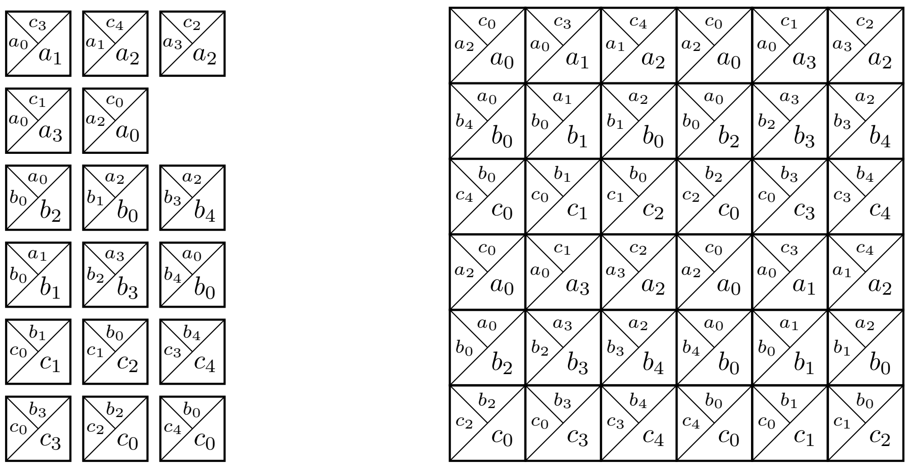
1. Using the previous nodes as arc labels a_0, a_1, a_2, a_0, a_3, and a_2, start from node b_4 to form a closed walk visiting nodes b_0, b_1, b_0, b_2, b_3, and b_4. This corresponds to the periodic propagation:
$\langle(b_4, a_0, b_0), (b_0, a_1, b_1), (b_1, a_2, b_0), (b_0, a_0, b_2), (b_2, a_3, b_3), (b_3, a_2, b_4)\rangle$
2. Using the previous nodes as arc labels b_0, b_1, b_0, b_2, b_3, and b_4, start from node c_4 to form a closed walk visiting nodes c_0, c_1, c_2, c_0, c_3, and c_4. This corresponds to the periodic propagation:
$\langle(c_4, b_0, c_0), (c_0, b_1, c_1), (c_1, b_0, c_2), (c_2, b_2, c_0), (c_0, b_3, c_3), (c_3, b_4, c_4)\rangle$
3. Use previous nodes as arc labels, start at node a_2, etc.
4. Use previous nodes as arc labels, start at node b_0, etc.
5. Use previous nodes as arc labels, start at node c_2, etc.
6. Using the previous nodes as arc labels c_0, c_3, c_4, c_0, c_1, and c_2, start from node a_2 to form a closed walk visiting nodes a_0, a_1, a_2, a_0, a_3, and a_2. We started with this same sequence of nodes, therefore we are done and have found the first periodic propagation to be:
$\langle(a_2, c_0, a_0), (a_0, c_3, a_1), (a_1, c_4, a_2), (a_2, c_0, a_0), (a_0, c_1, a_3), (a_3, c_2, a_2)\rangle$

To compactly illustrate all 6 propagations, follow the method of Remark 3.1 and construct a matrix M where, for each row i and column j, the elements $(M[i, j-1], M[i-1, j], M[i, j])$ form the jth action of the

ith propagation of our livelock.

$$M = \begin{bmatrix} a_0 & a_1 & a_2 & a_0 & a_3 & a_2 \\ b_0 & b_1 & b_0 & b_2 & b_3 & b_4 \\ c_0 & c_1 & c_2 & c_0 & c_3 & c_4 \\ a_0 & a_3 & a_2 & a_0 & a_1 & a_2 \\ b_2 & b_3 & b_4 & b_0 & b_1 & b_0 \\ c_0 & c_3 & c_4 & c_0 & c_1 & c_2 \end{bmatrix}$$

The equivalent periodic tiling problem has an input tile set and solution shown in Figure 5. Recall that the solution is a block of tiles which can have a copy of itself placed on any side without breaking the tiling rules, therefore it can be used to periodically tile the infinite plane.



(a) Wang Tile Set (b) Periodic Block

Figure 5: Instance and solution for the periodic domino problem which corresponds to finding a livelock in the "a-b-c" protocol.

4.3 Equivalent Tile Sets

The remainder of this section shows how to transform a NW-deterministic Wang tile set into a NW-deterministic action tile set which is equivalent with respect to the domino problems. This gives us the tools to reduce the periodic domino problem to livelock detection in the next section which proves that livelock detection is undecidable for unidirectional ring protocols of symmetric, deterministic, self-disabling processes.

Lemma 4.5. *For any set of SE-identical tiles which is not W-disabling, a W-disabling set of SE-identical tiles (i.e., an action tile set) exists which gives the same result to Problem 4.1 and Problem 4.2 and preserves NW-determinism.*

Proof. Recall that if a SE-identical tile set is W-disabling, then for every tile (a, b, c), there exists no tile (a, c, d) in the set for any d. If a tile set does contain such tiles, we can create a new tile set which is W-disabling. The new tile set has colors: $a_\rightarrow$ and $a_\uparrow$ for every color a in the original tile set, abc for every tile (a, b, c) in the original set, and a new color \$. The new set has tiles $(a_\rightarrow, b_\uparrow, abc)$, $(abc, \$, c_\rightarrow)$, $(\$, abc, c_\uparrow)$, and $(c_\uparrow, c_\rightarrow, \$)$ for each tile (a, b, c) in the original set. This reduction is shown clearly in Figure 6.

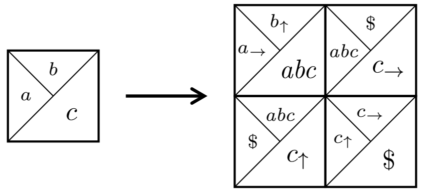


Figure 6: Transform (a, b, c) tile to 4 W-disabling tiles.

Tiling correspondence. Observe that if a tile with a color of the form abc is placed on the plane, we can determine three other tiles which must be placed near it to form the 2×2 arrangement shown in Figure 6. Two of these are determined since the color abc appears on exactly three tiles in the set (for the W, N, and SE edges). The third tile $(c_\uparrow, c_\rightarrow, \$)$ is determined since no other tile has a color $c_\uparrow$ on its west edge (or $c_\rightarrow$ on its north edge).

Conversely, if a tile of the form $(c_\uparrow, c_\rightarrow, \$)$ is placed on the plane, its west neighbor must have the form $(\$, abc, c_\uparrow)$ for some a and b corresponding to the original set of colors. After knowing these a and b, the two tiles to the north are determined due to the reasoning in the previous paragraph. Thus, any valid tiling T' using the new tile set consists of 2×2 blocks corresponding to the tiles in the original set. Further, since the $ colors must match across these 2×2 blocks, the blocks must be aligned.

For correspondence, it remains to show that a valid tiling T exists using the original tile set if and only if a valid tiling T' exists using the new set. This is easy to see since two tiles (a, b, c) and (x, y, z) in the original set can border each other if and only if their corresponding 2×2 blocks in the new set can border each other.

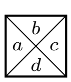
The new tile set is W-disabling. We want to show that for every tile (a, b, c) in the new set, there does not exist another tile (a, c, d) in the set for any d.

Partition the new tile set into four classes whose west edge colors have the form $\texttt{X}_\rightarrow$, $\texttt{XXX}$, $\texttt{X}_\uparrow$, and $ respectively. Note that the forms of the north and southeast edge colors are also the same across tiles of the same class. Within each of these classes, the form of the north color differs from the form of the southeast color. Thus, no tile has a north color which matches the southeast color of a tile in the same class. Since tiles of different classes have different west colors, this implies that the new tile set is W-disabling.

The new tile set preserves NW-determinism. Recall that a tile set is NW-deterministic when for every tile (a, b, c), there does not exist another tile (a, b, d) in the set for any $d \neq c$. If this is the case in the original set, then any tile in the new set with a west color of $a_\rightarrow$ and a north color of $b_\uparrow$ for any a and b has a uniquely determined southeast color.

Each other tile in the new set (those with $ on some edge) can be uniquely identified by its west or north color. For any abc, a tile whose west color is abc uniquely has the form $(abc, \$, c_\rightarrow)$. Similarly, a tile whose north color is abc uniquely has the form $(\$, abc, c_\uparrow)$. Lastly, for any c, a tile whose west color is $c_\uparrow$ uniquely has the form $(c_\uparrow, c_\rightarrow, \$)$. This covers all forms of tiles in the new set, therefore the new set preserves NW-determinism. ☐

Lemma 4.6. *For any set of Wang tiles, an equivalent set of SE-identical tiles exists which gives the same result to Problem 4.1 and Problem 4.2 and preserves NW-determinism.*

Proof. For each tile 

in the set of Wang tiles, create a new color $abcd$. Next, for each color $abcd$ in the new tile set, construct an SE-identical tile $(uvaw, xyzb, abcd)$ for every pair of colors $uvaw$ and $xyzb$ where a as the third letter of $uvaw$ and b is the fourth letter of $xyzb$.

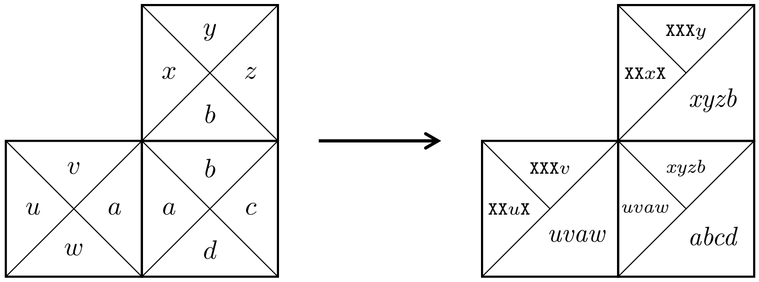


Figure 7: Transform Wang tiles to SE-identical tiles.

Recall from Problem 4.1 that a tiling is valid if and only if the following formula is satisfied for every tile $T[i, j]$.

$$(T[i, j].N = T[i-1, j].S) \wedge (T[i, j].W = T[i, j-1].E)$$

Consider a tiling T' using the new set of SE-identical tiles. For any $T'[i, j] = (uvaw, xyzb, abcd)$, we know that $T'[i, j-1].E = uvaw$ and $T'[i-1, j].S = xyzb$ must hold. By the construction of the new set, this is possible if and only if there are three Wang tiles from the original set whose edge colors correspond with $uvaw$, $xyzb$, and $abcd$ which can be placed on a plane at $T[i, j-1]$, $T[i-1, j]$, and $T[i, j]$ respectively (as illustrated on the left side of Figure 7). As such, a valid tiling T' exists using the new set if and only if a valid tiling T exists using the original set.

This transformation also preserves NW-determinism. If the input tile set is NW-deterministic, then we know a tile at $T[i, j]$ is fully determined if we know both tiles $T[i-1, j]$ and $T[i, j-1]$. Since one SE-identical tile $(uvaw, xyzb, abcd)$ is created for each of these cases where $T[i, j-1] = uvaw$, $T[i-1, j] = xyzb$, and $T[i, j] = abcd$, the new tile set is NW-deterministic when the input tile set is NW-deterministic. $\square$

5 Decidability Results

This section presents the undecidability of verifying livelock freedom in symmetric unidirectional ring and bidirectional chain protocols. The proofs of the theorems in this section heavily rely on the results of previous sections. Moreover, the results of this section assume a *locally-conjunctive invariant*, which means that the predicate defining legitimate states has the form $(\forall i : L(x_{i-1}, x_i))$, where $L(x_{i-1}, x_i)$ is a predicate checkable by process P_i.

Theorem 5.1. *Livelock detection on a unidirectional ring of symmetric, deterministic, finite-state, self-disabling processes is undecidable (Σ_1^0-complete).*

Proof. Given a set of NW-deterministic Wang tiles, we can form a protocol which has a livelock on a symmetric unidirectional ring if and only if the set can form a periodic tiling. We can transform a NW-deterministic tile set to also be SE-identical by Lemma 4.6, then transformed to be W-disabling Lemma 4.5 (i.e, an action tile set), and finally transformed into a symmetric unidirectional ring protocol by Lemma 4.3. $\square$

Corollary 5.2. *Detection of a livelock where exactly one process is enabled at all times on a unidirectional ring of symmetric, deterministic, finite-state, self-disabling processes is undecidable (Σ_1^0-complete).*

Proof. For brevity, we use the term *single-propagation livelock* to denote an infinite execution which has exactly one process enabled in all states. Our goal is, given a unidirectional ring protocol p, to construct another protocol p' such that p' has a single-propagation livelock if and only if p has a livelock.

Without loss of generality, we can assume each process P_i acting under p has a single variable x_i. Let each P_i have transition function ξ, and recall from Definition 2.1 that we can define all actions of P_i as:

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

For the reduction, we are only interested in single-propagation livelocks of p' and therefore ignore its other behaviors. In doing so, we find that all such livelocks of p' require that its c variables form a 2-coloring and that its e variables are used to circulate a token much like a binary token ring. Since no distinguished process exists for token circulation, we use a variable d to mark processes which are acting as a distinguished process. When a process P_i in p' acts, and $c_i = e_i$, it will change its x_i variable if a process under p would also change its x_i variable given the current values of x_{i-1} and x_i. Since $c_{i-1} \neq c_i$, we simulate p in a way which prevents one propagation from catching up to, and colliding with, another. Finally, a distinguished process P_i (marked by $d_i = 1$) stops circulating the token when no p action is enabled, which is flagged by the g_{i-1} variable. Thus, p' will have a single-propagation livelock if and only if some execution of p does not terminate.

Let each process P_i in p' have a variable x_i with the same domain as in p and four boolean variables c_i, d_i, e_i, and g_i. The p' protocol is defined as follows, where $d_i = (e_{i-1} = e_i)$ is equivalent to $(d_i \wedge (e_{i-1} = e_i)) \vee (\neg d_i \wedge (e_{i-1} \neq e_i))$.

$$
\begin{aligned}
&(c_{i-1} \neq c_i) \wedge (d_i = (e_{i-1} = e_i)) \wedge (d_i \implies \neg g_{i-1}) \\
&\qquad\qquad \longrightarrow \quad g_i := (d_i \vee g_{i-1}) \wedge ((x_{i-1}, x_i) \in \textsc{Pre}(\xi)); \\
&\qquad\qquad\qquad\quad\; x_i := \begin{cases} \xi(x_{i-1}, x_i); & \text{if } (c_i = e_i) \wedge ((x_{i-1}, x_i) \in \textsc{Pre}(\xi)) \\ x_i; & \text{otherwise} \end{cases} \\
&\qquad\qquad\qquad\quad\; e_i := \neg e_i;
\end{aligned}
$$

A livelock of p' simulates some maximal execution (either terminating or infinite) of p. In other words, a livelock of p' will change some x value if and only if that x value could be changed (in the same way) under p. It is sufficient to show that the following two statement hold: (1) If some P_i changes its x_i under p', then P_i could also change its x_i the same way under p. (2) In a p' livelock, if some process P_i would be enabled to under p given its current x_{i-1} and x_i values, then some (possibly different) process P_j will eventually change its x_j value.

Statement (1) trivially holds since each process P_i can change x_i by assigning $x_i := \xi((,x)_{i-1}, x_i)$ under both p and p'. This is also the only way that x_i can change under either protocol.

Statement (2) holds since every other time a process P_i acts under p', it will change x_i if $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$. This can be seen since P_i flips its e_i bit every time it acts, yet only can change x_i when $c_i = e_i$. Each process acts infinitely often in a livelock of p'. Therefore, some x_i which can change under p (i.e., if $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$) will eventually be changed under a livelock of p'.

A single-propagation livelock of p' requires some d value to be true. Let N equal the number of processes in the ring. Since all actions of a process P_i in p' require $c_{i-1} \neq c_i$, and c values do not change, the c values must form a 2-coloring. This also means that N is even.

Some d value must be true in (all states of) a single-propagation livelock. For a contradiction, assume a livelock of p' exists where $d_i = 0$ for all i. We know that $c_{i-1} \neq c_i$ for all i, therefore a process P_i is enabled when $e_{i-1} \neq e_i$. If all e values are the same, then no process is enabled, otherwise at least two processes are enabled. Thus, at least one d value must be true for the livelock to exist.

If p is livelock-free, then p' does not have a single-propagation livelock. To show this by contradiction, assume p is livelock-free and p' has a single-propagation livelock. We know that some d value must be true, therefore let P_q have this $d_q = 1$ value. Let P_s be a process for which $d_s = 1$ and the processes between P_q and P_s have $d_{q+1} = \cdots = d_{s-1} = 0$. Note that P_q and P_s are the same process when only one d value in the ring is true.

Consider a state of the p' livelock where P_q is enabled to act and $(x_{i-1}, x_i) \notin \textsc{Pre}(\xi)$ for all i. This state will eventually be reached since p' simulates an execution of p, and all executions of p are terminating. From this state, we will show that P_s cannot eventually act, thereby contradicting the assumption.

When P_q acts, it assigns $g_q := 1$ since $(x_{q-1}, x_q) \notin \textsc{Pre}(\xi)$. Next, P_{q+1} assigns $g_{q+1} := 1$ since $g_q \wedge ((x_q, x_{q+1}) \notin \textsc{Pre}(\xi))$. Likewise, each subsequent P_i assigns $g_i := 1$ for $i \in \{q+2, \ldots, s-1\}$. Since $g_{s-1} = 1$ after P_{s-1} acts, and $d_s = 1$, we know that P_s does not become enabled. No process is enabled at this point, thus a single-propagation livelock does not exist in p'. Our assumption is contradicted, therefore p' does not have a single-propagation livelock when p is livelock-free.

If p has a livelock, then p' has a single-propagation livelock. By Lemma 3.6 we know that a livelock of p is characterized by m propagations of period n (for some m and n) where the $(i-1)$th propagation leads the ith propagation for all i. Assume we have such a livelock, we want to find a single-propagation livelock of p' on a ring of size $N = mn$. We are able to assume n is even since the propagations are periodic and we can simply double their periods if n were odd.

We wish to find a predicate ψ such that, from an initial state satisfying ψ, any execution of p' will return to ψ after N actions. If states satisfying ψ have exactly one process enabled to act, and ψ is satisfiable, then p' has a single-propagation livelock. Let ψ be defined by a conjunction of the following constraints:

- $c_i = i \ \mathtt{mod}\ 2$ for each i.
- $d_0 = 1$ and all other d values are $d_1 = \cdots = d_{N-1} = 0$.
- $e_0 = \cdots = e_{N-1}$.
- $g_{N-1} = 0$.
- The x values form a state which exists in a livelock under p.
- For some k, for each i, the inequality $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$ holds if and only if $k = i \ \mathtt{mod}\ n$.

Clearly the first four constraints can be satisfied, but the remaining two constraints rely on the p protocol. Since p has such a livelock characterized by m propagations of period n, we can use the proof of Lemma 3.5 to find some x values to satisfy these two constraints. Thus ψ is satisfiable. Notice that the only enabled process is P_0 when ψ is satisfied, therefore it remains to show that any execution from ψ will return to ψ after N steps.

Each of the N processes $P_0, \ldots, P_{N-1}$ will act. We know P_0 acts first and assigns $e_0 := \neg e_0$. Each other process P_i has $d_i = 0$ and therefore become enabled when $e_{i-1} \neq e_i$. Since P_0 assigns e_0 to be different from all other e values, P_1 is the next to act and assigns $e_1 := e_0$. By the same reasoning, processes $P_2, \ldots, P_{N-1}$ are guaranteed to act next in order.

After N actions from a state satisfying ϕ, we know $g_{N-1} = 0$ since $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$ for some i. To elaborate, a process P_i assigns $g_i := (d_i \vee g_{i-1}) \wedge ((x_{i-1}, x_i) \notin \textsc{Pre}(\xi))$ when it acts. Since only $d_0 = 1$, this means $g_{N-1} = 1$ after the actions of $P_0, \ldots, P_{N-1}$ only if $(x_{i-1}, x_i) \notin \textsc{Pre}(\xi)$ for each index i.

When $P_0, \ldots, P_{N-1}$ act, either all or none of the processes P_i which initially (before P_0 acts) have $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$ will change their x_i values, and no other process P_j will change its x_j value. Since all e values are the same in states satisfying ψ, when $P_0, \ldots, P_{N-1}$ act in order, either all of the even-indexed or all of the odd-indexed processes P_i have $e_i = c_i$ when they act. As such, either all even-indexed or all odd-indexed processes are eligible to change their x values. Therefore, a process P_j which initially has $(x_{j-1}, x_j) \notin \textsc{Pre}(\xi)$ will not change its x_j value even if P_{j-1} changes x_{j-1} to satisfy $(x_{j-1}, x_j) \notin \textsc{Pre}(\xi)$. Since n is even, all processes P_i which initially have $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$ have even indices, or they all have odd indices. Thus, either all or none of the processes P_i which initially have $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$ will change their x_i values.

After $P_0, \ldots, P_{N-1}$ act, the x values form a state which exists in a livelock under p, and processes P_i for which $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$ will be spaced equally around the ring at every nth position (i.e., the last two conditions of ψ will be satisfied). We have seen that either all or none of the processes P_i which initially have $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$ will act. When each of these processes P_i acts to change its x_i value, it also changes $(x_i, x_{i+1}) \in \textsc{Pre}(\xi)$ from false to true for P_{i+1} since we started in a state where x values form a state in a p livelock. As such, after $P_0, \ldots, P_{N-1}$ act, processes P_i for which $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$ will still be spaced equally around the ring at every nth position. Further, the x values still form a state which exists in a livelock under p.

We have shown that if $P_0, \ldots, P_{N-1}$ act in order from a state in ψ, then the system reaches a state where all e values are equal, $g_{N-1} = 0$, and every nth process P_i has $(x_{i-1}, x_i) \in \textsc{Pre}(\xi)$. Since the c and d variables do not change under p', the system returns to ψ after N actions. Thus, p' has a single-propagation livelock when p has a livelock.

It has been shown that p has a livelock if and only if p' has a single-propagation livelock. It is therefore undecidable (Σ_1^0-complete) whether a single-propagation livelock exists in a unidirectional ring protocol of symmetric, deterministic, finite-state, self-disabling processes. $\square$

Corollary 5.3. *Verifying strong or weak stabilization on a unidirectional ring is undecidable (Π_1^0-complete).*

Proof. Given a unidirectional ring protocol p, we can define a locally-conjunctive invariant such that p is self-stabilizing if and only if p does not contain a livelock. This invariant is simply $(\forall i : L_i)$ where L_i is any state where process P_i is disabled, which means a state is legitimate if and only if process is not enabled to act. Obviously closure holds since the system has no action enabled in a legitimate state. Convergence of p holds if and only if no infinite execution (i.e., livelock) exists. Thus, p is strongly stabilizing if and only if it does not contain a livelock.

For weak stabilization [17], convergence is satisfied if, for every illegitimate state, there exists some execution which reaches a legitimate one. In other words, even if there are livelocks in illegitimate states (assuming the reachability of legitimate states), a strongly fair scheduler would guarantee the reachability of some legitimate state. We know that no action can increase the number of enabled processes due to the self-disabling property, but strong fairness can reliably bring any state to one with either zero or one enabled processes. Thus, p is weakly stabilizing if and only if it does not contain a livelock where exactly one process is enabled.

Verifying if p is not strongly or weakly stabilizing is therefore Σ_1^0-complete (semi-decidable) due to Theorem 5.1 and Corollary 5.2 respectively. Thus, deciding either type of stabilization for p is Π_1^0-complete (co-semi-decidable). We know Π_1^0 is also an upper bound due to Apt and Kozen [3] who show that verifying temporal properties of finite-state concurrent systems is Π_1^0-complete in general. Verifying either type of stabilization is therefore Π_1^0-complete. $\square$

Corollary 5.4. *Verifying strong or weak stabilization on a segment of a bidirectional chain of symmetric, deterministic, self-disabling processes is undecidable (Π_1^0-complete).*

Proof. Given a unidirectional ring protocol p, we can construct a bidirectional chain protocol p' which has a livelock if and only if p has a livelock where exactly one process is enabled at all times. Like in the proof of Corollary 5.2, we call this kind of p livelock a *single-propagation livelock*. Using the same approach as Corollary 5.3, we reduce our problem of verifying stabilization to verifying livelock freedom (of p') by defining a state of p' to be legitimate when no process is enabled to act.

We construct p' by a modification of Dijkstra's four-state token passing protocol [7] which operates on a chain topology. Dijkstra's protocol relies on distinguished bottom and top processes (the first and last processes). It ensures that eventually exactly one process has the token (is enabled to act) which is passed along the chain from the bottom to top and top to bottom indefinitely.

Though we leverage Dijkstra's protocol, the end processes of p' do not act as the bottom and top processes. In fact, they are defined to have no actions and therefore do not contribute to livelocks at all. We give each process P_i three new variables $wall_i$, x_i, and y_i in addition to the two boolean variables up_i and z_i used by Dijkstra's protocol. The $wall_i$ variable is boolean, and if it is true, then P_i acts as either the bottom or top process in Dijkstra's protocol when up_i is true or false respectively. In this way, the initial state determines which processes act as bottom and top processes in Dijkstra's protocol.

We show that if processes in a segment $P_q, \ldots, P_s$ are executing Dijkstra's protocol (where P_q and P_s behave as bottom and top processes respectively), then they can simulate the p protocol on a unidirectional ring by using their x and y variables. When a token is being passed from P_q up to P_s, each process P_i where $i \in \{q+1, \ldots, s-1\}$ can only act if it would also be enabled under p with its current x_{i-1} and x_i values. Similarly, the top process P_s acts when it has the token and would be enabled under p with its current x_{s-1} and y_s values (where y_s in p' stands for x_s in p). When the token is being passed from P_s down to P_q, the y variables are used to copy the current state of P_s, allowing P_q to assign its x_q value as if P_s were its predecessor in a unidirectional ring executing protocol p.

Without loss of generality, we can assume each process P_i acting under p has a single variable x_i. Let each P_i have transition function ξ, and recall from Definition 2.1 that we can define all actions of P_i as:

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

Let the end processes of p' have no actions, therefore they cannot contribute to a livelock. Define the actions of each other process P_i in p' as follows.

$$\begin{array}{rlll}
\mathit{Bot}: & \neg up_{i+1} \wedge \quad (z_i = z_{i+1}) \wedge & \phi_{Bot}(i) \longrightarrow x_i := \xi(y_{i+1}, x_i);\ z_i := \neg z_i; & \\
\mathit{Rise}: (z_{i-1} \neq z_i) \wedge & (up_{i+1} \vee (z_{i-1} \neq z_{i+1})) \wedge & \phi_{Rise}(i) \longrightarrow x_i := \xi(x_{i-1}, x_i);\ z_i := z_{i-1}; & up_i := 1; \\
\mathit{Top}: (z_{i-1} \neq z_i) \wedge & & \phi_{Top}(i) \longrightarrow y_i := \xi(x_{i-1}, y_i);\ z_i := z_{i-1}; & \\
\mathit{Fall}: (z_{i-1} = z_i) \wedge up_i \wedge \neg up_{i+1} \wedge & (z_{i-1} = z_{i+1}) \wedge & \phi_{Fall}(i) \longrightarrow y_i := y_{i+1}; & up_i := 0;
\end{array}$$

Disregarding the ϕ formulas and the x and y variables, these four actions make a version Dijkstra's token passing protocol which is modified to be deterministic and have self-disabling processes. The *Bot* and *Top* actions correspond to the bottom and top processes of that protocol, which are assumed to have constant up_i values of true and false respectively. The *Rise* and *Fall* actions are executed by the intermediate processes to pass the token up and down the chain respectively.

The ϕ predicates strengthen the guards of Dijkstra's token passing protocol. First, they ensure a process P_i acts as a bottom or top process when its $wall_i$ variable is true. When $wall_i$ is true, the value of up_i determines whether P_i acts as a bottom or top process. Next, there are conditions on the x and y variables which correspond to guards of actions in the p protocol. Lastly, the $\mathit{SegCheck}(i)$ predicate ensures that a process P_i will not act when P_{i-1} is a top process or when P_{i+1} is a bottom process. As such it is defined as: $\mathit{SegCheck}(i) = \neg(wall_{i-1} \wedge \neg up_{i-1}) \wedge \neg(wall_{i+1} \wedge up_{i+1})$. Define the ϕ predicates as follows.

$$\begin{aligned}
\phi_{Bot}(i) &= wall_i \wedge up_i \wedge ((y_{i+1}, x_i) \in \textsc{Pre}(\xi)) \wedge \mathit{SegCheck}(i) \\
\phi_{Rise}(i) &= \neg wall_i \qquad\quad \wedge ((x_{i-1}, x_i) \in \textsc{Pre}(\xi)) \wedge \mathit{SegCheck}(i) \\
\phi_{Top}(i) &= wall_i \wedge \neg up_i \wedge ((x_{i-1}, y_i) \in \textsc{Pre}(\xi)) \wedge \mathit{SegCheck}(i) \\
\phi_{Fall}(i) &= \neg wall_i \qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge \mathit{SegCheck}(i)
\end{aligned}$$

In a livelock of p', some segment $P_q, \ldots, P_s$ executes Dijkstra's four-state token passing protocol on the z and up variables. Assume p' has a livelock. Let $P_q, \ldots, P_s$ be a segment of the chain such that all processes in that segment act infinitely often, but P_{q-1} and P_{s+1} (eventually) never act. Obviously some segment $P_q, \ldots, P_s$ which acts infinitely often must exist in a livelock of p'. Processes P_{q-1} and P_{s+1} are also guaranteed to exist since, even if all other processes act infinitely often, the end processes of the chain are defined to never act.

It is the case that P_q acts as the bottom process and P_s acts as the top. For a contradiction, assume P_q does not act as the bottom process or P_s does not act as the top. By the properties of Dijkstra's protocol, if P_q acts infinitely often and is not the bottom process, then it will pass its token to P_{q-1} infinitely often. Likewise, if P_s acts infinitely often and is not the top process, then it will pass its token to P_{s+1} infinitely often. Since P_{q-1} and P_{s+1} do not act in this livelock, passing a token to either of them will destroy it. We clearly cannot destroy tokens infinitely often in a livelock of p', therefore by contradiction, P_q must act as the bottom process and P_s must act as the top.

Recall that, due to *SegCheck*, if a process acts as the bottom or top, one of its neighbors will never act. Therefore, since each process in the segment $P_q, \ldots, P_s$ acts infinitely often, none of $P_{q+1}, \ldots, P_{s-1}$ can act as a bottom or top process. Thus, in any livelock of p', some segment $P_q, \ldots, P_s$ of the chain is executing Dijkstra's token passing protocol.

If p' has a livelock, then p has a single-propagation livelock. Assume p' has a livelock. We know that a chain segment $P_q, \ldots, P_s$ exists which executes Dijkstra's token passing protocol on the z and up variables.

Eventually, one token will exist in this segment and process P_s obtains it. Since we know the segment $P_q, \ldots, P_s$ is executing Dijkstra's token passing protocol, the following actions are performed in order. We only show the constraints and effects that each action has on the x and y variables since we have already reasoned about the other variables.

Process	Action	Guard		Assignment
P_s	*Top*	$(x_{s-1}, y_s) \in \textsc{Pre}(\xi)$	$\longrightarrow$	$y_s := \xi(x_{s-1}, y_s);$
P_{s-1}	*Fall*	true	$\longrightarrow$	$y_{s-1} := y_s;$
$\vdots$	$\vdots$		$\vdots$	
P_{q+1}	*Fall*	true	$\longrightarrow$	$y_{q+1} := y_{q+2};$
P_q	*Bot*	$(y_{q+1}, x_q) \in \textsc{Pre}(\xi)$	$\longrightarrow$	$x_q := \xi(y_{q+1}, x_q);$
P_{q+1}	*Rise*	$(x_q, x_{q+1}) \in \textsc{Pre}(\xi)$	$\longrightarrow$	$x_{q+1} := \xi(x_q, x_{q+1});$
$\vdots$	$\vdots$		$\vdots$	
P_{s-1}	*Rise*	$(x_{s-2}, x_{s-1}) \in \textsc{Pre}(\xi)$	$\longrightarrow$	$x_{s-1} := \xi(x_{s-2}, x_{s-1});$

Notice that the sequence of *Fall* actions executed by $P_{s-1}, \ldots, P_{q+1}$ simply copy the value of y_s down to y_{q+1}. Thus, $y_{q+1} = y_s$ when P_q acts.

Consider a unidirectional ring of size $N = s - q$ where each process $P_0, \ldots, P_{N-2}, P_{N-1}$ around the ring has a variable whose current value is $x_q, \ldots, x_{s-1}, y_s$ respectively (taken from our state of the chain). It is easy to see that processes $P_{N-1}, P_0, \ldots, P_{N-2}$ can act in order from this state under protocol p if and only if the sequence of actions above can be performed under p'. Since we have assumed p' has a livelock, this sequence of actions continues indefinitely, thus p has a single-propagation livelock.

If p has a single-propagation livelock, then p' has a livelock. With respect to the x and y variables, our reasoning above shows equivalence between a livelock of p' and a single-propagation livelock of p. However, we assumed that the *wall*, z, and *up* variables could be assigned to admit such a livelock.

Such an assignment is already known which admits some chain segment $P_q, \ldots, P_s$ to execute Dijkstra's token passing protocol on the z and *up* variables under the rules of p'. For the bottom process P_q, let $wall_q = 1$ and $up_q = 1$. For the top process P_s, let $wall_s = 1$ and $up_s = 0$. For the other processes, let $wall_{q+1} = \cdots = wall_{s-1} = 0$.

Thus, we see a livelock of p' exists if and only if a single-propagation livelock of p exists. Determining if the p protocol has a single-propagation livelock is Σ^0_1-complete by Corollary 5.2, therefore determining if p' is livelock-free is at least Π^0_1. Fairness clearly does not affect the livelocks of p', thus the result holds for any fairness assumption. As stated previously, Π^0_1 is an upper bound due to Apt and Kozen [3], thus verifying strong or weak stabilization on a bidirectional chain segment of symmetric, deterministic, self-disabling processes is Π^0_1-complete (co-semi-decidable). □

6 Related Work

This section discusses related work regarding necessary and/or sufficient conditions for livelock freedom and decidability of livelock freedom in parameterized systems. Specifically, Farahat and Ebnenasir investigate sufficient conditions for livelock freedom in symmetric unidirectional ring protocols of self-disabling processes [14]. They also present necessary and sufficient conditions for deadlock detection in symmetric unidirectional and bidirectional ring protocols. This paper complements their work by showing that, even when assuming deterministic and self-disabling properties, livelock freedom on ring and chain topologies is undecidable in general.

Decidability. In [3], Apt and Kozen prove that verifying an LTL formula holds for a parameterized system is Π^0_1-complete. Suzuki [24] builds on this result, showing that the problem remains Π^0_1-complete on symmetric unidirectional ring protocols where only the number of processes is parameterized. Emerson and Namjoshi [11] show that the result holds even when a token which can take two different values is passed around such a ring.

Abello and Dolev [2] show that any Turing machine can be simulated on a bidirectional chain topology in a self-stabilizing manner. Among other variables in their protocol, each process has variables to represent an input tape cell, a working tape cell, and an output. When the Turing machine accepts, rejects, or fails to compute a result (due to cycles or insufficient tape cells) for the given input, the output value of each process will eventually be 1, 0, or $\perp$ respectively. Once a simulation of the Turing machine finishes, it begins again in case some fault corrupts an output value or the input is changed. It is reasonable to say that this protocol implies our Corollary 5.4, though a proof would be complicated by the fact that the protocol relies on distinguished end processes and all executions are infinite.

Regular Model Checking. In regular model checking [1, 5], system states are represented by strings of arbitrary length, and a protocol is represented by a transducer. Let R be the relation of this transducer and let R^+ be its transitive closure. A livelock can then be detected by checking if R^+ maps some string to itself, or in other words, checking if $R^+ \cap R_{id}$ is non-empty, where R_{id} is the identity relation. Of course no algorithm computes R^+ in all cases, therefore heuristic acceleration techniques are used such as widening [25].

Our reasoning strongly resembles that of regular model checking. Briefly, we can interpret the graph of a protocol p as a transducer, where arc and node labels denote input and output symbols respectively (i.e., a Moore machine). Further, let each state of this transducer be both initial and accepting (without generating initial output). A periodic propagation is a closed walk in the graph, or equivalently, it is some strings s and w such that the transducer accepts w^k and outputs s^k for all $k \in \mathbb{N}$. The protocol p therefore has a livelock if and only if some string w exists such that $(\forall k : (w^k, w^k) \in R^+)$, where R is the transducer's relation and R^+ is its transitive closure. We can check if such a string w exists by (1) computing $R^+ \cap R_{id}$, (2) finding the minimal DFA of its input language, and (3) checking if the initial state of the DFA (which will also be an accepting state) takes part in a cycle. While this formulation is interesting, it only applies to finding livelocks in symmetric unidirectional ring protocols, whereas regular model checking can be used with other topologies and LTL properties.

Cutoff Theorems. Emerson *et al.* [9, 10] present cutoff theorems for the verification of temporal logic properties in parameterized systems, where a property $\mathcal{P}$ holds for a parameterized protocol p if and only if $\mathcal{P}$ holds for an instantiation of p with a fixed number of processes k, called the *cutoff*. This method is mainly applicable for properties that are specified in terms of the locality of each process.

7 Conclusions and Future Work

We illustrated that verifying livelock freedom is undecidable for parameterized unidirectional ring and bidirectional chain protocols, where every process has similar code up to variable renaming. While Suzuki [24] shows that the verification of general case temporal logic properties is undecidable for symmetric unidirectional ring protocols, this paper illustrates that the verification problem remains undecidable even if processes are *self-disabling*; i.e., a process is disabled after acting. The proof of undecidability presented in this paper is based on a reduction from the periodic domino problem [26]. We also showed that verifying weak/strong self-stabilization is undecidable for these parameterized unidirectional ring and bidirectional chain protocols. As an extension of this work, we will investigate the design of a framework that will be an integration of our automated synthesis tools [13] and theorem proving. In this framework, we will first synthesize small instances of parameterized systems that are self-stabilizing for a specific number of processes. Then, we will use theorem proving techniques to generalize the synthesized systems for an arbitrary number of processes.

References

[1] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In P. Gardner and N. Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 2004.

[2] J. Abello and S. Dolev. On the computational power of self-stabilizing systems. *Theoretical Computer Science*, 182(1-2):159–170, 1997.

[3] K. R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, 1986.

[4] R. Berger. *The Undecidability of the Domino Problem*. Memoirs ; No 1/66. American Mathematical Society, 1966.

[5] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In E. A. Emerson and A. P. Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.

[6] D. Cachera and K. Morin-Allory. Verification of safety properties for parameterized regular systems. *ACM Transactions on Embedded Computing Systems*, 4(2):228–266, 2005.

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

[8] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. Elsevier, 1990.

[9] E. A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In D. A. McAllester, editor, *CADE*, volume 1831 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2000.

[10] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In R. K. Cytron and P. Lee, editors, *POPL*, pages 85–94. ACM Press, 1995.

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

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

[13] A. Farahat and A. Ebnenasir. A lightweight method for automated design of convergence in network protocols. *TAAS*, 7(4):38, 2012.

[14] A. Farahat and A. Ebnenasir. Local reasoning for global convergence of parameterized rings. In *ICDCS*, pages 496–505. IEEE, 2012.

[15] L. Fribourg and H. Olsén. Reachability sets of parameterized rings as regular languages. *Electronic Notes in Theoretical Computer Science*, 9:40, 1997.

[16] P. Funk and I. Zinnikus. Self-stabilization as multiagent systems property. In *AAMAS*, pages 1413–1414. ACM, 2002.

[17] M. G. Gouda. The theory of weak stabilization. In A. K. Datta and T. Herman, editors, *WSS*, volume 2194 of *Lecture Notes in Computer Science*, pages 114–123. Springer, 2001.

[18] M. G. Gouda and H. B. Acharya. Nash equilibria in stabilizing systems. In R. Guerraoui and F. Petit, editors, *SSS*, volume 5873 of *Lecture Notes in Computer Science*, pages 311–324. Springer, 2009.

[19] M. G. Gouda and N. J. Multari. Stabilizing communication protocols. *IEEE Trans. Computers*, 40(4):448–458, 1991.

[20] Y. Gurevich and I. O. Koriakov. A remark on Berger's paper on the domino problem. *Siberian Mathematical Journal*, 13(2):319–321, 1972.

[21] J. Kari. The nilpotency problem of one-dimensional cellular automata. *SIAM Journal on Computing*, 21(3):571–586, 1992.

[22] H. J. La and S. D. Kim. A self-stabilizing process for mobile cloud computing. In *SOSE*, pages 454–462. IEEE Computer Society, 2013.

[23] H. Rogers. *Theory of recursive functions and effective computability (Reprint from 1967).* MIT Press, 1987.

[24] I. Suzuki. Proving properties of a ring of finite-state machines. *Information Processing Letters*, 28(4):213–214, July 1988.

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

[26] H. Wang, A. Telephone, and T. Company. *Proving Theorems by Pattern Recognition -II.* American Telephone and Telegraph Company, 1961.