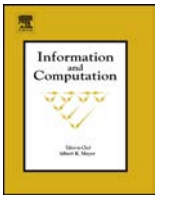
Information and Computation 285 (2022) 104829

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco



A combinatorial characterization of self-stabilizing population protocols

Shaan Mathur *, Rafail Ostrovsky

University of California, Los Angeles, United States of America

ARTICLE INFO

Article history:
Received 13 January 2021
Received in revised form 3 September 2021
Accepted 21 November 2021
Available online 24 November 2021

Keywords:
Population protocols
Self-stabilization
Anonymous
Finite-state
Chemical reaction networks

ABSTRACT

We characterize self-stabilizing functions in population protocols for complete interaction graphs. In particular, we investigate self-stabilization in systems of N finite state agents in which a malicious scheduler selects an arbitrary sequence of pairwise interactions under a global fairness condition. We show a necessary and sufficient condition for self-stabilization. Specifically we show that functions without certain set-theoretic conditions are impossible to compute in a self-stabilizing manner. Our main contribution is in the converse, where we construct a self-stabilizing protocol for all other functions that meet this characterization. Our positive construction uses Dickson's Lemma to develop the notion of the root set, a concept that turns out to fundamentally characterize self-stabilization in this model. We believe it may lend to characterizing self-stabilization in more general models as well.

1. Introduction

The population protocol computational model assumes a system of N identical finite state transducers (which we call *agents*) in which pairwise interactions between agents induce their respective state transitions. Each agent is provided a starting input and starting state, and an adversarial scheduler decides at each time step which two agents are to interact. In order to make the behavior of the scheduler precise, the scheduler is allowed to act arbitrarily so long as the *global fairness condition* (defined by Angluin, Aspnes, Diamadi, Fischer, and Peralta [1]) is satisfied: if a configuration of agent states appears infinitely often, then any configuration that can follow (say, after an interaction) must also appear infinitely often. We stress that agents individually do not have unique identifiers and a bound on N is not known. We call all N agents jointly a *population*. When an agent interacts with another agent, both agents change states as a function of each agent's input and state tuple. Each agent outputs some symbol at every time step as a function of their current state, and once every agent agrees on some common output for all subsequent time steps, we say the protocol has *converged* to that symbol. We say a protocol computes (or decides) some function f if distributing the input symbols of input x and running the protocol causes the population (i.e. all agents) to converge to $f(x)$. In the general model an accompanying *interaction graph* restricting which agents can ever interact may be provided as a constraint on the scheduler. In this paper, we will be considering the original, basic model introduced in [1], where we deal with complete interaction graphs and inputs that do not change with time. For the population protocol model on complete graphs, the characterization of computable predicates (not necessarily with

* Corresponding author.
E-mail addresses: shaan@cs.ucla.edu (S. Mathur), rafail@cs.ucla.edu (R. Ostrovsky).

https://doi.org/10.1016/j.ic.2021.104829
0890-5401/

self-stabilization) has been studied by Angluin, Aspnes, Eisenstat, and Ruppert, who proved it to be equivalent to the set of semilinear predicates [2].

It is desirable to have population protocols that can handle transient faults: specifically, we would like it to be the case that no matter what multiset of states the agents are initialized with, the protocol will eventually converge to the correct output. For this reason, such a protocol is called *self-stabilizing*, being able to converge after experiencing any adversarial fault that erroneously changes an agent state. Since fault-tolerance is a desirable property of any distributed system, we aim to determine exactly what computable functions in this model admit self-stabilizing solutions, and which do not. We prove the following main theorem for population protocols on complete interaction graphs.

Theorem *(Main). Let $f : \mathcal{X} \to Y$ be a function where $\mathcal{X}$ is any set of finite multisets on a finite alphabet. Then f has a self-stabilizing protocol $\iff$ (For any $A, B \in \mathcal{X}$, $A \subseteq B \implies f(A) = f(B)$).*

We remark on the definition of $\mathcal{X}$: Firstly, it is a set of multisets; thus $A \subseteq B$ refers to multiset inclusion, not set inclusion (e.g. $\{a, a, b\} \subseteq \{a, a, b, b\}$ but $\{a, a, b\} \nsubseteq \{a, b\}$). Moreover, the domain $\mathcal{X}$ can be *any* set of multisets; that is, the domain does not necessarily contain all possible nonempty finite multisets on a finite alphabet, but could be a subset of it. If $\mathcal{X}$ is all possible nonempty finite multisets on a finite alphabet, then the theorem states f is a constant function: the output of f on the singleton multisets is the output of f on the union of all singletons; since f agrees on all singletons, it agrees on all larger multisets. Thus self-stabilizing decision problems, where all possible inputs are included, will be a constant function. In contrast, self-stabilizing promise problems, where only a subset of all possible inputs are included, may be non-constant.

Our Techniques: The technique that we use to show when self-stabilization is not possible follows from the work of Angluin, Aspnes, Fischer, and Jiang in [3]. Informally, self-stabilizing functions must not allow subpopulations to "re-converge" to a different answer; so if $A \subseteq B$ but $f(A) \neq f(B)$, running a protocol with input B from any configuration could lead to the subpopulation with input A converging on erroneous output $f(A)$. The converse, that functions where subsets lead to the same output are self-stabilizing, is an unstudied problem that involves a technically intricate construction. Tools from partial order theory (Dickson's Lemma) are used to observe that the domain $\mathcal{X}$ will have a finite set of minimal elements under the $\subseteq$ partial order; these minimal elements completely determine the output of f, and so our protocol computes f by identifying which of these minimal elements is present in the population. A function is *self-stabilizing* if it admits a self-stabilizing protocol under the basic model. A nontrivial corollary of our main theorem is that for a fixed (possibly infinite) domain $\mathcal{X}$ and a finite output alphabet Y, there are only finitely many self-stabilizing functions $f : \mathcal{X} \to Y$.

Corollary 1. *Fix some set, $\mathcal{X}$, of finite multisets on a finite alphabet. Fix a finite output alphabet Y. There are only finitely many self-stabilizing functions of the form $f : \mathcal{X} \to Y$.*

This follows from the fact that the outputs of finitely many minimal elements in the domain fully characterize a self-stabilizing function. This validates the intuition that the set of all self-stabilizing functions is very limited in comparison to the set of computable functions. We note that our general self-stabilizing protocol is not efficient, and often specific problems have much faster self-stabilizing protocols. Unsurprisingly the functions we list in Section 1.3 admit faster solutions than our protocol. The proceedings version of this work appears in SSS 2020 [17].

1.1. Related work

The population protocol model was first introduced by Angluin, Aspnes, Diamadi, Fischer, and Peralta in [1] to represent a system of mobile finite-state sensors. Dijkstra was the first to formalize the notion of self-stabilization within a distributed system, although the models and problems he discussed imposed different constraints than that of population protocols, such as distinguishing agents with unique identifiers [13]. Self-stabilizing population protocols were first formalized in the work of Angluin, Aspnes, Fischer, and Jiang [3]. Their work generated self-stabilizing, constant-space protocols for problems including round-robin token circulation, leader election in rings, and 2-hop coloring in degree-bounded graphs. Moreover their work established a crucial method of impossibility result. Call a class of graphs *simple*, as defined in [3], if there does not exist a graph in the class which contains two disjoint subgraphs that are also in the class. Example of this includes the class of all rings or the class of all connected degree-d regular graphs. Angluin et al.'s work demonstrated that leader election in *non-simple* classes of graphs are impossible. Our paper's impossibility result follows from Angluin et al.'s technique, as the class of complete graphs that we work with is non-simple. Other impossibility results come from Cai, Izumi, and Wada [10] using *closed sets*, which are sets of states such that a transition on any two of the states results in a state within the set; impossibility in leader election is demonstrated by identifying a closed set excluding the leader state.

Many self-stabilizing population protocol constructions besides those from [3] tend to give the model additional properties to achieve self-stabilization. Beauquier, Burman, Clement, and Kutten introduce intercommunication speeds amongst agents, captured by the *cover time*; they also add a distinguished, non-mobile agent with unlimited resources called a *base station* [7]. Under this model, Beauquier, Burman, and Kutten design an *automatic tranformer* that takes a population protocol algorithm solving some static problem and transforms it into a self-stabilizing algorithm [8]. Izumi, Kinpara, Izumi,

and Wada also use this model to create efficient protocols for performing a self-stabilizing count of the number of agents in the network, where there is a known upper bound P on the number of agents. Their protocol converges under global fairness with $3 \cdot \lceil \frac{P}{2} \rceil$ agent states [16]. Fischer and Jiang introduce an *eventual leader detector* oracle into the model that allows self-stabilizing leader election in complete graphs and rings. The complete graph protocol works with local and global fairness conditions, while the ring protocol requires global fairness [14]. Beauquier, Blanchard, and Burman extend this work by presenting self-stabilizing leader election in arbitrary graphs when a composition of eventual leader detectors is introduced into the model [6]. Knowledge of the number of agents allows Burman, Doty, Nowak, Severson, and Xu to develop several efficient self-stabilizing protocols for leader election; with no silence or space constraints they achieve optimal expected parallel time of $\mathcal{O}(\log N)$ [9]. Loosely-stabilizing protocols relax the self-stabilization definition to allow the protocol to meet its specification for a long time, but not forever [19]; this can allow more tractable solutions, such as leader election protocols with polylogarithmic convergence time by Sudo, Ooshita, Kakugawa, Masuzawa, Datta, and Larmore [20]. Self-Stabilizing leader election (under additional symmetry-breaking assumptions) is possible without unique identifiers, as discussed in [5,18], but the communication happens over a fixed graph - unlike population protocols where interaction of agents is arbitrary and interaction pattern is controlled by an adversarial scheduler. Population self-stabilizing protocols are also related to biological systems self-stabilization, see [15] for further discussion.

In this work, we do not extend the basic model with any extra abilities. We demonstrate a universal self-stabilizing population protocol for any function $f : \mathcal{X} \to Y$ where for any $A, B \in \mathcal{X}$, $A \subseteq B$ implies that $f(A) = f(B)$. We do this by using a result from partial order theory by Dickson [12] that states that any set of finite dimensional vectors of natural numbers have finitely many minimal elements under the pointwise partial order.

1.2. The number of self-stabilizing functions depends on the number of minimal elements

In Definition 7 we define the *root set*. Formally, let $\mathcal{X}$ be a (possibly infinite) set of finite multisets over a finite alphabet (e.g. $\mathcal{X} = \{\{a\}, \{a, a\}, \ldots\}$ over alphabet $\Sigma = \{a\}$). The root set is some subset $\mathcal{R} \subseteq \mathcal{X}$ with the following property: for any element of the domain $A \in \mathcal{X}$, there is some element $R \in \mathcal{R}$ such that $R \subseteq A$. We call R a root. Section 2.1 uses Dickson's Lemma to prove that there exists a unique, finite, and minimally sized root set $\mathcal{R}$. We note the following corollary to the main theorem: to determine the output of any $A \in \mathcal{X}$ for a self-stabilizing function $f : \mathcal{X} \to Y$, it suffices to identify some root $R \subseteq A$ since $f(R) = f(A)$. In fact the entire output of f is determined by $f(R)$ for each $R \in \mathcal{R}$. The number of possible outputs is then upper bounded by the size of the smallest root set, which is an interesting fact in of its own right.

Corollary 2. *Let $\mathcal{X}$ be a set of finite multisets over a finite alphabet, let Y be a finite output alphabet, let $f : \mathcal{X} \to Y$ be a self-stabilizing function, and let $\mathcal{R}$ be the minimally sized root set of $\mathcal{X}$. Then the total number of possible outputs for f is upper bounded by $|\mathcal{R}|$.*

Fix some domain $\mathcal{X}$ and finite output alphabet Y, and let $\mathcal{R}$ be the minimally sized root set of $\mathcal{X}$. The number of self-stabilizing functions $f : \mathcal{X} \to Y$ are precisely the number of valid ways to assign an output to each root in $\mathcal{R}$, of which there are at most $n = |Y|^{|\mathcal{R}|}$ ways. It could be fewer than n, though, if there exists a $A \in \mathcal{X}$ has two roots $R \subseteq A$ and $R' \subseteq A$. In this case, they must be assigned the same output $f(R) = f(R') = f(A)$. In fact this can chain with other roots as well: if there is another root R'' such that $R' \subseteq B$ and $R'' \subseteq B$, then we also have that $f(R) = f(R') = f(R'') = f(A) = f(B)$. Otherwise, if every A has a unique root, then there is no overlap and each root has $|Y|$ choices for its output.

Call two roots R_0 and R_k *dependent* if there is a sequence of roots starting with R_0 and ending with R_k, $k \geqslant 1$, that chain as described above. That is, a sequence $R_0 R_1 \ldots R_{k-1} R_k$, such that for each $0 \leqslant i < k$, there is a $A_i^{i+1} \in \mathcal{X}$ such that $R_i \subseteq A_i^{i+1}$ and $R_{i+1} \subseteq A_i^{i+1}$. Notice that this makes every root in the chain map to the same output $f(R_0) = \ldots = f(R_k)$ for self-stabilizing f. In fact, this is an equivalence relation that partitions the root set into r equivalence classes, where every root in a single equivalence class must have the same output under f. This makes the number of self-stabilizing functions exactly $|Y|^r$.

Corollary 3. *Let $\mathcal{X}$ be a set of finite multisets over a finite alphabet, let Y be a finite output alphabet, and let $\mathcal{R}$ be the minimally sized root set of $\mathcal{X}$. The number of self-stabilizing functions $f : \mathcal{X} \to Y$ is finite. Specifically there are $|Y|^r$ self-stabilizing functions, where r is the number of equivalence classes of $\mathcal{R}$ under the dependence relation.*

1.3. Nontrivial examples of self-stabilizing functions

If we can restrict the domain $\mathcal{X}$ to exclude some inputs, then it can become easier to generate problems that admit self-stabilizing solutions.

- In **Chemical Reaction Networks (CRN)**, it can be desirable to compute boolean circuits. In CRNs one prevalent technique to compute some boolean function $g : \{0,1\}^k \to \{0,1\}^m$ is to have k different species, each with 2 sub-species [11]. That is, we have molecules $s_1^0, s_1^1, s_2^0, s_2^1, \ldots, s_k^0, s_k^1$, where molecule s_i^j signifies that the i^{th} bit has value $j \in \{0,1\}$. All species will appear in the input, but only one sub-species per species will appear (i.e. s_i^0 and s_i^1 will not both appear, but one of them will). This way of formulating the input allows for self-stabilizing computation of boolean functions! This is

because if $A \subseteq B$ for some input multisets, A and B, of molecules, they will include the same sub-species and hence have the same output $g(A) = g(B)$.

- Generalizing the former example, suppose we have an input alphabet of k symbols, but only $\tilde{k} < k$ of these symbols will ever show up in the population (though a given symbol could occur multiple times). Then we can compute *any* self-stabilizing function of the $\tilde{k}$ present members.
  For instance, suppose we have $\tilde{k}$ different classes of finite-state mobile agents, $a_1, \ldots, a_{\tilde{k}}$. There are a nonzero number of agents in each class a_i, and agents within the same class are running a (possibly not self-stabilizing) population protocol. Eventually every agent in class a_i will converge and be outputting the same common value o_i. These outputs $o_1, \ldots, o_{\tilde{k}}$ will then be the input to our self-stabilizing population protocol. The agents might perform some sort of protocol composition where a tuple of states (q, s) are used for each agent; q would correspond to the agent state in the first protocol, and then s would correspond to the agent state in the self-stabilizing protocol. If the first protocol was also self-stabilizing (it doesn't have to be), then the entire protocol composition would be self-stabilizing as well.
  As an example, we could have each class of agents a_i compute some boolean circuit C_i in a self-stabilizing way. Eventually each agent of class a_i will be outputting some m character string over the set $\{0_i, 1_i\}$. Notice that even both agents in class a_1 and in class a_2 intend to output 0110, the former agents will output $0_1 1_1 1_1 0_1$ and the latter agents will output $0_2 1_2 1_2 0_2$. Once all of these boolean circuit computations are done, all the agents across classes will calculate the majority output in a self-stabilizing way (and they would be able to do so since the outputs $o_1, \ldots, o_{\tilde{k}}$ are all distinguishable).
- Any computable function in which the number of agents is a fixed constant k admits a self-stabilizing solution (there can't be any subsets, so the condition is vacuously true). For instance, distribute bits amongst exactly k agents; we can create a self-stabilizing protocol to output 1 if any permutation of those k bits represents a prime number in binary.

1.4. The basic model

There are different formalizations of the basic population protocol model. We adopt the basic one first introduced by Angluin et al. [1], except where we impose that any two agents are allowed to interact.

A **population protocol** is a tuple $\mathcal{P} = (Q, \Sigma, Y, I, O, \delta)$ where Q is the finite set of agent states; Σ is a finite set of input symbols; Y is a finite set of output symbols; $I : \Sigma \to Q$ is an input function; $O : Q \to Y$ is the output function; and $\delta : (Q \times \Sigma) \times (Q \times \Sigma) \to (Q \times Q)$ is the transition function.

Note that population protocols have no know prior knowledge of the number of agents; rather, first a population protocol is specified and then it is run on some set of agents V. At the beginning of execution, an input assignment $\alpha : V \to \Sigma$ is provided, providing each agent an input symbol (we will observe that in complete graphs, we can view input assignments as merely a multiset of inputs). Since our model focuses on the computation of functions, we will enforce that the input does not change (i.e. the input is hardwired into every agent). If an agent is assigned input symbol $\sigma \in \Sigma$, it will determine its starting state via the input function as $I(\sigma)$ (note that in the basic model the input determines the starting states, but in self-stabilization we do not consider starting states). At each time step, a scheduler selects (subject to a global fairness condition) an agent pair (u, v) for interaction; semantically the scheduler is selecting agents u and v to interact, where u is called the *initiator* and v is called the *responder*. Agents u and v will then state transition: letting $q_u, q_v \in Q$ and $\sigma_u, \sigma_v \in \Sigma$ be the states and inputs for u and v respectively, the new respective states will be the output of $\delta((q_u, \sigma_u), (q_v, \sigma_v))$.

We use the notion of a *configuration* to describe the collective agent states. We define it as a mapping below. In complete graphs, however, we can also view a configuration as a multiset of states due to symmetry.

Definition 1. *Configuration.* Let V be the set of agents and Q be a set of states. A *configuration* of a system is a function $C : V \to Q$ mapping every agent to its current state.

When running the protocol we will go through a sequence of configurations. If C and C' are configurations under some population protocol such that C' can follow from an interaction by two agents in C, we write $C \to C'$. If a series of interactions takes us from C to C', we write $C \xrightarrow{*} C'$.

Definition 2. *Execution.* An *execution* of a population protocol is a sequence of configurations $\mathcal{C} = C_1 C_2 \ldots$ where for all i, $C_i \to C_{i+1}$.

The scheduler is subject to a global fairness condition, which states that if a configuration can follow from an infinitely occurring configuration, then it must also occur infinitely often.

Definition 3. *Global Fairness Condition* [1]. Let C and C' be configurations such that $C \to C'$. If C appears infinitely often during an execution, then C' appears infinitely often during that execution.

At each time step every agent outputs some symbol from Y via output function O. If all agents output the same symbol and continue to do so for each time step afterwards, we say the protocol's output is that symbol (interactions may continue,

but the output of the agents remains that symbol). Note that when computing functions, the protocol should output the same symbol when the same inputs are provided, irrespective of the globally fair scheduler's behavior. When the population has determined the final output, we say it has *converged*.

Definition 4. *Convergence* [1]. A population is said to have *converged* to an output $y \in Y$ during a population protocol's execution if the current configuration C is such that each agent's output is y and $C \to C'$ implies that C' has each agent with the same output y.

When agents u and v interact, their state transition is a function of both agent's current state and respective inputs; since this is all the information they receive, an agent does not learn the identity of the agent it interacts with, but merely the state and input of that agent. In this sense, agents with the same input that are in the same state are *indistinguishable* from one another. Furthermore, population protocols are independent of the number of agents, making it impossible to design the state set to give every agent a unique identifier, so agents are truly *anonymous*.

As noted earlier, our model accepts input via an input assignment $\alpha : V \to \Sigma$, where V is the set of agents and Σ is our finite input alphabet. It is useful to note, though, that we can actually view our input instead as some finite multiset A over alphabet Σ, with $|A| = |V|$ [4]. Though we omit the proof, the idea follows from the fact that any two agents can interact, so which agent gets what input symbol is less important than what input symbols are provided to the system in the first place. We use the notation $m_A(\sigma)$ to denote the *multiplicity*, the number of occurrences, of element σ in multiset A.

Definition 5. *Population Protocol Functions.* Let $f : \mathcal{X} \to Y$ be a function where $\mathcal{X}$ is a set of multisets over the finite alphabet Σ. A population protocol $\mathcal{P}$ computes f if and only if for any $A \in \mathcal{X}$, all executions of $\mathcal{P}$ with input A converge to $f(A)$.

In this paper when we say $\mathcal{P}$ is a population protocol (or simply protocol), we mean that it computes some function f. When we don't care about whether a population protocol computes some function, we will refer to it as a *sub-protocol*. This will be useful jargon in our protocol composition in Section 2.3.

We say that a population protocol computing a function is *self-stabilizing* when it can begin in any configuration and eventually converge (to the same output). Such a function is called a *self-stabilizing* function.

Definition 6. *Self-Stabilizing Protocol.* Let $\mathcal{P} = (Q, \Sigma, Y, I, O, \delta)$ be a population protocol computing some function $f : \mathcal{X} \to Y$. $\mathcal{P}$ is called *self-stabilizing* if and only if for any input multiset $A \in \mathcal{X}$, any set of agents V of cardinality $|A|$, and any starting configuration $C : V \to Q$, we have that any execution of $\mathcal{P}$ converges to $f(A)$.

2. Constructing a self-stabilizing protocol for the basic model

We aim to show that not only does Theorem 1 specify *necessary* conditions for computing self-stabilizing functions (shown in Appendix A), but they are also *sufficient* for self-stabilization. We do this by generating a self-stabilizing protocol for computing all functions of the form $f : \mathcal{X} \to Y$ where for all $A, B \in \mathcal{X}, A \subseteq B \implies f(A) = f(B)$. To do this, we must introduce a new notion known as the root set of a set of multisets.

2.1. The root set

The inputs for our agents are represented by a multiset of inputs, an element of domain $\mathcal{X}$. We are interested in a kind of subset $\mathcal{R} \subseteq \mathcal{X}$ such that all multisets in $\mathcal{X}$ are a superset of some multiset in $\mathcal{R}$. This section aims to show that all sets of multisets $\mathcal{X}$ actually have a *finite* $\mathcal{R}$.

Definition 7. *Root Set and its Roots.* Let $\mathcal{X}$ be a set of finite multisets. A subset $\mathcal{R} \subseteq \mathcal{X}$ is called a *root set* of $\mathcal{X}$ if and only if for all $A \in \mathcal{X}$, there exists $R \in \mathcal{R}$ such that $R \subseteq A$. We call a multiset $R \in \mathcal{R}$ a *root* of A.

For instance, take $\mathcal{X}$ over alphabet $\Sigma = \{a, b, c, d, e, f\}$ as

$$\mathcal{X} = \{\{a, a, b\}, \{a, a, b, b, c\}, \{e, e, e, f, f, f, b, d\}, \{d\}\}.$$

A root set $\mathcal{R} \subseteq \mathcal{X}$ could be

$$\mathcal{R} = \{\{a, a, b\}, \{d\}\},$$

where $\{a, a, b\}$ and $\{d\}$ are the roots.

Notice that $\mathcal{X}$ is always trivially its own root set. However, $\mathcal{X}$ can be infinitely large; for example the set of all nonempty finite multisets on alphabet $\Sigma = \{a\}$ is $\mathcal{X} = \{\{a\}, \{a, a\}, \ldots\}$. However, we are primarily interested in the existence of a *finite* root set over our function f's domain. Dickson's Lemma [12] provides us what we need.

First, consider a finite multiset over an alphabet, Σ, of n elements. An equivalent representation of a multiset is as a vector of multiplicities in $\mathbb{N}^n$. For instance, the multiset $\{a, a, b\}$ on ordered alphabet $\Sigma = \{a, b, c\}$ would be represented by $(2, 1, 0)$. Hence a set of finite multisets on an alphabet of size n could be considered a subset $S \subseteq \mathbb{N}^n$. Consider two vectors $\boldsymbol{n}, \boldsymbol{m} \in \mathbb{N}^n$, and denote n_i and m_i as the i^{th} component in the corresponding vectors. Define the pointwise partial order $\boldsymbol{n} \leqslant \boldsymbol{m} \iff n_i \leqslant m_i$ for all i. A minimal element of a subset $S \subseteq \mathbb{N}^n$ is an element that has no smaller element with respect to this partial order. Now we can state Dickson's Lemma.

Lemma 1. *Dickson's Lemma. In every subset $S \neq \emptyset$ of $\mathbb{N}^n$, there is at least one but no more than a finite number of elements that are minimal elements of S for the pointwise partial order.*

This is equivalent to the existence of a finite root set.

Corollary 4. *Let $\mathcal{X}$ be a set of finite multisets over a finite alphabet. $\mathcal{X}$ has a finite root set. In other words, there exists a finite subset $\mathcal{R} \subseteq \mathcal{X}$ such that for any $A \in \mathcal{X}$, there exists $R \in \mathcal{R}$ such that $R \subseteq A$.*

A interesting corollary is that the root set of minimal size is unique, making it legitimate to speak of *the* minimally-sized root set.

Corollary 5. *Let $\mathcal{X}$ be a set of finite multisets over a finite alphabet. The minimal length root set $\mathcal{R}$ of $\mathcal{X}$ is unique.*

Proof. Suppose there exists two root sets of minimal size, $\mathcal{R}$ and $\mathcal{R}'$. Then there is some root $R_i \in \mathcal{R}$ such that $R_i \notin \mathcal{R}'$. Since $R_i \in \mathcal{X}$, it must have a root $R' \in \mathcal{R}'$ such that $R' \subseteq R_i$. Since $R' \in \mathcal{X}$, it must have a root $R_j \in \mathcal{R}$ such that $R_j \subseteq R'$. Therefore we have that $R_j \subseteq R' \subseteq R_i$, for some $R_i, R_j \in \mathcal{R}$. If $R_i = R_j$ then we have that $R_i \subseteq R' \subseteq R_i$, which is a contradiction since this means $R' = R_i \in \mathcal{R}'$. If $R_i \neq R_j$ then $\mathcal{R}$ is not of minimal size since $\mathcal{R} - \{R_i\}$ is also a root set of $\mathcal{X}$. Thus any root in $\mathcal{R}$ is also in $\mathcal{R}'$; since $|\mathcal{R}| = |\mathcal{R}'|$ by assumption, the sets must be equal. □

It will also be convenient for us to refer to the maximum multiplicity, M, of all symbols in the root set. This will be useful later when designing counters that will increment modulo M.

Now suppose we have a function $f : \mathcal{X} \to Y$ over multisets such that for $A, B \in \mathcal{X}, A \subseteq B \implies f(A) = f(B)$. We have that there exists a finite, minimal length root set $\mathcal{R} \subseteq \mathcal{X}$. If the input to the population is A, it suffices to identify the root $R \in \mathcal{R}$ such that $R \subseteq A$, since the output would be $f(R) = f(A)$.

2.2. Self-stabilizing population protocol construction

Before we formally specify a universal self-stabilizing population protocol, it is much more helpful to first understand how it works at a high level. As a reminder, we will be working with functions of the form $f : \mathcal{X} \to Y$, where $\mathcal{X}$ is some set of finite multisets on a finite alphabet and Y is the finite output alphabet. The function f is also assumed to satisfy the following property:

$$\forall A, B \in \mathcal{X}, A \subseteq B \implies f(A) = f(B).$$

We need to determine how to design our protocol to compute f in a self-stabilizing manner. Since $\mathcal{X}$ is a set of finite multisets, it has a finite root set; let $\mathcal{R} \subseteq \mathcal{X}$ be the minimally sized, finite root set. $\mathcal{R}$ will be given some arbitrary fixed ordering so that we can index into it. Loosely speaking, we can compute f by having each agent iterate through every root in the finite root set to see if a given root is a subset of the population's input A. If we were not dealing with self-stabilization, one could imagine a solution where every agent counts the inputs they see to try and directly determine which root is present. However the issue in the self-stabilizing setting is that these counters might be initialized in an unfavorable way that tricks the agents into predicting the wrong root. Thus we need a way for an agent to identify that the wrong root has been selected so it can reset its counter and start over.

As an agent iterates through the root set, how can it tell if the current root R_i is not a subset of the population's input multiset, A? It turns out that the agents' counters will grow sufficiently large enough to show that another root R_j with different output $f(R_i) \neq f(R_j)$ could also be present in the population, which will be a signal to reset counters and iterate through the root set. More concretely, suppose an agent has guessed that root $R_i \subseteq A$. Consider some other root R_j where $f(R_i) \neq f(R_j)$. Naturally there are some symbols that occur more often in R_j than in R_i, and vice versa. Consider a symbol σ that occurs m_j times in R_j and m_i times in R_i, where $m_j > m_i$. Suppose agents start counting how many other agents with input σ they see, and manage to identify there are at least m_j of them. Now further suppose that this is the case for all such σ occurring more often in R_j, implying that there is a unique agent with input σ for each occurrence of σ in R_j. Then we now know that $f(R_i) \neq f(A)$ by contradiction. For any symbol σ occurring m_j times in R_j and m_i times in R_i, we have two cases. If $m_j > m_i$, then we know that input multiset A has at least m_j instances of σ. If $m_j \leqslant m_i$, since $R_i \subseteq A$ by our hypothesis then we also know that there are at least m_j instances of σ. Therefore we simultaneously have

that $R_j \subseteq A$ and $R_i \subseteq A$, which is a contradiction since $f(A) = f(R_j) \neq f(R_i)$! Therefore our initial assumption was wrong and R_i is not a subset of the input multiset A, and so we should increment our index i to guess root R_{i+1}. Note that we could also change our index i to become j, and the authors believe this would be a faster protocol; however since this is a paper about computability, we leave this optimization as an observation.

On the other hand if R_i really is a subset of A, then the existence of such a R_j would be impossible. If our counters are all initialized to 0, then no agent counter would ever increment high enough to identify such a R_j. However since the protocol may start in an arbitrary configuration, we have to make sure to reset counters whenever an agent increments and guesses a new root. Notice that agents try to compute the correct root by exchanging information about input counts, and that each agent is using that information independently to select a root. Agents with conflicting roots do not try and agree upon them after an interaction, but will eventually select the same root after both use this counting argument independently.

To formalize this idea, we need convenient notation. We define $MORE_{i,j}$ to be the set of symbols occurring more often in R_j than in R_i when $f(R_i) \neq f(R_j)$.

$$MORE_{i,j} = \begin{cases} \{\sigma \mid m_{R_i}(\sigma) < m_{R_j}(\sigma)\} & f(R_i) \neq f(R_j) \\ \emptyset & \text{otherwise} \end{cases}.$$

Each agent will have a table indexed by i and j, where each entry is a binary string of length $|MORE_{i,j}|$. The k^{th} bit of this string is set to 1 when the agent counts that the k^{th} symbol of $MORE_{i,j}$ occurs in the population as often as it does in R_j; otherwise it is 0. Once any entry in this table becomes a (nonempty) bit string of all 1's, then the agent tries a new root. To keep track of these counts, each agent will also maintain a nonnegative integer *count*; when two agents with the same *count* and same input symbol σ meet, the responder will increment its value. To keep the states finite, the count is bounded above by the maximum multiplicity that occurs in the root set. It's important to note that the fact that the root set is finite is crucial to keep the number of states here finite.

2.3. Sub-protocols for protocol composition

The self-stabilizing population protocol we construct will be a protocol composition $A \times B \times C$, where the input is given to A, the input to B is the output of A, the input to C is the output of B, and the composition output is the output of C. With the previous discussion as our design motivation, we decompose our protocol into three distinct sub-protocols:

1. *SymbolCount*. This sub-protocol implements a simple modular counting mechanism, where agents with the same input symbol σ compare their counts. The counts are 0-indexed since they are meant to represent how many *other* agents they've met with the same input symbol σ. If two agents with the same count meet, then the responder increments its count (modulo the maximum multiplicity of any symbol in the root set, M, to keep things finite). If there are at least $k < M$ agents with the same symbol, then some agent must eventually have their count *at least* $k-1$ by the Pigeonhole Principle, irrespective of initial configuration. The converse is only true if all the counts are initialized to 0, which is problematic if we are designing a self-stabilizing protocol that can initialize in any configuration. We will circumvent this by having the larger protocol composition reset this counter whenever moving on to the next root.
2. *WrongOutput*?. This sub-protocol maintains a table indexed by $i, j \in \{0, 1, \ldots, |\mathcal{R}| - 1\}$, where $\mathcal{R}$ is the minimally sized root set. Each entry will be a binary string of length $|MORE_{i,j}|$, where $MORE_{i,j}$ is the set of all input symbols occurring more often in R_j than in R_i. If the k^{th} symbol of $MORE_{i,j}$ occurs in the population at least as often as it does in R_j, then the corresponding bit in the binary string is set to 1. Notice that this table is only useful when it is initialized with all entries as binary strings of all 0's. Again, we will circumvent this by having the larger protocol composition reset the table whenever moving on to the next root.
3. *RootOutput*. This sub-protocol uses an index $root \in \{0, 1, \ldots, |\mathcal{R}| - 1\}$. Adopting the notation from the previous bullet and letting $root = i$, this protocol increments *root* if there is a j such that the (i, j) entry in the table has a binary string of all 1's. The protocol composition will use the incrementing of *root* to signal that the other sub-protocol states should reset.

We now list the three sub-protocols below.

Definition 8. *SymbolCount*. Let $f : \mathcal{X} \to Y$ be a function over finite multisets on a finite alphabet Σ, and let $\mathcal{R}$ be a finite and minimally sized root set of $\mathcal{X}$. Each agent has a state called *count*, where $count \in \{0, 1, \ldots, M-1\}$ and M is the maximum multiplicity of any symbol in the root set. Let $M = \max_{R \in \mathcal{R}, \sigma \in \Sigma} m_R(\sigma)$. Each agent takes as input some σ from some input multiset $A \in \mathcal{X}$. When an agent meets another agent with the same σ and same *count*, one of them will increment their *count* modulo M. This guarantees that if there are n agents with symbol σ, then eventually one agent will have $count \geqslant n - 1$.

$$(count, \sigma), (count, \sigma) \to (count, count + 1 \mod M)$$

An agent in state *count* with input σ outputs $(count, \sigma)$.

Note that even though output is a function of just the state, we can formally allow *SymbolCount* agents to output their input symbol as well, since we could take our current states Q and define a new set of states via the cross product $Q' = Q \times \Sigma$. Then transitions on this could be defined in a similar way, where we would also have to make agents transition into a state that reflects their own input symbol (which can happen after every agent interacts once).

For $WrongOutput?$, our transitions need to satisfy two properties.

1. First, it is natural to have agents share their tables with each other to share their collected information about the counts of the inputs. Whenever an agent meets another agent, they bitwise OR their tables.
2. Fix some ordering on $MORE_{i,j}$ and denote the k^{th} symbol by σ_k. Consider the bitstring entry at index i and j. By our previous discussion we want the k^{th} bit to be 1 if the number of occurrences of σ_k in the population is at least its multiplicity in $R_j \in \mathcal{R}$.
   This can be enforced by setting this bit to 1 when the *count* of some agent with input σ_k is sufficiently high enough; then the first property will ensure this bit is set for the other agents by bitwise OR'ing tables. This can formally be accomplished by bitwise OR'ing the agent's table with an indicator table of the same dimension. Let i and j be the indices into the table, k be the index into the bitstring entry, σ_k be the k^{th} symbol of $MORE_{i,j}$, and *count* and σ be the agent's inputs. Define

$$INDICATOR_{i,j,k} = \begin{cases} 1 & \text{if } \sigma = \sigma_k \text{ and } count \geqslant m_{R_j}(\sigma) - 1 \\ 0 & \text{otherwise} \end{cases}.$$

   We will denote this table as $INDICATOR(count, \sigma)$ to be explicit about the agent's inputs *count* and σ. Note that the *count* is 0-indexed, which is why we subtract by 1. Also, since agents can start in arbitrary states, the tables will only be updated with $INDICATOR$ after an agent makes its first transition.

Definition 9. $WrongOutput?$ Let $f : \mathcal{X} \to Y$ be a function over finite multisets on a finite alphabet, and let $\mathcal{R}$ be a finite and minimally sized root set of $\mathcal{X}$. Each agent has a state called *HAS-MORE*, a table indexed by i and j where each entry is a binary string of length $|MORE_{i,j}|$ (as described in previous discussion).

$$\textit{HAS-MORE}_{i,j} \in \{0,1\}^{|MORE_{i,j}|}.$$

Each agent takes as input (from *SymbolCount*) some $count \in \{0, 1, \ldots, M\}$, where M is the maximum multiplicity of any symbol in the root set, and some σ from the input multiset $A \in \mathcal{X}$.

Denote bitwise OR with symbol $\vee$. Our transition rule is

$$\begin{gathered} (\textit{HAS-MORE}^1, (count, \sigma)), (\textit{HAS-MORE}^2, (count', \sigma')) \\ \downarrow \\ (\textit{HAS-MORE}^3 \vee INDICATOR(count, \sigma), \\ \textit{HAS-MORE}^3 \vee INDICATOR(count', \sigma')) \\ \text{where } \textit{HAS-MORE}^3 = \textit{HAS-MORE}^1 \vee \textit{HAS-MORE}^2. \end{gathered}$$

An agent outputs their *HAS-MORE* table.

$RootOutput$ will have an integer *root* that is an index into the root set. An agent with input *HAS-MORE* increments its *root* modulo $|\mathcal{R}|$ if there is a j such that $\textit{HAS-MORE}_{root,j}$ is all 1's. Of course, this would mean that the state *root* keeps cycling every time an agent with such an input interacts. When we define the overall protocol composition after, this will be resolved by resetting the previous two sub-protocols' states when *root* increments. Notice that agents don't care about the states of the other agents in an interaction; instead the behavior depends on how sub-protocol $WrongOutput?$ changes its output over time. Notice that we don't allow protocols to have changing inputs, but our 3 protocol composition allows two of the sub-protocols to have a changing input as the states of the other sub-protocols change.

Definition 10. $RootOutput$. Let $f : \mathcal{X} \to Y$ be a function over finite multisets on a finite alphabet, and let $\mathcal{R}$ be a finite and minimally sized root set of $\mathcal{X}$. Each agent has a state called *root*, where $root \in \{0, 1, \ldots, |\mathcal{R}| - 1\}$. Each agent takes as input *HAS-MORE*, a table indexed by i and j where each entry is a binary string of length $|MORE_{i,j}|$ (as described in previous discussion).

$$\textit{HAS-MORE}_{i,j} \in \{0,1\}^{|MORE_{i,j}|}.$$

Our transition rules are:

$$((root_1, \textit{HAS-MORE}^1), (root_2, \textit{HAS-MORE}^2)) \to (root_1', root_2')$$
$$\text{where } i' = \begin{cases} i+1 \mod |\mathcal{R}| & \text{if } \exists j \text{ s.t. } \textit{HAS-MORE}_{i,j} = 1^{|MORE_{i,j}|} \\ i & \text{otherwise} \end{cases}.$$

An agent outputs $f(R_{root})$.

Putting these three sub-protocols together, we get the *SS-Protocol*, a self-stabilizing population protocol for f.

Definition 11. *SS-Protocol.* Let $f : \mathcal{X} \to Y$ be a function over finite multisets on a finite alphabet, and let $\mathcal{R}$ be a finite and minimally sized root set of $\mathcal{X}$. Define *SS-Protocol* as protocol composition

$$SymbolCount \times WrongOutput? \times RootOutput,$$

where we define protocol composition in the beginning of Section 2.3. We additionally modify this composition's transition function so that whenever an agent's $RootOutput$ state *root* gets incremented, then

- The *count* state from $SymbolCount$ becomes 0.
- The *HAS-MORE* state from $WrongOutput?$ becomes a table of all 0's.

When this modified transition occurs, we say the agent has been *reset* and is in a *reset state* (note there are multiple reset states as we allow *root* to be arbitrary).

2.4. Proof of correctness

We now prove that *SS-Protocol* is a self-stabilizing population protocol for $f : \mathcal{X} \to Y$ where for any $A, B \in \mathcal{X}$, $A \subseteq B \implies f(A) = f(B)$. The proof, at a high level, will argue the following. Say the input to the population is A with root R_i. If the protocol ever incorrectly outputs $f(R_j) \neq f(R_i)$, it will recognize this because the protocol would count enough symbols in $MORE_{i,j}$ to make $\textit{HAS-MORE}_{i,j}$ a bitstring of all 1's. If the protocol outputs $f(R_i)$ and begins in a reset state, then there will never be a bitstring entry with all 1's. This protocol composition is a self-stabilizing protocol for f.

To begin, we want to argue that if there are n agents with the same input, then the counting sub-protocol will achieve a *count* of at least $n-1$ (where we subtract 1 due to 0-indexing). This will be useful because we need *count* to be sufficiently high enough so that we can identify when to increment *root*. This lemma is almost immediate, but there is some subtle nuance since the counting mechanism may reset before becoming sufficiently large enough.

Lemma 2. *Let $f : \mathcal{X} \to Y$ be a function over finite multisets on a finite alphabet, and let $\mathcal{R} = \{R_0, R_1, \ldots, R_{|\mathcal{R}|-1}\}$ be a finite and minimally sized root set of $\mathcal{X}$. Let $A \in \mathcal{X}$ be the multiset whose elements are dispersed amongst the agents. Let $n \leqslant M$, where M is the maximum multiplicity of any symbol in the root set. Let C be a configuration in SS-Protocol where there are at least n agents with input σ. There is some configuration C', with $C \xrightarrow{*} C'$, where one of these agents has $SymbolCount$ state* count $\geqslant n-1$.

Proof. Consider n of these agents with input σ in some configuration C, and suppose all of these agents have $0 \leqslant count < n-1$. By the Pigeonhole Principle there will be at least two agents with the same *count*, and so it is valid to keep scheduling interactions between agents with the same *count* while no agent has $count \geqslant n-1$. If none of these agents reset when interacting with each other, then it follows that interactions between agents with duplicate values of *count* will eventually raise some agent's *count* to be at least $n-1$ (the subtraction by 1 due to 0-indexing). If an agent does reset when only interacting with agents with the same input, it can only do so finitely many times. Suppose some agent with $root = i$ resets during these interactions; it must be because the input σ has a corresponding bit in some $\textit{HAS-MORE}_{i,j}$ entry, and hence $\sigma \in MORE_{i,j}$. Either the agent stops resetting, or it keeps resetting until *root* increments to $root = j$; then since $MORE_{i,j}$ is the complement of $MORE_{j,i}$, σ won't have a corresponding bit in $\textit{HAS-MORE}_{j,i}$. Thus continued interactions with agents with input σ could not lead to another reset. Consequently all agents with input σ must eventually stop resetting, and so the agents with duplicate values of *count* will eventually drive some agent's *count* to be at least $n-1$. This final configuration C', where some agent with input σ has $count \geqslant n-1$, follows from C. □

The next lemma allows us to argue that any agent that is outputting the wrong answer (due to an incorrect choice of *root*) will eventually identify this, ultimately leading to *root* being incremented. The proof idea is that the input multiset A must have a root $R_j \subseteq A$, and the agents with symbols from R_j will have *count* sufficiently high enough to make the agent increment *root* and reset. A corollary to this is that eventually such an agent will start outputting the right answer as it keeps iterating through the root set. It's also crucial to note that every time *root* increments, the agent enters a reset state.

Lemma 3. *Let $f : \mathcal{X} \to Y$ be a function over finite multisets on a finite alphabet, and let $\mathcal{R} = \{R_0, R_1, \ldots, R_{|\mathcal{R}|-1}\}$ be a finite and minimally sized root set of $\mathcal{X}$. Let $A \in \mathcal{X}$ be the multiset whose elements are dispersed amongst the agents. Consider some execution of*

SS-Protocol. Suppose there is an infinitely occurring configuration in which an agent has RootCount state root $= i$ *such that* $f(R_i) \neq f(A)$. *Then there is an infinitely occurring configuration where this agent has* root $= i + 1 \mod |\mathcal{R}|$ *and is in a reset state.*

Proof. For sake of notational convenience, we will let $root = i$ throughout this proof. Suppose there is some infinitely occurring configuration with an agent with *RootOutput* state i such that $f(R_i) \neq f(A)$; that is, the agent is outputting the wrong answer. As $f(R_i)$ is not the right output, it must be that $R_i \not\subseteq A$. By definition of the root set there exists some different root R_j such that $R_j \subseteq A$, which in turn implies $f(R_j) = f(A)$. Therefore $f(R_i) \neq f(R_j)$, so $MORE_{i,j}$ is the set of symbols occurring more often in R_j than in R_i. Note that $MORE_{i,j}$ must be nonempty since otherwise $R_j \subseteq R_i$, which contradicts the minimality of the root set since the root set would be smaller if we excluded R_i. For each $\sigma \in MORE_{i,j}$, there will be at least as many agents with input σ as there are occurrences in R_j, since $R_j \subseteq A$. In particular there will be at least $m_{R_j}(\sigma)$ agents with input σ. By Lemma 2 and global fairness, there is an infinitely occurring configuration where one of these agents has $count \geqslant m_{R_j}(\sigma) - 1$. This agent will hence have the component corresponding to σ in *HAS-MORE*$_{i,j}$ set to 1. There will be an agent like this for each $\sigma \in MORE_{i,j}$; after our original agent interacts with each of them and bitwise OR's her *HAS-MORE* table with them, it would set her *HAS-MORE*$_{i,j}$ bitstring entry to $1^{|MORE_{i,j}|}$. The original agent would increment her *RootCount* state to $root = i + 1 \mod |\mathcal{R}|$, which would reset the agent. This resultant configuration follows from an infinite configuration, and so by global fairness it also must be infinitely occurring. □

Once an agent with input σ resets, we want their *count* to reflect a lower bound of the number of agents with input σ. Specifically, suppose all agents have reset at least once. Then if an agent with input σ has $count = n$, we want it to be the case that there are at least $n + 1$ agents with input σ in the population. Though this fact is intuitive, it turns out to be a little cumbersome to prove. We leave the details to Appendix B.

Lemma 4. *Let $f : \mathcal{X} \to Y$ be a function over finite multisets on a finite alphabet. Let $A \in \mathcal{X}$ be the multiset whose elements are dispersed amongst the agents. Suppose that there is an infinitely occurring configuration where all agents have reset at least once. If an agent with input σ has $count = n$, then there must be at least $n + 1$ agents with input σ in the population.*

One of the challenges of demonstrating protocol correctness is that the population can begin in any arbitrary configuration of states. For instance, it may be the case that *root* corresponds to the correct root, but the *count* and *HAS-MORE* table are so poorly initialized that we end up erroneously incrementing *root*! The following lemma captures what happens when we have a favorable initialization: when all agents have reset at least once, then no *count* for input σ can ever overestimate the actual number of agents with input σ in the population. When we know that all agents have reset at least once, we can guarantee convergence.

Lemma 5. *Let $f : \mathcal{X} \to Y$ be a function over finite multisets on a finite alphabet, and let $\mathcal{R} = \{R_0, R_1, \ldots, R_{|\mathcal{R}|-1}\}$ be a finite and minimally sized root set of $\mathcal{X}$. Let $A \in \mathcal{X}$ be the multiset whose elements are dispersed amongst the agents. Suppose that there is an infinitely occurring configuration where all agents have reset at least once. Then the protocol will converge with output $f(A)$.*

Proof. Suppose we are in an infinitely occurring configuration C where all agents have reset at least once. This means that all agents have had their counters reset to $count = 0$ at some point. By Lemma 4 this means that an agent with $count = n$ and input σ implies that there must be at least $n + 1$ agents with input σ in the population.

Fix an arbitrary agent, where we aim to show that this agent will output $f(A)$ forever. To do this, we will find it useful to:

- Reset the agent (again) so that its *WrongOutput*? table *HAS-MORE* has all 0's as its entries. This gets rid of any new bits that might've been set since the agent's last reset.
- Make the agent's *RootCount* state $root = i$, where $R_i \subseteq A$.

If the agent currently has the wrong output, then repeated use of Lemma 3 will increment the *root* until the agent has the correct output, $f(R_{root}) = f(A)$. Now we can assume the agent has the correct output in some infinitely occurring configuration. From this configuration, it either eventually outputs correctly for all time (in which case we are done) or it keeps changing output by repeatedly incrementing *root* and resetting. Therefore we can now assume that there is an infinitely occurring configuration where the agent has $root = i$ with table *HAS-MORE* having all 0 bitstrings. Since $root = i$, the agent correctly outputs $f(R_i) = f(A)$.

To show that this agent will eventually converge on this answer, consider the case where the agent changes its output again. For this to happen, it must be the case that its *RootCount* state $root = i$ was incremented yet again. This means that for some j, the *WrongOutput*? state entry *HAS-MORE*$_{i,j}$ becomes all 1's. Note that the definition states that $MORE_{i,j}$, the set of symbols occurring more often in R_j than in R_i, must be **nonempty** for this to happen. Since the agent was just reset with 0's in every component of *HAS-MORE*, this can only happen if for each $\sigma \in MORE_{i,j}$ there is some agent with a *count* sufficiently high enough: specifically, $count \geqslant m_{R_j}(\sigma) - 1$. By Lemma 4, it must be that there are at least $m_{R_j}(\sigma)$ agents with input symbol σ in the population.

We claim that now we have that $R_j \subseteq A$, which we show by considering two cases. If $\sigma \in MORE_{i,j}$, then we already know there are at least $m_{R_j}(\sigma)$ agents with input symbol σ, so the count of σ in R_j is at most the count in A. If $\sigma \notin MORE_{i,j}$, then the number of occurrences of σ in R_j is at most the number in R_i, which is at most the number in A since $R_i \subseteq A$. Thus the multiplicity of each element in R_j is at most the respective multiplicity in A. Therefore $R_j \subseteq A$ and hence $f(R_i) = f(A) = f(R_j)$. This is a contradiction because this would make $MORE_{i,j}$ the empty set. □

Finally, we can extend convergence to unfavorable initializations. Essentially, Lemma 3 tells us that agents that output the wrong answer will eventually reset. If agents outputting the right answer eventually reset as well, we would be done due to Lemma 5. We accomplish this by contradiction, where the protocol not converging would lead to at least one agent iterating through the root set. If this iterating agent is always able to set a bitstring entry to all 1's for any choice of *root*, then any other agent that hasn't reset yet should be able to as well.

Theorem *(Self-Stabilizing Population Protocol Theorem). Let $f : \mathcal{X} \to Y$ be a function computable with population protocols on a complete interaction graph, where $\mathcal{X}$ is a set of multisets. Then*

$$f \text{ has a self-stabilizing protocol} \iff (\forall A, B \in \mathcal{X}, A \subseteq B \implies f(A) = f(B)).$$

Proof. The forward direction is shown in Appendix A. We claim that *SS-Protocol* is a self-stabilizing protocol for f. Consider some infinitely occurring configuration C in an execution of *SS-Protocol* with input multiset $A \in \mathcal{X}$; we will argue that all agents will eventually output $f(A)$ forever.

By Lemma 3, all agents that currently output the wrong answer will eventually be reset. If all agents that currently output the correct answer eventually reset, then Lemma 5 proves that the protocol converges. Otherwise, there is an agent with $root = j$ currently outputting the correct answer that will never reset; call this agent *NoReset*. If the protocol converges anyway, we are done. If not, then there is some other agent that eventually outputs the wrong answer an infinite number of times; call this agent *RootIncrementor*. By Lemma 3 *RootIncrementor* will have to output correctly eventually, and so this agent oscillates between correct and wrong outputs. This can only happen if the *RootIncrementor*'s *root* keeps getting incremented for all time, resetting the agent each time. This means after cycling through values, *RootIncrementor* will eventually also have its $root = j$, just like *NoReset*. Since *RootIncrementor* will increment and reset yet again after interacting with some agents, it must be the case that *NoReset* could also interact with these same agents to satisfy the conditions for an increment and reset (both *RootIncrementor* and *NoReset* will have an entry in their table *HAS-MORE* with all 1's). Therefore by global fairness *NoReset* does eventually reset, a contradiction. Thus the protocol must converge. □

3. Conclusions and generalizations

A function $f : \mathcal{X} \to Y$ in the basic computational model of population protocols is self-stabilizing if and only if for any multisets A and B in the domain, $A \subseteq B \implies f(A) = f(B)$. The principle insight yielding the forward implication is that we cannot have the scheduler isolate a subpopulation and have that subpopulation restabilize to another output. The converse holds because we only need to parse the root set of the domain and find a root that is a subset of the population to determine what the output is, as our protocol does.

The notion of a root set should be applicable to arbitrary interaction graphs as well. Angluin et al. [3] demonstrated that leader election in *non-simple* classes of graphs are impossible, which is effectively applying the idea that different subgraphs may converge on different answers. We can view input assignments as interaction graphs where the nodes are the input symbols; if there exists a subgraph of an input assignment that maps to a different output under f, then f could not admit a self-stabilizing protocol. We believe that the converse should hold as well. If we consider the class of all graphs of input assignments, we can use the subgraph relation $\subseteq$ as our partial order and find its corresponding minimal elements. Unfortunately there are infinite minimal elements in the class of rings, leading to an infinite root set. Perhaps taking a quotient on the domain of possible input assignments (e.g. calling all rings equivalent) may lead to a finite root set, though this is a subject for future research, as well as on characterizing self-stabilization in general population protocols and other distributed models.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. This work is supported in part by DARPA under Cooperative Agreement No: HR0011-20-2-0025, NSF grant CNS-2001096, NSF grant CNS-2001096, US-Israel BSF grant 2015782, Google Faculty Award, JP Morgan Faculty Award, IBM Faculty Research Award, Xerox Faculty Research Award, OKAWA Foundation Research Award, B. John Garrick Foundation Award, Teradata Research Award, Lockheed-Martin Research Award and

Sunday Group. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, the Department of Defense, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright annotation therein.

Appendix A. Self-stabilization impossibility from Angluin

Angluin et al.'s [3] impossibility result for self-stabilization informally asserts that if a self-stabilizing protocol converges in a population, any subgraph of the population should converge to the same answer had it run under the same protocol. If not, the scheduler could isolate that subgraph after the convergence of the population, causing agents in the subgraph to self-stabilize and change their output, thereby contradicting convergence.

This proof technique is known, but we include it below for sake of completeness.

Theorem 1. *Let $\mathcal{P} = (Q, \Sigma, Y, I, O, \delta)$ be a self-stabilizing population protocol computing some function $f : \mathcal{X} \to Y$, where $\mathcal{X}$ is a set of multisets over finite alphabet Σ. Then*

$$\forall A, B \in \mathcal{X}, A \subseteq B \implies f(A) = f(B)$$

Proof. Suppose $f(A) \neq f(B)$ for some $A, B \in \mathcal{X}$ where $A \subseteq B$. We run protocol $\mathcal{P}$ on an arbitrary configuration of $|B|$ agents where each agent is given an input symbol from B. Since $\mathcal{P}$ is self-stabilizing and computes f, it will eventually converge to the output $f(B)$.

Since the scheduler can act arbitrarily for a finite amount of time (by global fairness), after convergence the scheduler can isolate members[1] of the population whose input symbols together form multiset A. Since this protocol is self-stabilizing and $A \in \mathcal{X}$, the protocol will eventually have this population self-stabilize and converge to $f(A)$. However this is a contradiction, since by the definition of convergence, all agents must continue to output $f(B)$ for all time after converging and yet this subpopulation is now outputting $f(A) \neq f(B)$. Thus by contradiction, it must be that $f(A) = f(B)$. □

The principle notion is that we must disallow the possibility of a subpopulation converging to a different answer than what the entire population converged to. In a setting where only certain ordered pairs of agents can interact, which interactions are allowed could be described by an interaction graph on the agents. In this setting we still cannot have a subgraph converge in isolation to a different answer. See Conclusion and Generalizations for more discussion.

Appendix B. Reset agents and *count* lower bound

Suppose all agents during some execution of our protocol have reset (i.e. at some point had $count = 0$). Then if an agent with input σ has $count = n$, then there must be at least $n + 1$ agents with input σ in the population. Though this feels intuitive, the proof takes some care.

Suppose an agent a with input σ starts off with $count = 0$. It might interact with another agent b with input σ and $count = 0$, which will lead to a incrementing to $count = 1$. Subsequently it may interact with an agent c with input σ and $count = 1$, leading a to increment to $count = 2$. However it could be that $b = c$. So we need to actually not double count b, but count the latent agent d that allowed b to increment its $count$. We do this by maintaining a set of agents $UNIQUE$, which will keep track of all $count + 1$ unique agents with input σ. As the $count$ of a increments, we will design specific rules for how $UNIQUE$ will grow. For instance:

1. In the beginning, $UNIQUE = \{a\}$.
2. When a increments to 1 after interacting with b, $UNIQUE = \{a, b\}$.
3. When b increments to 1 after interacting with d, $UNIQUE = \{a, d\}$.
4. When a increments to 2 after interacting with b, $UNIQUE = \{a, b, d\}$.

However, another wrinkle is that agents might reset their $count$ at any point due to the other subprotocols at play. Moreover, we need to guarantee that every time a interacts with another agent, it can't be any agent from $UNIQUE$. For this last fact, we must argue that it is always the case that a has the strictly largest $count = k$ in $UNIQUE$; doing this requires characterizing the distribution of $count$ amongst the agents of $UNIQUE$. In particular there is always no other agent with $count = k$, at most 1 other agent with $count \geqslant k - 1$, at most 2 other agents with $count \geqslant k - 2$, and so on. This characterization of the distribution excludes a from ever interacting with another agent in $UNIQUE$, so the incrementing of $count$ always corresponds to meeting a new agent.

[1] A globally fair scheduler can isolate any subset of the population. If C is an infinitely occurring configuration, let C' be the configuration that follows in which finitely many interactions occur only within that subpopulation. Then $C \to C'$ and so C' occurs infinitely often.

Lemma 6. *Let $f : \mathcal{X} \to Y$ be a function over finite multisets on a finite alphabet. Let $A \in \mathcal{X}$ be the multiset whose elements are dispersed amongst the agents. Suppose that there is an infinitely occurring configuration where all agents have reset at least once. If an agent with input σ has count $= n$, then there must be at least $n+1$ agents with input σ in the population.*

Proof. First we show that if a single agent has been reset, then $count = n$ for this agent means $n+1$ agents have the same input symbol. If all agents are reset once, then this must be the case for every agent, and we are done.

Suppose an agent a with input σ has been reset so that $count = 0$. Using a set $UNIQUE$, we will add new agents with input σ as this agent's *count* increments. So when a resets, set $UNIQUE \leftarrow \{a\}$. For the rest of this proof, we will refer to a as the *main* agent, any agent in $UNIQUE - \{a\}$ as a *inside* agent, and all other agents as *outside* agents. As interactions continue to occur, we will maintain the following two rules. First, say an interaction occurs between an inside agent b and an outside agent c; if b increments its *count*, then replace b with c in $UNIQUE$ (i.e. $UNIQUE \leftarrow UNIQUE - \{b\} \cup \{c\}$). Notice that this operation keeps the number of agents in $UNIQUE$ with b's old *count* the same, since b and c had the same *count* before the interaction. Second, say an interaction occurs between our main agent a and some outside agent $b \notin UNIQUE$. If a increments its *count*, then add b to $UNIQUE$ (i.e. $UNIQUE \leftarrow UNIQUE \cup \{b\}$).

To do our proof, we first prove the following invariant on the distribution of *count* on the inside agents: there are 0 inside agents with the same *count* as a, at most 1 inside agent with *count* at least one less than a, at most 2 inside agents with *count* at least two less than a, and so on. Formally, suppose main agent a has $count = k$. Then there are at most m inside agents with $count \geqslant k - m$, where $0 \leqslant m \leqslant k$. We see how this invariant is maintained in all possible interactions that change the *count* of agents in $UNIQUE$:

- When main agent a resets, this is trivial since $UNIQUE = \{a\}$. This is $UNIQUE$'s initial state.
- Suppose an inside agent with $count = j$ resets. The number of inside agents with $count \geqslant 0$ trivially stays the same. In all other cases, the number of inside agents either decreases by 1 (if another inside agent reset after the interaction) or stays the same.
- Suppose the main agent increments its *count* by meeting another agent. Notice that since there are 0 inside agents with $count \geqslant k$, this interaction must be with an outside agent. If the main agent increments its *count* to $k+1$, the outside agent is added to $UNIQUE$ with $count = k$. Now we need to show that there are at most m inside agents with $count \geqslant (k+1) - m$ for $0 \leqslant m \leqslant k+1$. Before the interaction we had that there were at most $m-1$ inside agents with $count \geqslant k - (m-1) = (k+1) - m$ for $1 \leqslant m \leqslant k+1$, so making this outside agent an inside agent would now make it at most $m - 1 + 1 = m$ inside agents. The only unconsidered case is $m = 0$, but this is straightforward: if there was an agent with $count \geqslant k+1$ after the interaction then it must have been there before the interaction, which is impossible given our invariant.
- Suppose an inside agent increments its *count* by meeting another agent. As noted earlier, the other agent cannot be the main agent due to our invariant. If it is an outside agent, then our rule states that $UNIQUE$ replaces the inside agent with the outside agent; as noted earlier, this maintains the *count* distribution of $UNIQUE$ and hence maintains the invariant. Now suppose an inside agent meets an inside agent with the same $count = k - m$, resulting in one incrementing to $count = k - m + 1$, where $2 \leqslant m \leqslant k$. Notice we exclude $m = 0, 1$ since there are never two inside agents both with $count = k$ or $count = k-1$. Since this only changes the number of inside agents with $count = k-m$ and $count = k-m+1$, this does not change the number of inside agents with *count* at least $0, 1, \ldots, k-m-1, k-m+2, k-m+3, \ldots, k$. If the invariant is violated, it can only be because there are now too many agents with $count = k-m+1$. Specifically a violation means there must be at least m agents with $count = k - m + 1$; this means that before the interaction there were at least $m-1$ agents with $count = k-m+1$ and 2 agents with $count = k-m$. This gives a total of at least $m+1$ agents with $count \geqslant k-m$ before the interaction, a contradiction. Thus the invariant is not violated.

Therefore at all points this invariant on the inside agents holds. A corollary is that when a's $count = k$, then $|UNIQUE| = k+1$ and hence there are $k+1$ agents with input σ. When $count = 0$, then $UNIQUE$ is the singleton $\{a\}$. Suppose agent a just incremented to $count = k+1$, having met some other agent b with $count = k$. By the invariant, no inside agent could have that *count*, so $b \notin UNIQUE$. Therefore b gets added to $UNIQUE$, making $|UNIQUE| = k+1$. □

References

[1] D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, R. Peralta, Computation in networks of passively mobile finite-state sensors, Distrib. Comput. (2006) 235–253.
[2] D. Angluin, J. Aspnes, D. Eisenstat, E. Ruppert, The computational power of population protocols, Distrib. Comput. 20 (4) (Nov 2007) 279–304.
[3] D. Angluin, J. Aspnes, M. Fischer, H. Jiang, Self-stabilizing population protocols, ACM Trans. Auton. Adapt. Syst. 3 (2008).
[4] J. Aspnes, E. Ruppert, An introduction to population protocols, in: Middleware for Network Eccentric and Mobile Applications, 2009.
[5] B. Awerbuch, R. Ostrovsky, Memory-efficient and self-stabilizing network {RESET}, in: PODC 1994, 1994, pp. 254–263.
[6] J. Beauquier, P. Blanchard, J. Burman, Self-stabilizing leader election in population protocols over arbitrary communication graphs, in: International Conference on Principles of Distributed Systems, 2013, pp. 38–52.
[7] J. Beauquier, J. Burman, J. Clement, S. Kutten, Brief announcement, in: PODC 2009, ACM Press, 2009.
[8] J. Beauquier, J. Burman, S. Kutten, Making population protocols self-stabilizing, in: SSS 2009, 2009.
[9] J. Burman, D. Doty, T. Nowak, E.E. Severson, C. Xu, Efficient Self-Stabilizing Leader Election in Population Protocols, 2020.

[10] S. Cai, T. Izumi, K. Wada, How to prove impossibility under global fairness: on space complexity of self-stabilizing leader election on a population protocol model, Theory Comput. Syst. (2012) 433–445.
[11] M. Cook, D. Soloveichik, E. Winfree, J. Bruck, Programmability of chemical reaction networks, in: Algorithmic Bioprocesses, Springer Berlin Heidelberg, 2009, pp. 543–584.
[12] L.E. Dickson, Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors, Am. J. Math. 35 (4) (1913) 413–422.
[13] E. Dijkstra, Self-stabilizing systems in spite of distributed control, Commun. ACM (1974) 643–644.
[14] M. Fischer, H. Jiang, Self-stabilizing leader election in networks of finite-state anonymous agents, in: Lecture Notes in Computer Science, 2006, pp. 395–409.
[15] S. Goldwasser, R. Ostrovsky, A. Scafuro, A. Sealfon, Population stability: regulating size in the presence of an adversary, in: PODC 2018, 2018, pp. 397–406.
[16] T. Izumi, K. Kinpara, T. Izumi, K. Wada, Space-efficient self-stabilizing counting population protocols on mobile sensor networks, Theor. Comput. Sci. 552 (Oct 2014) 99–108.
[17] S. Mathur, R. Ostrovsky, A Combinatorial Characterization of Self-Stabilizing Population Protocols, in: SSS 2020, 2020.
[18] A. Mayer, Y. Ofek, R. Ostrovsky, M. Yung, Self-stabilizing symmetry breaking in constant-space (extended abstract), in: Proc. 24th ACM Symp. on Theory of Computing, 1992, pp. 667–678.
[19] Y. Sudo, J. Nakamura, Y. Yamauchi, F. Ooshita, H. Kakugawa, T. Masuzawa, Loosely-stabilizing leader election in population protocol model, in: S. Kutten, J. Žerovnik (Eds.), Structural Information and Communication Complexity, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 295–308.
[20] Y. Sudo, F. Ooshita, H. Kakugawa, T. Masuzawa, A.K. Datta, L.L. Larmore, Loosely-stabilizing leader election with polylogarithmic convergence time, in: OPODIS 2018, vol. 125, 2018, pp. 30:1–30:16.