Fast and Exact Majority in Population Protocols[1]

Dan Alistarh, Rati Gelashvili, Milan Vojnović

February 2015

Technical Report
MSR-TR-2015-13

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

[1]Dan Alistarh is with Microsoft Research, Cambridge, United Kingdom (daalista@microsoft.com). Rati Gelashvili is with MIT, Cambridge, MA (gelash@mit.edu). His work was conducted in part while an intern with Microsoft Research. Milan Vojnović is with Microsoft Research, Cambridge, United Kingdom (milanv@microsoft.com).

Abstract – Population protocols, roughly defined as systems consisting of large numbers of simple identical agents, interacting at random and updating their state following simple rules, are an important research topic at the intersection of distributed computing and biology. One of the fundamental tasks that a population protocol may solve is *majority*: each node starts in one of two states; the goal is for all nodes to reach a correct consensus on which of the two states was initially the majority. Despite considerable research effort, known protocols for this problem are either exact but slow (taking linear parallel time to converge), or fast but approximate (with non-zero probability of error).

In this paper, we show that this trade-off between precision and speed is not inherent. We present a new protocol called *Average and Conquer (AVC)* that solves majority *exactly* in expected parallel convergence time $O(\log n/(s\epsilon) + \log n \log s)$, where n is the number of nodes, ϵn is the initial node advantage of the majority state, and s is the number of states the protocol employs. This shows that the majority problem can be solved exactly in time poly-logarithmic in n, provided that the memory per node is $s = \Omega(1/\epsilon)$. On the negative side, we establish a lower bound of $\Omega(1/\epsilon)$ on the expected parallel convergence time for the case of four memory states per node, and a lower bound of $\Omega(\log n)$ parallel time for protocols using any number of memory states per node.

1 Introduction

Population protocols [AAD+06], roughly defined as systems consisting of large numbers of randomly interacting identical agents, which update their state using simple rules, have recently become an important topic of research in distributed computing. In particular, considerable research effort has gone into determining the set of tasks solvable by population protocols, e.g., [AAD+06, AAER07], and into understanding the convergence bounds achievable by these algorithms in the case of fundamental tasks, e.g., [AAE08, PVV09, DV12, MNRS14]. Recently, research in biology and nanotechnology has shown that such algorithms can in fact be implemented at the level of molecules [CDS+13], and that there are non-trivial connections between population protocols and the computational tasks which biological cells must solve in order to function correctly [CCN12].

One fundamental task in the context of population protocols is *majority computation*: given a set of n nodes, starting in one of two distinct states (A or B), the task requires the nodes to reach consensus on a value associated to one of the two initial states. Crucially, the final value the nodes converge on should be associated to the *majority initial state*.

Evidence suggests that the cell cycle switch in eukaryotic cells solves an approximate version of majority [CCN12]; a three-state population protocol for approximate majority was empirically studied as a model of epigenetic cell memory by nucleosome modification [DMST07]; also, this task is a key component when simulating register machines via population protocols [AAE08]. The majority task has been studied by different research communities by considering population protocols in computation models defined in discrete and continuous time. In the discrete model, e.g., [AAE08], in each *round*, a pair of nodes is drawn at random to interact. Alternatively, other sources, e.g., [PVV09, DV12] assume that pairs of nodes interact at instances of a Poisson process in continuous time. The two models are essentially equivalent. Possible interactions are usually specified as a graph on n nodes, where the complete graph is an interesting special case, especially in the context of biological computations. In both models, we are interested in the amount of time until the algorithm converges to the correct state. In the discrete-time model, the metric of interest is *parallel time*, defined as number of rounds to convergence divided by the number of nodes n, which corresponds to "real-time" until convergence in the continuous-time model.

The complexity landscape of the majority task is quite intriguing. Assuming a system of n nodes, where one of the initial states (say, A) has an initial relative advantage over the other state (B) of ϵn nodes, there exists an elegant four-state protocol that can solve the problem in expected $O(\log n/\epsilon)$ parallel time [DV12,MNRS14]. This algorithm solves *exact* majority, i.e. has 0 probability of error; however, its running time is polynomial in n for small initial discrepancy between the two states, e.g., $\epsilon n = \Theta(n^\alpha)$, for $\alpha < 1$. At the same time, there exists an even simpler *three-state* protocol which can compute *approximate majority* in $O(\log n)$ parallel time, with high probability in n, assuming that the initial relative advantage is $\epsilon n = \omega(\sqrt{n}\log n)$ [AAE08, PVV09]. For small ϵ, this can be exponentially faster than the four-state protocol, but the price of fast convergence is *approximation*: the failure probability of the three-state protocol is $\exp(-c\epsilon^2 n)$ for constant $c > 0$ and small ϵ [PVV09], which becomes constant for small ϵ. Furthermore, it has recently been shown that it is impossible to solve majority *exactly* if each node only has three memory states [MNRS14].

Given this apparent trade-off between correctness and performance, it is natural to ask: is there a population protocol for majority which is both *fast* (converging in logarithmic or poly-logarithmic time for any ϵ), and *exact* (with 0 probability of error)?

In this paper, we answer this question in the affirmative. We present a new protocol, called AVC (for *Average-and-Conquer*), which solves *exact* majority in expected parallel time $O(\log n/(s\epsilon) + \log n \log s)$, where n is the number of nodes, ϵ is the margin, and s is the number of *states* employed by the protocol. Specifically, for number of states $s \geq 1/\epsilon$, the expected convergence time of the protocol is always poly-logarithmic in n. (Similar bounds hold with high probability in n.) In other words, AVC uses $O(\log(1/\epsilon))$ local bits to solve exact majority in poly-logarithmic time. Our protocol exhibits a new trade-off between the algorithm's error probability and convergence time, versus the number of states s employed by each node, and the discrepancy ϵ.

The AVC protocol roughly works as follows. Let $m \geq 1$ be an odd integer parameter. Each state in the protocol will correspond to an integer value between $-m$ and m, which will be either odd, or 0. The *sign* of the value corresponds to the node's current opinion (by convention, $+$ corresponds to initial majority A), and its absolute value corresponds will be the *weight* of this opinion. Therefore, in the following, we will use the terms *state* and *value* interchangeably. Initially, nodes with initial state A start with initial value $+m$, and nodes with initial value B start with initial value $-m$.

At a high level, the protocol combines two components: an *averaging* reaction, by which nodes with values > 1 average their initial values whenever interacting, and a *neutralization* reaction, by which nodes with absolute value 1 and opposite signs become neutralized and stop influencing the decision value. More precisely, whenever a node with weight > 1 interacts with a node with weight > 0, their updated states will correspond the *average* of their initial values, rounded to odd integers. (For example, input states 5 and -1 will yield output states 1 and 3.) Nodes with the same weight but different signs do not average directly to 0 states, but to intermediate states corresponding to values $+1$ and -1, respectively. States corresponding to value 0 (with positive or negative sign) are created by a *neutralization* reaction, when nodes in states $+1$ and -1 interact. Zero states are *weak*, in that they will not influence the *weight* of their interaction partner, and automatically adopt its *opinion*. (For example, input states 3 and -0 will yield output states 3 and 0.)

While the protocol is relatively simple, its analysis is non-trivial. To illustrate, take $m = 1$, and notice that in this special case the protocol would be identical to the four-state algorithm of [DV12, MNRS14]: nodes start in states $+1$ or -1; whenever two such nodes meet, they are downgraded to weak states $+0$ and -0, respectively. Weak 0 nodes shift their opinion whenever meeting a node with weight > 0, and preserve their weight. The process dynamics on a clique are relatively simple: given an initial state with discrepancy ϵ in favor of state A $(+1)$, the minority state B (-1) will eventually be depleted (downgraded to state -0) in expected $O(\log n/\epsilon)$ parallel time [DV12]. At this point, there still exist ϵn nodes in state $+1$, and it will take additional $O(\log n/\epsilon)$ expected parallel time to shift all the -0 states to the correct opinion $+0$.

Our algorithm leverages additional states to speed up *both* components of this dynamics, and overcome the linear dependence on $1/\epsilon$. This works for two main reasons: first, initial states, corresponding to large initial weight $\pm m$, will always be depleted fast, since reactions of the type $+m$ meets $-m$ are frequent at first, while, as the algorithm progresses, high values will start interacting often with small values of the opposite sign, again decreasing their weight.

Further, our algorithm is designed to keep the *total sum of values in the system invariant*. This is initially ϵmn (assuming that A is the initial majority by a margin of ϵn nodes). In particular, as the absolute values of states in the majority decrease, their *number* must increase to maintain the invariant. Our analysis formalizes this augmentation in the number of majority nodes to prove that the expected convergence time is $O(\log n/(s\epsilon) + \log s \log n)$, where $s = \Theta(m + d)$ is the number of states employed by the protocol. The protocol's correctness follows by a variation on the total sum invariant.

It is interesting to contrast AVC with a trivial algorithm in which each node maintains an array of n initial values, one for each other possible interaction partner, and nodes merge their arrays on each interaction, deciding when they have all input values. Such an algorithm would converge in $O(\log n)$ parallel time, by standard gossip analysis. However, there are two issues: first, nodes would need *distinct identifiers* for this to work, which is not standard in population protocols. Moreover, even with identifiers, the trivial protocol would require an exponential number of states per process, to cover all possible input combinations.

We complement our upper bound by two lower bounds showing that the trade-off between the number of states and convergence time is tight at its two extremes. The first lower bound uses a complex case analysis to prove that any dynamics based on agents with four states must have parallel convergence time $\Omega(1/\epsilon)$. At the other extreme, we leverage an information-propagation argument to prove that any population protocol solving exact majority converges in $\Omega(\log n)$ parallel time, irrespective of the number of states it employs.

Finally, we provide an implementation of the AVC protocol, and compare its empirical performance both with the four-state protocol of [DV12, MNRS14], and with the three-state algorithm of [AAE08, PVV09]. The simulation

results confirm the tightness of our theoretical analysis, and show that AVC provides significant speedup over the four-state protocol by increasing the number of memory states per node, with the convergence time being competitive to that of the three-state protocol for large enough number of memory states per node.

Related Work: The framework of population protocols was first formally introduced by [AAD+06]. A two-state population protocol was studied by [HP99] in a discrete-time model such that in each round every node picks a neighbor node with probability of selection allowed to be specific to this pair of nodes, and adopts the observed opinion of the selected neighbor. For the complete-graph pairwise interactions and continuous-time model where each node picks a neighbor at instances of a Poisson process, this corresponds to *the voter model* studied in the context of interacting particle systems, see e.g. Chapter 5 in [Lig85]. In particular, for the case of complete graph pairwise interactions, [HP99] showed that the error probability of the two-state protocol is equal to the portion of nodes in the initial minority state $(1-\epsilon)/2$, and that the expected parallel convergence time is $\Omega(n)$.

[AAER07] studied a three-state population protocol for computing approximate majority in a discrete-time computation model. They established the parallel time convergence bound $O(\log n)$ with high probability given that the initial margin is $\epsilon n = \omega(\sqrt{n \log n})$. The same three-state population protocol was empirically studied as a model of epigenetic cell memory by nucleosome modification [DMST07]. Independently, the same three-state population protocol for approximate majority was studied by [PVV09] in a continuous-time computation model. They established a tight bound on the probability of error for convergence to the correct terminal state $\exp(-D((1+\epsilon)/2||1/2)n)$ where $D(p||q)$ is the Kullback-Leibler divergence between two Bernoulli distributions with parameters p and q. This implies the asymptotic bound on the error probability $\exp(-c\epsilon^2 n)$ for constant $c > 0$ and small ϵ. Moreover, they established a parallel convergence time bound $O(\log(1/\epsilon) + \log n)$ for the limit system dynamics described by a system of ordinary differential equations as the number of nodes n grows large. [CCN12] showed that the dynamics of this three-state population protocol is under certain initial conditions equivalent of the cell cycle switch and other dynamics.

[DV12] first proposed a four-state protocol for exact majority computation and studied it for the case of arbitrary rates of pairwise interactions, which are such that the underlying graph G of pairwise interactions is connected. They showed an expected parallel time convergence bound of $(\log n + 1)/\delta(G, \epsilon)$ where $\delta(G, \epsilon) > 0$ is the smallest eigenvalue gap for a family of pairwise interaction rate matrices. For the case of a clique, this yields the expected parallel time convergence bound $O(\log n/\epsilon)$. A similar four-state protocol was also studied by [MNRS14] in a discrete-time computation model, who established that four memory states are necessary for exact majority computation.

2 Preliminaries

Population Protocols: We assume a population consisting of n agents, or cells, each executing as a deterministic state machine with states from a finite set Q, with a finite set of input symbols $X \subseteq Q$, a finite set of output symbols Y, a transition function $\delta : Q \times Q \to Q \times Q$, and an output function $\gamma : Q \to Y$. Initially, each agent starts with an input from the set X, and proceeds to update its state following interactions with other agents, according to the transition function δ. For simplicity of exposition, we assume that agents have identifiers from the set $V = \{1, 2, \ldots, n\}$, although these identifiers are not known to agents, and not used by the protocol.

The agents' interactions proceed according to a directed *interaction graph* G without self-loops, whose edges indicate possible agent interactions. Usually, the graph G is considered to be the complete graph on n vertices, a convention we also adopt in this work.

The execution proceeds in *steps*, where in each step a new edge (u, w) is chosen uniformly at random from the set of edges of G. Each of the two chosen agents updates its state according to the function δ.

We say that a population protocol computes a function $g : X^V \to Y$ within ℓ steps with probability $1 - \phi$ if for all $x \in g^{-1}(Y)$, the configuration $c : V \to Q$ reached by the protocol after ℓ steps satisfies the following two properties with probability $1 - \phi$:

1. For all $v \in V$, $g(x) = \gamma(c(v))$. Specifically, all agents have the correct output in c.
2. For every configuration c' reachable from c, and for all $v \in V$, $g(x) = \gamma(c'(v))$.

Parallel Time: The above setup considers sequential interactions; however, in general, interactions between pairs of distinct agents are independent, and are usually considered as occurring in parallel. In particular, it is customary to define one unit of *parallel time* as n consecutive steps of the protocol.

The Majority Problem: In the *majority problem*, agents start with arbitrary initial states in the input set $X = \{A, B\}$. Let a be the number of agents starting in state A, and b be the number of agents starting in state B, and let $\epsilon = |a-b|/n$ denote the relative advantage of an initial state. The output set is $Y = \{0, 1\}$.

A population protocol solves the majority problem within ℓ steps with probability $1 - \phi$, if, for any configuration $c : V \to Q$ reachable by the protocol after $\geq \ell$ steps, it holds with probability $1-\phi$ that (1) If $a > b$ for the given input, then for any agent i, $\gamma(c(i)) = 1$, and, conversely, (2) If $b > a$ for the given input, then for any agent i, $\gamma(c(i)) = 0$.

3 The Average and Conquer Algorithm

In this section, we describe the *Average and Conquer* (AVC) algorithm, which solves majority problem in population protocols *exactly*, using s states. Throughout the rest of the exposition, we fix an odd integer parameter $m \geq 1$, which will determine the number of states of the protocol and its running time.

Overview: The intuition behind the protocol is as follows. Each node state is associated with a *sign* ($+$ or $-$) and a *weight* (an non-negative odd integer between 0 and m), whose product determines the *value* corresponding to each state. Throughout the exposition, we will identify the node *state* with corresponding *integer value*, making the distinction where appropriate. (The formal mapping is given for each state in lines 1–3 of the pseudocode in Figure 1.)

Nodes start in one of two distinguished states. By convention, nodes starting in state A have initial value (state) $+m$, while nodes starting in state B have initial value (state) $-m$. The protocol is based on the following idea: whenever two nodes holding non-zero values interact, they *average* their current values. (Since states only represent odd values, the algorithm rounds the average if needed.) States $+0$ and -0 are *weak*: a node in this state does not influence other values in interactions, and will change its state's sign to match the that of the interaction partner. Also, for technical reasons, value 1 corresponds to d distinct *intermediate states*, which we denote by $1_1, 1_2, \ldots, 1_d$ (and, symmetrically, states $-1_1, -1_2, \ldots, -1_d$ for value -1). A node in such a state must interact several times with other nodes of opposite sign and intermediate value in order to eventually downgrade to 0.

Nodes continue to interact according to these simple rules. We prove that all nodes will converge to values of the same sign as the initial majority, and that convergence is fast with high probability. This solves the majority problem since we can define γ as mapping each state to the output based on its sign.

We present the algorithm in detail below.

State Parameters: Each state in the protocol corresponds to an integer value, composed by a $sign$ ($+$ or $-$) and a and a $weight$, which is either 0, or an odd positive integer between 1 and m. Intuitively, a node's sign gives its tentative output, whereas its $weight$ gives the "confidence" in that specific output. *Strong* states (with weight ≥ 3) can drive weaker states, of lower absolute value, to change sign. *Weak* states (absolute value 0) will not influence other states' sign. States with weight 1 (denoted by $\pm 1_j$, where j is a positive integer denoting the state's level) are *intermediate*, and are interpreted as having weight 1, and sign ± 1. (Intuitively, the reason why we grade values of 1 is to prevent nodes from becoming weak too quickly.) The total number of states in the protocol is $s = m + 2d + 1$.

Nodes start in one of two initial states: state A, which we identify as having $sign = +1$ and $weight = m$, and state B, which corresponds to $sign = -1$ and $weight = m$. The algorithm, specified in Figure 1, consists of a set of simple deterministic update rules for the node state, depending on the $sign$ and $weight$ of the interacting node, and will ensure that, after some time, all nodes converge to the correct (majority) opinion. In the pseudocode, the node states before an interaction are denoted by x and y, while their new states are given by x' and y'.

Interactions: The core of the update rule is the *averaging* of nodes' values after each interaction, where for each state x, $value(x) = sgn(x) \cdot weight(x)$, where states $\pm 1_j$ have weight 1. Intuitively, the values of the two participants after an interaction will equal the average of their previous values, where we round values up and down to the closest odd numbers. Note that this rule always preserves the total sum of values in the system. Values 0 and 1 are special, and therefore we consider them separately in the update rules. We have three cases for the update rules: *strong meets non-zero* (line 11), *zero meets non-zero* (lines 12—14), and *intermediate meets intermediate* (lines 15—19), with a special case if one of the nodes is 1_d (lines 15—17). (The *zero meets zero* reaction does not change the participants' states.) Figure 2 gives examples of reactions.

If a node in a strong state (with weight > 1) meets a node in strong or intermediate state (with weight > 0), their new values will correspond to the average of their initial value (line 11). Specifically, if the average of the two node values is even, the updated states will encode values $average - 1$ and $average + 1$. If the average is odd, then both

Parameters:
m, an odd integer > 0, and d, integer > 0
State Space:
$StrongStates = \{-m, -m+2, \ldots, -3, 3, 5, \ldots, m-2, m\}$,
$IntermediateStates = \{-1_1, -1_2, \ldots, -1_d, 1_d, 1_{d-1}, \ldots, 1_1\}$,
$WeakStates = \{-0, +0\}$
Input: States of two nodes, x and y
Output: Updated states x' and y'
Auxiliary Procedures:
`/* Functions mapping states to integers */`
1 $weight(x) = \begin{cases} |x| & \text{if } x \in StrongStates \text{ or } x \in WeakStates; \\ 1 & \text{if } x \in IntermediateStates. \end{cases}$
2 $sgn(x) = \begin{cases} 1 & \text{if } x \in \{+0, 1_d, \ldots, 1_1, 3, 5, \ldots, m\}; \\ -1 & \text{otherwise.} \end{cases}$
3 $value(x) = sgn(x) \cdot weight(x)$
`/* Functions for rounding state interactions */`
4 $\phi(x) = -1_1$ if $x = -1$; 1_1 if $x = 1$; x, otherwise
5 $R_\downarrow(k) = \phi(k$ if k odd integer, $k-1$ if k even$)$
6 $R_\uparrow(k) = \phi(k$ if k odd integer, $k+1$ if k even$)$
7 $Shift\text{-}to\text{-}Zero(x) = \begin{cases} -1_{j+1} & \text{if } x = -1_j \text{ for some index } j < d \\ 1_{j+1} & \text{if } x = 1_j \text{ for some index } j < d \\ x & \text{otherwise.} \end{cases}$
8 $Sign\text{-}to\text{-}Zero(x) = \begin{cases} +0 & \text{if } sgn(x) > 0 \\ -0 & \text{oherwise.} \end{cases}$
9 **procedure** update$\langle x, y \rangle$
10 **if** $(weight(x) > 0$ *and* $weight(y) > 1)$ ***or*** $(weight(y) > 0$ *and* $weight(x) > 1)$ **then**
11 $x' \leftarrow R_\downarrow\left(\frac{value(x)+value(y)}{2}\right)$ and $y' \leftarrow R_\uparrow\left(\frac{value(x)+value(y)}{2}\right)$
12 **else if** $weight(x) \cdot weight(y) = 0$ *and* $value(x) + value(y) > 0$ **then**
13 **if** $weight(x) \neq 0$ **then** $x' \leftarrow Shift\text{-}to\text{-}Zero(x)$ and $y' \leftarrow Sign\text{-}to\text{-}Zero(x)$
14 **else** $y' \leftarrow Shift\text{-}to\text{-}Zero(y)$ and $x' \leftarrow Sign\text{-}to\text{-}Zero(y)$
15 **else if** $(x \in \{-1_d, +1_d\}$ *and* $weight(y) = 1$ *and* $sgn(x) \neq sgn(y))$ ***or***
16 $(y \in \{-1_d, +1_d\}$ *and* $weight(x) = 1$ *and* $sgn(y) \neq sgn(x))$ **then**
17 $x' \leftarrow -0$ and $y' \leftarrow +0$
18 **else**
19 $x' \leftarrow Shift\text{-}to\text{-}Zero(x)$ and $y' \leftarrow Shift\text{-}to\text{-}Zero(y)$

Figure 1: The state update rules for the majority computation.

states encode *average*. When a node reaches value 1 or -1, its state becomes either 1_1 or -1_1. This rounding is performed through the functions $R_\uparrow$ and $R_\downarrow$.

Second, if a node with 0 weight meets a node with non-zero weight, then it will adopt the sign of the interaction partner via the *Sign-to-Zero* function, whose weight remains unchanged (lines 12—14). Nodes $\pm 1_j$ with $j < d$ are a slight exception, as they become downgraded to $\pm 1_{j+1}$.

Third, if the participants have different signs, weight 1, and one of them is in state 1_d or -1_d, then both are downgraded to value 0, albeit with different signs (lines 15—17).

In the last case unless both values are zero (and nothing changes) both participants have weight 1 but none of them is in state 1_d or -1_d. Then, both keep weight 1, but are downgraded to the next level of state 1 via the *Shift-to-Zero* function. For example, a node in a state 1_j with $j < d$ now moves to state 1_{j+1}.

4 Analysis of the Average and Conquer Algorithm

In this section, we provide a complete analysis of the AVC algorithm. We first prove correctness, showing that the algorithm always converges to a correct answer, and also characterize the distribution of its convergence time.

Notation: Throughout this proof, we denote the set of nodes executing the protocol by V. We measure execution time in discrete steps (rounds), where each step corresponds to an interaction. The *configuration* at a given time t is a

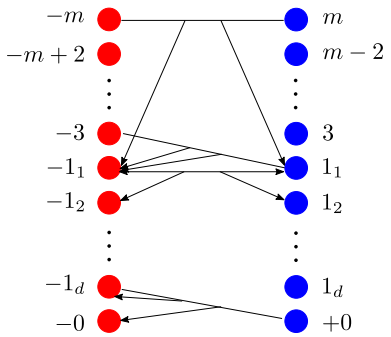

Figure 2: Structure of the states, and some reaction examples. Initially, nodes are in state $-m$ or $+m$. Each line corresponds to a reaction between two states; the two arrows point to the resulting states. For example, states m and $-m$ react to produce states -1_1 and 1_1.

function $c : V \to Q$, where $c(v)$ is the state of the node v at time t. (We omit the explicit time t when clear from the context.) We use the quantities $sgn(c(v))$, $weight(c(v))$ and $value(c(v))$ associated with the state $c(v)$ of the node v in configuration c, as defined on Figure 1 lines 1-3. We will refer to them as the sign, weight and value of the node, respectively.

Analysis Overview: The analysis first considers correctness, and then speed of convergence. For correctness, we prove that nodes never converge to the sign of the initial minority (safety), and that they eventually converge to the sign of the initial majority (termination). The first statement follows since the algorithm is designed to keep the total sum of values in the system *invariant*: for instance, if the initial sum is positive, then positive values must always exist in the system. Further, we prove that, eventually, all minority values are eliminated, implying termination.

The convergence bound is more involved. We first focus on the *extremal* weights in the system, showing that, roughly, either their weight is halved every $\log n$ parallel time steps, or the number of nodes in strong states is depleted for one of the signs. Further, this halving process, which takes expected $O(\log n \log m)$ parallel time, still preserves the initial system sum: therefore, instead of having a few deciding nodes with high values, after the halving completes, we will have many deciding nodes of small value. We then show that these nodes sway the remaining minority states to the correct decision within expected $O(\log n/(\epsilon m))$ parallel time, completing the bound. Our main result is the following theorem.

Theorem 4.1. *The* AVC *algorithm solves majority with probability* 1. *There is a setting of parameters, such that it uses* $s = m + O(\log n \log m)$ *states with* $\log n \log\log n \leq m \leq n$ *and has parallel convergence time* $O(\log n/(s\epsilon) + \log n \log s)$ *in expectation and* $O(\log^2 n/(s\epsilon) + \log^2 n)$ *with high probability.*

An interesting setting of parameters $s = 1/\epsilon$ gives the following.

Corollary 4.2. *There is a setting of parameters, such that the* AVC *algorithm uses* $s = 1/\epsilon + O(\log 1/\epsilon \log n)$ *states, and has parallel convergence time* $O(\log 1/\epsilon \log n)$ *in expectation and* $O(\log^2 n)$ *w.h.p.*

Correctness: We observe that, given the interaction rules of the algorithm, the sum of the encoded values stays constant as the algorithm progresses. The proof follows by the structure of the algorithm. In particular, the averaging reaction is designed to maintain the sum. Same holds for all other types of interactions.

Invariant 4.3. *The sum* $\sum_{v \in V} value(c(v))$ *never changes, for all reachable configurations* c *of the protocol.*

This invariant implies that the algorithm may never converge to a wrong decision value, as at least a node with the sign of the initial majority must exist in the system. We therefore only need to show that the algorithm converges to a state where all nodes have the same sign, which we do via a rough convergence bound, assuming an arbitrary starting configuration.

Lemma A.1. *Let* c *be an arbitrary starting configuration for the the* AVC *algorithm and define* $S := \sum_{v \in V} value(c(v)) \neq 0$. *With probability* 1, *the algorithm will reach a configuration* f *such that* $sgn(f(v)) = \mathrm{sgn}(S)$ *for all nodes* $v \in V$. *Moreover, in all later configurations* e *reachable from* f, *no node can ever have a different sign, i.e.* $\forall v \in V : sgn(e(v)) = sgn(f(v))$. *The convergence time to* f *is at most* $n^3(m + d + 1)$ *expected communication rounds, i.e. parallel time* $n^2(m + d + 1)$.

Convergence Time Bounds: We now focus on bounding the time until all nodes converge to the correct sign. We begin by establishing some notation. We call a round a *negative-round* if, in the configuration c at the beginning of the round, at least a *third* of the nodes encode a strictly negative value, i.e. $|\{v \in V : value(c(v)) < 0\}| \geq |V|/3$. Analogously, we call a round a *positive-round* if a third of the nodes encode a strictly positive value. A round can be simultaneously negative *and* positive. Our first claim bounds the speed at which the maximum weight present in the system decreases. The proof, given in full in the Appendix, follows by bounding the probability that a node with the maximum weight meets a node with non-zero value and opposite sign in a round. This decreases the weight of the node, and we use Chernoff and Union Bounds to bound the probability that the node avoids such a meeting for logarithmic parallel time.

Claim A.2. *Let $w > 1$ be the maximum weight among the nodes with a negative (resp., positive) sign. For constant $\beta \geq 54$, after $\beta n \log n$ positive-rounds (resp., negative-rounds) the maximum weight among the nodes with a negative (resp., positive) sign will be at most $\lfloor w/2 \rfloor$ with probability at least $1 - \log n/n^{\beta/54}$.*

In the following, we consider the AVC algorithm with parameters $\log n \log\log n \leq m \leq n$ and $d = 1000 \log m \log n$, for the total of $s = m + O(\log m \log n)$ states. We first prove that in this setting, no node will reach value 0 within the first $432n \log m \log n$ rounds, w.h.p. The proof follows from Chernoff Bound by showing that the probability that a node gets selected at least d times (necessary for it to downgrade to 0) during this interval is extremely low.

Claim A.3. *The probability that any node gets weight 0 during the first $432n \log m \log n$ rounds is $\leq 1/n^4$.*

Claim A.2 gave a condition for halving the maximum positive and the minimum negative values in the system with high probability. The initial maximum weight in the system is m, which can only be halved $\log m$ times for each sign. This motivates the following claim:

Claim 4.4. *During the first $432n \log m \log n$ rounds, if no node gets value 0, then with probability at least $1 - 2 \log n \log m/n^4$, one of the following three events occurs at some point during these rounds:*

1. *Nodes only have values in $\{-1, 1\}$;*
2. *There are less than $n/3$ nodes with negative values, all with value -1,*
3. *There are less than $n/3$ nodes with positive values, all with value 1.*

Proof. Since no node gets value 0 during the first $432n \log m \log n$ rounds, each round during this interval is a negative-round, a positive-round, or both. Unless the maximum positive value in the system is 1, by Claim A.2, a stretch of $216n \log n$ negative-rounds halves the maximum positive value available, with probability at least $1 - \log n/n^4$. The same occurs for stretches of $216n \log n$ positive-rounds and the minimum negative value.

Assume that none of the three events hold at any time during the first $432n \log m \log n$ rounds. In that case, each round can be classified as either a negative round where the maximum positive value is strictly larger than 1, or a positive round where the minimum negative value is strictly smaller than -1. Thus, each round contributes to at least one of the stretches of $216n \log n$ rounds that halve the maximum positive of minimum negative value, w.h.p. However, this may happen at most $2 \log m$ times. By applying Claim A.2 $2 \log m$ times (using $\beta = 216$) and the Union Bound we get that after the first $432n \log m \log n$ rounds, with probability at least $1 - 2 \log n \log m/n^4$ only values -1 and 1 will remain.

However, this is the same as the first event above. Thus, the probability that none of these events happen is at most $2 \log n \log m/n^4$. □

Returning to the proof, recall that we assumed without loss of generality that the initial majority of nodes was in A (positive) state, i.e. $a > b$. The following claim provides the last piece of the puzzle.

Claim 4.5. *Consider a configuration after the first $432n \log m \log n + 4dn$ rounds. With probability at least $1 - (2 \log n \log m + 2)/n^4$, all nodes with a strictly negative value will be in state -1_d while at least $n \cdot \min(1/3, \epsilon m)$ more nodes will encode a strictly positive value than the nodes in state -1_d.*

Proof. By our assumption about the initial majority and Invariant 4.3, $\sum_{v \in V} value(c(v)) = \epsilon nm$ holds in every reachable configuration c. Let us focus on the high probability events in Claim A.3 and Claim 4.4: no node gets a value 0 within the first $432n \log m \log n$ rounds, and during these rounds the execution reaches a configuration where one of the three events from Claim 4.4 holds. Consider this point T in the execution.

Since no node has value 0, the third event is impossible, because the total sum would be negative. Second event implies that there are at least $n/3$ more strictly positive values than strictly negative values. In the first event, the total sum is $\geq \epsilon nm$ and values are all -1 and 1, so there must be at least ϵmn more strictly positive than negative values. Hence, at point T during the first $432n \log n \log m$ rounds there are at least $n \cdot \min(1/3, \epsilon m)$ more strictly positive than strictly negative values.

In both cases, -1's are the only strictly negative values of the nodes at point T, and this will be the case for the rest of the execution because of the update rules. Also, for any level $i < d$, every time one of the nodes in state -1_i interacts with another node with value not equal to -1 (has to be value ≥ 0), either both nodes get values ≥ 0, or the node moves to state -1_{i+1} via the *Shift-to-Zero* function.

In each round, the probability that a given node with value -1 interacts with a node with value ≥ 0 is at least $\min(2/3, (1 + \epsilon m)/2)/n \geq 1/(2n)$. Let us describe the number of times the given node is selected in for such an interaction during the at least $4dn$ rounds after T by a random variable $Y \sim \text{Bin}(4dn, 1/2n)$. Then, by Chernoff Bound, the probability that the node is selected less than d times is at most:

$$\Pr[Y \leq d] = \Pr\left[Y \leq 2d\left(1 - \frac{1}{2}\right)\right] \leq \exp\left(-\frac{2d}{2 \cdot 2^2}\right) \leq 2^{-d/4} < \frac{1}{n^5},$$

where we have used that d is set to $1000 \log m \log n$. Union Bound over all the $\leq n$ nodes that had value -1 at point T, we get that after $4dn$ more rounds, with probability $1 - 1/n^4$, all these nodes interacted at least d times with nodes with non-negative values. Hence, they either have value ≥ 0 or are in state -1_d.

After T, the number of nodes with strictly positive values only decreases in an interaction with a node in state -1_i, in which case the number of nodes with strictly negative values also decreases by one. Hence there must still be at least $n \cdot \min(1/3, \epsilon m)$ more nodes with strictly positive that strictly negative values.

A Union Bound over Claim A.3, Claim 4.4 and the above argument completes the proof. □

By Claim 4.5, with high probability after $432n \log m \log n + 4dn$ rounds we reach a configuration, from which the convergence bares many similarities to the four state protocol of [DV12] and can be analyzed similarly. We describe its convergence below, and the proof is given in Appendix A.

Claim A.4. *Consider a configuration where all nodes with strictly negative values are in state -1_d, while at least $n\delta$ more nodes encode a strictly positive value than the nodes in state -1_d. The number of rounds until convergence is $O(\frac{n \log n}{\delta})$ in expectation and $O(\frac{n \log^2 n}{\delta})$ with high probability.*

Thus, after the first $432n \log m \log n + 4dn$ rounds, in the high probability case, plugging $\delta = \min(1/3, \epsilon m)$ in Claim A.4, we get that the remaining number of rounds is at most $O(n \log n + (n \log n)/(m\epsilon))$ in expectation and $O(n \log^2 n + (n \log^2 n)(m\epsilon))$ w.h.p. The high probability statement immediately follows by the Union Bound. In our setting of parameters m, d and s, $s = \Theta(m)$ which leads to the desired bounds.

We only have to bound expectation in the remaining low probability event. With probability at most $(2 \log n \log m + 2)/n^4$, the expected remaining number of rounds is at most $O(n^3(m + d))$ by Lemma A.1 and since in our parameter setting $m + d = O(n)$, it is negligible compared to the previous case.

5 Lower Bounds

We explore the convergence-memory trade-off suggested by our algorithm at two extremal points: algorithms with four states, and algorithms using arbitrarily many states. We prove the following lower bounds.

5.1 Convergence Time with Four States

We prove an $\Omega(1/\epsilon)$ lower bound on the convergence time of any four-state protocol for exact majority. The proof extends the approach introduced by [MNRS14] to show impossibility of exact three-state majority protocols, in that it explores all possible state transitions and output mappings. However, there are considerably more possibilities to consider, among which, unlike the three-state case, some yield correct four-state protocols. We combine indistinguishability arguments to demonstrate that certain algorithms could converge to the incorrect output, with potential-based arguments to show that certain algorithms may never converge, and with convergence analysis of the correct algorithms.

For simplicity of presentation, we slightly alter notation in this section. We call the initial states S_1 (corresponding to A) and S_0 (corresponding to B). The output (formally defined by the mapping γ) should be $i \in \{0, 1\}$ if a majority of the nodes start in state S_i. We define C_i as a set of all *absorbing* configurations c for output i: for all configurations c_i reachable from $c \in C_i$, we must have $\forall v \in V : \gamma(c_i(v)) = i$.

Theorem B.1. *Consider any majority algorithm using four states, providing the following correctness properties for any $|V| = n \geq 1$:*

- *For each possible outcome $i \in \{0, 1\}$, the set C_i of absorbing configurations is non-empty.*
- *The system never converges to the wrong decision: if in the initial configuration the majority of nodes are in S_i for $i \in \{0, 1\}$, it must be impossible to reach any configuration $c \in C_{1-i}$ under* any *schedule of interactions.*
- *It is always possible to converge to the correct solution: if a majority of nodes start in S_i for $i \in \{0, 1\}$, from every reachable configuration there is a sequence (schedule) of interactions leading to a configuration $c \in C_i$.*

Then, if in the initial configuration ϵn more nodes start in state S_i than in S_{1-i} for $i \in \{0, 1\}$, the expected parallel convergence time to reach some configuration $c \in C_i$ is $\Omega(1/\epsilon)$.

5.2 Convergence Lower Bound for Arbitrary Number of States

We now prove that any algorithm solving exact majority must take expected $\Omega(\log n)$ parallel time until convergence on a worst-case input, irrespective of the number of states it uses. The structure is as follows: we start from an input state where the outcome is decided by a *constant* number of nodes (the initial states of other nodes are balanced). We then bound expected parallel time for the initial states of these nodes to propagate to every node by $\Omega(\log n)$ by a standard information-based argument, e.g. [CDR86, Jay98]. We then argue that any node that does not have this decisive information may only have converged with *constant probability*, completing the proof. The complete argument is given in the appendix.

Theorem C.1. *Any algorithm $\mathcal{A}$ solving exact majority takes expected $\Omega(\log n)$ parallel time until convergence under a worst-case input.*

6 Experiments and Discussion

Empirical Results: We have tested the performance of our protocol through implementation, and compared it with the three-state protocol of [AAE08, PVV09], which has non-zero probability of error, and with the exact four-state protocol of [DV12, MNRS14]. (Please see Appendix D for a detailed description.)

For $\epsilon = 1/n$, the n-state version of AVC is orders of magnitude faster than the four-state protocol, and has comparable running time with the three-state protocol, noting that the latter has constant failure probability in this case (see Figure 3). Experiments varying parameters s and ϵ support the claim that the running time of the protocol is linear in these parameters, and that the constant factors are small (see Figure 4).

Discussion: We have given the first sub-linear time population protocol solving exact majority, which highlights a new trade-off between the number of memory states of the algorithm (alternatively, number of local bits), the convergence time, and the error probability of the protocol. We have also shown that this trade-off is tight up to logarithmic factors when considered at its extremal points in s.

The above experiments were performed setting $d = 1$, i.e., with a single version of the 1 states. This suggests that this variant states is also correct and efficient. The multiple levels of 1 and -1 are necessary in the analysis; however, setting $d > 1$ does not significantly affect the running time of the protocol in the experiments. It is therefore an interesting question whether the protocol can be further simplified.

The main open question left by our work is that of matching lower bounds. We conjecture that the $\Omega(1/(s\epsilon))$ convergence time dependence is in fact inherent for exact majority. In general, the question of what can be computed in a population protocol with limited memory per node is intriguing. It would also be interesting to explore whether the average-and-conquer technique would also be useful in the context of other problems, such as leader election in population protocols. Further, we highlight the question of obtaining a fast *approximate* majority protocol which is correct with high probability for all possible values of the discrepancy ϵ.

References

[AAD+06] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed computing*, 18(4):235–253, 2006.

[AAE08] Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, July 2008.

[AAER07] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, November 2007.

[CCN12] Luca Cardelli and Attila Csiksz-Nagy. The cell cycle switch computes approximate majority. *Nature Scientific Reports*, 2:656, 2012.

[CDR86] Stephen Cook, Cynthia Dwork, and Ruediger Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.*, 15(1):87–97, February 1986.

[CDS+13] Yuan-Jyue Chen, Neil Dalchau, Niranjan Srnivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from dna. *Nature Nanotechnology*, 8(10):755–762, 2013.

[DMST07] Ian B. Dodd, A. M. Micheelsen, Kim Sneppen, and Geneviéve Thon. Theoretical analysis of epigenetic cell memory by nucleosome modification. *Cell*, 129(4):813–822, 2007.

[DV12] Moez Draief and Milan Vojnovic. Convergence speed of binary interval consensus. *SIAM Journal on Control and Optimization*, 50(3):1087–1109, 2012.

[HP99] Yehuda Hassin and David Peleg. Distributed probabilistic polling and applications to proportionate agreement. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, ICAL '99, pages 402–411, London, UK, UK, 1999. Springer-Verlag.

[Jay98] Prasad Jayanti. A time complexity lower bound for randomized implementations of some shared objects. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '98, pages 201–210, New York, NY, USA, 1998. ACM.

[Lig85] Thomas M. Liggett. *Interacting Particle Systems*. Springer, 1985.

[MNRS14] George B. Mertzios, Sotiris E. Nikoletseas, Christoforos Raptopoulos, and Paul G. Spirakis. Determining majority in networks with local interactions and very small local memory. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, ICALP '14, pages 871–882, 2014.

[PVV09] Etienne Perron, Dinkar Vasudevan, and Milan Vojnovic. Using three states for binary consensus on complete graphs. In *INFOCOM 2009, IEEE*, pages 2527–2535. IEEE, 2009.

A Deferred Proofs

Lemma A.1. *Let c be an arbitrary starting configuration for the the* AVC *algorithm and define $S := \sum_{v \in V} value(c(v)) \neq 0$. With probability 1, the algorithm will reach a configuration f such that $sgn(f(v)) = \mathrm{sgn}(S)$ for all nodes $v \in V$. Moreover, in all later configurations e reachable from f, no node can ever have a different sign, i.e. $\forall v \in V : sgn(e(v)) = sgn(f(v))$. The convergence time to f is at most $n^3(m + d + 1)$ expected communication rounds, i.e. parallel time $n^2(m + d + 1)$.*

Proof. Assume without loss of generality that the sum S is positive. At any time, given the current configuration is r, let us define $P := \sum_{v \in V \mid sgn(r(v))>0} weight(r(v))$ and $N := \sum_{v \in V \mid sgn(r(v))<0} weight(r(v))$. Then, by Invariant 4.3, $P - N = S > 0$ holds throughout the execution. Also, $N \leq nm$, as each of the at most n contributing weights to the sum is at most m.

We estimate the expected convergence time by splitting the execution into three phases. The first phase starts at the beginning of the execution, and lasts until either i) N becomes 0, *or* ii) until all nodes are in states that encode values in $\{-1, 0, 1\}$. Therefore, throughout this phase, there are always nodes that encode non-zero values with different signs and at least one node in a strong state (with weight > 1).

Consequently, in any communication round in the first phase, with at least $1/n^2$ probability this node of large weight interacts with a node with opposite sign and non-zero value, and therefore N (and P) both decrease by at least 1. At the same time, because of the update rules of the algorithm, N can never increase. In expected n^2 rounds, N will decrease by at least 1. By $N < nm$ and the linearity of expectation, expected number of rounds until N becomes 0, or until no nodes remain in a strong state (maximum weight of nodes ≤ 1), is at most $n^3 m$. This upper bounds the expected number of rounds in the first phase.

The second phase starts immediately after the first, and ends when N becomes 0. Note that if the first phase ended because of $N = 0$, the second phase is trivially empty. Now consider the case when all nodes are in states with values

-1, 0 and 1 at the beginning of the second phase. There are clearly N nodes with value -1, P nodes with value 1 ($N < P < n$) and the remaining nodes must be in weak states (weight 0).

Under these circumstances, because of the update rules, no node will ever be in a strong state (weight > 1) again in any future configuration. Also, the number of nodes in intermediate states (weight 1) can only decrease. In each round, with probability at least $1/n^2$, two nodes with weights 1_i and 1_j and opposite signs meet. Either $\max(i, j) = d$ and both new values become zero, meaning N (and P) decreases by 1, or the weights become 1_{i+1} and 1_{j+1} without affecting the signs. But this can only happen nd times, since there can be at most n nodes encoding value -1, and each becomes 0 after at most d meetings with a nodes of opposite sign. Therefore, the expected number of rounds in the second phase is at most $n^3 d$.

After the second phase, $P = S$ holds and all nodes with negative sign ought to have weight 0. From this time, all rounds belong to the final, third phase, throughout which N clearly remains equal to zero. The third phase lasts until the system converges, that is, until all nodes with weight 0 get a positive sign. Since $S \geq 1$ there is at least one node with a positive sign and non-zero weight, and there are at most $n - 1$ *conflicting* nodes with a negative sign, all with value zero as $N = 0$. Thus, independently in each round, with probability at least $1/n^2$, a conflicting node meets a node with strictly positive value and the number of conflicting nodes decreases by one. The number of conflicting nodes never increases because of the update rules and when it becomes zero, the system has converged to the desired configuration f. Therefore, the expected number of rounds in the third phase is at most n^3.

Combining the results, the total expected number of rounds is at most $n^3(m + d + 1)$. The expectation being finite implies that the algorithm converges with probability 1. Finally, when two nodes with positive sign meet, they both remain positive, so any configuration e reachable from f also contains every node with the correct sign. □

Claim A.2. *Let $w > 1$ be the maximum weight among the nodes with a negative (resp., positive) sign. For constant $\beta \geq 54$, after $\beta n \log n$ positive-rounds (resp., negative-rounds) the maximum weight among the nodes with a negative (resp., positive) sign will be at most $\lfloor w/2 \rfloor$ with probability at least $1 - \log n / n^{\beta/54}$.*

Proof. We will prove the claim for nodes with strictly negative values. (The converse claim follows analogously.) Fix a round r, and recall that w is the maximum weight of a node with a negative value at the beginning of the round. Let U be the set of nodes encoding negative values between $-\lceil w/2 \rceil$ and $-w$, with $w > 1$, at the beginning of the round, and let $u = |U|$. We call these nodes *target nodes*.

By the structure of the AVC algorithm, the number of target nodes never increases, and decreases by one in every *eliminating* round where a target node meets a node with a strictly positive value. Consider a set of αn consecutive positive-rounds after r, for $\alpha \geq 18$. In each round, if there are still at least $\lceil u/2 \rceil$ target nodes, then the probability of this round being eliminating is at least $\lceil u/2 \rceil / 3n$ (recall that by definition of a positive round at least third of the nodes have strictly positive value). Let us describe the process by considering a random variable $X \sim \mathrm{Bin}(\alpha n, \frac{\lceil u/2 \rceil}{3n})$, where each success event corresponds to an eliminating round. By a Chernoff Bound, the probability of having αn iterations ($\alpha \geq 18$) with at most $\lceil u/2 \rceil$ eliminations is at most:

$$\Pr\left[X \leq \lceil u/2 \rceil\right] = \Pr\left[X \leq \frac{\alpha \lceil u/2 \rceil}{3}\left(1 - \frac{\alpha - 3}{\alpha}\right)\right] \leq \exp\left(-\frac{\alpha \lceil u/2 \rceil (\alpha - 3)^2}{6\alpha^2}\right) \leq 2^{-\frac{u\alpha}{18}}.$$

For $u \geq \log n$, the probability of this event is at most $\frac{1}{n^{\alpha/18}}$ for αn positive-rounds. Applying the same rationale iteratively as long as $u \geq \log n$, we obtain by using a Union Bound that the number of target nodes will become less than $\log n$ within $\alpha n(\log n - \log\log n)$ positive-rounds, with probability at least $1 - (\log n - \log\log n)/n^{\alpha/18}$.

Finally, we wish to upper bound the remaining number of positive-rounds until no target node remains. When $u < \log n$, we get from the same argument as above that the number of target nodes is reduced to $\lfloor u/2 \rfloor$ within $\frac{\alpha n \log n}{u}$ consecutive positive-rounds with probability $1/n^{\alpha/18}$. So we consider increasing numbers of consecutive positive-rounds, and obtain that no target nodes will be left after at most $\alpha n + 2\alpha n + \ldots + \alpha n \log n \leq 2\alpha n \log n$ positive-rounds, with probability at least $1 - \log\log n / n^{\alpha/18}$, where we have taken the union bound over $\log\log n$ events. The original claim follows by setting $\beta = 3\alpha$, and taking the Union Bound over the above two events ($u \geq \log n$ and $u < \log n$). □

Claim A.3. *The probability that any node gets weight 0 during the first $432n \log m \log n$ rounds is $\leq 1/n^4$.*

Proof. In each round, the probability that a particular node is selected is $2/n$. Let us describe the number of times a given node is selected in $432n \log m \log n$ rounds by considering a random variable $Z \sim \mathrm{Bin}(432n \log m \log n, 2/n)$.

Given that $d = 1000 \log m \log n$, by the Chernoff Bound, the probability that the node gets selected more than d times in these rounds is at most:

$$\Pr[Z \geq d] = \Pr\left[Z \geq (864 \log m \log n) \cdot \left(1 + \frac{136}{864}\right)\right] \leq \exp\left(-\frac{864 \cdot 136^2}{3 \cdot 864^2} \log m \log n\right) \leq \frac{1}{(mn)^5}$$

A node cannot get weight 0 during the first $432n \log m \log n$ rounds unless it is selected for interactions at least d times in total. Thus, the probability that a given node gets value zero is at most $1/(mn)^5$. Applying the Union Bound over all nodes, we conclude that no node gets $weight = 0$ during the first $432n \log m \log n$ rounds, with probability at least $1 - 1/(m^5 n^4) \geq 1 - 1/n^4$. □

Claim A.4. *Consider a configuration where all nodes with strictly negative values are in state -1_d, while at least $n\delta$ more nodes encode a strictly positive value than the nodes in state -1_d. The number of rounds until convergence is $O(\frac{n \log n}{\delta})$ in expectation and $O(\frac{n \log^2 n}{\delta})$ with high probability.*

Proof. In any configuration, let us call *conflicting* any node that is in state -1_d, and *target* node any node that has a strictly positive value. Because of the structure of the algorithm, and that in configuration c the only nodes with negative sign are in states -1_d or -0, in all configurations reachable from c nodes with negative values will also only be in states -1_d or -0. Moreover, the number of conflicting nodes can never increase after an interaction. As in Claim 4.5, observe that the number of target nodes can only decrease after an interaction where a node with value 1 meets a node in state -1_d, in which case the number of conflicting nodes also decreases by one. There are $n\delta$ more target nodes than conflicting nodes in c, therefore, in every later configuration, there must always be at least $n\delta$ target nodes.

Every time a conflicting node meets a target node, both nodes get value ≥ 0, so the number of conflicting nodes decreases by one. Let us estimate the number of rounds until each conflicting node has interacted with a target node, at which point no more conflicting nodes may exist. Let us say there were x conflicting nodes in configuration c. The expected number of rounds until the first conflicting node meets a target node is at most $\frac{n}{x\delta}$, since the probability of such an interaction happening in each round is at least $\frac{x}{n} \cdot \frac{n\delta}{n}$. The expected number of rounds for the second node is then $\frac{n}{(x-1)\delta}$, and so on. By linearity of expectation, the expected number of rounds until all conflicting nodes are eliminated is $O(\frac{n \log x}{\delta}) \leq O(\frac{n \log n}{\delta})$.

At this point, all nodes that have a negative sign must be in state -0. If we redefine *conflicting* to describe these nodes, it is still true that an interaction of a conflicting node with a target node brings the conflicting node to state $+0$, thus decreasing the number of conflicting nodes. As we discussed at least $n\delta$ target nodes are still permanently present in the system. By the structure of the algorithm no interaction can increase the number of nodes with negative sign, the system will converge when all conflicting nodes are eliminated. But this takes expected $O(\frac{n \log n}{\delta})$ rounds by exactly the same argument as above.

To get the high probability claim, simply observe that when there are x conflicting nodes in the system, a conflicting node will interact with a target node within $\frac{nO(\log n)}{x\delta}$ rounds, with high probability. The same applies for the next conflicting node, etc. Taking Union Bound over these events gives the desired result. □

B Lower Bound Proofs

Theorem B.1. *Consider any majority algorithm using four states, providing the following correctness properties for any $|V| = n \geq 1$:*

- *For each possible outcome $i \in \{0, 1\}$, the set C_i of absorbing configurations is non-empty.*
- *The system never converges to the wrong decision: if in the initial configuration the majority of nodes are in S_i for $i \in \{0, 1\}$, it must be impossible to reach any configuration $c \in C_{1-i}$ under* any *schedule of interactions.*
- *It is always possible to converge to the correct solution: if a majority of nodes start in S_i for $i \in \{0, 1\}$, from every reachable configuration there is a sequence (schedule) of interactions leading to a configuration $c \in C_i$.*

Then, if in the initial configuration ϵn more nodes start in state S_i than in S_{1-i} for $i \in \{0, 1\}$, the expected parallel convergence time to reach some configuration $c \in C_i$ is $\Omega(1/\epsilon)$.

Proof. Let the four states be S_0, S_1, X and Y, where S_0 and S_1 are the two starting states. Without the loss of generality, we can assume that the output mapping γ assigns 0 or 1 to each of the 4 states. The algorithm should be correct for $n = 1$ and a single node that starts in state S_0 remains in state S_0 forever. Therefore $\gamma(S_0) = 0$ must hold and analogously, we also have $\gamma(S_1) = 1$.

Let us start by the following useful claim

Claim B.2. *Let Z and W be configurations for n nodes: in configuration Z there are z nodes in state S_0 and $n-z$ nodes in S_1, while in W we have w nodes in S_0 ($n-w$ in S_1). Let R_z and R_w be the sets of configurations reachable from Z and W, respectively. Then if $z \neq w$, $R_Z \cap R_W = \varnothing$.*

Proof. Without the loss of generality $z > w$. Assume contrary that there exists a configuration $r \in R_z \cap R_w$. Now consider a system of $2n-1$ nodes, with n *special* nodes starting in Z or W, $n-1-w$ more nodes starting in S_0 and the remaining w nodes starting in S_1. No matter if the special nodes start in configuration Z or W, we can start by a sequence of interactions involving only the n special nodes and lead them to the same r, as these nodes will not observe a difference from running alone. However, when special nodes start in Z, the initial majority out of all $2n-1$ nodes is S_0 (with $z + (n-1-w) \geq n$ nodes) and when special nodes start in W, the initial majority is S_1 (with $(n-w)+w = n$ nodes). Thus, we can reach the same intermediate configuration (n nodes according to r, $n-1-w$ in S_0 and w in S_1) from two starting configurations in a system of $2n-1$ nodes, one with majority S_0 and another with S_1.

As the intermediate configuration is reachable from a starting configuration with majority S_0, by the third property, there exists a sequence of interactions that leads to a configuration $c \in C_0$ from the intermediate configuration. But we can also start with majority S_1, still reach the intermediate configuration, and from there reach $c \in C_0$. However, this contradicts with the second correctness property for the starting configuration with majority S_1 and completes the proof. □

As an example, above claim has the following interesting corollary.

Corollary B.3. *Consider $n \geq 2$ nodes in a configuration c where each node is in either S_0 or S_1 state, and there is at least one node in each of these states. Then no sequence of interactions among these nodes may lead to a configuration with all nodes in state S_0 or all in state S_1.*

There are two cases based on whether outputs $\gamma(X)$ and $\gamma(Y)$ are the same or not. Let us first consider the simple case $\gamma(X) = \gamma(Y)$ and assume without the loss of generality that both of these are equal to 1. The only possible absorbing configuration in C_0 must then have every node in state S_0, as all other states map to output 1. By the first property, C_0 cannot be empty, so this configuration is actually in C_0. By the third property, from every starting configuration with the majority nodes in S_0, this sole absorbing configuration of all nodes in S_0 must be reachable by a sequence of interactions. This contradicts with Corollary B.3 when for example $n = 3$ with with two nodes in S_0 and one node in S_1. Notice that if we considered X and Y as two names for the same third state (other than S_0 and S_1), this is essentially the contradiction argument that shows as in [MNRS14] that no algorithm can solve exact majority with only 3 states.

Therefore, any algorithm must have $\gamma(X) \neq \gamma(Y)$. Assume without the loss of generality that $\gamma(X) = 0$ and $\gamma(Y) = 1$. Next step is the following

Claim B.4. *There exists a system with some number of nodes, such that the set C_0 contains a* mixed *absorbing configuration in which at least two nodes are in each of the states S_0 and X. Similarly, for some other system, C_1 contains a configuration with at least two nodes in S_1 and two nodes in Y.*

Proof. Let us prove the claim for C_0, as the other case is analogous. In all configurations in C_0, all nodes must be in states S_0 or X, because of the first property and that γ maps other states to 1.

Now assume contrary, and consider the following four starting configurations with $n = 9$ nodes. In configuration O, a single node starts in state S_1 and eight nodes start in S_0. In configuration T, two nodes start in S_1 (seven in S_0), while configuration H has three in S_1 (six in S_0). Finally, in configuration F, four nodes start in S_1 and five in S_0.

For all of these configurations majority of nodes start in S_0 and by the third property, there exists a schedule of interactions that leads the system to an absorbing configuration in C_0. If a mixed configuration (with two nodes in each X and S_0) in C_0 can be reached from either O, T, H or F, we are done. Also, by Corollary B.3, no $c \in C_0$ reachable from any of these starting configurations may have all nodes in S_0. This leaves only three possibilities for a configuration in C_0 that can be reached: with all nine nodes in state X, one in X and eight in S_0 or with one in S_0 and eight in X.

Thus, from each of the four starting configurations O, T, H and F, it is possible to reach one of the three possible absorbing states in C_0. By pigeon-hole principle, we can reach the same absorbing configuration from two of these starting configurations. This gives the desired contradiction with Claim B.2 for those two configurations and $n = 9$, completing the proof. □

Now we have all the tools to see how the states of nodes evolve after interactions. Let us use an example to introduce some necessary notation. We show in the next claim that if in any correct protocol, a node in S_0 interacts with a node in X, one node must again be in S_0 and one in X after the interaction. To make this statement, we will henceforth use the formal notation $[S_0, X] \to \{[S_0, X]\}$, whereby interacting nodes are not ordered ($[S_0, X]$ is the same as $[X, S_0]$) and the set on the right hand side describes possible states that the two nodes can be in after the interaction as specified by any correct (satisfying our three properties) 4-state exact majority algorithm.

Claim B.5. *In any correct (satisfying our three properties) 4-state algorithm for exact majority, if two nodes are in states that map to the same outputs according to γ and these nodes interact, they either swap their states or both remain in their states. Formally, $[S_0, S_0] \to \{[S_0, S_0]\}$, $[S_0, X] \to \{[S_0, X]\}$, $[X, X] \to \{[X, X]\}$, $[S_1, S_1] \to \{[S_1, S_1]\}$, $[S_1, Y] \to \{[S_1, Y]\}$ and $[Y, Y] \to \{[Y, Y]\}$.*

Proof. Let us focus on states S_0 and X, for which γ is the same and equal to 0. This is without the loss of generality, because the case for S_1 and Y is completely symmetric and works analogously.

By definition of C_0 in the first correctness property, no node can get into a state S_1 or Y after any interaction from a configuration in C_0, as γ maps these states to an incorrect value 1. In Claim B.4, we established existence of a mixed absorbing configuration C_0 with at least two nodes in each of the states S_0 and X. In this configuration all three interactions $[S_0, S_0], [S_0, X]$ and $[X, X]$ are possible. Thus, when two nodes both in states S_0 or X interact, both nodes must remain in either S_0 or X.

We can now prove $[S_0, X] \to \{[S_0, X]\}$ by contradiction, showing that after such interaction (node in S_0 meeting a node in X) in any correct algorithm, both nodes may not end up in state S_0, and both may not end up in X. Consider two starting configurations for $n = 5$ nodes. In configuration O one node starts in state S_1 and four nodes in S_0 and in configuration T, two nodes start in S_1 and three in S_0. In both cases majority is S_0, so by the third property it is possible to reach some absorbing configuration in C_0. Let $c(O) \in C_o$ be one such configuration in C_0 reachable from O, and let $c(T) \in C_0$ be the configuration reachable from T. By Corollary B.3, a configuration with all nodes in S_0 cannot be reached from neither O nor T, thus both $c(O)$ and $c(T)$ contain at least one X. Now, if there was an algorithm in which $[S_0, X]$ led to $[X, X]$, we could apply this transition repeatedly from $C(O)$ and get all nodes in X state. We could do the same from $C(T)$. Hence, starting from different starting configurations O and T it would be possible to reach the same state of all nodes in X (by first reaching configurations $C(O)$ and $C(T)$, respectively), contradicting Claim B.2. Finally, if there is an algorithm that turns $[S_0, X]$ into $[S_0, S_0]$, neither $c(O)$ nor $c(T)$ may contain even a single node in S_0, as the configuration with all nodes in S_0 would then be reachable by applying the interaction multiple consecutive times, contradicting Corollary B.3. In the only remaining case both $c(O)$ and $c(T)$ have all nodes in state X, immediately contradicting Claim B.2.

Notice that almost the same argument shows that no algorithm can turn $[X, X]$ into $[S_0, S_0]$ or into $[S_0, X]$. Assume contrary. Since it is impossible to reach a configuration with all nodes in S_0 from $C(O)$ or $C(T)$, by repeatedly applying the interaction for $[X, X]$ for as long as possible, we must reach the same configuration with one node in X and four nodes in S_0 from both $C(O)$ and $C(T)$, contradicting Claim B.2 for starting configurations O and T.

To show that no algorithm can turn $[S_0, S_0]$ into $[X, X]$ or $[S_0, X]$, we need a slight modification. We should now consider a system of $n = 7$ nodes, and define O and T similar to before, with one and two nodes starting in S_1 (rest in S_0), respectively. Let H be a configuration where three nodes start in S_1 and four in S_0. Majority is still S_0 in all of these three starting configurations, so we can again define $c(O)$, $c(T)$ and $c(H)$ this time for the system of seven nodes as the configurations in C_0 reachable from O, T and H, respectively. We again assume for contradiction that we have a correct algorithm that turns $[S_0, S_0]$ into $[X, X]$ or $[S_0, X]$. Repeatedly applying the interaction for $[S_0, S_0]$ for as long as possible from any configuration in C_0, we will then reach a configuration with at most one S_0. By the pigeonhole principle, this process leads to the same final configuration for two of the three intermediate configurations $c(O)$, $c(T)$ and $c(H)$. Hence, from two different starting configurations (out of O, T and H), we can, through the respective configurations in C_0, reach the same final configuration (with zero or one nodes in S_0), contradicting Claim B.2. □

The behavior described in Claim B.5 is consistent with known 4-state protocols, and now we know that no other behavior may exist. Let us next consider the case of $[S_0, S_1]$.

Claim B.6. *In any correct (satisfying our three properties) 4-state algorithm for exact majority, if two nodes interact, one of which is in state S_0 and another in S_1 before the interaction, none of these two nodes can be in S_0 or S_1 after the interaction. Formally, $[S_0, S_1] \to \{[X, X], [X, Y], [Y, Y]\}$.*

Proof. We derive contradiction if any algorithm can turn two nodes from $[S_0, S_1]$ into anything other than $[X, X]$, $[X, Y]$, or $[Y, Y]$ after an interaction. Towards this purpose, consider a system of $n = 3$ nodes, where starting configuration O has one node in S_1 and two nodes in S_0, while starting configuration T has two nodes in S_1 and one in S_0. Clearly in O the majority is S_0 and in T it is S_1.

To see that no correct algorithm may lead two nodes from $[S_0, S_1]$ back into $[S_0, S_1]$ after an interaction, observe that for that algorithm, every reachable configuration from O must also have two nodes in S_0 and one in S_1. This is because by Claim B.5 we know $[S_0, S_0] \to \{[S_0, S_0]\}$ and $[S_1, S_1] \to \{[S_1, S_1]\}$ for the other possible interactions in the system. Since $\gamma(S_1) = 1$ the system can never reach an absorbing configuration in C_0 from the starting configuration O with initial majority S_0, and thus, the algorithm does not satisfy the third correctness property.

If an algorithm leads from $[S_0, S_1]$ to $[S_0, S_0]$, then from the starting configuration O, we trivially reach the configuration with all nodes in S_0 by using this interaction only once, contradicting Corollary B.3. Analogously, if an algorithm leads from $[S_0, S_1]$ to $[S_1, S_1]$, then it is possible to reach a configuration with all nodes in S_1 from O, again a contradiction with Corollary B.3.

Next, consider an algorithm that turns two nodes in $[S_0, S_1]$ into $[S_1, Y]$ after an interaction. Then, it is possible to start in configuration O and with only two interactions reach a configuration with two nodes in Y and one node in S_1. However, we know from Claim B.5 that from that point on, in any correct algorithm, the nodes will remain in states Y and S_1. Since $\gamma(Y) = \gamma(S_1) = 1$, we have actually reached an absorbing configuration in C_1 from the starting configuration O with majority S_0, contradicting the second correctness property. The case of an algorithm that leads from $[S_0, S_1]$ to $[S_0, X]$ is symmetric, we just consider the starting configuration T with majority S_1 from which we similarly reach a configuration in C_0 (with two nodes in X and one in S_0).

Finally, consider an algorithm that turns nodes from $[S_0, S_1]$ into $[S_0, Y]$ (the case when the states after interaction are $[S_1, X]$ is symmetric and works analogously). Consider $n = 4$ nodes and a configuration Z with two nodes in S_0 and two nodes in S_1, and a configuration W with one node in S_0 and three nodes in S_1. If we focus only on three nodes out of four in configuration Z, two of them in S_1 and one in S_0, and treat this as a starting configuration for 3 nodes, then by a sequence of interactions only among these nodes it is possible to reach a 3-node absorbing configuration in C_1 for the given 3 nodes. This is because the fourth node does not participate in any interactions, and can be safely disregarded by treating the selected 3 nodes as a system of 3 nodes with majority S_1. So, from Z we reach a configuration with three nodes each in S_1 or Y, and fourth node in S_0 as in the beginning. Let $y \leq 3$ be the number of nodes in state Y in this configuration that we reached from Z. However, this configuration is also reachable starting from W, simply by interacting a node in S_0 one-by-one with y nodes in S_1: one node will remain in S_0, y nodes will turn Y, and $3 - y$ nodes will be in S_1 as desired. The fact that with some sequence of interactions it is possible to reach the same configuration from both Z and W contradicts with Claim B.2[1]. □

Next, we do a case analysis to find all potentially correct algorithms, whose convergence we will then analyze. As we will see, these potential algorithms will fall into two categories and we will use the following two results to analyze them:

Claim B.8. *Denote by* majority state *the state S_i that was the state of the majority of the nodes in the starting configuration, and let S_{1-i} be the* minority state*. Every algorithm for which the difference between number of nodes in the majority state and the number of nodes in the minority state always stays invariant throughout any execution, has parallel convergence time at least $\Omega(1/\epsilon)$.*

Claim B.9. *Consider assigning four different potentials of -3, -1, 1 and 3 to the four states of the algorithm S_0, S_1, X and Y, such that S_0 and X get positive potentials. Consider any algorithm that has the property that after any interaction, the sum of potentials of the states of two interacting nodes never changes. Such an algorithm violates third property and cannot be correct.*

Let us start our case analysis by universally eliminating some possibilities.

Claim B.7. *No correct algorithm can lead two nodes from $[S_1, X]$ into $[S_0, S_0]$, $[S_0, X]$ or $[X, X]$ after the interaction. Also, no correct algorithm leads from $[S_0, Y]$ to $[S_1, S_1]$, $[S_1, Y]$ or $[Y, Y]$.*

Proof. Assume for contradiction that an algorithm leads from $[S_1, X]$ into $[S_0, S_0]$, $[S_0, X]$ or $[X, X]$. (The other case is symmetric and can be worked out analogously). Consider a system of $n = 5$ nodes and a starting configuration with two nodes in S_0 and three nodes in S_1, and a subset of 3 selected nodes two of which are in S_0. By the third property,

[1] even number of nodes $n = 4$ in configurations Z and W is not a problem here, because the claim uses properties for a system of $2n - 1$ nodes, where the majority is always defined

there exists a sequence of interactions only between the selected nodes such that they each end up either in state S_0 or X. This is because the selected nodes cannot observe a difference from running alone and the majority among them is S_0. Now, as long as we have any nodes in state X, by interacting with the node in S_1, we bring both nodes in either S_0 or X. No node will ever end up in state Y after these interactions. If we can do two such interactions we will reach a configuration with all five nodes each in either S_0 or X, which is an absorbing configuration by Claim B.5 and thus a configuration in C_0, contradicting the second property and that we started with a majority of S_1 in five nodes.

Otherwise, since we had three nodes each in S_0 or X, two nodes in S_1 and could not perform an interaction from $[S_1, X]$ twice, we must have ran out of nodes in X before we ran out of nodes in S_1. However, in that configuration, all nodes must be in S_0 or S_1 with majority S_0. This contradicts Claim B.2 for the system of five nodes. □

In the following, we take for granted our knowledge that all correct algorithms satisfy conditions from Claim B.5 and Claim B.7. We start the case analysis by considering the possibilities from Claim B.6.

Case 1 - $[S_0, S_1]$ **into** $[X, Y]$: Here we consider algorithms that take two nodes from $[S_0, S_1]$ into $[X, Y]$ after the interaction. In this case, the algorithm should take two nodes from $[X, Y]$ to either $[X, X]$, $[X, Y]$, $[Y, Y]$ or $[S_0, S_1]$, because in all other cases the same contradictions as in Claim B.6 works by using two consecutive interactions (instead of one) on two nodes in $[S_0, S_1]$.

Moreover, in this case, no algorithm can take two nodes from $[S_0, Y]$ into $[S_0, S_0]$. Assume contrary and this time consider a system of $n = 5$ nodes with three nodes in S_1 and two nodes in S_9. We start by an interaction $[S_0, S_1]$ followed by $[S_0, Y]$ which leads to a configuration with one node in X, two nodes in S_0 and two nodes in S_1. Essentially, doing this sequence of interactions leads to a configuration with one less node in S_1 and one more node in X. Hence, doing $[S_0, S_1]$ followed by $[S_0, Y]$ two more times leads to a configuration with three nodes in X and two nodes in S_0. This is an absorbing configuration in C_0 and contradicts with the initial majority S_1. Analogously we can show that no algorithm takes two nodes from $[S_1, X]$ into $[S_1, S_1]$.

Case 1.1 - $[X, Y]$ **into** $[X, Y]$: Now we are considering further subset of algorithms, that take nodes from $[X, Y]$ back into $[X, Y]$. Notice that there are only 4 possible interactions between states that map to different outputs according to γ. In our case after two of such interactions, from $[S_0, S_1]$ and from $[X, Y]$, nodes remain in states that map to different outputs, and by Claim B.5, when two nodes that map to the same output meet, they both also map to the same output as before after the interaction. The initial majority can be S_0 or S_1, but the initial configuration may not be absorbing. Thus, for each output 0 and 1, there must be an interaction that increases the number of nodes in states that map to that output. Hence, the interactions from the remaining two interactions $[S_0, Y]$ and $[S_1, X]$ should lead to the both nodes in states that map to the same outputs. Due to Claim B.7, the only possibility is that the algorithm takes $[S_0, Y]$ into $[S_0, X]$ or $[X, X]$ and takes $[S_1, X]$ to $[S_1, Y]$ or $[Y, Y]$ ($[S_0, Y]$ to $[S_0, S_0]$ and $[S_1, X]$ to $[S_1, S_1]$ are impossible as discussed above in **Case 1**).

Let us now demonstrate that no correct algorithm can actually take $[S_0, Y]$ into $[X, X]$. Assume contrary and consider a system of $n = 5$ nodes, three of them starting in S_0 and two in S_1. After two interactions both from $[S_0, S_1]$ we get two nodes in X, two nodes in Y and a node in S_0. Now, we can lead from $[S_0, Y]$ into $[X, X]$ and have four nodes in X and one in Y. The only types of interactions are now from $[X, X]$ which leaves both nodes in X, and from $[X, Y]$ which in the current case also goes back into $[X, Y]$. This means that the system will always have a node whose state maps to 1 and a node whose state maps to 0, and can never reach configuration in C_0 despite initial majority S_0, contradicting the third property. Analogous argument shows that no algorithm takes $[S_1, X]$ into $[Y, Y]$.

To summarize, in this case the only correct algorithm takes $[S_0, Y]$ into $[S_0, X]$, $[S_1, X]$ into $[S_1, Y]$, and both $[S_0, S_1]$ and $[X, Y]$ into $[X, Y]$. Convergence of this algorithm follows by Claim B.8.

Case 1.2 - $[X, Y]$ **into** $[Y, Y]$: Here we consider further subset of algorithms from **Case 1**, that take two nodes from $[X, Y]$ into $[Y, Y]$. Notice that in order to be able to converge to a configuration in C_0 from an initial configuration that has nodes in both S_0 and S_1, but with majority S_0, there must be an interaction that increases the number of nodes in states that map to 0. Because of Claim B.7, in our case only such interaction can be from $[S_0, Y]$. Thus after the interaction both nodes should be either in S_0 or X.

However, no algorithm can take two nodes from $[S_0, Y]$ into $[S_0, S_0]$ because of the discussion in the beginning of **Case 1**. Now consider an algorithm takes $[S_0, Y]$ into $[X, X]$ and the same system of $n = 5$ nodes as in **Case 1.1**, with the initial majority of S_0 from where we can reach a state with four nodes in X and one node in Y. However, we can now use the interaction from $[X, Y]$ to $[Y, Y]$ four times and end up with all five nodes in state Y. This is an absorbing configuration in C_1 despite the initial majority S_0 contradicting the second property. Therefore, any correct algorithm must take two nodes from $[S_0, Y]$ into $[S_0, X]$ as in the previous case.

Let us prove that no correct algorithm takes $[S_1, X]$ into $[S_0, S_1]$ or $[S_0, Y]$. Assume contrary and consider a system of $n = 5$ nodes with three nodes in S_1 and two nodes in S_0. After two separate interactions from $[S_0, S_1]$ into $[X, Y]$ we reach a configuration where two nodes are in X, two nodes in Y and one node in S_1. If an interaction from $[S_1, X]$ takes nodes into $[S_0, Y]$, we can immediately reach a configuration with one node in S_0, one node in X and three nodes in Y. If the interaction from $[S_1, X]$ leads to $[S_0, S_1]$, we can use this interaction twice, leading to two nodes (that were in X) in S_0, one node in S_1 and two nodes in Y. Then, using the interaction from $[S_0, S_1]$ to $[X, Y]$ we can again reach the same configuration with one node in S_0, one node in X and three nodes in Y. However, from here we can use the interaction from $[S_0, Y]$ three times (for each node in Y) and lead to a configuration with one node in S_0 and four nodes in X. This is an absorbing configuration in C_0 and contradicts with initial majority being S_1.

Next, let us prove that no correct algorithm takes two nodes from $[S_1, X]$ into $[X, Y]$ or $[Y, Y]$. Assume contrary and note that in the first case it is also possible to reach $[Y, Y]$ from $[S_1, X]$ albeit using two consecutive interactions between the same nodes (first into $[X, Y]$ and then into $[Y, Y]$). Now consider the same system as before of $n = 5$ nodes with three nodes in S_1 and two nodes in S_0, i.e. majority S_1. First, let one node in S_0 and one node in S_1 interact, turning to states X and Y. Then we lead $[S_1, X]$ into $[Y, Y]$ at which point we have one node in S_0, three nodes in Y and one node in S_1. Now we can use the interaction from $[S_0, Y]$ three times and turn all nodes in Y into X. Then we again lead from $[S_1, X]$ into $[Y, Y]$ and again by $[S_0, Y]$ turn the two nodes in Y into X. The final configuration we reached contains one node in S_0 and four nodes in X, so it is an absorbing configuration in C_0 contradicting the initial majority of S_1.

Finally, by Claim B.7, no correct algorithm takes two nodes from $[S_1, X]$ to states that both map to 0 by γ, and as discussed in **Case 1**, it is also impossible to lead from $[S_1, X]$ into $[S_1, S_1]$

Hence, in this case the only correct algorithm takes $[S_0, Y]$ into $[S_0, X]$, $[S_1, X]$ into $[S_1, Y]$ or back into $[S_1, X]$, $[S_0, S_1]$ into $[X, Y]$ and $[X, Y]$ into $[Y, Y]$. Convergence bound follows by Claim B.8.

Case 1.3 - $[X, Y]$ **into** $[X, X]$: This case is completely analogous to the previous **Case 1.2** and the resulting possible state transitions for correct algorithms are also completely symmetric and can be analyzed exactly the same way.

Case 1.4 - $[X, Y]$ **into** $[S_0, S_1]$: Here we consider algorithms that take two nodes from $[S_0, S_1]$ into $[X, Y]$ and two nodes from $[X, Y]$ to $[S_0, S_1]$. Exactly as in **Case 1.1**, we get that any algorithm must take $[S_0, Y]$ into $[S_0, X]$ or $[X, X]$ and take $[S_1, X]$ to $[S_1, Y]$ or $[Y, Y]$. There are four cases

Case 1.4.1 - $[S_0, Y]$ **into** $[S_0, X]$ **and** $[S_1, X]$ **into** $[S_1, Y]$: This algorithm satisfies the invariant that the difference between the number of nodes in the majority state and minority state (nodes in states S_i and S_{1-i}) stays invariant, and hence convergence lower bound follows from Claim B.8.

Case 1.4.2 - $[S_0, Y]$ **into** $[X, X]$ **and** $[S_1, X]$ **into** $[S_1, Y]$: Let us prove that such an algorithm is not correct. Consider a system of $n = 7$ nodes with four nodes starting in S_0 and three nodes in S_1. It is possible to reach a configuration with six nodes in state X and one in S_1 after the following sequence of interactions: twice from $[S_0, S_1]$, which creates two nodes in X and two nodes in Y, and then twice $[S_0, Y]$, which leads to the configuration described above. However, from this configuration with six nodes in X and one in S_1 we can use the interaction from $[S_1, X]$ six times and reach an absorbing configuration in C_1 with six nodes in Y and one node in S_1 despite starting with majority S_0. This contradiction with the second property completes the proof.

Case 1.4.3 - $[S_0, Y]$ **into** $[S_0, X]$ **and** $[S_1, X]$ **into** $[Y, Y]$: This case is symmetric to **Case 1.4.2**. To get the contradiction consider the initial configuration of $n = 7$ nodes but with four nodes in S_1 and three in S_0, and reach an absorbing configuration in C_0 contradicting correctness of the algorithm.

Case 1.4.4 - $[S_0, Y]$ **into** $[X, X]$ **and** $[S_1, X]$ **into** $[Y, Y]$: We can assign potential of 3 to S_0, potential 1 to X, -3 to S_1 and -1 to Y. In our case, this assignment satisfies the invariant necessary to apply Claim B.9 and get that the algorithm in this case is not correct.

Case 2 - $[S_0, S_1]$ **into** $[Y, Y]$: Here we consider algorithms that take two nodes from $[S_0, S_1]$ into $[Y, Y]$ after the interaction. Recall that by Claim B.7, no correct algorithm can take two nodes from $[S_0, Y]$ to $[S_1, S_1]$, $[S_1, Y]$ or $[Y, Y]$. Moreover, let us eliminate some more possibilities.

- No correct algorithm can take $[S_0, Y]$ to $[S_0, S_0]$. Assume contrary. Consider a system of $n = 5$ nodes with two nodes starting in state S_0 and three nodes in S_1. After one interaction from $[S_0, S_1]$ into $[Y, Y]$ and then using interaction from $[S_0, Y]$ to $[S_0, S_0]$ twice, we reach a configuration with three nodes in S_0 and two nodes in S_1, contradicting Claim B.2.
- No correct algorithm can take $[S_0, Y]$ to $[S_0, S_1]$. Assume contrary. Consider a system of $n = 3$ nodes with

two nodes starting in state S_0 and one node in S_1. After one interaction from $[S_0, S_1]$ into $[Y, Y]$ and then using interaction from $[S_0, Y]$ to $[S_0, S_1]$ twice, we reach a configuration with one node in S_0 and two nodes in S_1, contradicting Claim B.2.

- No correct algorithm can take $[S_0, Y]$ to $[S_0, Y]$. Assume contrary. Consider a system of $n = 3$ nodes with two nodes starting in state S_0 and one node in S_1. After one interaction from $[S_0, S_1]$ we reach a configuration with one node in S_0 and two nodes in Y. Any interaction from this configuration leaves two nodes in Y and one node in S_0, meaning that since $\gamma(Y) = 1$, no configuration in C_0 is reachable, contradicting the third property.

Below, we will consider the four remaining cases for an interaction from $[S_0, Y]$ one-by-one:

Case 2.1 - $[S_0, Y]$ **into** $[S_1, X]$: Here we consider further subset of algorithms from **Case 2**, that take two nodes from $[S_0, Y]$ into $[S_1, X]$.

Note that for each output, any correct algorithm must have an interaction that increases the number of nodes in states that map to that output. Otherwise, it would be impossible to reach an absorbing state when starting with the respective majority, and the algorithm would violate the third property. Currently, no interactions that we have already considered increase the number of nodes in states that map to output 0. By Claim B.7, an interaction from $[S_1, X]$ will also not lead to two nodes in such states. This implies that an interaction from $[X, Y]$ must lead to both nodes in S_0 or X. Moreover, in the current case, no correct algorithm can lead from $[X, Y]$ into $[S_0, X]$. Assume contrary and consider a configuration of $n = 5$ nodes, with two nodes starting in S_0 and three nodes starting in S_1. Let us use an interaction from $[S_0, S_1]$ followed by an interaction from $[S_0, Y]$ and from $[X, Y]$, which allows us to reach a configuration with one node in X, one node in S_0 and three nodes in S_1. However, now an interaction from $[S_0, S_1]$ immediately followed by two interactions from $[X, Y]$ lead to a configuration with one node in X, two nodes in S_0 and two nodes in S_1. Doing the same sequence two more times leads to a configuration with one node in X and four nodes in S_0, which is an absorbing configuration in C_0 and contradicts with the initial majority S_1 by the second property. So, any correct algorithm must lead to $[S_0, S_0]$ or $[X, X]$ from $[X, Y]$.

Before we consider these two sub-cases, let us again eliminate some possibilities for an interaction from $[S_1, X]$. By Claim B.7, such an interaction may not lead into $[S_0, S_0]$ or $[S_0, X]$ or $[X, X]$. In our case, such an interaction also may not lead into $[S_1, S_1]$ or $[S_1, Y]$ or $[Y, Y]$, because these would also then be reachable by two consecutive interactions from $[S_0, Y]$, and we can get the same contradiction as in Claim B.7 for $[S_0, Y]$. Finally, for a similar reason an interaction from $[S_1, X]$ may not lead to $[S_0, S_1]$, since we can apply two consecutive interactions from $[S_0, Y]$ and get to $[S_0, S_1]$. This was discussed earlier in **Case 2** and exactly the same argument can be reused to get a contradiction. So, an interaction from $[S_1, X]$ can only lead into $[S_1, X]$ or $[S_0, Y]$ or $[X, Y]$.

Now we consider the two possible cases for an interaction from $[X, Y]$.

Case 2.1.1 - $[X, Y]$ **into** $[X, X]$: We are considering algorithms that lead from $[S_0, S_1]$ into $[Y, Y]$, from $[S_0, Y]$ into $[S_1, X]$ and from $[X, Y]$ into $[X, X]$. We have just established in **Case 2.1** that an interaction from $[S_1, X]$ can only lead into $[S_1, X]$ or $[S_0, Y]$ or $[X, Y]$. Let us prove that no correct algorithms fall in the current case by deriving a contradiction for all these options.

Consider a system of $n = 5$ nodes, two starting in state S_0 and three starting in S_1. Notice that the majority is S_1, so no configuration in C_0 must be reachable. From this starting configuration, after an interaction from $[S_0, S_1]$ followed by an interaction from $[S_0, Y]$, we can reach a configuration I with one node in X, one node in Y and three nodes in S_1.

Assume a correct algorithm takes $[S_1, X]$ to $[X, Y]$. Then we can use this interaction three times from I and reach a configuration with one node in X and four nodes in Y. However, from there four successive applications of an interaction from $[X, Y]$ to $[X, X]$ leads to all nodes in state X, which is an absorbing configuration in C_0 and contradicts with the second property.

Now consider another configuration I' reachable from I by an interaction from $[X, Y]$, with two nodes in X and three nodes in S_1. If the algorithm takes $[S_1, X]$ to $[S_1, X]$, then every configuration reachable from I' contains nodes in S_1 and nodes in X that map to different outputs. Since I' is reachable (through I) from the starting configuration with majority S_1, this contradicts the third property, because from I' no configuration in C_1 is reachable.

Finally, consider an algorithm that takes $[S_1, X]$ to $[S_0, Y]$. Starting in configuration I', an application of an interaction from $[S_1, X]$ followed by an interaction from $[Y, X]$ to $[X, X]$ leads to a configuration with two nodes in X, one node in S_0 and two nodes in S_1. Applying these two consecutive interactions two more times leads to two nodes in X and three nodes in S_0. This is again an absorbing configuration in C_0 reachable (though I') from the starting configuration with majority S_1, contradicting the third property.

Case 2.1.2 - $[X, Y]$ **into** $[S_0, S_0]$: Here we consider subset of algorithms from **Case 2.1** that lead two nodes from

$[X, Y]$ into $[S_0, S_0]$. Let us show that under these circumstances, no correct algorithm can take $[S_1, X]$ into $[X, Y]$. This is because we could then take two nodes from $[S_0, Y]$ into $[S_1, X]$ into $[X, Y]$ and into $[S_0, S_0]$, so by applying three consecutive interactions to two nodes in states S_0 and Y, we could bring both nodes to state S_0. However, we can reuse the scenario and argument that we used in the beginning of **Case 2** to eliminate algorithms with interactions that immediately take $[S_0, Y]$ into $[S_0, S_0]$ to get a contradiction.

Let us assign potential 3 to X, 1 to S_0, -3 to S_1 and -1 to Y. Thus, any algorithm must lead from $[S_1, X]$ into $[S_0, Y]$ or back into $[S_1, X]$. In both cases we can apply Claim B.9 to establish that these algorithms are not correct.

Case 2.2 - $[S_0, Y]$ **into** $[X, Y]$: Algorithms that take $[S_0, S_1]$ to $[Y, Y]$ and $[S_0, Y]$ to $[X, Y]$.

Exactly the same argument as in the beginning of **Case 2.1** implies that any correct algorithm must lead from $[X, Y]$ into either $[S_0, X]$ or $[S_0, S_0]$ or $[X, X]$. Moreover, similar scenario shows that taking $[X, Y]$ to $[S_0, X]$ cannot actually be a correct behavior - starting from a system of $n = 5$ nodes with two nodes in S_0 and three in S_1, after interactions from $[S_0, S_1]$ followed by $[S_0, Y]$ we reach a state with one node in X, two in Y and two in S_1. But now since interaction from $[X, Y]$ leads to $[X, S_0]$ we can do this twice, then use $[S_0, S_1]$ to $[Y, Y]$ twice and then finally, the interaction from $[X, Y]$ to $[X, S_0]$ four times. The end result is one node in X and four nodes in S_0, an absorbing configuration in C_0 contradicting the initial majority of S_1 by the second property.

Additionally, no correct algorithm takes $[X, Y]$ to $[S_0, S_0]$. This is because then we can use two consecutive interactions and lead two nodes from $[S_0, Y]$ first to $[X, Y]$ and then to $[S_0, S_0]$. However, reaching $[S_0, S_0]$ from $[S_0, Y]$ after a few interactions leads to exactly the same contradiction as if this was the immediate interaction behavior from $[S_0, Y]$, which was discussed in **Case 2**. So, we know that any correct algorithm must lead to $[X, X]$ from $[X, Y]$.

Let us now consider possible interactions from $[S_1, X]$. By Claim B.7, it may not lead to $[S_0, S_0]$ or $[S_0, X]$ or $[X, X]$. Also, if an interaction from $[S_1, X]$ leads to $[X, Y]$, a second consecutive interaction from $[X, Y]$ will lead to $[X, X]$. If an interaction from $[S_1, X]$ leads to $[S_0, Y]$, then another interaction from there leads to $[X, Y]$ and then again, from $[X, Y]$ to $[X, X]$. So, in these cases, it is possible to take $[S_1, X]$ to $[X, X]$ albeit not directly but rather with a few consecutive interactions between the same nodes. However, the same counterexample from Claim B.7 as for direct interaction applies.

Next, assume a correct algorithm that takes $[S_1, X]$ to $[S_1, S_1]$ or $[S_1, Y]$. Consider a system of $n = 5$ nodes with three nodes starting in S_0 and two nodes in S_1. After an interaction from $[S_0, S_1]$ to $[Y, Y]$, we get two nodes in Y and two nodes in S_0, so we can use an interaction from $[S_0, Y]$ to $[X, Y]$ twice. This leads to a configuration with two nodes in X, two nodes in Y and one node remaining in S_1. However, applying an interaction from $[S_1, X]$, we get one node in S_1 and another in S_1 or Y, and it will be possible to use $[S_1, X]$ again, after which all nodes will be in S_1 or Y. This is an absorbing configuration in C_1, contradicting original majority S_0 and the second property.

Finally, consider a system of $n = 5$ nodes with two nodes starting in S_0 and three nodes starting in S_1. From here using an interaction from $[S_0, S_1]$, then an interaction from $[S_0, Y]$ and then an interaction from $[X, Y]$ twice, we reach a configuration I with three nodes in X and two nodes in S_1. A correct algorithm may not lead from $[S_1, X]$ to $[S_1, X]$ because configuration I would then never converge to any output and violate the third property. If an algorithm leads from $[S_1, X]$ to $[S_0, S_1]$, then applying this interaction from I three times leads to a configuration with three nodes in S_0 and two nodes in S_1, contradicting Claim B.2. The last possibility is that an algorithm leads from $[S_1, X]$ to $[Y, Y]$. In this case, consider applying this interaction twice from configuration I, which will lead to a configuration with one node in X and four nodes in Y. However, from here applying an interaction that takes $[X, Y]$ to $[X, X]$ four times leads to an absorbing configuration from C_0 with all nodes in X, and this configuration was reached from starting configuration with majority S_1, contradicting the second property.

Case 2.3 - $[S_0, Y]$ **into** $[X, X]$: Algorithms that take $[S_0, S_1]$ to $[Y, Y]$ and $[S_0, Y]$ to $[X, X]$.

Let D be a configuration of four nodes, where three nodes are in S_0 and one node in S_1. From D, we can use an interaction that takes $[S_0, S_1]$ to $[Y, Y]$ followed by two interactions from $[S_0, Y]$, which will let us reach a configuration with all nodes in X. Notice that even if there are more than four nodes in the system, for any four nodes that comply to D (i.e. three in S_0 and one in S_1), this shows we can to turn them all to be in state X.

Let us now consider an interaction of a correct algorithm from $[S_1, X]$. By Claim B.7, it may not lead to $[S_0, S_0]$ or $[S_0, X]$ or $[X, X]$. As we have also seen before, the same contradiction applies if we can reach $[X, X]$ from $[S_1, X]$ with multiple consecutive interactions, thus an interaction from $[S_1, X]$ may not lead to $[S_0, Y]$ either.

Consider a system of $n = 5$ nodes with three nodes in S_0 and two nodes in S_1. If we choose four nodes that comply with configuration D (all nodes except one in S_1), we can turn them all to X after a sequence of interactions. In this fashion, from the starting configuration we reach a configuration I with four nodes in X and one node in S_1. If an algorithm takes $[S_1, X]$ to $[S_1, X]$ then from configuration I it is impossible to converge to an output, violating the

third property (we had initial majority S_0, but cannot reach C_0). If an algorithm takes $[S_1, X]$ to $[S_1, S_1]$ or $[S_1, Y]$, then applying this interaction four times from I leads to each node in S_1 or Y, i.e. an incorrect absorbing configuration (from C_1), violating the second property.

Next let us prove that no algorithm can take $[S_1, X]$ to $[S_1, S_0]$. Consider a system of $n = 7$ nodes with three nodes starting in C_0 and four nodes in S_1. From here, using the interactions on D, we can again turn three nodes in C_0 and a node in S_1 all into state X. However, from here, we can apply the interaction from $[S_1, X]$ four times, which will lead to a configuration with four nodes in S_0 and three nodes in S_1, contradicting Claim B.2.

Therefore, in any correct algorithm an interaction from $[S_1, X]$ leads to either $[Y, Y]$ or $[X, Y]$. We now focus on these two sub-cases.

Case 2.3.1 - $[S_1, X]$ **into** $[Y, Y]$: Algorithms from **Case 2.3** that also take $[S_1, X]$ into $[Y, Y]$.

Consider a system of $n = 3$ nodes with two nodes in S_0 and one node in S_1. After an interaction from $[S_0, S_1]$ followed by an interaction from $[S_0, Y]$ we reach an intermediate configuration with two nodes in X and one node in Y. Therefore, no correct algorithm can lead from $[X, Y]$ to $[X, Y]$, as it would not be possible to converge to an output from the intermediate configuration, contradicting the third property. Moreover, no correct algorithm can take $[X, Y]$ into $[S_1, S_1]$, $[S_1, Y]$ or $[Y, Y]$, since after one application of this interaction we get one node in X and two nodes each in S_1 or Y. But after interactions from $[S_1, X]$ and $[Y, X]$ we end up with both interacting nodes each in S_1 or Y, thus all three nodes will be in S_1 or Y, an absorbing configuration from C_1, violating second property. This also means that no algorithm can take $[X, Y]$ to $[S_1, X]$ or $[S_0, S_1]$, since then two consecutive interactions on nodes from $[X, Y]$ leads to $[Y, Y]$ and the same contradiction as above for an interaction from $[X, Y]$ into $[Y, Y]$ applies.

In the last example in **Case 2.3** was a system with $n = 7$ nodes with three nodes starting in S_0 and four in S_1. We saw that that we can reach a configuration with four nodes in X and three nodes in S_1, from where we could do an interaction from $[S_1, X]$ for three independent pairs of nodes. In the current case, this leads to a configuration with one node in X and six nodes in Y. If an algorithm leads from $[X, Y]$ into $[S_0, S_0]$ or $[S_0, X]$ or $[X, X]$, we can repeatedly apply interactions from $[X, Y]$ or $[S_0, Y]$ and after six interactions reach an absorbing configuration from C_0 while initial majority was S_1, contradicting the second correctness property. Finally, if an algorithm leads from $[X, Y]$ to $[S_0, Y]$, we do as above, except apply two consecutive interactions from $[X, Y]$ to get to $[X, X]$ and continue until all nodes are in X and configuration is in C_0.

Case 2.3.2 - $[S_1, X]$ **into** $[X, Y]$: Algorithms from **Case 2.3** that also take $[S_1, X]$ into $[X, Y]$.

In the beginning of our analysis of **Case 2.3** we eliminated multiple possibilities for an interaction from $[S_1, X]$. In particular, if an algorithm takes $[X, Y]$ to $[S_0, S_0]$, $[S_0, X]$, $[X, X]$, $[S_1, S_1]$, $[S_1, Y]$, $[S_1, S_0]$ or $[S_0, Y]$, then clearly it is also possible to take two nodes there from $[S_1, X]$ after two consecutive interactions (first turning them into $[X, Y]$). It is easy to check that the same counterexamples that established that no correct algorithm could take nodes there from $[S_1, X]$, applies even if we replaced the interaction with two consecutive interactions. Therefore, no correct algorithm can turn $[X, Y]$ into any of the above described combinations.

In **Case 2.3.1** we have seen how to reach a configuration with two nodes in X and one node in Y from a starting configuration for $n = 3$ of two nodes in S_0 and one node in S_1. It is impossible to converge to any output from this configuration if the algorithm takes $[X, Y]$ back into $[X, Y]$ or into $[S_1, X]$ (since $[S_1, X]$ again leads to $[Y, X]$ and in any case, we are stuck with nodes in S_1 or Y and node in X, mapping to different outputs).

Finally, let us show that no correct algorithm can lead two nodes from $[X, Y]$ into $[Y, Y]$. Assume contrary and consider a system with $n = 5$ nodes with three nodes starting in S_0 and two nodes in S_1. We have seen in **Case 2.3** that from this starting configuration a configuration with four nodes in X and one node in S_1 can be reached. From there, an interaction from $[S_1, X]$ followed by four interactions from $[X, Y]$ leads to all nodes in Y, an absorbing configuration in C_1 contradicting the second property.

Case 2.4 - $[S_0, Y]$ **into** $[S_0, X]$: Algorithms that take $[S_0, S_1]$ to $[Y, Y]$ and $[S_0, Y]$ to $[S_0, X]$.

A correct algorithm from may not lead from $[S_1, X]$ to $[S_0, S_0]$ or $[S_0, X]$ or $[X, X]$ because of Claim B.7. Like in **Case 2.3**, neither can a correct algorithm lead from $[S_1, X]$ to $[S_0, Y]$, because this allows getting to $[S_0, X]$ after two consecutive interactions.

Let us prove that no correct algorithm can lead two nodes from $[S_1, X]$ into $[S_1, S_1]$. Assume contrary and consider a system of $n = 5$ nodes, three starting in S_0 and two in S_1. After an interaction from $[S_0, S_1]$ followed by two interactions from $[S_0, Y]$ we reach a configuration with two nodes in S_0, two nodes in X and one node in S_1. However, after two interactions from $[S_1, X]$ we reach a configuration with three nodes in S_1 and two nodes in S_0, contradicting Claim B.2.

Next, no correct algorithm can lead from $[S_1, X]$ into $[S_0, S_1]$. Assume contrary and consider a system of $n = 5$

nodes, this time two starting in S_0 and three in S_1. From this system through the same interactions as above from $[S_0, S_1]$ and twice from $[S_0, Y]$ we reach a configuration with one node in S_0, two nodes in X and two nodes in S_1. However, after two interactions from $[S_1, X]$ we reach a configuration with three nodes in S_0 and two nodes in S_1, contradicting Claim B.2.

We will now prove that no correct algorithm can lead from $[S_1, X]$ into $[Y, Y]$ by showing that the same configuration r with six nodes in Y is reachable from two different starting configurations of six nodes, Z and W, contradicting Claim B.2. In configuration W, two nodes start in S_0 and four nodes start in S_1. A sequence of interactions leading to configuration r is as follows: first from $[S_0, S_1]$ into $[Y, Y]$, then twice from $[S_0, Y]$ into $[S_0, X]$, (at this point we have one node in S_0, two nodes in X, and three nodes in S_1) followed by an interaction from $[S_0, S_1]$ into $[Y, Y]$ and two interactions from $[S_1, X]$ into $[Y, Y]$. In configuration Z, three nodes start in S_0 and three nodes start in S_1. Reaching r is trivial simply by three independent interactions from $[S_0, S_1]$ into $[Y, Y]$.

We will now use the same contradiction approach again to show that no correct algorithm can take $[S_1, X]$ into $[X, Y]$: but this time the configuration r' reachable from both W and Z will contain one node in X and five nodes in Y. A sequence of interactions leading from configuration W to r' is as follows: first from $[S_0, S_1]$ into $[Y, Y]$, and then from $[S_0, Y]$ into $[S_0, X]$, (at this point we have one node in S_0, one node in X, one node in Y and three nodes in S_1) followed by an interaction from $[S_0, S_1]$ into $[Y, Y]$ and two interactions from $[S_1, X]$ into $[X, Y]$. A sequence of interactions leading from configuration Z to r' is as follows: twice from $[S_0, S_1]$ into $[Y, Y]$, and then from $[S_0, Y]$ into $[S_0, X]$, (at this point we have one node in S_0, one node in X, three nodes in Y and one node in S_1) followed by a single interaction from $[S_0, S_1]$ into $[Y, Y]$. Since r' is reachable from two different starting configurations Z and W we get the desired contradiction with Claim B.2.

Thus, a correct algorithm in this case must take $[S_1, X]$ back into $[S_1, X]$ or into $[S_1, Y]$. Let us consider these two possibilities next.

Case 2.4.1 - $[S_1, X]$ **into** $[S_1, X]$: Algorithms from **Case 2.4** that also take $[S_1, X]$ into $[S_1, X]$.

Consider a system of $n = 5$ nodes, two nodes starting in S_0 and three in S_1. From this initial configuration, after an interaction from $[S_0, S_1]$ into $[Y, Y]$, then two interaction from $[S_0, Y]$ into $[S_0, X]$, and finally, again an interaction from $[S_0, S_1]$ leads to an intermediate configuration I with two nodes in X, two nodes in Y and one node in S_1.

No correct algorithm can lead two nodes from $[X, Y]$ into $[S_1, X]$, because this interaction from I would lead to a configuration with two nodes in X and three nodes in S_1, from where it is impossible to converge to an output despite the initial majority S_1, contradicting the third property.

No correct algorithm can lead from $[X, Y]$ into $[S_0, S_0]$, because two such interactions from I would lead to a configuration with four nodes in S_0 and one node in S_1, contradicting Claim B.2.

It is also impossible to lead from $[X, Y]$ into $[S_0, X]$, because after two such interactions from I we reach a configuration with two nodes in X, two nodes in S_0, and a node in S_1. Now, an interaction from $[S_0, S_1]$ followed by two interactions from $[X, Y]$ leads to two nodes in X and three in S_0, an absorbing configuration in C_0, contradicting the second property as initial majority was S_1.

Finally, no algorithm can turn $[X, Y]$ into $[S_0, Y]$. Again, consider two such interactions from I, leading to a configuration with two nodes in Y, two nodes in S_0 and a node in S_1. An interaction from $[S_0, S_1]$ leads to four nodes in Y and one node in S_0, and four interactions from $[S_0, Y]$ into $[S_0, X]$ allows reaching an absorbing configuration in C_0, again contradicting the third property.

Now consider a system of $n = 5$ nodes, but three nodes starting in S_0 and two nodes in S_1. Analogously to before, from this starting configuration, after an interaction from $[S_0, S_1]$ followed by two interactions from $[S_0, Y]$ and another interaction from $[S_0, S_1]$ we can reach an intermediate configuration I' with two nodes in X, two nodes in Y and one node in S_0.

This allows us to show that no correct algorithm can lead two nodes from $[X, Y]$ into $[S_1, S_1]$ or $[S_1, Y]$. To see this, consider two applications of an interaction from $[X, Y]$ in I'. If it leads to $[S_1, S_1]$, we reach a configuration with four nodes in S_1 and one node in S_0, contradicting Claim B.2. If it leads to $[S_1, Y]$, we reach a configuration with two nodes in Y, two nodes in S_1 and one node in S_0. From there, one interaction from $[S_0, S_1]$ four nodes in Y and one node in S_1, an absorbing configuration in C_1, contradicting the second property as the initial majority was S_0.

Hence, a correct algorithm must lead two nodes from $[X, Y]$ into $[X, Y]$, $[X, X]$, $[Y, Y]$ or $[S_0, S_1]$. All these algorithms satisfy the invariant that lets us analyze their convergence by Claim B.8.

Case 2.4.2 - $[S_1, X]$ **into** $[S_1, Y]$: Algorithms from **Case 2.4** that also take $[S_1, X]$ into $[S_1, Y]$.

Exactly as in **Case 2.4.1**, from a starting configuration of $n = 5$ nodes with two nodes in S_0 and three nodes in S_1 we can reach configuration I, and from a starting configuration with three nodes in S_0 and two nodes in S_1 we

can reach configuration I'. This is because in reaching these configurations we have never used an interaction from $[S_1, X]$ that is the only difference between the cases. Configuration I has two nodes in X, two nodes in Y and a node in S_1, while I' has two nodes in X, two in Y and one node in S_0.

Moreover, the same arguments from **Case 2.4.1** show that no correct algorithm can take $[X, Y]$ into $[S_0, S_0]$ or $[S_1, S_1]$, because those counterexamples also never involved interaction from $[S_1, X]$.

Consider any correct algorithm that leads from $[X, Y]$ into $[S_0, X]$ or $[S_0, Y]$. If the algorithm leads to $[S_0, Y]$, then we can also lead from $[X, Y]$ into $[S_0, X]$ albeit after two consecutive interactions (first into $[S_0, Y]$, then from $[S_0, Y]$ into $[S_0, X]$). If we do so twice from $[X, Y]$ in configuration I we reach a configuration with two nodes in S_0, two nodes in X and one node in S_1. Now, an interaction from $[S_0, S_1]$ leads to one node in S_0, two nodes in X and two nodes in Y. But, again using the fact that either one or two consecutive interactions from $[X, Y]$ leads into $[S_0, X]$, and doing so twice, leads to three nodes in S_0 and two nodes in X, absorbing configuration in C_0, contradicting majority of S_1 in the starting configuration from where we reached I and the second property.

Since $[S_1, X]$ leads into $[S_1, Y]$, we can use a similar argument to above to prove that no correct algorithm leads from $[X, Y]$ into $[S_1, Y]$ or $[S_1, X]$. First we establish that either one or two interactions leads from $[X, Y]$ into $[S_1, Y]$, then we start at I', and reach $[S_1, Y]$ twice from $[X, Y]$. This way, we reach a configuration with one node in S_0, two nodes in S_1 and two nodes in Y. However, here only one interaction from $[S_0, S_1]$ is sufficient to end up with four nodes in Y and one node in S_1. This is an absorbing configuration in C_1 contradicting the majority of S_0 in the starting configuration (from where I' was reached) and the second property.

We again have that a correct algorithm must lead from $[X, Y]$ into $[X, Y]$, $[X, X]$, $[Y, Y]$ or $[S_0, S_1]$, and as in **Case 2.4.1** get the desired bounds for these algorithms using Claim B.8.

Case 3 - $[S_0, S_1]$ **into** $[X, X]$: This case is symmetric to **Case 2**, and the whole analysis from there applies simply by treating S_0 as S_1 and X as Y and vice versa. □

Finally, we give the omitted proofs for convergence lower bounds for certain types of algorithms.

Claim B.8. *Denote by* majority state *the state S_i that was the state of the majority of the nodes in the starting configuration, and let S_{1-i} be the* minority state*. Every algorithm for which the difference between number of nodes in the majority state and the number of nodes in the minority state always stays invariant throughout any execution, has parallel convergence time at least $\Omega(1/\epsilon)$.*

Proof. The algorithm may not converge until no node has state S_{1-i} because γ maps this state to the incorrect output. Because of the invariant, when two nodes interact and one of them is in state S_{1-i}, if the other node is not in state S_i, one of the nodes will necessarily be in S_{1-i} after the interaction. Let us measure the expected number of *clutch* rounds at the beginning of which there is a single node in S_{1-i}. As discussed above, in order to converge (to a configuration with no nodes in S_{1-i}), there must be at least one such round in every execution. The expected number of clutch rounds will be sufficient to establish the claim.

Initially, there were ϵn more nodes in S_i than in S_{1-i}. Hence, due to the invariant, in every clutch round, the only way to decrease the number of nodes in S_{1-i} to 0, is to select the only node in S_{1-i} for interaction with one of the $\epsilon n + 1$ nodes in S_i. Probability of this happening is $\frac{\epsilon n+1}{n^2}$, and otherwise, at least one node remains in S_{1-i} and for the algorithm to converge there will be another clutch round. Thus, the expected number of clutch rounds before eliminating the last S_{1-i} in the system is $\Omega(n/\epsilon)$, meaning at least $\Omega(1/\epsilon)$ parallel convergence time of the algorithm. □

Claim B.9. *Consider assigning four different potentials of -3, -1, 1 and 3 to the four states of the algorithm S_0, S_1, X and Y, such that S_0 and X get positive potentials. Consider any algorithm that has the property that after any interaction, the sum of potentials of the states of two interacting nodes never changes. Such an algorithm violates third property and cannot be correct.*

Proof. The invariant implies that in every reachable configuration, the sum of potentials of the states of all nodes will be the same as it was for the starting configuration. Assume without the loss of generality that the potential of S_0 plus the potential of S_1 (which is negative) is at most 0. (otherwise, the symmetric argument works). Let us consider a system of $n = 5$ nodes, three in state S_0 and two in state S_1. Then, the sum of potentials of the states of nodes in every reachable configuration can be at most 3, meaning that all nodes can never be in states S_0 and X. Thus, the state of some node maps to the wrong output 1 in every reachable configuration which means that no configuration in C_0 is reachable. The algorithm violates the third property and cannot be correct. □

C Lower Bound for Arbitrary Number of States

Theorem C.1. *Any algorithm $\mathcal{A}$ solving exact majority takes expected $\Omega(\log n)$ parallel time until convergence under a worst-case input.*

Proof. Let us consider an arbitrary algorithm $\mathcal{A}$, solving the majority problem in a system of n nodes connected as a clique. For simplicity, we give nodes an arbitrary labeling from 1 to n, and assume that n is even. We measure time in rounds, where a round consists of a new interaction between two nodes. Define the set T to contain the first three nodes in the labeling, and we denote by K_t the set of nodes which may "know" the initial value of a node in T at time t.

Formally, K_t is defined inductively as follows. Initially, $K_0 = T$. In new round $t \geq 1$, we consider the pair of nodes (i, j) which have just interacted. If only one of the interacting nodes is in K_{t-1}, but *not both*, then $K_t = K_{t-1} \cup \{i, j\}$. Otherwise, the set stays unchanged, i.e., $K_t = K_{t-1}$.

Our plan for the rest of the proof will be to show two claims. First, we show that $|K_t| < n$, w.h.p., for $t < \alpha n \log n$, where $\alpha < 1$ is a constant. Since the schedule is independent on the initial assignment of states, this claim holds irrespective of the initial state. Further, we notice that we can build input assignments for which the nodes in T *decide* the majority. In other words, a node should not be able to set on its final value without having some causal relation with nodes in T, as it may otherwise settle on the wrong value. This will imply the lower bound.

We now prove the first claim.

Claim C.2. *For every initial state, there exists a constant $\alpha < 1$ such that for $t < \alpha n \log n$, $|K_t| < n$, with probability at least $1 - O(1/\log^2 n)$.*

Proof. For integer $n \geq i \geq 4$, we define random variable X_i to count the time elapsed until $|K_t| = i$, from the first time t_0 when $|K_{t_0}| = i - 1$. We wish to characterize the random variable $Y = \sum_{3 \leq i \leq n} X_i$. To characterize X_i, we notice that the probability that we add a new element to the set K_t, given that we have exactly $i - 1$ elements, is $p_i = (i-1)(n-i+1)/(n(n-1))$. Therefore, $\mathsf{E}[X_i] = 1/p_i$. Also, $Var[X_i] = 1/p_i^2 - 1/p_i$.

Since $p_i \leq (i-1)/n$, it follows that there exists a constant $\beta < 1$ such that $\mathsf{E}[Y] \geq \beta n \log n$. Similarly, there exists a constant $\beta' < 1$ such that $Var[Y] \leq \beta' n^2$. Fixing a small constant $\alpha < 1$, we can use Chebyshev's inequality to get that

$$\Pr[Y \leq \alpha n \log n] \leq \frac{C}{\log^2 n},$$

where $C \geq 1$ is a constant.

Hence, the probability that $|K_t| = n$ for $t < \alpha n \log n$ is $O(1/\log^2 n)$, as claimed. □

We now move to the second claim. We consider the following setup: nodes in T have initial values either all A or all B, with probability $1/2$ each. The next $n/2 - 2$ nodes in the labeling are in initial state A, and the remaining $n/2 - 1$ nodes are in initial state B.

By Claim C.2, given an arbitrary initial assignment, the probability that there exists a node v with no causal relation to nodes in T by round $\alpha n \log n$ is at least $1 - O(1/\log^2 n)$, where the probability is only taken over the scheduler choices. Let us consider the possible states of node v at round $\alpha n \log n$. If the node is not in a state which can be mapped to an output value, then the algorithm has not yet converged, and we are done. The remaining case is when the node is in a state which can be mapped to an output value. Then, notice that the probability that the node has the correct output value is at most $1/2$, since either output is the majority with probability $1/2$ (taken over our initial choice of output), and node v's output choice is independent of the initial value of nodes in T. Therefore, with probability at least $1/2$, node v will still have to change its output value due to algorithm correctness. Hence, the algorithm has not yet converged at round $\alpha n \log n$, with probability $> 1/2$.

Therefore, with probability at least $(1 - O(1/\log^2 n))/2$, the parallel running time of the algorithm is $\Omega(\log n)$, which completes the proof. □

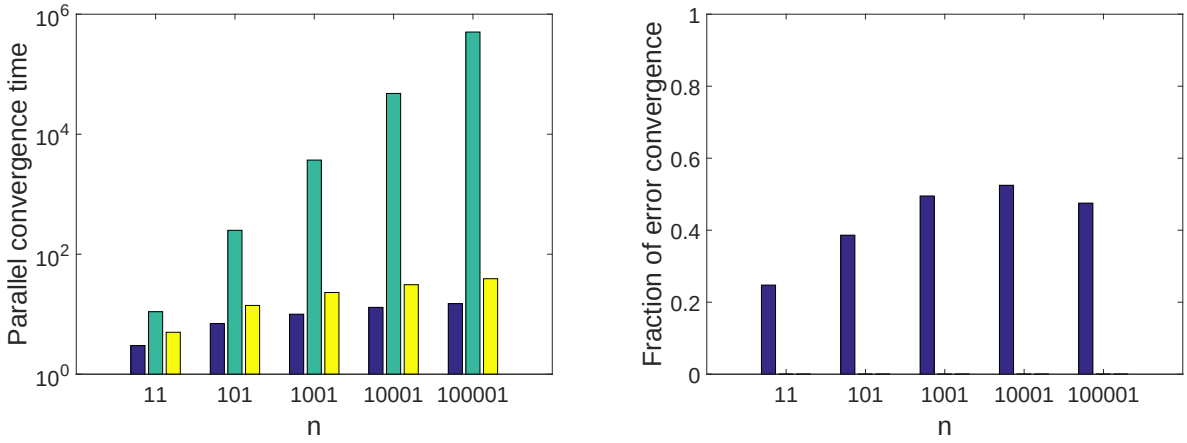

Figure 3: Comparison of performance of different protocols (left bar) 3-state protocol, (middle bar) 4-state protocol, and (right bar) n-state protocol. (Left) convergence time and (right) fraction of runs to error final state.

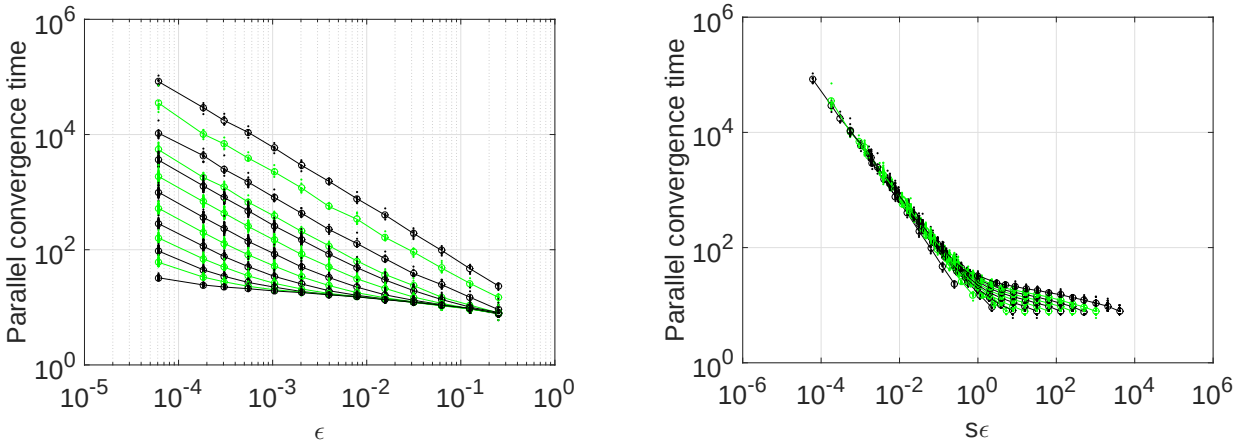

Figure 4: (left) Convergence time vs. parameter ϵ for for the number of states per node $s = 4, 6, 12, 24, 34, 66, 130, 258, 514, 1026, 2050, 4098, 16340$ for the respective curves in the graph from the top to bottom. (right) Convergence time vs. the product $s\epsilon$.

D Experiments

In this section we present the results of performance evaluation by simulations, which serve the purpose of illustrating the correctness and the speed of convergence of the population protocol AVC introduced in this paper. We conducted two types of simulation experiments whose results we summarize as follows.

In our first type of simulation experiments, we consider a system with n nodes, where the initial majority is decided by a single node, i.e. $\epsilon = 1/n$. We compare the performance of n-state AVC algorithm with the four-state protocol and the three-state protocol. We report the mean values computed based on 101 independent simulation runs for each setting of the parameters. The results in the left graph in Figure 3 demonstrate that the n-state AVC algorithm is much faster than the four-state protocol, by several orders of magnitude, and that its speed is comparable to that of the three-state protocol (which may fail to converge to the correct final state). In the right graph in Figure 3, we observe that for the given setting of parameters, the three-state protocol can err with a sizable probability. The n-state protocol, however, guarantees convergence to the correct state with probability 1.

In our second type of simulation experiments, we examine the performance of AVC as we vary the margin ϵ and the number of states per node s. The results in the left graph in Figure 4 show the mean convergence time versus the parameter ϵ, for several values of parameter s that cover the range of four states per node to the number of states per node being in the order of the total number of nodes n. The results demonstrate the speed up of of the protocol as we increase the number of states per node. The results provide support to the claim that the convergence time is $\Theta(1/\epsilon)$

asymptotically for small ϵ with values of other parameters held fixed. As the number of states becomes in the order of the number of nodes, we observe that the convergence time is less sensitive to the margin ϵ. The right graph in Figure 4 shows the same results but versus the product $s\epsilon$. This provides support to the claim that the asymptotically dominant term for the convergence time is $\tilde{\Theta}(1/(s\epsilon))$, which conforms with our theoretical analysis.