# Self-stabilizing Population Protocols

Dana Angluin, James Aspnes*, Michael J. Fischer, and Hong Jiang**

Yale University, Department of Computer Science

**Abstract.** Self-stabilization in a model of anonymous, asynchronous interacting agents deployed in a network of unknown size is considered. Dijkstra-style round-robin token circulation can be done deterministically with constant space per node in this model. Constant-space protocols are given for leader election in rings, local-addressing in degree-bounded graphs, and establishing consistent global direction in an undirected ring. A protocol to construct a spanning tree in regular graphs using $O(\log D)$ memory is also given, where $D$ is the diameter of the graph. A general method for eliminating nondeterministic transitions from the self-stabilizing implementation of a large family of behaviors is used to simplify the constructions, and general conditions under which protocol composition preserves behavior are used in proving their correctness.

## 1 Introduction

In some practical scenarios, a large (and sometimes unknown) number of devices are deployed over a region without fine control of their locations, communication and movement patterns. The devices are all indistinguishable and have only a few bits of memory each. Such scenarios are modeled by the *population protocols* introduced in [1], where families of predicates computable in this model are explored. Graph properties computable in the same model are discussed in [2]. Communication in population protocols occurs through pairwise interaction of anonymous finite-state agents. The number of agents is finite but unbounded. A communication graph describes which pairs of nodes may interact.

In the theoretical literature on distributed computing, a weak fairness condition is usually assumed. Informally, in an infinite fair execution each process or node is given a turn infinitely often. We call this definition *local fairness*. The environment/scheduler is viewed as a powerful adversary who can strategically determine the sequence in which processes are activated, as long as local fairness is preserved. Many impossibility results rely on this assumption. For instance, the impossibility of deterministic self-stabilizing token circulation in uniform rings [3] follows from the assumption that the scheduler can activate the nodes in a round-robin fashion, preserving symmetry and achieving local fairness.

However, in practical distributed systems, such a powerful scheduler seldom exists. The global ordering of computational steps depends on a variety of elements. Temperature and power-supply affect the efficiency of electronic devices.

* Supported in part by NSF grants CNS-0305258 and CNS-0435201.
** Supported by the PORTIA project (NSF grant ITR-0331548).

J.H. Anderson, G. Prencipe, and R. Wattenhofer (Eds.): OPODIS 2005, LNCS 3974, pp. 103–117, 2006.

Local clock frequency influences the progress of each node. For ad hoc networks, the movement of nodes determines possible sequences of interactions. Random delay is usually used in practical leader election and collision detection protocols, which can be viewed as a way to randomize the scheduling of the system.

In the model of population protocols, an alternative fairness condition is assumed, *global fairness*, which better reflects the scheduling properties of many distributed systems. Global fairness puts more constraints on the scheduler, so problems proved impossible under global fairness are also impossible under local fairness. Global fairness also provides a simple conceptual framework for protocol design. For instance, once a task is known to be possible in our model, randomization techniques can be applied to make the protocol work under some weaker fairness conditions.

The responsibility of many systems is to meet certain specifications for well-behavedness such as avoidance of deadlock, fairness among processes, fault tolerance, and other global system properties that cannot be simply modeled as I/O behavior. We extend the model of population protocols to accommodate such tasks. We focus on *self-stabilizing* systems that can start in any global configuration and achieve behavior meeting the task specification by itself. Such systems can tolerate worst-case transient faults.

### 1.1 Other Related Work

Self-stabilizing systems were first introduced by Dijkstra [3]. In his seminal paper, Dijkstra gives three protocols to achieve process mutual-exclusion in rings in a self-stabilizing way. Leader election and token management are fundamental problems in self-stabilization and have been extensively studied in various other models. Each of these result differs from our work in at least one of these aspects: there is an external timeout mechanism to detect deadlocks [4]; each node can access the states of all neighbors at the same time to determine its next state [5]; the protocol is randomized [6, 7, 8], knows the size of the network [6], or has per-node space or message size in $O(\log n)$ [6, 9]; or nodes have unique IDs [10].

Herman [8] proposed a probabilistic synchronous self-stabilizing token-circulation algorithm for identical nodes in an odd ring. Johnen [7] presents a randomized self-stabilizing token circulation protocol on unidirectional anonymous rings. Fairness is enforced by randomization and the fair circulation of *privileges*. The scheduler can only choose nodes that hold a privilege token to make the next step. In our model, nodes are deterministic, and the nondeterminism in the environment (scheduler) is utilized to break symmetry.

Itkis and Levin [11] present a self-stabilizing leader-election protocol for asynchronous networks of identical nameless nodes with arbitrary topology. In their model, each node can set points to neighbors whose state satisfies given properties. Our impossibility result for leader election in connected interaction graphs with arbitrary topology in Section 5.5 shows that their model and ours differ. However, it is an open problem whether our model can simulate theirs in some special classes of interaction graphs.

Distance-2 coloring, also known as neighborhood unique naming, was defined in [12] as the graph labeling problem with conditions at distance 2. The communication networks community have studied variants of this problem, for applications such as assigning radio frequency ranges or time slots to wireless-signal transmitters to avoid interference. The existing results are different from ours in the following aspects: nodes have unique IDs [13] or have the ability to associate incoming messages with their sources [14, 15]; and the protocol has probabilistic rules [13, 14, 15]. We impose a strong anonymity condition: A node does not have innate ability to distinguish different neighbors. To our knowledge no existing protocol is applicable in this model.

## 2 Basic Model

We represent a network by a directed graph $G = (V, E)$ with $n$ vertices numbered 0 through $n-1$ and no multi-edges or self-loops. Each vertex represents a finite-state sensing device, and an edge $(u, v)$ indicates the possibility of a communication between $u$ and $v$ in which $u$ is the *initiator* and $v$ is the *responder*.[1] An "undirected" communication graph refers to a network in which for every edge $(u, v)$, interactions of the forms $(u, v)$ and $(v, u)$ are both possible.

A *protocol* $P(Q, \mathcal{C}, X, Y, O, \delta)$ consists of a finite set of *states* $Q$, a set of *initial configurations* $\mathcal{C}$, a finite set $X$ of *input symbols*, an *output function* $O : Q \to Y$, where $Y$ is a finite set of *output symbols*, and a *transition function* $\delta$ mapping each element of $(Q \times X) \times (Q \times X)$ to a nonempty subset of $Q \times Q$. If $(p', q') \in \delta((p, x), (q, y))$, we call $((p, x), (q, y)) \to (p', q')$ a *transition*. The transition function, and the protocol, is *deterministic* if $\delta((p, x), (q, y))$ always contains just one pair of states. The inputs provide a way for a protocol to interact with an external entity, be it the environment, a user, or another protocol.

A *configuration* is a mapping $C : V \to Q$ specifying the state of each device in the network, and an *input assignment* is a mapping $\alpha : V \to X$. A *trace* $T_G(Z)$ on a graph $G(V, E)$ is an infinite sequence of assignments from $V$ to the symbol set $Z$: $T_G = \lambda_0, \lambda_1, \ldots$ where $\lambda_i$ is an assignment from $V$ to $Z$. $Z$ is called the alphabet of $T_G$. If $Z = X$, we say $T_G$ is an *input trace* of the protocol. Let $C$ and $C'$ be configurations, $\alpha$ be an input assignment, and $u$, $v$ be distinct nodes. We say that $(C, \alpha)$ goes to $C'$ via pair $e = (u, v)$, denoted $(C, \alpha) \xrightarrow{e} C'$, if the pair $(C'(u), C'(v))$ is in $\delta((C(u), \alpha(u)), (C(v), \alpha(v)))$ and for all $w \in V - \{u, v\}$ we have $C'(w) = C(w)$. We say that $(C, \alpha)$ can go to $C'$ in one step, denoted $(C, \alpha) \to C'$, if $(C, \alpha) \xrightarrow{e} C'$ for some edge $e \in E$. Given an input trace $IT = \alpha_0, \alpha_1, \ldots$ we write $C \xrightarrow{*} C'$ if there is a sequence of configurations $C = C_0, C_1, \ldots, C_k = C'$, such that $(C_i, \alpha_i) \to C_{i+1}$ for all $i$, $0 \le i < k$, in which case we say that $C'$ is *reachable* from $C$ given input trace $IT$.

An *execution* is an infinite sequence of configurations and input assignments $(C_0, \alpha_0)$, $(C_1, \alpha_1)$, ... such that $C_0 \in \mathcal{C}$ and for each $i$, $(C_i, \alpha_i) \to C_{i+1}$. An

[1] The distinct roles of the two devices in an interaction is a fundamental assumption in our model.

execution is *fair* if for every $\alpha$, $C$ and $C'$ such that $(C, \alpha) \rightarrow C'$, if $(C, \alpha)$ occurs infinitely often in the execution, then $C'$ also occurs infinitely often in the execution. We extend the output function $O$ to take a configuration $C$ and produce an *output assignment* $O(C)$ defined by $O(C)(v) = O(C(v))$. Let $E = (C_0, \alpha_0), (C_1, \alpha_1), \ldots, (C_i, \alpha_i), \ldots$ be an execution of $P$. We define the *output trace* of an execution as $OT(E) = O(C_0), O(C_1), \ldots, O(C_i), \ldots$.

A self-stabilizing system can start at an arbitrary configuration and eventually exhibit "good" behavior. We define a *behavior* $B$ on a network $G(V, E)$ to be a set of traces on $G$ that have the same alphabet. We write $B(Z)$ to be explicit about the common alphabet $Z$. A behavior $B$ is *constant* if every trace in $B$ is constant. If given the constraint that every input trace is contained in some behavior $B_{\text{in}}(X)$, the output trace of every fair execution of a protocol $P(Q, \mathcal{C}, X, Y, O, \delta)$ starting from any configuration in $\mathcal{C}$ is in some behavior $B_{\text{out}}(Y)$, we say $P$ is an *implementation* of output behavior $B_{\text{out}}$ given input behavior $B_{\text{in}}$. If $P$ does not have any restriction on inputs, we simply say $P$ is an implementation of $B_{\text{out}}$. Given a behavior $B(Z)$, we define the corresponding *stable behavior* $B^s(Z)$: $T \in B^s$ if and only if $Z$ is $T$'s alphabet, and there exists $T' \in B$ such that $T'$ is a suffix of $T$. Thus, an execution in a stable behavior may have a completely arbitrary finite prefix followed by an execution with the desired properties. If $P(Q, \mathcal{C}, X, Y, O, \delta)$ is an implementation of $B^s$, and $\mathcal{C}$ is the set of all possible configurations, we say that $P$ is a *self-stabilizing implementation* of $B$.

## 3 Nondeterministic Protocols

In [2], we showed that nondeterminism in the transition function does not increase the class of stably computable predicates. In this section, we extend the result to self-stabilizing algorithms.

We define the *repetition closure* of a sequence $t$ to be set of sequences obtainable from $t$ by repeating each element one or more times. In other words, given any sequence $t = a_1 a_2 \ldots a_i \ldots$, the repetition closure $R(t)$ is $a_1^+ a_2^+ \ldots a_i^+ \ldots$ in regular expression notation. We extend the definition of $R$ to a behavior $B$ by taking the union of $R(t)$ for all $t \in B$. We say a behavior $B$ is *elastic* if $B = R(B)$.

**Theorem 1.** *If a nondeterministic protocol $P$ is a self-stabilizing implementation of a behavior $B$, there exists a deterministic protocol $P'$ that is a self-stabilizing implementation of $R(B)$.*

*Proof.* The proof is similar to that of the theorem in [2] corresponding to computations. We construct a compiler using a nondeterminizer to convert every nondeterministic protocol to a deterministic one. For the compiler to preserve self-stabilization, the nondeterminizer itself must be self-stabilizing.

Let $P_1$ be a nondeterministic protocol with states $Q$, input alphabet $X$, and transition function $\delta$. We describe a simulation of $P_1$ that works in graphs with at least 3 vertices. Let $m$ be the maximum cardinality of any of the sets $\delta((q, x), (q', x'))$ for $q, q' \in Q$ and $x, x' \in X$. For each $q, q' \in Q$ and $x, x' \in X$,

select an arbitrary surjective function $f_{(q,x),(q',x')}$ mapping $\{0, 1, \ldots, m-1\}$ to $\delta((q,x),(q',x'))$.

We describe a protocol $P_2$ to simulate each step of $P_1$ by multiple deterministic steps. and the state components used in $P_1$ only change in the last of the corresponding steps in $P_2$. The state consist of three components: (1) a nondeterminizer mark $\circ$ or its absence $-$, (2) a state $q \in Q$, (3) a choice counter, consisting of an integer between 0 and $m-1$ inclusive. The transitions are:

1. $((\circ qc, x), (\circ q'c', x')) \to (-qc, \circ q'c')$
2. $((\circ qc, x), (-q'c', x')) \to (-qc, \circ q'(c'+1))$
3. $((-q'c', x), (\circ qc, x')) \to (\circ q'(c'+1), -qc)$
4. $((-qc, x), (-q'c', x')) \to (-rc, \circ r'c')$

where the increments are made modulo $m$ and the pair of states $(r, r')$ is the element of $\delta((q,x),(q',x'))$ selected by the function $f_{(q,x),(q',x')}(c)$. Thus, in transitions of type 4, the value of the choice counter of the initiator is used to make a deterministic choice of an element of $\delta((q,x),(q',x'))$. The role of the nondeterminizers is to hop around the graph incrementing choice counters as they go. accomplish this purpose. Transitions of type 4 ensure that deadlock is impossible: even if we start from a configuration with no nondeterminizers, the rule will generate new nondeterminizers. Transitions of type 1 ensure that the nondeterminizers have room to move around by merging two adjacent nondeterminizers. □

If $B$ is an elastic behavior, $B = R(B)$. The following corollary is immediate:

**Corollary 1.** *If a nondeterministic protocol $P$ is a self-stabilizing implementation of an elastic behavior $B$, there exists a deterministic protocol $P'$ that is a self-stabilizing implementation of $B$.*

## 4 Protocol Composition

It is desirable to be able to combine protocols to obtain new protocols. Parallel execution of protocols is easily achieved by taking the Cartesian product of their state sets and updating the states for each protocol independently when a transition occurs. In this section we introduce one technique of protocol composition in our model. We want to compose protocols $P_1$, $P_2$, …, $P_n$, so that the self-stabilizing behavior of $P_1$, $P_2$, …, $P_i$ is used as an assumption in $P_{i+1}$.

For $n = 2$, assuming $P_1$ and $P_2$ access different components of the node's state, we run $P_1$ and $P_2$ in parallel, except that whenever $P_2$ is executed, it uses the current output of $P_1$ as its current input. When an edge is fired, it is nondeterministically determined which protocol gets the chance to execute. Recall that a behavior is constant if it contains only constant traces.

**Theorem 2.** *Suppose $B_1$ is a constant behavior. If $P_2$ is a self-stabilizing implementation of an elastic behavior $B_2$ given input behavior $B_1$, and $P_1$ is a self-stabilizing implementation of $B_1$, the composition of $P_1$ and $P_2$ (written as $P_1 \circ P_2$) is a self-stabilizing implementation of $B_2$.*

*Proof.* Let $S = C_0, C_1, \ldots$ be any fair execution of $P_1 \circ P_2$. Define the projection $\Pi_1(C)$ of a configuration $C$ to be the sub-configuration produced by taking each node's state components that are accessed by $P_1$. $\Pi_2$ is defined similarly for $P_2$. Define $S' = C'_0, C'_1, \ldots$ the maximal subsequence of $S$ in which for each $i$, the transition immediately after $C'_i$ is defined in $P_1$. $S'' = C''_0, C''_1, \ldots$ is defined similarly for $P_2$. Because $P_1$ is self-stabilizing, and $\Pi_1(C'_0), \Pi_1(C'_1), \ldots$ is a fair execution of $P_1$, there exists some $i$ such that the output trace of $\Pi_1(C'_i), \Pi(C'_{i+1}), \ldots$ satisfies $B_1$. Let $C''_j$ be any configuration that appears after $C'_i$ in $S$, and let $C''_j, C''_{j+1}, \ldots$ be the sequence starting from $C''_j$ in $S''$. Because the output trace of $P_1$ in $C''_j, C''_{j+1}, \ldots$ is constant and satisfies $B_1$, $\Pi_2(C''_j), \Pi_2(C''_{j+1}), \ldots$ is a fair execution of $P_2$ whose output trace satisfies $B_2$. Because $B_2$ is elastic, the output trace of $P_1 \circ P_2$ in the subsequence of $S$ starting from $C''_j$ satisfies $B_2$. Therefore $P_1 \circ P_2$ is a self-stabilizing implementation of $B_2$. □

## 5 Self-stabilizing Protocols

### 5.1 Token-Circulation in a Directed Ring

As an simple example, we discuss the token circulation problem in an interaction graph whose topology is a directed ring. The protocol uses the same idea as in Dijkstra's first algorithm in [3], but we only use 2 colors (0 and 1). Readers from the self-stabilization community will find the protocol familiar.

The *token-circulation behavior* $TC$ on graph $G(V, E)$ is the set of all traces $t = \beta_0, \beta_1, \ldots$ with alphabet $\{T, \phi\}$ such that:

1. For all $m \geq 0$, $\exists v \in V$ such that $\beta_m(v) = T$ and $\forall u \in V - \{v\}$, $\beta_m(u) = \phi$.
2. For all $0 \leq m < k < n$, if $\exists v, v' \in V$ $(v \neq v')$ such that $\beta_m(v) = \beta_k(v') = \beta_n(v) = T$, then $\forall u \in V - \{v\}$, $\exists l$ such that $m < l < n$ and $\beta_l(u) = T$.
3. For all $v \in V$, $\beta_k(v) = T$ for infinitely many $k$.

A node owns a token in a configuration if its output is $T$. For any trace in $TC$, exactly one node has a token in each configuration, and after a node releases a token, it does not obtain a token again until every other node has obtained a token once.

We describe a self-stabilizing implementation of $TC$ given the *leader-election behavior* $LE$. The description of a self-stabilizing implementation of $LE$ is postponed to Section 5.5. $LE$ on graph $G = (V, E)$ is the set of all constant traces $\beta, \beta, \ldots$ such that for some $v \in V$, $\beta(v) = L$ and for all $u \neq v$, $\beta(u) = N$. Informally, there is a static node with the leader mark $L$, and all other nodes have the nonleader mark $N$ in every configuration. Given the $LE$ input behavior, the leader receives input $L$ and all other nodes receive input $N$.

Node states are pairs in $\{-, +\} \times \{0, 1\}$. "+" indicates the presence of a token and "−" indicates the absence of token. The second component of a node is called the *label* of that node. The interaction rules are:

1. $((*b, N), (*b, L)) \rightarrow (-b, +\bar{b})$;
2. $((*b, *), (*\bar{b}, N)) \rightarrow (-b, +b)$.

We use the convention that $*$ on the left side of a rule matches any value for the component, that $b$ on the left side matches either 0 or 1, and $\bar{b}$ means the complement of $b$. The output rules are $+* \to T$ and $-* \to \phi$. (Output $T$ if and only if the first component is "+".)

Because of space limitation and the simplicity of the protocol, we state the following theorem without giving the proof.

**Theorem 3.** *There exists a constant-space self-stabilizing implementation of output behavior TC given input behavior LE in a directed ring.*

This protocol does not need a non-constant number of colors as in [3] or randomized transition rules as in [7], because of the stronger fairness condition.

### 5.2 Distance-2 Coloring in Bounded-Degree Graphs

To extend the token circulation algorithm to undirected rings, we need a protocol that imposes direction on the ring. A necessary condition is that each node be able to recognize its two different neighbors. Here we describe a more general algorithm that enables each node in a degree bounded graph to distinguish between its neighbors.

Suppose an undirected graph has a degree bound $d$ and we want to color the graph such that any two nodes whose distance is 2 have different colors. After a graph is properly colored, the neighbors of any node bear different colors and thus are distinguishable. It is not difficult to see that $d(d-1)+1$ colors suffice for a distance-2 coloring of a graph with degree bound $d$.

The *distance-2 coloring behavior* $D2C$ on graph $G = (V, E)$ with color set $C$ is defined as the set of constant traces $\lambda, \lambda, \ldots$ where the alphabet of $\lambda$ is $C$ and whenever $u, v, w \in V$ are such that $(u, v) \in E$ and $(v, w) \in E$ and $u \neq w$, we have $\lambda(u) \neq \lambda(w)$.

A node $i$ has the following state components:

| | |
|---|---|
| $color_i$ | An integer encoding the color of node $i$; its value is between 0 and $d(d-1)$. |
| $F_i$ | A boolean array whose size is $d(d-1)+1$, indexed by colors. |

Each node $i$ outputs the current value of its $color_i$ component.

In this and the following sections, we describe our algorithms by specifying the interaction between two adjacent nodes $i$ and $j$ when the edge $(i, j)$ is activated. The intuition behind the protocol is that if a node $i$ has only one neighbor $j$ with a given color and vice versa, then interactions between $i$ and $j$ will flip the bits $F_i[color_j]$ and $F_j[color_i]$ synchronously. If there is a second neighbor $j'$ with the same color as $j$, then an interaction with $j'$ will set the bit at $i$ to the opposite value of the bit at $j$; this will be detected in a later interaction between $i$ and $j$, causing a recoloring of either $i$ or $j$ and a resynchronization of the bits $i$ and $j$ use to follow each other. After enough nondeterministic recolorings, the protocol will eventually reach a state in which all distance-2 neighbors have distinct colors and all alternating bits are properly synchronized.

**Protocol 1.** Distance-2 coloring with degree bound $d$

1: **if** $F_i[color_j] \neq F_j[color_i]$ **then** ▷ possibly conflicting colors
2: $color_i \leftarrow color'_i$ ▷ nondeterministic coloring
3: $F_i[color_j] \leftarrow F_j[color_i]$
4: **else** ▷ valid coloring
5: $F_i[color_j] \leftarrow \overline{F_i[color_j]}$
6: $F_j[color_i] \leftarrow \overline{F_j[color_i]}$
7: **end if**

For a formal argument, we define the *safe configurations* to be the set of configurations that satisfy the following conditions:

1. Let $(u, v)$ and $(v, w)$ be any two edges in the network, it holds that $color_u \neq color_w$.
2. Let $u$ and $v$ be any two adjacent nodes, $F_u[color_v] = F_v[color_u]$.

**Lemma 1.** *The trace of any execution of Protocol 1 starting from a safe configuration is in D2C.*

*Proof.* Let $C_0$ be a safe configuration and $(u, v)$ be any edge. Suppose $C_0 \xrightarrow{(u,v)} C_1$. Because $F_u[color_v] = F_v[color_u]$, no change of color occurs. Because both $F_u[color_v]$ and $F_v[color_u]$ are complemented in the interaction, it still holds that $F_u[color_v] = F_v[color_u]$. Notice that if the first safety condition holds, among $u$'s neighbors only $v$ has the color $color_v$, and $u$ is the only one of $v$'s neighbor with $color_u$, therefore $F_u[color_v]$ and $F_v[color_u]$ cannot be changed unless $(u, v)$ is activated. Therefore both requirements of safety are preserved. Because $C_0$ and $(i, j)$ are chosen arbitrarily, we can conclude that the coloring does not change in any execution starting from any safe configuration. □

**Lemma 2.** *Starting from an arbitrary configuration, there exists a finite execution fragment that reaches a safe configuration.*

*Proof (sketch).* Suppose the second safety condition is violated in the starting configuration. There exists an edge $(u, v)$ such that $F_u[color_v] \neq F_v[color_u]$. When $(u, v)$ is activated, node $v$ will change its color and ensure that $F_u[color_v] = F_v[color_u]$ holds for the new value of $color_u$.

If the first safety condition is violated, there exist two edges $(u, v)$ and $(v, w)$ such that $color_u = color_w$. For any initial states of $u$, $v$, and $w$, there is a sequence of activations of $(u, v)$ and $(w, v)$ which will cause the second condition to be violated and either $u$ or $w$ to change its color.

Therefore the coloring cannot stabilize until a safe configuration is reached. By fairness, the nondeterministic coloring rule will eventually choose colors that lead to a safe configuration.

According to Corollary 1, there is a deterministic version of this protocol. We remark without proof that one way to turn the nondeterministic protocol to a deterministic one is to change line 2 to $color_i \leftarrow (color_i + 1) \bmod (d(d-1)+1)$. □

The following theorem follows from the lemmas and establishes the correctness of the protocol.

**Theorem 4.** *For each $d$, there exists a constant-space self-stabilizing implementation of the distance-2 coloring behavior in communication graphs of degree bounded by $d$.*

### 5.3 Directing an Undirected Ring

Given a graph colored by Protocol 1, Protocol 2 gives a sense of direction to each edge on an undirected ring and guarantees global consistency.

Formally, the *ring direction behavior* $RD$ on $G(V, E)$ is defined as the set of all constant traces $t = \lambda, \lambda, \ldots$ over an alphabet $C \times C \times C$, where we denote $\lambda(v)$ by $(c_v, c_{v,0}, c_{v,1})$, satisfying the conditions:

1. For all $v \in V$, $c_{v,0} \neq c_{v,1}$.
2. For all $(u, v) \in E$, there exists $b \in \{0, 1\}$ such that $c_v = c_{u,b} \wedge c_u = c_{v,\overline{b}}$.

We think of $c_v$ as the color of node $v$, $c_{v,0}$ as the color of its left neighbor and $c_{v,1}$ as the color of its right neighbor, and the conditions ensure global consistency.

In the protocol, each node $i$ has the following components:

| | |
|---|---|
| $color_i$ | the color of node $i$ (we assume this value is provided by the input behavior $D2C$.) |
| $color_{i,0}$ | the color of the left neighbor |
| $color_{i,1}$ | the color of the right neighbor |

Node $i$ outputs $(color_i, color_{i,0}, color_{i,1})$. A configuration is safe if its output assignment satisfies the requirement of $RD$.

**Protocol 2.** Directing an undirected ring

```
1: if color_j = color_{i,0} and color_j ≠ color_{i,1} then
2:     color_{j,1} ← color_i
3: else if color_j = color_{i,1} and color_j ≠ color_{i.0} then
4:     color_{j,0} ← color_i
5: else
6:     color_{i,0} ← color_j
7:     color_{j,1} ← color_i
8: end if
```

**Lemma 3.** *In the executions of Protocol 2, given input behavior $D2C$, all reachable configurations from any safe configuration are also safe configurations.*

*Proof.* Let $C$ be a safe configuration, that is, for all $(i, j) \in E$, there exists $b \in \{0, 1\}$ such that $color_j = color_{i,b}$ and $color_i = color_{j,\bar{b}}$, and for all $i$, $color_{i,0} \neq color_{i,1}$. Depending on the value of $b$, the condition in either line 1 or line 3 is true, and the assignments in line 2 and 4 do not modify the states, since the components already have the assigned values. Therefore, starting from a safe configuration, the state of each node does not change. □

**Lemma 4.** *Starting from an arbitrary unsafe configuration given input behavior D2C, there exists a finite execution fragment of Protocol 2 that ends at a safe configuration.*

*Proof (sketch).* Starting from an arbitrary configuration, after each edge is activated once, all dangling pointers are eliminated, which means the "left neighbor" pointer and "right neighbor" pointer of each node points to its actual neighbors. Under this assumption, consider an arbitrary node and label it 0. Label the ring sequentially in a direction such that $color_{0,0} = color_1$. If the scheduler activates $(0, 1)$, it must hold afterwards that $color_{0,0} = color_1$ and $color_{1,1} = color_0$, because either of the conditions on line 1 or line 5 applies. Let the scheduler activate $(0, 1), (1, 2), \ldots, (n-1, 0)$ sequentially. Each activation $(i, (i+1) \bmod n)$ ensures that $color_{i,0} = color_{(i+1) \bmod n}$ and $color_{((i+1) \bmod n),1} = color_i$. After this sequence of activations, all edges are directed consistently and a safe configuration is reached. □

**Theorem 5.** *Given the distance-2-coloring input behavior, there exists a constant-space self-stabilizing implementation of ring direction.*

### 5.4 Self-stabilizing Spanning Trees in Regular Graphs

Assuming the existence of a special node and the local addresses assigned by the distance-2 coloring protocol, a spanning tree rooted at the special node can be constructed in a self-stabilizing fashion in a regular graph of degree $d$. Our protocol uses $O(\log D)$ bits of memory, where $D$ is the diameter of the graph.

Let $N$ be a set of labels and $\phi \notin N$ be a special element. The *spanning tree behavior ST* on graph $G(V, E)$ consists of all constant traces $t = \lambda, \lambda, \ldots$ such that:

1. For $v \in V$, $\lambda(v)$ is a pair $(c, p)$ where $c \in N$ and $p \in N \cup \{\phi\}$, and there exists a unique $r \in V$ such that the second component of $\lambda(r)$ is $\phi$.
2. For all $v_0 \neq r$, there exists $v_0, v_1, \ldots,$ and $v_k = r$, where $v_i \in V$, $\lambda(v_i) = (c_i, p_i)$ and $p_i = c_{i+1}$ for all $0 \leq i < k$.

Informally, $N$ is the set of possible colors of nodes. If $\lambda(v) = (c, p)$, $c$ is the color of $v$ and $p$ is the color of its parent in the spanning tree. For the root node $r$ in the spanning tree, $p = \phi$.

We define the *first spanning tree* of a distance-2-colored graph with a unique leader to be the spanning tree satisfying the following conditions:

1. The root of the tree is the leader.
2. The parent of each node is the neighbor closest to the root. Ties are broken by an ordering of the colors.

It is easy to see that the first spanning tree is unique, if the coloring and the leader is fixed. Due to space limitations, we leave the detailed specification of the protocol to the full version of the paper and only informally describe the protocol to construct the first spanning tree.

The protocol consists of two parts. Each node keeps a *neighbors* queue of size $d$ which records the distinct colors of the nodes it interacts with. When the queue is full and the node interacts with a node whose color is not in the queue, the oldest value is removed from the queue and the new value is recorded. After the coloring protocol stabilizes, each node will eventually have the $d$ distinct colors of all its neighbors.

When node $i$ interacts with node $j$, if $i$ is the root, $j$ sets its state variable $dist_j$ to 1, which records the length of the shortest path from $j$ to the root that has been discovered, and it sets the *parent* variable $parent_j$ to $color_i$, the color of node $i$. Otherwise, if $parent_j$ is undefined or if $parent_j \neq color_i$ and $dist_i < dist_j - 1$, $j$ sets $parent_j$ to $color_i$ and $dist_j$ to $dist_i + 1$. If $dist_i = dist_j - 1$ but $color_i < parent_j$, $j$ also sets $parent_j$ to $color_i$. If $parent_j = color_i$, $j$ sets $dist_j$ to $dist_i + 1$.

**Theorem 6.** *Given input behaviors LE and D2C, the above protocol is a self-stabilizing implementation of output behavior ST for all regular graphs of degree d.*

*Proof.* Given input behaviors $D2C$ and $LE$, we may assume that the interaction graph $G$ is properly distance-2 colored with one node marked $L$ and all other nodes marked $N$. Let $T$ denote the unique first spanning tree of $G$. Starting from an arbitrary configuration, after every edge has been activated in each direction, the queue of neighbors of each node consists of the $d$ colors of its neighbors, and the parent pointer of every node except the root points to some neighbor of the node. Define graph $H$ to be: $(i, j)$ is in $H$ if $parent_i = j$. We look at each edge that is in $H$ but not in $T$ and show each such edge will be corrected. For ease of description, we associate a number $N_i$ with each node $i$: the higher bits of $N_i$ gives the distance from the root (the real distance in $G$, not the current value of $dist_i$), and the lower bits give $i$'s color.

Let $i$ be the node such that:

1. $parent_i = j$ ($(i, j) \in H$), but $(i, j) \notin T$.
2. $N_i$ has the smallest value among those that satisfy 1.

Let $k$ be the node such that $(i, k) \in T$. That is, $i$'s parent should be $k$, but $i$ currently thinks $j$ is his parent. There are two cases:

1. In graph $H$, the root is reachable from $j$. The scheduler activates the edges on the path from the root to $k$ in $H$ sequentially. These edges are in both $T$ and $H$. Because each node will set its distance variable to be the distance variable of its parent plus one, $dist_k$ will be the real distance of $k$ from the root. The scheduler then activates the edges on the path from the root to $j$ in $H$ sequentially. After that it must holds that $dist_j > dist_k$ or the distances are equal but the color of $k$ precedes the color of $j$. Then $(i, k)$ is activated, and $i$ will set $parent_i = k$.
2. In graph $H$, the root is not reachable from $j$. Let's only look at $H$, and let $C$ consist of $j$ and the nodes reachable from it. $C$ does not contain the root. Because all nodes in $C$ have out-degree one, there must be a directed cycle in $C$. By the definition of $H$, every node in the cycle think the next node is its

parent. By letting the scheduler keep activating the edges in the cycle, the *dist* values of the nodes in that cycle can be increased to become arbitrarily large. By activating the edges on the path from $j$ to any of the nodes on the cycle in the reversed order, the large *dist* value will be propagated back to $j$. Thus $i$ will switch the *parent* pointer to another node which has a smaller *dist* component than $j$. If the root is reachable from the new parent, do (1), otherwise repeat (2).

The process is repeated until $H = T$. □

We remark that a traversal of the tree can simulate a directed ring. Therefore, token-circulation can be done in a regular graph by running the ring token-circulation protocol in parallel with the distance-2 coloring protocol and the spanning tree protocol.

### 5.5 Leader Election

Two of the above protocols assume a pre-designated special node. In our model, self-stabilizing leader election is possible in some classes of interaction graphs and impossible in others. In this section, we first describe a family of leader-election protocols in directed rings. We also present an impossibility result for leader election in general graphs. The formal definition of the leader-election behavior ($LE$) is given in Section 5.1.

We first consider rings of odd size. Supposing each node has a *label* bit, we call a maximal sequence of alternating labels a *segment*. Since the size of the ring is odd, there is at least one pair of adjacent nodes with the same label. We define the *head* and *tail* of a segment in the natural way according to the direction of the ring. One edge of the form $(0, 0)$ or $(1, 1)$ connects the tail of one segment to the head of another segment. We call such edges *barriers*.

The protocol consists of several parts. At the base is the "unstable clock" protocol, in which the barriers move forward around the ring (which we call "clockwise"). When two barriers collide, one of them is eliminated. There exists a sequence of activations that remove all but one barrier. By fairness, eventually there is a single barrier which rotates clockwise around the ring forever.

The remainder of the protocol manipulates the leader marks and two kinds of tokens, *bullet* and *probe*. Probes move faster than barriers. Probes are sent out by the barrier in a clockwise direction and absorbed by any leader they run into. If a probe makes it all the way back to the barrier, it is converted to leader. Leaders fire bullets counterclockwise around the ring. Bullets are absorbed by the barrier, but they kill any leaders they encounter along the way.

Call a configuration "clean" if it contains exactly one barrier, exactly one leader, and there are no *bullet* or *probe* marks on any node in the interval starting from the leader and proceeding clockwise to the barrier. Thus, any *bullet* and *probe* marks are confined to the interval starting from the barrier and proceeding clockwise to the leader. As the barrier rotates, this region gets squeezed smaller and smaller until finally the barrier passes leader, at which point there are no *bullet* or *probe* marks at all. We leave the pseudocode specification of the protocol

to the full version, and only give proof sketches of correctness according to the description above.

**Lemma 5.** *All configurations reachable from a clean configuration are also clean, and the same node is marked* leader *in each.*

*Proof.* No *probe* ever encounters the barrier, because there are no *probe* marks anywhere in the region between leader and the barrier, hence, no new leader is created. No *bullet* ever encounters the leader because there are no *bullet* marks anywhere in the region starting from the barrier and going counterclockwise to the barrier, hence the leader is never killed. Newly created *bullet* and *probe* marks are both confined to the region from the barrier to the leader. □

**Lemma 6.** *For any configurations $C$ there exists a clean configuration $C'$ reachable from $C$.*

*Proof.* It follows from our fairness condition that every fair computation contains a clean configuration. From the above claim, all configurations following the first clean configuration are also clean and have the same node marked as leader.

Here's how to reach a clean configuration starting from an arbitrary configuration $C$. First, pick a barrier edge and rotate the barrier once around the ring as described above. This eliminates any other barriers that might have been present. Next, take any bullet in the forbidden region starting from the barrier and proceeding counterclockwise to the first leader (or the entire ring if no node is marked leader) and propagate the bullets counterclockwise around the ring until they are absorbed by the barrier. Some or all leaders may die in the process. If any leader remains, take the farthest leader from the barrier (in the counterclockwise direction), fire a bullet, and propagate it until it is absorbed by the barrier. Now at most one leader remains. Next, let the barrier create a *probe* mark, then propagate all *probe* marks clockwise around the ring until they are absorbed by the leader or they encounter the barrier and are converted to leader. At this point, we have a ring with one barrier, one leader, and no other marks, so it is clean. □

Lemmas 5 and 6 complete the proof of correctness.

This protocol is a special case of a family of protocols. For any ring of size $n$, we can pick an integer $k > 1$ that is relatively prime to $n$. Each node is labeled by an integer between 0 and $k-1$ inclusive. Call an edge a "barrier" if it is $(i, j)$ where $i+1 \not\equiv j \pmod{k}$. Because k is relatively prime to n, there is at least one barrier. The barrier advancement rule would be $(i, j) \to (i, i+1 \bmod k)$, where $i + 1 \not\equiv j \pmod{k}$. Calling this protocol $P_k$, then the protocol we detailed in this section is $P_2$. We thus have a family of protocols $P_2$, $P_3$, ... such that for any ring, $P_k$ accomplishes self-stabilizing leader election whenever $k$ does not divide the size of the ring.

**Theorem 7.** *For each integer $k \geq 2$, there exists a constant-space self-stabilizing implementation of the leader-election behavior on all rings whose sizes are not multiples of $k$.*

Finally, we present the following impossibility result:

**Theorem 8.** *There does not exist a self-stabilizing protocol for leader election in interaction graphs with general topology.*

*Proof.* Assuming such a protocol $A$ exists, we consider how it would behave in directed lines. Let $e$ be an arbitrary edge. If $e$ were removed, the interaction graph would become two directed lines, and by the correctness assumption of $A$, the two shorter lines would each elect a leader. Therefore from any configuration $C$ there is a reachable configuration $C'$ in which there are two leaders, because from $C$ the scheduler just stops activating $e$ and only activates other edges for a certain amount of time to reach $C'$. By fairness, in any fair execution of $A$, some configuration $C'$ with two leaders occurs infinitely often. Therefore the output trace of any fair execution of $A$ cannot have a suffix in the behavior $LE$. □

A class $C$ of graphs is *simple* if there does not exist a graph in $C$ which contains two disjoint subgraphs that are also in $C$. Notable simple classes of graphs include rings, or, more generally, connected degree-$d$ regular graphs. Directed lines, connected graphs with a certain degree bound and strongly connected graphs are non-simple classes of graphs. The proof above shows that there is no self-stabilizing leader election protocol that works for all the graphs in any non-simple class.

## 6 Conclusion and Open Problems

In this paper, we extended the population protocol model of [1] to allow for inputs at each step, and we defined general classes of behaviors. We studied self-stabilization protocols for token-circulation, distance-2 coloring, ring orientation, spanning tree, and leader election in this extended model.

We remark that one of the applications of the self-stabilizing protocols is to combine them with the protocols in [1, 2] to compute algebraic predicates or graph properties, with the additional benefit of transient-fault tolerance. For instance the token-circulation protocol could be augmented to compute predicates such as $n > k$ or expressions like $n \bmod k$ in regular graphs in which $n$ is the size of the network and $k$ is a constant. We leave the detailed discussion to the full version of the paper.

The leader election protocol we presented in this paper depends on the size of the ring. There are impossibility results and space bounds on self-stabilizing leader election in general rings in various other models [3, 10]. Because of the difference between our model and that of the previous papers, those results cannot be easily extended to our model. The existence of a uniform constant-space leader election protocol on the class of all rings or on the class of regular communication graphs of degree $d > 2$ is still open for future research.

## References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: Twenty-Third ACM Symposium on Principles of Distributed Computing. (2004) 290–299

2. Angluin, D., Aspnes, J., Chan, M., Fischer, M.J., Jiang, H., Peralta, R.: Stably computable properties of network graphs. In Prasanna, V.K., Iyengar, S., Spirakis, P., Welsh, M., eds.: Distributed Computing in Sensor Systems: First IEEE International Conference, DCOSS 2005, Marina del Rey, CA, USE, June/July, 2005, Proceedings. Volume 3560 of Lecture Notes in Computer Science., Springer-Verlag (2005) 63–74
3. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Communications of the ACM **17**(11) (1974) 643–644
4. Mayer, A., Ofek, Y., Ostrovsky, R., Yung, M.: Self-stabilizing symmetry breaking in constant-space (extended abstract). In: Proc. 24th ACM Symp. on Theory of Computing. (1992) 667–678
5. Itkis, G., Lin, C., Simon, J.: Deterministic, constant space, self-stabilizing leader election on uniform rings. In: Workshop on Distributed Algorithms. (1995) 288–302
6. Higham, L., Myers, S.: Self-stabilizing token circulation on anonymous message passing rings. Technical report, University of Calgary (1999)
7. Johnen, C.: Bounded service time and memory space optimal self-stabilizing token circulation protocol on unidirectional rings. In: Procedings of the 18th International Parallel and Distributed Processing Symposium. (2004) 52a
8. Herman, T.: Probabilistic self-stabilization. Information Processing Letters **35**(2) (1990) 63–67
9. Dolev, S., Israeli, A., Moran, S.: Uniform dynamic self-stabilizing leader election. IEEE Transactions on Parallel and Distributed Systems **8** (1997) 424–440
10. Beauquier, J., Gradinariu, M., Johnen, C.: Memory space requirements for self-stabilizing leader election protocols. In: Eighteenth ACM Symposium on Principles of Distributed Computing. (1999) 199–207
11. Itkis, G., Levin, L.A.: Fast and lean self-stabilizing asynchronous protocols. In: Proceeding of 35th Annual Symposium on Foundations of Computer Science, IEEE Press (1994) 226–239
12. Griggs, J.R., Yeh, R.K.: Labeling graphs with a condition at distance two. Journal of Discrete Mathematics **5** (1992) 586–595
13. Herman, T., Tixeuil, S.: A distributed TDMA slot assignment algorithm for wireless sensor networks. Lecture Notes in Computer Science **3121** (2004) 45–58
14. Gradinariu, M., Johnen, C.: Self-stabilizing neighborhood unique naming under unfair scheduler. Lecture Notes in Computer Science **2150** (2001) 458–465
15. Moscibroda, T., Wattenhofer, R.: Coloring unstructured radio networks. In: Proceedings of the 17th annual ACM symposium on Parallelism in algorithms and architectures, ACM Press (2005) 39–48