Running Time Analysis of Broadcast Consensus Protocols*

Philipp Czerner, Stefan Jaax

{czerner, jaax}@in.tum.de
Department of Informatics, TU München, Germany

February 13, 2021

Broadcast consensus protocols (BCPs) are a model of computation, in which anonymous, identical, finite-state agents compute by sending/receiving global broadcasts. BCPs are known to compute all number predicates in $\mathsf{NL} = \mathsf{NSPACE}(\log n)$ where n is the number of agents. They can be considered an extension of the well-established model of population protocols. This paper investigates execution time characteristics of BCPs. We show that every predicate computable by population protocols is computable by a BCP with expected $\mathcal{O}(n \log n)$ interactions, which is asymptotically optimal. We further show that every log-space, randomized Turing machine can be simulated by a BCP with $\mathcal{O}(n \log n \cdot T)$ interactions in expectation, where T is the expected runtime of the Turing machine. This allows us to characterise polynomial-time BCPs as computing exactly the number predicates in ZPL, i.e. predicates decidable by log-space bounded randomised Turing machine with zero-error in expected polynomial time where the input is encoded as unary.

1. Introduction

In recent years, models of distributed computation following the *computation-by-consensus* paradigm attracted considerable interest in research (see for example [9, 26, 27, 8, 13]). In such models, network agents compute number predicates, i.e. Boolean-valued functions of the type $\mathbb{N}^k \to \{0, 1\}$, by reaching a stable consensus whose value determines the outcome of the computation. Perhaps the most prominent model following this paradigm

*This work was supported by an ERC Advanced Grant (787367: PaVeS) and by the Research Training Network of the Deutsche Forschungsgemeinschaft (DFG) (378803395: ConVeY).

are *population protocols* [5, 6], a model in which anonymous, identical, finite-state agents interact randomly in pairwise rendezvous to agree on a common Boolean output.

Due to anonymity and locality of interactions, it is an inherent property of population protocols that agents are generally unable to detect with absolute certainty when the computation has stabilized. This makes sequential composition of protocols difficult, and further complicates the implementation of control structures such as loops or branching statements. To overcome this drawback, two kinds of approaches have been suggested in the literature: 1.) Let agents guess when the computation has stabilized, leading to composable, but merely *approximately correct* protocols [7, 24], or 2.) extend population protocols by global communication primitives that enable agents to query global properties of the agent population [13, 8, 27].

Approaches of the first kind are for the most part based on simulations of global broadcasts by means of *epidemics*. In epidemics-based approaches the spread of the broadcast signal is simulated by random pairwise rendezvous, akin to the spread of a viral epidemic in a population. When the broadcasting agent meets a certain fraction of "infected" agents, it may decide with reasonable certainty that the broadcast has propagated throughout the entire population, which then leads to the initiation of the next computation phase. Of course, the decision to start the next phase may be premature, in which case the rest of the execution may be faulty. However, epidemics can also be used to implement phase clocks that help keep the failure probability low (see e.g. [7]).

In [13], Blondin, Esparza, and one of the authors of this paper introduced *broadcast consensus protocols* (BCPs), an extension of population protocols by reliable, global, and atomic broadcasts. BCPs find their precursor in the broadcast protocol model introduced by Emerson and Namjoshi in [17] to describe bus-based hardware protocols. This model has been investigated intensely in the literature, see e.g. [18, 19, 15, 29]. Broadcasts also arise naturally in biological systems. For example, Uhlendorf *et al.* analyse applications of broadcasts in the form of an external, global light source for controlling a population of yeasts [12].

The authors of [13] show that BCPs compute precisely the predicates in $\mathsf{NL} = \mathsf{NSPACE}(\log n)$, where n is the number of agents. For comparison, it is known that population protocols compute precisely the *Presburger predicates*, which are the predicates definable in the first-order theory of the integers with addition and the usual order; a class much less expressive than the former.

An epidemics-based approach was used in [7] to show that population protocols can simulate with high probability a step of a virtual register machine with expected $\mathcal{O}(n \log^5(n))$ interactions, where n is the number of agents. This result stimulated further research into time bounds for classical problems such as leader election (see e.g. [21, 1, 16, 30, 11]) and majority (see e.g. [4, 2]). In their seminal paper [5], Angluin *et al.* already showed that population protocols can stably compute Presburger predicates with $\mathcal{O}(n^2 \log n)$ interactions in expectation. Belleville *et al.* further showed that leaderless protocols require a quadratic number of interactions in expectation to stabilize to the correct output for a wide class of predicates [10]. The aforementioned bounds apply to *stabilisation time*: the time it takes to go from an initial configuration to a stable consensus that cannot be destroyed by future interactions. In [24], Kosowski

and Uznanski considered the weaker notion of *convergence time*: the time it takes on average to ultimately transition to the correct consensus (although this consensus could in principle be destroyed by future interactions), and they show that sublinear convergence time is achievable.

By contrast, to the best of our knowledge, time characteristics of BCPs have not been discussed in the literature. The NL-powerful result presented in [13] does not establish any time bounds. In fact, [13] only considers a non-probabilistic variant of BCPs with a global fairness assumption instead of probabilistic choices.

Contributions of the paper This paper initiates the runtime analysis of BCPs in terms of expected number of interactions to reach a stable consensus. To simplify the definition of probabilistic execution semantics, we introduce a restricted, deterministic variant of BCPs without rendezvous transitions. In Section 2, we define probabilistic execution semantics for the restricted version of BCPs, and we provide an introductory example for a fast protocol computing majority in Section 3.

In Section 4, we show that these restrictions of our BCP model are inconsequential in terms of expected number of interactions: both rendezvous and nondeterministic choices can be simulated with a constant runtime overhead.

In Section 5, we show that every Presburger predicate can be computed by BCPs with $\mathcal{O}(n \log n)$ interactions and with constant space, where n denotes the number of agents in the population. This result is asymptotically optimal.

In more generality, in Section 6, we use BCPs to simulate Turing machines (TMs). In particular, we show that any randomised, logarithmically space-bound, polynomial-time TM can be simulated by a BCP with an overhead of $\mathcal{O}(n \log n)$ interactions per step. Conversely, any polynomial-time BCP can be simulated by such a TM. This result can be considered an improvement of the NL bound from [13], now in a probabilistic setting. We also give a corresponding upper bound, which yields the following succinct characterisation: polynomial-time BCPs compute exactly the number predicates in ZPL, which are the languages decidable by randomised log-space polynomial-time TMs with zero-error (the log-space analogue to ZPP).

Bounding the time requires a careful analysis of each step in the simulation of the Turing machine. Thus, our proof diverges in significant ways from the proof establishing the NL lower bound in [13]. Most notably, we now make use of epidemics in order to implement clocks that help reduce failure rates.

2. Preliminaries

Complexity classes As is usual, we define NL as the class of languages decidable by a nondeterministic log-space TM. Additionally, by ZPL we denote the set of languages decided by a randomised log-space TM A, s.t. A only terminates with the correct result (zero-error) and that it terminates within $\mathcal{O}(\operatorname{poly} n)$ steps in expectation, as defined by Nisan in [28].

Multisets A *multiset* over a finite set E is a mapping $M\colon E \to \mathbb{N}$. The set of all multisets over E is denoted $\mathbb{N}^E$. For every $e \in E$, $M(e)$ denotes the number of occurrences of e in M. We sometimes denote multisets using a set-like notation, e.g. $\wr f, g, g \wr$ is the multiset M such that $M(f) = 1$, $M(g) = 2$ and $M(e) = 0$ for every $e \in E \setminus \{f, g\}$. Addition, comparison and scalar multiplication are extended to multisets componentwise, i.e. $(M + M')(e) \stackrel{\text{def}}{=} M(e) + M'(e)$, $(\lambda M)(e) \stackrel{\text{def}}{=} \lambda M(e)$ and $M \leq M' \stackrel{\text{def}}{\iff} M(e) \leq M'(e)$ for every $M, M' \in \mathbb{N}^Q$, $e \in E$, and $\lambda \in \mathbb{N}$. For $M' \leq M$ we also define componentwise subtraction, i.e. $(M - M')(e) \stackrel{\text{def}}{=} M(e) - M'(e)$ for every $e \in E$. For every $e \in E$, we write $\boldsymbol{e} \stackrel{\text{def}}{=} \wr e \wr$. We lift functions $f\colon E \to E'$ to multisets by defining $f(M)(e') \stackrel{\text{def}}{=} \sum_{f(e)=e'} M(e)$ for $e' \in E'$. Finally, we define the *support* and *size* of $M \in \mathbb{N}^E$ respectively as $[\![M]\!] \stackrel{\text{def}}{=} \{e \in E : M(e) > 0\}$ and $|M| \stackrel{\text{def}}{=} \sum_{e \in E} M(e)$.

Broadcast Consensus Protocols A *broadcast consensus protocol* [13] (BCP) is a tuple $\mathcal{P} = (Q, \Sigma, \delta, I, O)$ where

- Q is a non-empty, finite set of *states*,
- Σ is a non-empty, finite *input alphabet*,
- δ is the *transition function* (defined below),
- $I\colon \Sigma \to Q$ is the *input mapping*, and
- $O \subseteq Q$ is a set of *accepting states*.

The function δ maps every state $q \in Q$ to a pair (r, f) consisting of the *successor state* $r \in Q$ and the *response function* $f\colon Q \to Q$.

Configurations A *configuration* is a multiset $C \in \mathbb{N}^Q$. Intuitively, a configuration C describes a collection of identical finite-state *agents* with Q as set of states, containing $C(q)$ agents in state q for every $q \in Q$. We say that $C \in \mathbb{N}^Q$ is a 1-*consensus* if $[\![C]\!] \subseteq O$, and a 0-*consensus* if $[\![C]\!] \subseteq Q \setminus O$.

Step relation A broadcast $\delta(q) = (r, f)$ is executed in three steps: (1) an agent at state q broadcasts a signal and leaves q; (2) all other agents receive the signal and move to the states indicated by the function f, i.e. an agent in state s moves to $f(s)$; and (3) the broadcasting agent enters state r.

Formally, for two configurations C, C' we write $C \to C'$, whenever there exists a state $q \in Q$ s.t. $C(q) \geq 1$, $\delta(q) = (r, f)$, and $C' = f(C - \boldsymbol{q}) + \boldsymbol{r}$ is the configuration computed from C by the above three steps. By $\xrightarrow{*}$ we denote the reflexive-transitive closure of $\to$.

For example, consider a configuration $C \stackrel{\text{def}}{=} \wr a, a, b \wr$ and a broadcast transition $a \mapsto b, \{a \mapsto c, b \mapsto d\}$. To execute this transition, we move an agent from state a to state b and apply the transition function to all other agents, so we end up in $C' \stackrel{\text{def}}{=} \wr b \wr + \wr c, d \wr$.

Broadcast transitions We write broadcast transitions as $q \mapsto r, S$ with S a set of expressions $q' \mapsto r'$. This refers to $\delta(q) = (r, f)$, with $f(q') = r'$ for $(q' \mapsto r') \in S$. We usually omit identity mappings $q' \mapsto q'$ when specifying S.

For graphic representations of broadcast protocols we use a different notation, which separates sending and receiving broadcasts. There we identify a transition $\delta(q) = (r, f)$ with a name α and specify it by writing $q \xrightarrow{!\alpha} r$ and $q' \xrightarrow{?\alpha} r'$ for $f(q') = r'$. Intuitively, $q' \xrightarrow{?\alpha} r'$ can be understood as an agent transitioning from q' to r' upon receiving the signal α, and $q \xrightarrow{!\alpha} r$ means that an agent in state q may transmit the signal α and simultaneously transition to state r.

As defined, δ is a total function, so each state is associated with a unique broadcast. If we do not specify a transition $\delta(q) = (r, f)$ explicitly, we assume that it simply maps each state to itself, i.e. $q \mapsto q, \{r \mapsto r : r \in Q\}$. We refer to those transitions as *silent*.

Executions An *execution* is an infinite sequence $\pi = C_0 C_1 C_2 ...$ of configurations with $C_i \to C_{i+1}$ for every i. It has some fixed number of agents $n \stackrel{\text{def}}{=} |C_0| = |C_1| = ...$. Given a BCP and an initial configuration $C_0 \in \mathbb{N}^Q$, we generate a random execution with the following Markov chain: to perform a step at configuration C_i, a state $q \in Q$ is picked at random with probability distribution $p(q) = C_i(q)/|C_i|$, and the (uniquely defined) transition $\delta(q)$ is executed, giving the successor configuration C_{i+1}. We refer to the random variable corresponding to the trace of this Markov chain as *random execution*.

Stable Computation Let π denote an execution and $\texttt{inf}(\pi)$ the configurations occurring infinitely often in π. If $\texttt{inf}(\pi)$ contains only b-consensuses, we say that π *stabilises* to b. For a predicate $\varphi : \mathbb{N}^\Sigma \to \{0, 1\}$ we say that $\mathcal{P}$ *(stably) computes* φ, if for all inputs $X \in \mathbb{N}^\Sigma$, the random execution of $\mathcal{P}$ with initial configuration $C_0 = I(X)$ stabilises to $\varphi(X)$ with probability 1.

Finally, for an execution $\pi = C_0 C_1 C_2 ...$ we let T_π denote the smallest i s.t. all configurations in $C_i C_{i+1} ...$ are $\varphi(X)$-consensuses, or ∞ if no such i exists. We say that a BCP $\mathcal{P}$ *computes* φ *within* $f(n)$ *interactions*, if for all initial configurations C_0 with n agents the random execution π starting at C_0 has $\mathbb{E}(T_\pi) \leq f(n) < \infty$, i.e. $\mathcal{P}$ stabilises within $f(n)$ steps in expectation. If $f \in \mathcal{O}(\text{poly}(n))$, then we call $\mathcal{P}$ a *polynomial-time* BCP.

Global States Often, it is convenient to have a shared global state between all agents. If, for a BCP $\mathcal{P} = (Q, \Sigma, \delta, I, O)$ we have $Q = S \times G$, $I(\Sigma) \subseteq Q \times \{j\}$ for some $j \in G$, and $f((s, j)) \in Q \times \{j'\}$ for each $\delta((q, j)) = ((r, j'), f)$, then we say that $\mathcal{P}$ has *global states* G. A configuration C has *global state* j, if $[\![C]\!] \subseteq Q \times \{j\}$ for $j \in G$. Note that, starting from a configuration with global state j, $\mathcal{P}$ can only reach configurations with a global state. Hence for $\mathcal{P}$ we will generally only consider configurations with a global state. To make our notation more concise, when specifying a transition $\delta(q) = (r, f)$ for $\mathcal{P}$, we will write f as a mapping from S to S, as q, r already determine the mapping of global states.

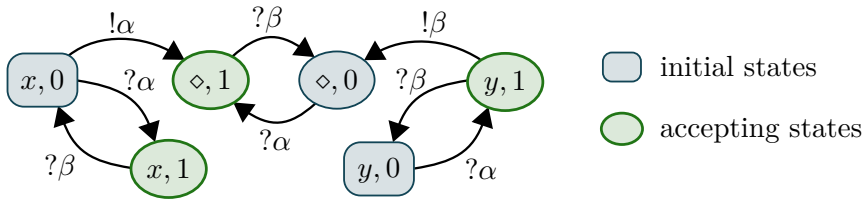

Figure 1: A fast broadcast consensus protocol computing the majority predicate.

Population Protocols A population protocol [5] replaces broadcasts by local rendezvous. It can be specified as a tuple $(Q, \Sigma, \delta, I, O)$ where Q, Σ, I, O are defined as in BCPs, and $\delta\colon Q^2 \to Q^2$ defines *rendezvous transitions*. A step of the protocol at C is made by picking two agents uniformly at random, and applying δ to their states: first $q_1 \in Q$ is picked with probability $C(q_1)/|C|$, then $q_2 \in Q$ is picked with probability $C'(q_2)/|C'|$, where $C' \stackrel{\text{def}}{=} C - \wr q_1 \int$. The successor configuration then is $C - \wr q_1, q_2 \int + \wr r_1, r_2 \int$ where $\delta(q_1, q_2) = (r_1, r_2)$.

Broadcast Protocols Later on we will construct BCPs out of smaller building blocks which we call *broadcast protocols (BPs)*. A BP is a pair (Q, δ), where Q and δ are defined as for BCPs. We extend the applicable definitions from above to BPs, in particular the notions of configurations, executions, and global states.

3. Example: Majority

As an introductory example, we construct a broadcast consensus protocol for the *majority predicate* $\varphi(x, y) = x > y$. Figure 1 depicts the protocol graphically. We have the set of states $\{x, y, \diamond\} \times \{0, 1\}$, with global states $\{0, 1\}$, where the states $O \stackrel{\text{def}}{=} \{(x, 1), (y, 1), (\diamond, 1)\}$ are accepting, and $I(x) = (x, 0)$ and $I(y) = (y, 0)$. The transitions are

$$(x, 0) \mapsto (\diamond, 1),\ \emptyset \qquad (\alpha)$$
$$(y, 1) \mapsto (\diamond, 0),\ \emptyset \qquad (\beta)$$

Note that we use the more compact notation for transitions in the presence of global states, written in long form (α) would be

$$(x, 0) \mapsto (\diamond, 1),\ \{(x, 0) \mapsto (x, 1), (y, 0) \mapsto (y, 1), (\diamond, 0) \mapsto (\diamond, 1)\} \qquad (\alpha)$$

To make the presentation of the following sample execution more readable, we shorten the state (i, j) to i_j. For input $x = 3$ and $y = 2$, an execution could look like this:

$$\wr x_0, x_0, x_0, y_0, y_0 \int \xrightarrow{\alpha} \wr \diamond_1, x_1, x_1, y_1, y_1 \int \xrightarrow{\beta} \wr \diamond_0, x_0, x_0, \diamond_0, y_0 \int$$
$$\xrightarrow{\alpha} \wr \diamond_1, \diamond_1, x_1, \diamond_1, y_1 \int \xrightarrow{\beta} \wr \diamond_0, \diamond_0, x_0, \diamond_0, \diamond_0 \int \xrightarrow{\alpha} \wr \diamond_1, \diamond_1, \diamond_1, \diamond_1, \diamond_1 \int$$

Intuitively, there is a preliminary global consensus, which is stored in the global state. Initially, it is rejecting, as $x > y$ is false in the case $x = y = 0$. However, any x agent is enough to tip the balance, moving to an accepting global state. Now any y agent could speak up, flipping the consensus again.

The two factions initially belonging to x and y, respectively, alternate in this manner by sending signals α and β. Strict alternation is ensured as an agent will not broadcast to confirm the global consensus, only to change it.

After emitting the signal, the agent from the corresponding faction goes into state $\diamond$, where it can no longer influence the computation. In the end, the majority faction remains and determines the final consensus.

Considering these alternations with shrinking factions, the expected number of steps of the protocol until stabilization can be bounded by $2\sum_{k=1}^{n} n/k = \mathcal{O}(n \log n)$. To see that this holds, we consider the factions separately: let n_0 denote the number of agents the first faction starts with (i.e. agents initially in state $(x, 0)$), and n_1 the number at the end. When we are waiting for the first transition of this faction all n_0 agents are enabled, so we wait n/n_0 steps in expectation until one of them executes a broadcast. For the next one, we wait $n/(n_0 - 1)$ steps. In total, this yields $\sum_{k=n_1+1}^{n_0} n/k \leq \sum_{k=1}^{n} n/k$ steps for the first faction, and via the same analysis for the second as well.

In contrast to the $\mathcal{O}(n \log n)$ interactions this protocol takes, constant-state population protocols require n^2 interactions in expectation for the computation of majority [4]. However, these numbers are not directly comparable: broadcasts may not be parallelizable, while it is uncontroversial to assume that n rendez-vous occur in parallel time 1.

4. Comparison with other Models

To facilitate the definition of an execution model, we only consider deterministic BCPs, in the sense that for each state there is a unique transition to execute. Blondin, Esparza and Jaax [14] analysed a more general model, i.e. they allow multiple transitions for a single state, picking one of them uniformly at random when an agent in that state sends a broadcast. Additionally, as they consider BCPs as an extension of population protocols, they include rendez-vous transitions. We now show that we can simulate both extensions within a constant-factor overhead.

4.1. Non-Deterministic Broadcast Protocols

The following construction allows for two broadcast transitions to be executed uniformly at random from a single state. This can easily be extended to any constant number of transitions using the usual construction of a binary tree with rejection sampling.

Now assume that we are given a BCP $(Q, \Sigma, \delta_0, I, F)$ with another set of broadcast transitions δ_1 and we want each agent to pick one transition uniformly at random from δ_0 or δ_1 whenever it executes a broadcast.

We implement this using a synthetic coin, i.e. we are utilising randomness provided by the scheduler to enable individual agents to make random choices. This idea has also

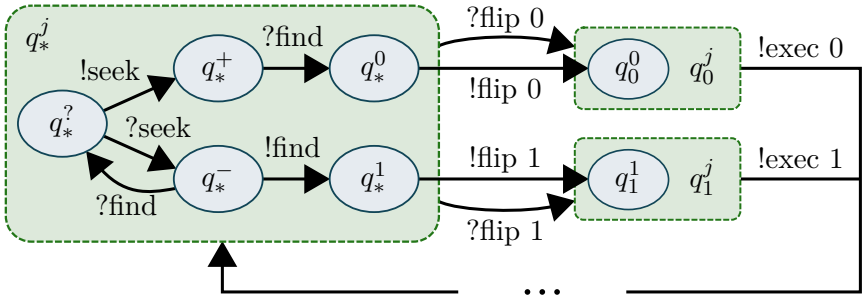

Figure 2: Transition diagram for implementing multiple broadcasts per state, for $q \in Q$, with (q, i, j) written as q_j^i. Dashed nodes represent multiple states, with $j \in T$. Transitions resulting from executing the broadcasts in δ_0, δ_1 are not shown.

been used for population protocols [1, 3]. Compared to these implementations, broadcasts allow for a simpler approach.

The idea is that we partition the agents into types, so that half of the agents have type 0 and the other half have type 1. Additionally, there is a global coin shared across all agents. To flip the coin, a random agent announces its type (the coin is set to heads if the agent is type 0, tails if it is type 1) and a second random agent executes a broadcast transition from either δ_0 or δ_1, depending on the state of the global coin that has just been set. These two steps repeat, the former flipping the coin fairly and the latter then executing the actual transitions. Figure 2 sketches this procedure.

Intuitively, we start with no agents having either type 0 or 1. When such a typeless agent is picked by the scheduler to announce its type (to flip the global coin) it instead broadcasts that it is searching for a partner. Once this has happened twice, these two agents are matched, one is assigned type 0 and the other type 1. Thus we ensure that there is the exact same number of type 0 and type 1 agents at all times, meaning that we get a perfectly fair coin. Additionally we make progress regardless of whether an agent with or without a type is chosen.

To describe the construction formally, we introduce a set of types $T \stackrel{\text{def}}{=} \{?, +, -, 0, 1\}$, and choose the set of states $Q' \stackrel{\text{def}}{=} Q \times T \times \{*, 0, 1\}$, with global states $\{*, 0, 1\}$ used to represent the state of the synthetic coin. We use $(q, ?)$ as initial state instead of $q \in I$, and start with global state $*$. To pick types, we need transitions

$$(q, ?, *) \mapsto (q, +, *), \{(r, ?) \mapsto (r, -) : r \in Q\} \qquad \text{for } q \in Q \qquad \text{(seek)}$$

$$(q, -, *) \mapsto (q, 1, *), \{(r, -) \mapsto (r, ?) : r \in Q\} \cup \{(r, +) \mapsto (r, 0) : r \in Q\} \qquad \text{for } q \in Q \qquad \text{(find)}$$

So an agent of type ? announces that it seeks a partner, moving itself to type + and the others to type −. Then any type − agent may broadcast that a match has been found, moving itself to type 1 and the type + agent to type 0. The other type − agents revert to type ?. This ensures that the number of type 0 and 1 agents is always equal. Note that there may be an odd number of agents, in which case one agent of type +

remains.

The following transitions effectively flip the global coin, by having an agent of type 0 or 1 announce that we now execute a broadcast transition from respectively δ_0 or δ_1. Here, we have $q \in Q, \circ \in \{0, 1\}$.

$$(q, \circ, *) \mapsto (q, \circ, \circ), \emptyset \qquad \text{(flip } \circ\text{)}$$

Then we actually execute the transition $\delta_\circ(q) = (r, f)$, for each $(q, i) \in Q \times T$.

$$(q, i, \circ) \mapsto (r, i, *), \{(s, j) \mapsto (f(s), j) : (s, j) \in Q \times T\} \qquad \text{(exec } \circ\text{)}$$

As the number of type 0 and 1 agents is equal, we select transitions from δ_0 and δ_1 uniformly at random. It remains to show that the overhead of this scheme is bounded.

Executing transition (exec 0) or (exec 1) is the goal. Transitions (flip 0) and (flip 1) ensure that the former are executed in the very next step, so they cause at most a constant-factor slowdown. Transitions (seek) and (find) can be executed at most n times, as they decrease the number of agent of type ?. All that remains is the implicit silent transition of states $(q, +, j)$, which occurs with probability at most $1/n$ in each step.

Hence, to execute $m \geq n$ steps of the simulated protocol our construction takes at most $(2m + 2n) \cdot n/(n - 1) \leq 8m$ steps in expectation.

4.2. Population Protocols

Another extension to BCPs is the addition of rendez-vous transitions. Here we are given a map $R : Q^2 \to Q^2$. At each step, we flip a coin and either execute a broadcast transition as usual, or pick two distinct agents uniformly at random, in state q and r, respectively. These interact and move to the two states $R(q, r)$.

Again, we can simulate this extension with only a constant-factor increase in the expected number of steps. Given a BCP (Q, Σ, B, I, F), the idea is to add states $\{\tilde{q} : q \in Q\} \cup \{r_q : r, q \in Q\}$ and insert "activating" transitions $q \mapsto \tilde{q}, \{r \mapsto r_q : r \in Q\}$ for $q \in Q$ and "deactivating" transitions $r_q \mapsto s, \{\tilde{q} \mapsto t\} \cup \{u_q \mapsto u : u \in Q\}$ for each $R(q, r) = (s, t)$. So a state q first signals that it wants to start a rendez-vous transition. Then, any other state r answers, both executing the transition and signalling to all other states that it has occurred.

Each state in Q has exactly 2 broadcast transitions, so (using the scheme described above) the probability of executing any "activating" transition is exactly $\frac{1}{2}$, the same as doing one of the original broadcast transitions in B. After doing an activating transition we may do nothing for a few steps by executing the broadcast transition on $\tilde{q}$, but eventually we execute a "deactivating" transition and go back. The probability of executing a broadcast on $\tilde{q}$ is $1/n$, so simulating a single rendez-vous transition takes $1 + n/(n - 1) \leq 3$ steps in expectation.

5. Protocols for Presburger Arithmetic

While Blondin, Esparza and Jaax [14] show that BCPs are more expressive than population protocols, they leave the question open whether BCPs provide a runtime speed-up for

the class of Presburger predicates computable by population protocols. We already saw that Majority can be computed within $\mathcal{O}(n \log n)$ interactions in BCPs. This also holds in general for Presburger predicates:

Theorem 1. *Every Presburger predicate is computable by a BCP within at most $\mathcal{O}(n \log n)$ interactions.*

We remark that the $\mathcal{O}(n \log n)$ bound is asymptotically optimal: e.g. the stable consensus for the parity predicate ($x = 1 \bmod 2$) must alternate with configuration size, which clearly requires every agent to perform at least one broadcast in the computation, and thus yields a lower bound of $\sum_{k=1}^{n} \frac{n}{k} = \Omega(n \log n)$ steps like in the coupon collector's problem [20].

It is known [22] that every Presburger predicate can be expressed as Boolean combination of linear inequalities and linear congruence equations over the integers, i.e. as Boolean combination of predicates of the form $\sum_i \alpha_i x_i < c$, and $\sum_i \alpha_i x_i = c \bmod m$, where the α_i, c and m are integer constants. In Section 5.1 we construct BCPs that compute arbitrary linear inequalities, before we sketch the construction for congruences and Boolean combinations in Section 5.2.

5.1. Linear Inequalities

Proposition 2. *Let $\alpha_1, \ldots, \alpha_k, c \in \mathbb{Z}$ and let $\varphi(x_1, \ldots, x_k) \overset{def}{\iff} \sum_{i=1}^{k} \alpha_i x_i < c$ denote a linear inequality. There exists a broadcast consensus protocol that computes φ within $\mathcal{O}(n \log n)$ interactions in expectation.*

Proof. We assume wlog that $\alpha_i \neq 0$ for $i = 1, ..., k$ and that $\alpha_1, ..., \alpha_k$ are pairwise distinct. Let $A \overset{\text{def}}{=} \max\{|\alpha_1|, |\alpha_2| \ldots, |\alpha_k|, |c|\}$. We define a BCP $\mathcal{P} = (Q \times G, \Sigma, \delta, I, O)$ with global states G, where

$$Q \overset{\text{def}}{=} \{0, \alpha_1, ..., \alpha_k\} \qquad \Sigma \overset{\text{def}}{=} \{x_1, \ldots, x_k\}$$
$$G \overset{\text{def}}{=} [-2A, 2A] \qquad O \overset{\text{def}}{=} \{(q, v) : v < c\}$$

As inputs we get $I(x_i) \overset{\text{def}}{=} (\alpha_i, 0)$ for each $i = 1, ..., k$. The transitions δ are constructed as follows. For every $v \in [-2A, 2A]$ and every α_i satisfying $v + \alpha_i \in [-2A, 2A]$, we add the following transition to T:

$$(\alpha_i, v) \mapsto (0, v + \alpha_i), \emptyset \qquad (\alpha_i)$$

Intuitively, in the first component of its state an agent stores its contribution to $\sum_i \alpha_i x_i$, the left-hand side of the inequality. The global state is used to store a counter value, initially set to 0. Each agent adds its contribution to the counter, as long as it does not overflow. The counter goes from $-2A$ to $2A$, which allows it to store the threshold plus any single contribution. The final counter value then determines the outcome of the computation.

Correctness Let $\mathsf{ctr}(C)$ denote the global state (and thus current counter value) of configuration C. Further, let

$$\mathsf{sum}(C) \stackrel{\text{def}}{=} \sum_{(\alpha,v)\in Q} C(\alpha, v) \cdot \alpha + \mathsf{ctr}(C)$$

denote the sum of all agents' contributions and the current value of the counter. Every initial configuration C_0 has $\mathsf{ctr}(C) = 0$ and thus $\mathsf{sum}(C) = \sum_i \alpha x_i$. Each transition α increases the counter by α but sets the agent's contribution to 0 (from α), so $\mathsf{sum}(C)$ is constant throughout the execution.

Recall that our output mapping depends only on the value of the counter, so our agents always form a consensus (though not necessarily a stable one). If this consensus and $\varphi(C_0)$ disagree, then, we claim, a non-silent transition is enabled.

To see this, note that the current consensus depends on whether $\mathsf{ctr}(C) < c$. If that is the case, but $\varphi(C_0) = 0$, then $\mathsf{sum}(C) \geq c$ and some agent with positive contribution $\alpha > 0$ exists. Due to $\mathsf{ctr}(C) < c$, transition α is enabled. Conversely, if $\mathsf{ctr}(C) \geq c$ and $\varphi(C_0) = 1$, some transition α with $\alpha < 0$ will be enabled.

Finally, note that each non-silent transition increases the number of agents with contribution 0 by one, so at most n can be executed in total. So the execution converges and reaches, by the above argument, a correct consensus.

Convergence time Each agent executes at most one non-silent transition. To estimate the total number of steps, we partition the agents by their current contribution: for a configuration C let $C^+ \stackrel{\text{def}}{=} C \restriction \{(q, v) \in Q : q > 0\}$ denote the agents with positive contribution, and define C^- analogously. We have that either $\mathsf{ctr}(C) < 0$ and all transitions of agents in C^+ would be enabled, or $\mathsf{ctr}(C) \geq 0$ and the transitions of C^- could be executed.

If C^+ is enabled, then we have to wait at most $n/|C^+|$ steps in expectation until a transition is executed, which reduces $|C^+|$ by one. In total we get $n/|C_0^+| + n/(|C_0^+| - 1) + ... + n/1 \in \mathcal{O}(n \log n)$. The same holds for C^-, yielding our overall bound of $\mathcal{O}(n \log n)$. □

5.2. Modulo Predicates and Boolean Combinations

Proposition 3. *Let $\varphi(x_1, ..., x_k) \stackrel{def}{\iff} \sum_{i=1}^k \alpha_i x_i \equiv c \pmod{l} < c$ denote a linear inequality, with $\alpha_1, \ldots, \alpha_k, c, l \in \mathbb{Z}, l \geq 2$. There exists a broadcast consensus protocol that computes φ within $\mathcal{O}(n \log n)$ interactions in expectation.*

Proof sketch. The idea is the same as for Proposition 2, but instead of taking care not to overflow the counter we simply perform the additions modulo l. □

Proposition 4 (Boolean combination of predicates)**.** *Let φ be a Boolean combination of predicates $\varphi_1, ..., \varphi_k$, which are computed by BCPs $\mathcal{P}_1, ..., \mathcal{P}_k$, respectively, within $\mathcal{O}(n \log n)$ interactions. Then there is a protocol computing φ within $\mathcal{O}(n \log n)$ interactions.*

Proof sketch. We do a simple parallel composition of the k BCPs, which is the same construction as used for ordinary population protocols (see for example [5, Lemma 6]). A detailed proof can be found in the full version of this paper. □

6. Protocols for all Predicates in ZPL

BCPs compute precisely the predicates in NL with input encoded in unary, which corresponds to NSPACE(n) when encoded in binary. The proof of the NL lower bound by Blondin, Esparza and Jaax [14] goes through multiple stages of reduction and thus does not reveal which predicates can be computed *efficiently*. We will now take a more direct approach, using a construction similar to the one by Angluin, Aspnes and Eisenstat [7]. A step of a randomised Turing machine (RTM) can be simulated using variants of the protocols for Presburger predicates from Section 5, which we combine with a clock to determine whether the step has finished, with high probability.

Instead of simulating RTMs directly, it is more convenient to first reduce them to counter machines. Here, we will use counter machines that are both randomised and capable of multiplying and dividing by two, with the latter also determining the remainder. This ensures that the reduction is performed efficiently, i.e. with overhead of $\mathcal{O}(n \log n)$ interactions per step.

We first show the other direction: simulating BCPs with RTMs.

Lemma 5. *Polynomial-time BCPs compute at most the predicates in* ZPL *with input encoded in unary.*

Proof. An RTM can store the number of agents in each state as binary counters. Picking an agent uniformly at random can be done in $\mathcal{O}(\log n)$ time by picking a random number between 1 and n and comparing it to the agents in the different states. Simulating a transition can also be done with logarithmic overhead. It can further be shown that stabilization of the execution is decidable in time $\mathcal{O}(\log n)$ (see the full version of this paper for details). As the BCP uses only $\mathcal{O}(\operatorname{poly} n)$ interactions (in expectation) the RTM is also $\mathcal{O}(\operatorname{poly} n)$ time-bounded. □

Theorem 6. *Polynomial-time BCPs compute exactly the predicates in* ZPL *with input encoded in unary.*

The proof of Theorem 6 will take up the remainder of this section.

Counter machines Let $\mathrm{Cmd} \stackrel{\text{def}}{=} \{\mathsf{mul}_2, \mathsf{inc}, \mathsf{divmod}_2, \mathsf{iszero}\}$ denote a set of commands, and $\mathrm{Ret} \stackrel{\text{def}}{=} \{\mathsf{done}_0, \mathsf{done}_1\}$ a set of completion statuses. A *multiplicative counter machine with k counters (k-CM)* $A = (S, \mathcal{T}_1, \mathcal{T}_2)$ consists of a finite set of states S with $\mathsf{init}, 0, 1 \in S$ and two transition functions $\mathcal{T}_1, \mathcal{T}_2$ mapping a state $q \in S$ to a tuple (i, j, q'_0, q'_1) where $i \in \{1, ..., k\}$ refers to a counter, $j \in \mathrm{Cmd}$ is a command, and $q'_0, q'_1 \in S$ are successor states (q'_1 is not used for mul_2 and inc operations). Additionally, we require that $\mathcal{T}_1, \mathcal{T}_2$ map $q \in \{0, 1\}$ to $(1, \mathsf{iszero}, q, q)$, effectively executing no operation from those states.

The idea is that A, starting in state init, picks transitions uniformly at random from either $\mathcal{T}_1$ or $\mathcal{T}_2$. Apart from this randomness, the transitions are deterministic. Eventually, A ends up in either state 0 or 1, at which point it cannot perform further actions, thereby indicating whether the input is accepted or rejected.

Step-execution function A *CM-configuration* is a tuple $K = (q, x_1, ..., x_k) \in Q \times \mathbb{N}^k$. We define the *step-execution function* step as follows, with $x \in \mathbb{N}$:

- $\text{step}(\mathsf{mul}_2, x) \stackrel{\text{def}}{=} (\mathsf{done}_0, 2x)$,
- $\text{step}(\mathsf{inc}, x) \stackrel{\text{def}}{=} (\mathsf{done}_0, x+1)$,
- $\text{step}(\mathsf{divmod}, 2x+b) \stackrel{\text{def}}{=} (\mathsf{done}_b, x)$, for $b \in \{0,1\}$, and
- $\text{step}(\mathsf{iszero}, x) \stackrel{\text{def}}{=} (\mathsf{done}_b, x)$, where b is 1 if $x > 0$ and 0 else.

For two CM-configurations $K = (q, x_1, ..., x_k)$ and $K' = (q', x'_1, ..., x'_k)$ where $\mathcal{T}_\circ(q) = (i, j, q'_0, q'_1)$ for $\circ \in \{1,2\}$ we write $K \xrightarrow{\circ} K'$ if $\text{step}(j, x_i) = (\mathsf{done}_b, x'_i)$, $q' = q'_b$ for some $b \in \{0,1\}$, and $x_r = x'_r$ for $r \neq i$. Note that for each K and $\circ$ there is exactly one K' with $K \xrightarrow{\circ} K'$.

The reasoning for introducing the step-execution function is that we want to construct a broadcast protocol (BP) which simulates just one step of the CM. Later on we can use this BP as a building block in a more general protocol.

Computation Let $\varphi : \mathbb{N}^l \to \{0,1\}$ denote a predicate, for $l \leq k$, and $C \in \mathbb{N}^l$ an input to φ. We sample a *random (CM-)execution $\pi = K_0 K_1 K_2 ...$ for input C*, where $K_0, ...$ are CM-configurations, via a Markov chain. For the initial configuration we have $K_0 \stackrel{\text{def}}{=} (\mathsf{init}, C(1), ..., C(l), 0, ..., 0)$, and K_i is determined as the unique configuration with $K_{i-1} \xrightarrow{\circ} K_i$, where $\circ \in \{1,2\}$ is chosen uniformly at random. (So π is the random variable defined as trace of the Markov Chain.)

We say that *A computes φ within $f(n)$ steps* if for each $C \in \mathbb{N}^l$ with $|C| = n$ the random execution for input C reaches a configuration in $\{\varphi(C)\} \times \mathbb{N}^k$ after at most $f(n)$ steps in expectation. Finally, A is *n-bounded* if the random executions for inputs C with $|C| = n$ can only reach configurations in $Q \times \mathbb{N}^k_{\leq n}$.

Theorem 7. *Let φ be a predicate decidable by a log-space bounded RTM within $\mathcal{O}(f(n))$ steps in expectation with unary input encoding. There exists an n-bounded CM that accepts φ within $\mathcal{O}(f(n)\log(n))$ steps in expectation.*

Proof sketch. This can be shown by first representing the Turing machine by a stack machine with two stacks that contain the tape content to the left/right of the current machine head position. In this representation, head movements and tape updates amount to performing pop/push operations on the stack. Moreover, we can simulate an $c \cdot n$-bounded stack by c many n-bounded stacks. An n-bounded stack, in turn, can be represented in a counter machine with a constant number of 2^n-bounded counters. The stack content is represented as the base-2 number corresponding to the binary sequence

stored in the stack. Popping then amounts to a divmod_2 operation, and pushing amounts to doubling the counter value, followed by adding 1 or 0, respectively.

A detailed proof can be found in the full version of this paper. □

We formally define two types of BPs, ones that simulate a step of the CM, and ones behaving like a clock.

Definition 1. *Let BP $\mathcal{P} = (Q \times G, \delta)$ denote a BP with global states G where $0, 1, \perp \in Q$ and $\mathrm{Cmd}, \mathrm{Ret} \subseteq G$. We define the injection $\varphi : G \times \mathbb{N}_{\leq n} \to \mathbb{N}^{Q \times G}$ as $\varphi(j, x) \stackrel{def}{=} x \cdot \wr(1, j)\wr + (n - x) \cdot \wr(0, j)\wr$. The configurations in $\varphi(\mathrm{Cmd} \times \mathbb{N})$ are called* initial*, the ones in $\varphi(\mathrm{Ret} \times \mathbb{N})$* final*. We call a configuration C* failing*, if $C(\perp, i) > 0$ for some $i \in G$.*

We say that $\mathcal{P}$ is CM-simulating *if the sets of final and failing configurations are closed under reachability, and from every initial configuration $\varphi(j, w)$ the only reachable final configuration is $\varphi(\mathrm{step}(j, w))$, if both are well-defined.*

Definition 2. *Let $\mathcal{P} = (Q, \delta)$ denote a BP with $0, 1 \in Q$ and $\mathrm{Time}(\mathcal{P})$ the number of steps until $\mathcal{P}$, starting in configuration $\wr 0, ..., 0\wr$, reaches $\wr 1, ..., 1\wr$, or ∞ if it does not. If $\mathrm{Time}(\mathcal{P})$ is almost surely finite and no agent is in state 1 before $\mathrm{Time}(\mathcal{P})$, then we call $\mathcal{P}$ a* clock-BP*.*

Now we begin by constructing a CM-simulating BP. The value of a given counter is scattered across the population: each agent stores its contribution to this counter value in its state. The counter value is the sum of all contributions. Usually, an agent's contribution is either 1 or 0, thus n agents can maximally store a counter value equal to n, which is not problematic, since the counter machine is assumed to be n-bounded. The difficult part is multiplying and dividing the counter by two. Besides contributions 0 and 1, we will also allow intermediate contributions $\frac{1}{2}$ and 2. By executing a single broadcast, we can multiply (or divide) all the individual contributions by 2, by setting all contributions of value 1 to $\frac{1}{2}$, or 2, respectively. Then, over time, we "normalise" the agents to all have contribution 0 or 1 again in a manner which is specified below. This process takes some time, and we cannot determine with perfect reliability whether it is finished, so we only bound the time with high probability. Here and in the following, we say that some event (dependent on the population size n) happens *with high probability*, if for *all* $k > 0$ the event happens with probability $1 - \mathcal{O}(n^{-k})$.

In this and subsequent lemmata we use $\mathcal{G}(p)$, for $0 < p < 1$, to denote the geometric distribution, that is the number of *trials* until a coin flip with probability p succeeds, which has expectation $1/p$. We start with a statement about the tail distributions of sums of geometric variables.

Lemma 8. *Let $n \geq 3$ and $X_1, ..., X_n$ denote independent random variables with sum X and $X_i \sim \mathcal{G}(i/n)$. Then for any $k \geq 1$ there is an l s.t.*

$$\mathbb{P}(X \geq l \cdot n \ln n) \leq n^{-k}$$

Proof. See the full version of this paper. □

Lemma 9. *There is a CM-simulating BP s.t. starting from an initial configuration it reaches a final configuration within $\mathcal{O}(n \log n)$ steps with high probability.*

Proof. Let $\mathcal{P} = (Q \times G, \delta)$ denote our BP, with $Q \stackrel{\text{def}}{=} \{0, \frac{1}{2}, 1, 2, *\}$ and $G \stackrel{\text{def}}{=} \mathrm{Cmd} \cup \mathrm{Ret} \cup \{\mathsf{high}\}$. The following transitions initialise the computation, with $b \in \{0, 1\}$:

$$(b, \mathsf{mul}_2) \mapsto (2b, \mathsf{done}_0),\ \{1 \mapsto 2, 0 \mapsto 0\} \tag{α_1}$$
$$(b, \mathsf{divmod}_2) \mapsto (\tfrac{b}{2}, \mathsf{done}_0),\ \{1 \mapsto \tfrac{1}{2}, 0 \mapsto 0\} \tag{α_2}$$
$$(b, \mathsf{inc}) \mapsto (b, \mathsf{high}),\ \emptyset \tag{α_3}$$

Additionally, we need transitions that move agents back into states 0 and 1.

$$(0, \mathsf{high}) \mapsto (1, \mathsf{done}_0),\ \emptyset \tag{β_1}$$
$$(2, \mathsf{done}_0) \mapsto (1, \mathsf{high}),\ \emptyset \tag{β_2}$$
$$(\tfrac{1}{2}, \mathsf{done}_0) \mapsto (0, \mathsf{done}_1),\ \emptyset \tag{β_3}$$
$$(\tfrac{1}{2}, \mathsf{done}_1) \mapsto (1, \mathsf{done}_0),\ \emptyset \tag{β_4}$$

This requires some explanation. Basically, we have the invariant that for a configuration C the current value of the counter is $b + \sum_{i \in Q, j \in G} i \cdot C((i, j))$, where b is 1 if the global state is high and 0 else. There is a "canonical" representation of each counter value, where $b = 0$ and the individual contributions $i \in Q$ are only 0 and 1. The transitions (α_1-α_3) update the represented counter value in a single step, but cause a "noncanonical" representation. The transitions (β_1-β_4) preserve the value of the counter and cause the representation to eventually become canonical.

This corresponds to final configurations from Definition 1: as long as the representation is noncanonical, i.e. an agent with value $\frac{1}{2}$, 2 or $*$ exists, the configuration is not final. Conversely, once we reach a final configuration our representation is canonical, and, as the value of the counter is preserved, we reach the correct final configuration.

$$(1, \mathsf{iszero}) \mapsto (1, \mathsf{done}_1),\ \emptyset \tag{α_4}$$
$$(0, \mathsf{iszero}) \mapsto (0, \mathsf{done}_0),\ \{1 \mapsto *\} \tag{α_5}$$
$$(*, \mathsf{done}_0) \mapsto (1, \mathsf{done}_1),\ \{* \mapsto 1\} \tag{β_5}$$

For iszero we do something similar, but the value of the counter does not change. If the initial transition is executed by an agent with value 1, we can go to the global state done_1 directly. Otherwise, we replace 1 by $*$ and go to done_0, so if no agents with value 1 exist, we are finished. Else some agent with value $*$ executes (β_5) and we move to the correct final configuration.

Final configurations can only contain states $\{0, 1\} \times \mathrm{Ret}$. As we have no outgoing transitions from those states, they are indeed closed under reachability.

It remains to be shown that starting from a configuration C_0 we reach a final configuration within $\mathcal{O}(n \log n)$ steps with high probability. Note that transitions (α_1-α_5) are executed at most once. Moreover, these are the only transitions enabled at C_0, so let C_1

denote the successor configuration after executing (α_1-α_5), i.e. $C_0 \to C_1$. From now on, we consider only transitions (β_1-β_5).

Let $M \stackrel{\text{def}}{=} \{\frac{1}{2}, 2, *\} \times G$ denote the set of "noncanonical" states, and, for a configuration C, let $\Phi(C) \stackrel{\text{def}}{=} 2\sum_{q \in M} C(q) + b$ denote a potential function, with b being 1 if the global state of C is high and 0 else. Now we can observe that executing a (β_1-β_5) transition strictly decreases Φ, and that $0 \leq \Phi(C) \leq 2n$ for any configuration C. So after at most $2n$ non-silent transitions, we have reached a final configuration.

Fix some transition (β_j), let $q \in Q \times G$ denote the state initiating (β_j), and let C, C', C'' denote configurations with $C \xrightarrow{\beta_j} C' \xrightarrow{*} C''$, meaning that C'' is a configuration reachable from C after executing (β_j). Then, we claim, $C(q) > C''(q)$.

To see that this holds for transitions (β_2-β_5), note that for $i \in \{\frac{1}{2}, 2, *\}$ the number of agents with value i can only decrease when executing transitions (β_1-β_5). For (β_1) this is slightly more complicated, as (β_3) increases the number of agents with value 0. However, (β_1) is reachable only after (α_1) or (α_3) has been executed, while (β_3) requires (α_2). Thus, our claim follows.

Let X_k denote the number of silent transitions before executing (β_j) for the k-th time, $k = 1, ..., l$, and let r_k denote the number of agents in state q at that time. Then $n \geq r_1 > r_2 > ... > r_l \geq 1$ and X_k is distributed according to $\mathcal{G}(r_k/n)$. So we can use Lemma 8 to show that the sum of X_k is $\mathcal{O}(n \log n)$ with high probability. There are only 5 transitions (β_j), so the same holds for the total number of steps until reaching a final state. □

Our next construction is the clock-BP, which indicates that some amount of time has passed (with high probability). Angluin, Aspnes and Eisenstat used epidemics for this purpose [7], as do we. The idea is that one agent initiates an epidemic and waits until it sees an infected agent. Similar to standard analysis of the coupon collector's problem, this is likely to take $\Theta(n \log n)$ time.

Lemma 10. *There is a clock-BP* $\mathcal{P} = (Q, \delta)$ *s.t.* $\mathbb{E}(\text{Time}(\mathcal{P})) \in \mathcal{O}(n \log n)$ *and* $\text{Time}(\mathcal{P}) \in \Omega(n \log n)$ *with probability* $1 - \mathcal{O}(n^{-1/2})$.

Proof sketch. For a clock we use states $\{0, 1, c_1, c_2, c_3, c_1^+, c_2^+\}$ and transitions

$$0 \mapsto c_1^+, \{0 \mapsto c_2^+\} \tag{α}$$
$$c_2^+ \mapsto c_3, \{c_2^+ \mapsto c_2, c_1^+ \mapsto c_1\} \tag{β}$$
$$c_3 \mapsto c_3, \{c_2 \mapsto c_2^+, c_1 \mapsto c_1^+\} \tag{γ}$$
$$c_1^+ \mapsto 1, \{c_2^+ \mapsto 1, c_3 \mapsto 1\} \tag{ω}$$

State 0 is the initial state, 1 the final state. States c_1 and c_2 denote "uninfected" agents, state c_3 "infected" ones. The former can become activated (moving to c_1^+ and c_2^+), causing one of them to become infected. Transition (α) marks a leader c_1, once they are infected the clock ends (via (ω)). In (β), a single activated agent becomes infected, deactivating the other agents. They get activated again via transition (γ). The state diagram is shown in Figure 3.

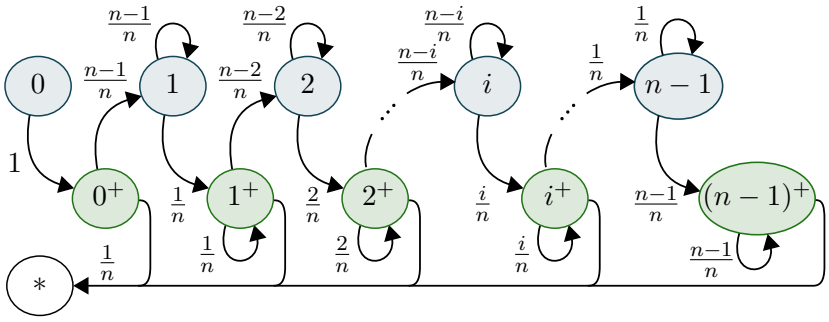

Figure 3: State diagram of the clock implementation. Nodes with i agents in state c_3 are labelled i or i^+, the latter denoting that the other agents are in states c_1^+ and c_2^+. The final state $*$ has all agents in state 1. Arcs are labelled with transition probabilities.

It remains to show that this protocol fulfils the stated time bounds. We prove $\mathbb{E}(\text{Time}(\mathcal{P})) \in \mathcal{O}(n \log n)$ by using that, in expectation, the protocol spends at most n/j steps in state j and at most $n/(n-j)$ in state j^+. For the lower bound we make a case distinction: either state $\lfloor \sqrt{n} \rfloor$ is not visited (i.e. the leader is one of the first $\sqrt{n}$ agents to be infected), or the total number of steps is at least $X_1 + ... + X_{\lfloor \sqrt{n} \rfloor}$, where X_j is the number of steps the protocol spends in state i. As X_j is geometrically distributed with mean n/j, we apply a tail bound from Janson [23] to get the desired result.

A detailed proof can be found in the full version of the paper. □

While the above clock measures some interval of time with some reliability, we want a clock that measures an "arbitrarily long" interval with "arbitrarily high" reliability. Constructions for population protocols use phase clocks for this purpose, but broadcasts allow us to synchronise the agents, so we can directly execute the clock multiple times in sequence instead.

Lemma 11. *Let $k \in \mathbb{N}$ denote some constant. Then there is a clock-BP $\mathcal{P}$ s.t. $\mathbb{E}(\text{Time}(\mathcal{P})) \in \mathcal{O}(n \log n)$, and $\text{Time}(\mathcal{P}) < kn \log n$ with probability $\mathcal{O}(n^{-k})$.*

Proof sketch. The idea is that we run $28k^2$ clocks in sequence, in groups of $2k$. Then it is likely that at least one clock in each group works, yielding the overall minimum running time. A detailed proof can be found in the full version of this paper. □

As mentioned earlier, we combine the clock with the construction in Lemma 9. While we cannot reliably determine whether the operation has finished, we can use a clock to measure an interval of time long enough for the protocol to terminate with high probability. The next construction does just that. In particular, in contrast to Lemma 9, it uses its global state to indicate that it is done.

Lemma 12. *There is a CM-simulating BP s.t. starting from an initial configuration it reaches either a final or a failing configuration C almost surely and within $\mathcal{O}(n \log n)$*

steps in expectation, and C is final with high probability. Additionally, all reachable configurations with global state in Ret *are final or failing.*

Proof. Fix some $k \in \mathbb{N}$ and let $\mathcal{P} = (Q \times G, \delta)$ denote the BP we want to construct. Further, let $\mathcal{P}_1 = (Q_1 \times G_1, \delta_1)$ denote the BP from Lemma 9 and choose some c s.t. $\mathcal{P}_1$ reaches a final configuration after at most $cn \log n$ steps with probability at least $1 - n^{-k}$.

Now we use Lemma 11 to get a clock $\mathcal{P}_2 = (Q_2, \delta_2)$ that runs for at least $cn \log n$ steps with probability at least $1 - n^{-k}$.

We do a parallel composition of $\mathcal{P}_1$ and $\mathcal{P}_2$ to get $\mathcal{P}$. In particular, $Q \stackrel{\text{def}}{=} Q_1 \times Q_2$, $G \stackrel{\text{def}}{=} \{j_\circ : j \in G_1\} \cup \text{Ret}$, where for Q we identify $(i, 0)$ with i for $i \in \{0, 1\, \bot\}$, and for G we identify j with $j_\circ$ for $j \in \text{Cmd}$.

Intuitively, we use $\circ$ to rename the global states of $\mathcal{P}_1$, meaning that the global state $j \in G_1$ of $\mathcal{P}_1$ is now called $j_\circ$ in our protocol. We want $\mathcal{P}_1$ to start with the same initial state we have, which is why we identified j with $j_\circ$ for $j \in \text{Cmd}$. However, we only want to enter a final configurations once the clock has run out, so the completion statuses of $\mathcal{P}_1$ are renamed into $j_\circ$ for $j \in \text{Ret}$ and we enter a final configuration by setting to global state to a $j \in \text{Ret}$.

For each $(q_1, j) \in Q_1 \times G_1$ and $q_2 \in Q_2$ with $\delta_1(q_1, j) = ((r_1, j'), f_1)$ and $\delta_2(q_2) = (r_2, f_2)$ we get the transition

$$(q_1, q_2, j_\circ) \mapsto (r_1, r_2, j'_\circ),\ \{(t_1, t_2) \mapsto (f_1(t_1), f(t_2)) : t_1 \in Q_1, t_2 \in Q_2\} \tag{α}$$

These transitions, together with the way we identified states, ensure that $\mathcal{P}_1$ and $\mathcal{P}_2$ run normally, with the input being passed through to $\mathcal{P}_1$ transparently. However, note that the final configurations of $\mathcal{P}_1$ are not final for $\mathcal{P}$, meaning that the protocol never ends. Hence, for $q_1 \in Q_1, j \in \text{Ret}$ we add the transition

$$\begin{aligned}(q_1, 1, j_\circ) \mapsto (q_1, 0, j), \{(b, 1) \mapsto (b, 0) : b \in \{0, 1\}\} \\ \cup \{(i, 1) \mapsto (\bot, 0) : i \in Q_1 \setminus \{0, 1\}\}\end{aligned} \tag{β}$$

This terminates the protocol once the clock has run out. If $\mathcal{P}_1$ was in a final state, we will now enter a final state as well, else we move into a failing state. □

Finally, we use the above BP to simulate the full l-CM.

Lemma 13. *Fix some predicate $\varphi : \mathbb{N}^k \to \{0, 1\}$ computable by an n-bounded l-CM within $\mathcal{O}(f(n)) \subseteq \mathcal{O}(\text{poly}\, n)$ steps. Then there is a BCP computing φ in $\mathcal{O}(f(n)\, n \log n)$ steps.*

Proof sketch. For each counter we need n agents, so ln in total, but we can simply have each agent simulate a constant number of agents. To execute a step of the CM, we use the BP from Lemma 12. It succeeds only with high probability, but in the case of failure at least one agent will have local state $\bot$, from which that agent initiates a restart of the whole computation.

As the CM takes only a polynomial number of steps, we can fix a k s.t. a computation of our BCP without failures (i.e. one that succeeds on the first try) takes $\mathcal{O}(n^k)$ steps. A

single step succeeds with high probability, so we can require it to fail with probability at most $\mathcal{O}(n^{-k-1})$. In total, the restarts increase the running time by a factor of $1/(1-\mathcal{O}(n^{-1}))$, which is only a constant overhead.

A detailed proof can be found in the full version of this paper. □

This completes the proof of Theorem 6. By Theorem 7, each predicate in ZPL (with input encoded in unary) is computable by a bounded l-CM. Lemma 13 then yields a polynomial-time BCP for that predicate.

We remark that our reductions also enable us to construct efficient BPPs for specific predicates. The predicate PowerOfTwo for example, as described in [14, Proposition 3], can trivially be decided by an $\mathcal{O}(\log n)$-time bounded RTM with input encoded as binary, so there is also a BCP computing that predicate within $\mathcal{O}(n \log^2 n)$ interactions.

References

[1] Alistarh, D., Aspnes, J., Eisenstat, D., Gelashvili, R., Rivest, R.L.: Time-space trade-offs in population protocols. In: Proceedings of the twenty-eighth annual ACM-SIAM symposium on discrete algorithms. pp. 2560–2579. SIAM (2017)

[2] Alistarh, D., Aspnes, J., Gelashvili, R.: Space-optimal majority in population protocols. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 2221–2239. SIAM (2018)

[3] Alistarh, D., Gelashvili, R.: Recent algorithmic advances in population protocols. ACM SIGACT News **49**(3), 63–73 (2018)

[4] Alistarh, D., Gelashvili, R., Vojnović, M.: Fast and exact majority in population protocols. In: Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing. pp. 47–56 (2015)

[5] Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed computing **18**(4), 235–253 (2006), https://www.cs.yale.edu/homes/aspnes/papers/podc04passive-dc.pdf

[6] Angluin, D., Aspnes, J., Eisenstat, D.: Stably computable predicates are semilinear. In: Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing. pp. 292–299. PODC '06, ACM, New York, NY, USA (2006). https://doi.org/10.1145/1146381.1146425

[7] Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. Distributed Computing **21**(3), 183–199 (2008), https://www.cs.yale.edu/homes/aspnes/papers/disc2006-journal.pdf

[8] Aspnes, J.: Clocked population protocols. In: Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing (PODC). pp. 431–440 (2017). https://doi.org/10.1145/3087801.3087836

[9] Aspnes, J., Ruppert, E.: An introduction to population protocols. In: Middleware for Network Eccentric and Mobile Applications, pp. 97–120. Springer (2009)

[10] Belleville, A., Doty, D., Soloveichik, D.: Hardness of computing and approximating predicates and functions with leaderless population protocols. In: 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)

[11] Berenbrink, P., Giakkoupis, G., Kling, P.: Optimal time and space leader election in population protocols. In: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing. pp. 119–129 (2020)

[12] Bertrand, N., Dewaskar, M., Genest, B., Gimbert, H.: Controlling a population. In: Proc. 28$^{\text{th}}$ International Conference on Concurrency Theory (CONCUR). vol. 85, pp. 12:1–12:16 (2017). https://doi.org/10.4230/LIPIcs.CONCUR.2017.12

[13] Blondin, M., Esparza, J., Jaax, S.: Expressive Power of Broadcast Consensus Protocols. In: Proceedings of the 30th International Conference on Concurrency Theory (CONCUR). pp. 31:1–31:16 (2019). https://doi.org/10.4230/LIPIcs.CONCUR.2019.31, http://drops.dagstuhl.de/opus/volltexte/2019/10933

[14] Blondin, M., Esparza, J., Jaax, S.: Expressive power of broadcast consensus protocols (2019), https://arxiv.org/abs/1902.01668.pdf

[15] Delzanno, G., Raskin, J., Begin, L.V.: Towards the automated verification of multithreaded Java programs. In: Proc. 8$^{\text{th}}$ International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 173–187 (2002)

[16] Doty, D., Soloveichik, D.: Stable leader election in population protocols requires linear time. Distributed Computing **31**(4), 257–271 (2018)

[17] Emerson, E.A., Namjoshi, K.S.: On model checking for non-deterministic infinite-state systems. In: Proc. Thirteenth Annual IEEE Symposium on Logic in Computer Science (LICS). pp. 70–80 (1998). https://doi.org/10.1109/LICS.1998.705644

[18] Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: Proc. 14$^{\text{th}}$ Annual IEEE Symposium on Logic in Computer Science (LICS). pp. 352–359 (1999). https://doi.org/10.1109/LICS.1999.782630

[19] Finkel, A., Leroux, J.: How to compose Presburger-accelerations: Applications to broadcast protocols. In: Proc. 22nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). pp. 145–156 (2002)

[20] Flajolet, P., Gardy, D., Thimonier, L.: Birthday paradox, coupon collectors, caching algorithms and self-organizing search. Discrete Applied Mathematics **39**(3), 207–229 (1992)

[21] Gąsieniec, L., Staehowiak, G.: Fast space optimal leader election in population protocols. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 2653–2667. SIAM (2018)

[22] Ginsburg, S., Spanier, E.H.: Semigroups, presburger formulas, and languages. Pacific J. Math. **16**(2), 285–296 (1966), https://projecteuclid.org:443/euclid.pjm/1102994974

[23] Janson, S.: Tail bounds for sums of geometric and exponential variables. Statistics & Probability Letters **135**, 1–6 (2018), https://arxiv.org/pdf/1709.08157.pdf

[24] Kosowski, A., Uznanski, P.: Brief announcement: Population protocols are fast. In: Newport, C., Keidar, I. (eds.) Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018. pp. 475–477. ACM (2018), https://dl.acm.org/citation.cfm?id=3212788

[25] Kruskal, J.B.: The theory of well-quasi-ordering: A frequently discovered concept. Journal of Combinatorial Theory, Series A **13**(3), 297–305 (1972)

[26] Michail, O., Chatzigiannakis, I., Spirakis, P.G.: Mediated population protocols. Theoretical Computer Science **412**(22), 2434–2450 (2011)

[27] Michail, O., Spirakis, P.G.: Terminating population protocols via some minimal global knowledge assumptions. Journal of Parallel and Distributed Computing pp. 1–10 (2015). https://doi.org/10.1016/j.jpdc.2015.02.005

[28] Nisan, N.: On read once vs. multiple access to randomness in logspace. Theoretical Computer Science **107**(1), 135–144 (1993)

[29] Schmitz, S., Schnoebelen, P.: The power of well-structured systems. In: Proc. 24$^{\text{th}}$ International Conference on Concurrency Theory (CONCUR). pp. 5–24 (2013)

[30] Sudo, Y., Masuzawa, T.: Leader election requires logarithmic time in population protocols. Parallel Processing Letters **30**(01), 2050005 (2020)

A. Proof of Theorem 7: From TMs to Counter Machines

In this section, we sketch the proof that every log-space-bounded probabilistic Turing machine with expected polynomial run-time can be simulated by an input-bounded, randomised, multiplicative counter machines with negligible runtime overhead. The proof is a reduction in several steps:

1. We start with a randomised Turing machine M that takes input in *unary* encoding and requires *logarithmic* space.

2. We then show how to transform M into an equivalent RTM M' that takes input in *binary* encoding and requires *linear* space on a single tape.

3. Then we transform M' into an equivalent multi-stack machine S, and S into an equivalent stack machine S' where the length of each stack can be bounded *precisely* by the length of the input.

4. Finally, we show how to transform S' into an equivalent counter machine K.

The rest of this section is structured as follows. In Subsection A.1, we formally introduce randomised Turing machines. In Subsection A.2, we introduce randomised stack machines and sketch how to simulate randomised Turing machines efficiently in stack machines whose stacks are bounded by the input size. Finally, in Subsection A.3, we show how to simulate stack machines in multiplicative counter machines.

A.1. Turing machines

For our reduction, we need to define a Turing machine model.

Definition 3 (Randomised Turing Machine)**.** *A randomised Turing machine (RTM) is a tuple* $(Q, \Sigma, \Gamma, q_0, q_a, q_r\delta_1, \delta_2)$ *where*

- Q *is a finite set of states,*
- Γ *is the tape alphabet, with* $\square, 0, 1 \in \Gamma$*,*
- $q_0, q_f, q_r \in Q$ *are the initial, accepting, and rejecting states, respectively.*
- $\delta_i \colon Q \times \Gamma \mapsto Q \times \Gamma \times \{-1, 0, +1\}$ *are the transition functions.*

Additionally, we require that $\delta_i(q, \alpha) = (q, \alpha, 0)$ *for all* $q \in \{q_f, q_r\}$*,* $i \in \{1, 2\}$ *and* $\alpha \in \Gamma$*.*

So once the RTM has reached q_f or q_r, it performs no further actions.

Configurations and step relation A *configuration* of an RTM M is a tuple $C = (q, i, \tau)$ consisting of a control state $q \in Q$, read/write head position $i \in \mathbb{Z}$, and tape $\tau \colon \mathbb{Z} \to \Gamma$. The configuration C is acceping if $q = q_f$, and rejecting if $q = q_r$. For $\circ \in \{1, 2\}$ and two configurations $C = (q, i, \tau)$, $C' = (q', i', \tau')$, a *step* $C \xrightarrow{\circ} C'$ is valid if and only if the following holds:

- $\delta_\circ(q, \tau(i)) = (q', \alpha, d)$ for some α and some d,
- $i' = i + d$,
- $\tau'(i) = \alpha$ and $\tau'(j) = \tau(j)$ for all $j \neq i$.

Input encoding For a given input $x \in \mathbb{N}^k$ and configuration $C \stackrel{\text{def}}{=} (q_0, 0, \tau)$, we say that C is x *encoded as unary (binary)* if $\tau(1)...\tau(n) = \square 1^{x_1} \square ... \square 1^{x_k} \square$ (resp. $\tau(1)...\tau(n) = \square (x_1)_2 \square ... \square (x_1)_2 \square$), where $(r)_2$ denotes the binary representation of $r \in \mathbb{N}$, and $\tau(i) = 0$ for $i \notin \{1, ..., n\}$.

Random executions We define the *random exection for input x encoded as unary (binary)* as a Markov chain $\pi = C_0C_1C_2...$ where $C_0, ...$ are RTM-configurations, C_0 is x encoded as unary (binary), and C_i is determined as the unique configuration with $C_{i-1} \xrightarrow{\circ} C_i$, where $\circ \in \{1, 2\}$ is chosen uniformly at random.

Acceptance/rejection For a predicate $\varphi : \mathbb{N}^k \to \{0, 1\}$, we say that M *computes* φ *within* $\mathcal{O}(f(n))$ *steps*, if for all $x \in \mathbb{N}^k$ the random execution $C_0...$ for input x reaches an accepting or rejecting configuration C almost surely and after at most $f(n)$ steps in expectation, and C is accepting iff $\varphi(x) = 1$. Such an RTM is called $f(n)$*-time bounded*. We say M is $f(n)$*-space bounded* if all configurations (q, i, τ) reachable from C_0 satisfy $|i| \leq f(|x|)$.

Log-space Turing machines The definition of the Turing machine model given above does not have a separate reading/writing tape, and is thus not suitable for the definition of log-space complexity classes. For log-space Turing machines we assume a modified model, where instead of one tape τ, we have two tapes: An immutable reading tape τ_{read}, and a working tape τ_{work}, and instead of one head we have two heads, one for each tape, which can be moved independently. A configuration then is a tuple $(q, i, j, \tau_{\text{read}}, \tau_{\text{work}})$, where q is the current state, i is the position of the reading head, and j is the position of the working head. The other definitions are adapted in the obvious manner.

Lemma 14. *Every predicate decidable by an $\mathcal{O}(\log n)$-space bounded, $\mathcal{O}(f(n))$-time bounded RTM with input encoded as unary is decidable by an $\mathcal{O}(N)$-space bounded and $\mathcal{O}(N \cdot f(2^N))$-time bounded RTM with input encoded as binary.*

Proof sketch. Let M be an $\mathcal{O}(\log n)$-space bounded, $\mathcal{O}(f(n))$-time bounded RTM with input encoded as unary. We sketch the construction of an RTM M' with binary input encoding that simulates M in $\mathcal{O}(N)$ space and within $\mathcal{O}(f(2^N))$ steps in expectation, where $N = \log_2 n$ denotes the size of the input of M' for clarity.

Movements and updates on the working tape of M can clearly be simulated by M' within the given space. Further, M' can simulate the movement of the head on the input tape of M by keeping a single binary counter to encode its position and performing a constant number of additions/comparisons when M moves the input head. This means that M' simulates a step of M with an overhead of N steps.

As M takes $f(n) = f(2^N)$ steps, we end up with M' deciding the predicate in $Nf(2^N)$ steps. □

A.2. Stack Machines

We now sketch how a RTMs M can be simulated efficiently with randomised stack machines.

Definition 4. *A (randomised) l-stack-machine M is a tuple $(Q, \delta_1, \delta_2, q_0, q_f, q_r)$ where*

- *Q is a finite set of states,*

- $\delta_i \colon Q \to ([l] \times \{0,1\} \times Q) \cup ([l] \times Q \times Q \times Q))$ *are transition functions*
- $q_0, q_f, q_r \in Q$ *are the initial, accepting, and rejecting states, respectively.*

Additionally, we require that $\delta_i(q) = (1, q, q, q)$ *for* $i \in \{1, 2\}$ *and* $q \in \{q_f, q_r\}$*, meaning that from states* q_f, q_r *no (significant) actions are performed.*

Configurations and transitions A configuration of the l-stack machine M is a tuple $C = (q, s_1, \ldots, s_l)$, consisting of a control state $q \in Q$, and l binary stacks $s_1, \ldots, s_k \in \{0,1\}^*$. We say C is *accepting* if $q = q_f$, and *rejecting* if $q = q_r$. Let $C' = (q', s'_1, \ldots, s'_l)$ be a configuration of M. We use $C \xrightarrow{\text{PUSH } k\,\alpha}_\circ C'$ and $C \xrightarrow{\text{POP } k}_\circ C'$ to denote that C' results from pushing α to or popping from the kth stack, respectively, as specified in $\delta_\circ$: Formally, we write $C \xrightarrow{\text{PUSH } k\,\alpha}_\circ C'$ whenever $\delta_\circ(q) = (k, \alpha, q')$ and $s'_k = \alpha \cdot s_k$. We write $C \xrightarrow{\text{POP } k}_\circ C'$ whenever $\delta(q) = (k, q_1, q_2, q_3)$ and $s_i = s'_i$ for every $i \neq k$, and one of following three constraints is satisfied:

$$q' = q_1 \text{ and } s_k = 0 \cdot s'_k, \tag{1}$$
$$q' = q_2 \text{ and } s_k = 1 \cdot s'_k, \tag{2}$$
$$q' = q_3 \text{ and } s_k = s'_k = \epsilon. \tag{3}$$

We write $C \xrightarrow{\circ} C'$ to indicate that $C \xrightarrow{\text{PUSH } k\,\alpha}_\circ C'$ or $C \xrightarrow{\text{POP } k}_\circ C'$ holds.

Input encoding These encodings are technical artefacts of the reductions we use, hence they are quite unintuitive. For a given input $x \in \mathbb{N}^k$ and configuration $C \stackrel{\text{def}}{=} (q_0, s_1, ..., s_l)$, we say that C is x *two-symbol encoded* if $s_1 = f(\tau(0))...f(\tau(n))$ and $s_2 = ... = s_l = \epsilon$, where τ are the tape contents of x encoded as binary and $f(0) \stackrel{\text{def}}{=} 01$, $f(1) \stackrel{\text{def}}{=} 10$, $f(\square) \stackrel{\text{def}}{=} 11$. We say that C is x *multi-stack encoded* if $s_{ji} = r_{3(i-1)+j}$ for $j \in \{1,2,3\}$ and $i \in \mathbb{N}$ with $1 \leq 3(i-1) + j \leq n$, where $(q_0, r, \epsilon, ..., \epsilon)$ is x two-symbol encoded.

The idea of the multi-stack encoding is that each stack has length at most n, which we can ensure by distributing the symbols in a round-robin fashion.

Acceptance condition and boundedness Random executions and acceptance are defined as in RTMs, now for configurations of the stack and the step relation of the stack machine. An l-stack machine M is $f(n)$-*bounded* if any initial configuration given by an encoded input x can only reach configurations $(q, s_1, ..., s_l)$ with $|s_1|, ..., |s_l| \leq f(n)$, where $n \stackrel{\text{def}}{=} \log_2(x(1) + ... + x(k))$. (So n is roughly the length of x in binary encoding.)

For subsequent propositions, we fix a predicate $\varphi : \mathbb{N}^k \to \{0,1\}$ and a function $f \colon \mathbb{N} \to \mathbb{N}$. We will now show that every $\mathcal{O}(n)$-bounded stack machine can be simulated by an n-bounded stack machine.

Proposition 15. *Let* $c \in \mathbb{N}$*. For every* cn*-bounded* l*-stack machine that decides* φ *in* $\mathcal{O}(f(n))$ *steps with input two-symbol encoded, there exists an* n*-bounded* $(c(l-1)+3)$*-stack machine that decides* φ *in* $\mathcal{O}(f(n))$ *steps with input multi-stack encoded.*

Proof sketch. The idea is to replace each stack s of the $c \cdot n$-bounded machine by c many n-bounded auxiliary stacks $s_1, ..., s_c$. For each stack s of the original machine, the new machine stores in its control state a round counter $i \in \{1, ..., c\}$ that specifies the currently used auxiliary stack s_i, starting with $i = 1$. Pushing some symbol α onto s and transitioning from states q to q' is implemented by updating q to q' and updating i to its successor i', where $i' = i + 1$, if $i < c$, else $i' = 1$, and by pushing α onto $s_{i'}$. This way, the first element of s is pushed onto s_1, the second onto s_2, and so forth, until the element $c + 1$, which again goes onto s_1, and so forth. Symmetrically, popping from s amounts to popping from s_i and decrementing i (resp. setting $i = c$, if $i = 1$).

For the input stack we use three auxiliary stacks instead of c, as that suffices for those stacks to be n-bounded. Conveniently, the multi-stack encoding means that we do not have to perform any work to convert the input. □

Proposition 16. *Let M be an $\mathcal{O}(n)$-space bounded, $\mathcal{O}(f(n))$-time bounded RTM deciding φ with input encoded as binary. Then there exists an n-bounded stack machine S that decides φ in $\mathcal{O}(f(n))$ steps with input multi-stack encoded.*

Proof sketch. We assume wlog that M uses $\Gamma = \{\square, 0, 1\}$ as tape alphabet. Let $C = (q, i, \tau)$ be a configuration of M. Then C can be represented as triple (q, l, r) where $l, r \in \{0, 1\}^\omega$, $l \stackrel{\text{def}}{=} f(\tau(i-1))f(\tau(i-2))...$ represents the side of the tape left to the head, and $r \stackrel{\text{def}}{=} f(\tau(i))f(\tau(i+1))...$ represents the side of the tape to the right of the head, where f is defined as for the two-symbol encoding. Moving the head to the left and updating the tape then amounts to removing the first two left-most symbols from l, and prepending the resulting updated digits to r, and symmetrically for moving the head to the right.

Since M is $\mathcal{O}(n)$-space bounded, we may represent l and r by linearly bounded stacks in a stack machine S. Again, the two-symbol input encoding means that we do not have to perform any work for initialisation, as long as we use r as input stack.

The aforementioned updates can then be performed using corresponding pop/push operations. This gives an $\mathcal{O}(n)$-bounded stack machine S that decides φ in $\mathcal{O}(f(n))$ steps. By Proposition 15, we can transform S into a n-bounded stack machine that decides φ in $\mathcal{O}(f(n) + n)$ steps. □

A.3. From Stack Machines to Counter Machines

Proposition 17. *Let M be an $\mathcal{O}(n)$-space bounded RTM that decides φ in $\mathcal{O}(f(n))$ steps, where the input is given in unary. Then there exists an n-bounded counter machine that computes φ in $\mathcal{O}(f(n)\log(n))$ steps.*

Proof. By Lemma 14 and Proposition 16, there exists an N-bounded l-stack-machine S that decides φ in $\mathcal{O}(f(2^N) \cdot N)$ steps with input in multi-stack encoding, where $N = \log_2 n$ is roughly the length of the input encoded as binary.

We now describe how the stack content is represented in the counters, how the counters can be initialized, and how push/pop operations can be implemented efficiently in that representation.

Representing stack content Each stack s_i of S is represented by two counters x_i, x_i' in the counter machine, where x_i contains the actual symbols of s_i, and x_i' simply contains a 1 for each symbol in s_i and serves to determine whether x_i is empty. Formally, for $s_i = w_0 ... w_m$ we get $x_i = \sum_j w_j 2^j$ and $x_i' = \sum_j 2^j$. As S is N-bounded, each stack has at most $N = \log_2 n$ symbols, and thus each of our counters is at most n large. Therefore we are indeed n-bounded.

Hence we can simply push a symbol $b \in \{0, 1\}$ onto s_i by performing operations mul_2 on x_i and x_i', inc on x_i', and if $b = 1$ also inc on x_i. Similarly, to pop from a stack we determine whether it is empty via iszero on x_i', and do a divmod_2 on both x_i and x_i' if it is not.

This means that we simulate a single operation with only constant overhead, so we execute $\mathcal{O}(f(n) \log(n)$ steps of the stack machine.

Initialization of stacks Recall that S accepts input in the multi-stack encoding, while the counter machine simply has k counter initialised to the input values.

We can determine bits of the binary representation of the value stored in a counter by repeatedly performing divmod_2, which are straightforward to encode into their two-symbol encoding, which we then push onto the correct stacks in round-robin fashion to construct the multi-stack encoding. Here, we use iszero to determine whether any bits are left in a counter. This initialisation procedure takes $\mathcal{O}(\log n)$ steps, which is subsumed in the overall running time. □

B. Proof: Stability can be decided in logarithmic time

Call a configuration C *stable* if it is in a b-consensus for some $b \in \{0, 1\}$, and every configuration reachable from C is also in a b-consensus.

Proposition 18. *For every BCP $\mathcal{P} = (Q, \Sigma, \delta, I, O)$, there exists a (deterministic) Turing machine $T_\mathcal{P}$ that decides in constant time whether a given configuration $C \in \mathbb{N}^Q$ is stable.*

Proof. Let $U \subseteq \mathbb{N}^Q$ be the set of unstable configurations of $\mathcal{P}$. It is straightforward to see that U is upwards-closed, that is, $C \in U$ implies $C' \in U$ for every $C' \geq C$ (if a configuration is not stable, then adding additional agents cannot make it stable).

By Dickson's lemma [25], U has finitely many minimal elements. Since $\mathcal{P}$ is not part of the input, we may assume that the minimal elements are precomputed and $T_\mathcal{P}$ can iterate over the minimal elements in constant time. In order to decide whether $C \in U$ holds, $T_\mathcal{P}$ only needs to verify whether $C' \leq C$ holds for some minimal unstable configuration C'. By the previous considerations, this can be done with a constant number of comparisons, so in time $\mathcal{O}(\log n)$. □

C. Proof of Proposition 4: Boolean Combinations of Predicates

Proposition 4 (Boolean combination of predicates)**.** *Let φ be a Boolean combination of predicates $\varphi_1, ..., \varphi_k$, which are computed by BCPs $\mathcal{P}_1, ..., \mathcal{P}_k$, respectively, within $\mathcal{O}(n \log n)$ interactions. Then there is a protocol computing φ within $\mathcal{O}(n \log n)$ interactions.*

Proof. It suffices to show the statement for $k = 2$ and $\varphi = \neg\varphi_1 \wedge \neg\varphi_2$.

Let $\mathcal{P}_i =: (Q_i, \Sigma, \delta_i, I_i, O_i)$, $i = 1, 2$. We assume that both protocols work on the same input alphabet Σ. This is without loss of generality, as we can always unify alphabets by adding missing symbols and identity mappings. We define our resulting protocol $\mathcal{P} = (Q, \Sigma, \delta, I, O)$ as follows:

$$\begin{aligned} Q &\stackrel{\text{def}}{=} Q_1 \times Q_2, \\ \delta((q_1, q_2)) &\stackrel{\text{def}}{=} ((r_1, r_2), f) \text{ with } \delta_i(q_i) = (r_i, f_i) \text{ and } f((q_1, q_2)) = (f_1(q_1), f_2(q_2)) \\ I(x) &\stackrel{\text{def}}{=} (I(x), I(x)) \text{ for every } x \in \Sigma, \\ O &\stackrel{\text{def}}{=} \{(q_1, q_2) : q_1 \notin O_1 \wedge q_2 \notin O_2\} \end{aligned}$$

Every execution of $\mathcal{P}$ maps onto one of $\mathcal{P}_1$ and one of $\mathcal{P}_2$ by projecting onto either the first of the second element of each state, so every fair execution of $\mathcal{P}$ consists of two executions of $\mathcal{P}_1$ and $\mathcal{P}_2$ which stabilize to the value of φ_1 and φ_2 within $\mathcal{O}(n \log n)$ steps in expectation, respectively. Hence $\mathcal{P}$ stabilizes to $\neg\varphi_1(C_0) \wedge \neg\varphi_2(C_0) = \varphi(C_0)$ within $\mathcal{O}(n \log n)$ steps in expectation. □

D. Sums of Geometric Random Variables

We use the following theorems to estimate lower and upper tail probabilities for sums of geometrically distributed random variables. As before, we use $\mathcal{G}(p)$, for $0 < p < 1$, to denote the geometric distribution (with expectation $1/p$).

Theorem 19 (see [23, Theorem 2.1])**.** *Let $X_1, ..., X_n$ denote independent random variables with $X_i \sim \mathcal{G}(p_i)$ for $i = 1, ..., n$ and $0 < p_i \leq 1$, and set $X \stackrel{def}{=} X_1 + ... + X_n$, $\mu \stackrel{def}{=} \mathbb{E}(X)$, and $p_* \stackrel{def}{=} \min_i p_i$. Then, for any $\lambda \geq 1$*

$$\mathbb{P}(X \geq \lambda\mu) \leq e^{-p_*\mu(\lambda - 1 - \ln\lambda)}$$

Theorem 20 (see [23, Theorem 3.1])**.** *Let $X_1, ..., X_n$ denote independent random variables with $X_i \sim \mathcal{G}(p_i)$ for $i = 1, ..., n$ and $0 < p_i \leq 1$, and set $X \stackrel{def}{=} X_1 + ... + X_n$, $\mu \stackrel{def}{=} \mathbb{E}(X)$, and $p_* \stackrel{def}{=} \min_i p_i$. Then, for any $\lambda \leq 1$*

$$\mathbb{P}(X \leq \lambda\mu) \leq e^{-p_*\mu(\lambda - 1 - \ln\lambda)}$$

In the following lemma we use the first of these bounds to show that the sum $X_1 + ... + X_n$ with $X_i \sim \mathcal{G}(i/n)$ is in $\Omega(n \log n)$ with high probability. This is the same analysis as used for the coupon collector's problem.

Lemma 8. *Let $n \geq 3$ and $X_1, ..., X_n$ denote independent random variables with sum X and $X_i \sim \mathcal{G}(i/n)$. Then for any $k \geq 1$ there is an l s.t.*

$$\mathbb{P}(X \geq l \cdot n \ln n) \leq n^{-k}$$

Proof. First, note that $\mathbb{E}(X) = nH_n$, where $H_n := 1/1 + ... + 1/n$ is the n-th harmonic number. Using Theorem 19 with λ chosen s.t. $\lambda - 1 - \ln \lambda = k$ we get

$$\mathbb{P}(X \geq \lambda n H_n) \leq e^{-1/n \cdot nH_n k} = e^{-kH_n}$$

We have $\ln n \leq H_n \leq 1 + \ln n \leq 2 \ln n$, for $n \geq 3$, which yields the desired bound when choosing $l := 2\lambda$. □

E. Proof of Lemma 11: Arbitrarily Good Clocks

Lemma 11. *Let $k \in \mathbb{N}$ denote some constant. Then there is a clock-BP $\mathcal{P}$ s.t. $\mathbb{E}(\text{Time}(\mathcal{P})) \in \mathcal{O}(n \log n)$, and $\text{Time}(\mathcal{P}) < kn \log n$ with probability $\mathcal{O}(n^{-k})$.*

Proof. To simplify notation, we will use the precise constants proved in Lemma 10 here, although the proof does not require them.

Choose $l \stackrel{\text{def}}{=} 28k^2$ as the number of clocks we want to run in sequence and let (Q', δ') denote the BP from Lemma 10. We construct our BP $\mathcal{P}$ as $(Q' \times \{1, ..., l\}, \delta)$, with global states $\{1, ..., l\}$ and transitions δ as follows.

$$(q, i) \mapsto (r, i),\ f \qquad \text{for } \delta(q) = (r, f), i = 1, ..., l \qquad (\alpha)$$
$$(1, i) \mapsto (0, i + 1), \{1 \mapsto 0\} \qquad \text{for } i = 1, ..., l - 1 \qquad (\beta)$$

Additionally, we identify state $(0, 0)$ with 0 and $(1, l)$ with 1.

For the analysis, we are going to divide the l clocks into groups of $2k$. As shown in Lemma 10, each clock has at most a $2n^{-1/2}$ probability of "failing", i.e. taking fewer than $n/14 \cdot \ln n$ steps. So the probability that all $2k$ clocks in a group fail is at most $2^{2k} n^{-k}$.

By union bound, the probability that there exists a group which has all of its clocks failing is at most $14 \cdot 2^{2k} k n^{-k}$. Conversely, if each group has a single non-failing clock, we take at least $kn \ln n$ steps in total. □

F. Proof of Lemma 13: Simulating CMs efficiently

Lemma 13. *Fix some predicate $\varphi : \mathbb{N}^k \to \{0, 1\}$ computable by an n-bounded l-CM within $\mathcal{O}(f(n)) \subseteq \mathcal{O}(\text{poly}\, n)$ steps. Then there is a BCP computing φ in $\mathcal{O}(f(n)\, n \log n)$ steps.*

Proof. Let $A = (S, \mathcal{T}_1, \mathcal{T}_2)$ denote the CM. We will construct our BCP assuming to work with ln agents instead of n, as each agent can simulate a constant number of agents. Additionally, we will ignore errors for now, and modify the BCP later to deal with them.

So let $\mathcal{P} = (Q \times G, \Sigma, \delta, I, O)$ denote our BCP with global states G, where we set $\Sigma \stackrel{\text{def}}{=} \{1, ..., k\}$. We want to use Lemma 12 to simulate the steps of the CM, so let $\mathcal{P}' = (Q' \times G', \delta')$ denote a BP from that Lemma s.t. $\mathcal{P}'$ reaches a final configuration with probability $1 - \mathcal{O}(n^{-r-1})$, where r is chosen s.t. $f \in \mathcal{O}(n^r)$.

We use local states $Q \stackrel{\text{def}}{=} \{1_1, ..., 1_k\} \cup Q'$, where $C(1_i)$ represents the value of counter $i \in \{1, ..., l\}$ of the CM in a configuration C. As global states, we have $G \stackrel{\text{def}}{=} (G' \cup \mathsf{init}) \times S$. Note that we have states $0, 1 \in Q'$ as well, with 1 representing a "working register" in the same manner as the other counters. Agents not belonging to a counter are in state 0. This means that the initial states are simply given by $I(i) \stackrel{\text{def}}{=} 1_i$ for $i \in \{1, ...k\}$.

To perform a step of the CM, we load the affected counter into the "working register" represented by state $1 \in Q'$ and let $\mathcal{P}'$ run. After it has terminated, we write the value back into the original counter and move to the next state. We will now describe these transitions formally, so let $\mathcal{T}_\circ(s) = (i, j, s'_0, s'_1)$ denote a transition of the CM, with $\circ \in \{1, 2\}$, $s \in S$. To initialise the working register, we need

$$\begin{aligned} &(1_i, \mathsf{init}, s) \mapsto (1, j, s),\ \{1_i \mapsto 1\} \\ &(0, \mathsf{init}, s) \ \mapsto (0, j, s),\ \{1_i \mapsto 1\} \end{aligned} \tag{init}$$

Then we faithfully execute the transitions of $\mathcal{P}'$.

$$(q, j, s) \mapsto (q', j', s),\ f \qquad \text{for } \delta'(q, j) = ((q', j'), f) \tag{run}$$

Once $\mathcal{P}'$ reaches a final state, we move to the next state of A.

$$\begin{aligned} &(1, \mathsf{done}_b, s) \mapsto (1_i, \mathsf{init}, s'_b), \{1 \mapsto 1_i\} \\ &(0, \mathsf{done}_b, s) \mapsto (0, \mathsf{init}, s'_b),\ \{1 \mapsto 1_i\} \end{aligned} \qquad \text{for } b \in \{0, 1\} \tag{finish}$$

Note that we have two outgoing (init) transitions, one for each $\circ$. Matching the execution of CMs, we pick one of these uniformly at random, using our construction from Section 4.1. We also remark that the BP $\mathcal{P}'$ is only guaranteed to simulate the CM if the counter values do not exceed n. This is ensured by A being bounded.

We know that A eventually reaches either state 0 or 1, which determines the result of its computation. So we define the accepting states of $\mathcal{P}$ based on that as $O \stackrel{\text{def}}{=} \{(q, j, s) \in Q \times G : s = 1\}$.

As mentioned, the above disregards error conditions. When executing $\mathcal{P}'$ via transition (run) it might end up in a failing configuration, meaning that at least one agent has local state $\bot$. There are no transitions which would cause an agent to leave state $\bot$ (in particular, recall that the set of failing configurations is closed under reachability for $\mathcal{P}'$). Hence any agent with local state $\bot$ remains so, and we can modify $\mathcal{P}$ to cause such an agent to initiate a reset. For this we have each agent remember its initial state, and state $\bot$ sends a broadcast which reverts each agent to its initial state, effectively restarting the computation.

Formally, we do this transformation (as well as simulating l agents by a single one) as follows. Let $\mathcal{P}^* = (Q^*, \Sigma, \delta^*, I^*, O^*)$ denote the new BCP. We use states $Q^* \stackrel{\text{def}}{=} (Q \times G)^{l+1}$, where the first component stores the initial state and the latter l components simulate l

agents. The input mapping is $I(i) \stackrel{\text{def}}{=} (1_i, 1_i, 0, ..., 0)$ for $i \in \{1, ..., k\}$, and the accepting states are $O^* \stackrel{\text{def}}{=} \{(q_0, ..., q_l) \in Q^* : q_1 \in O\}$.

For $i \in \{1, ..., l\}, q \in Q \times G$ we execute a transition $\delta(q_i) = (q_i', f)$ of $\mathcal{P}$ as

$$(q_0, ..., q_l) \mapsto (q_0, q_1', ..., q_l'),\ \{(r_0, ..., r_l) \mapsto (r_0, f(r_1), ..., f(r_l))\}$$

where $q_m' \stackrel{\text{def}}{=} f(q_m)$ for $m \in \{1, ..., l\} \setminus \{i\}$. To perform the resets, we add a transition for each $(q_0, ..., q_l) \in Q^*$ with $q_i = \perp$ for some i.

$$(q_0, ..., q_l) \mapsto (q_0, q_0, 0, ..., 0),\ \{(r_0, ..., r_l) \mapsto (r_0, r_0, 0, ..., 0)\}$$

We execute one step of the CM every $\mathcal{O}(n \log n)$ interactions, in expectation. If no failure occurs, the BCP stabilises after $\mathcal{O}(n^{r+1} \log n) \subseteq \mathcal{O}(f(n)\, n \log n)$ steps, in expectation. A single step can fail with probability $\mathcal{O}(n^{-r-1})$, so the probability that all steps succeed is at least $1 - \mathcal{O}(n^{-1})$. If a step fails, the computation will restart (after $\mathcal{O}(n)$ steps in expectation), increasing the expected running time by a factor of $1/(1 - \mathcal{O}(n^{-1})) \in \mathcal{O}(1)$. □