The Expressive Power of Uniform Population Protocols with Logarithmic Space

Philipp Czerner
Technical University of Munich, Germany

Vincent Fischer
Technical University of Munich, Germany

Roland Guttenberg
Technical University of Munich, Germany

Abstract

Population protocols are a model of computation in which indistinguishable mobile agents interact in pairs to decide a property of their initial configuration. Originally introduced by Angluin et. al. in 2004 with a constant number of states, research nowadays focuses on protocols where the space usage depends on the number of agents. The expressive power of population protocols has so far however only been determined for protocols using $o(\log n)$ states, which compute only semilinear predicates, and for $\Omega(n)$ states. This leaves a significant gap, particularly concerning protocols with $\Theta(\log n)$ or $\Theta(\operatorname{polylog} n)$ states, which are the most common constructions in the literature. In this paper we close the gap and prove that for any $\varepsilon > 0$ and $f \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$, both uniform and non-uniform population protocols with $\Theta(f(n))$ states can decide exactly those predicates, whose unary encoding lies in $\mathsf{NSPACE}(f(n) \log n)$.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Population Protocols, Uniform, Expressive Power

Digital Object Identifier 10.4230/LIPIcs.SAND.2025.1

Related Version *Preprint*: `https://arxiv.org/abs/2408.10027`
Previous Brief Announcement in DISC 2024: `https://doi.org/10.4230/LIPIcs.DISC.2024.44`

1 Introduction

Population protocols are a model of computation in which indistinguishable mobile agents randomly interact in pairs to decide whether their initial configuration satisfies a given property. The decision is taken by *stable consensus*; eventually all agents agree on whether the property holds or not, and never change their mind again. While originally introduced to model sensor networks [4], population protocols are also very close to chemical reaction networks [24], a model in which agents are molecules and interactions are chemical reactions.

Originally agents were assumed to have a finite number of states [4–6], however many predicates then provably require at least $\Omega(n)$ time to decide [1, 7, 20], as opposed to recent breakthroughs of $\mathcal{O}(\log n)$ time using $\mathcal{O}(\log n)$ or even fewer states for important tasks like leader election [9] and majority [18]. Limitting the number of states to logarithmic is important in most applications, especially the chemical reaction setting, since a linear in n number of states would imply the unrealistic number of approximately 10^{23} different chemical species. Therefore most recent literature focuses on the polylogarithmic time and space setting, and determines time-space tradeoffs for various important tasks like majority [1–3, 8, 18, 21], leader election [1, 9, 21] or estimating/counting the population size [10, 15–17, 19].

This leads to the interesting open problem of characterizing the class of predicates which can be computed in polylogarithmic time using a logarithmic or polylogarithmic number of states. There is however a fundamental problem with working on this question: Despite the focus on $\mathcal{O}(\log n)$ number of states in recent times, the expressive power for this number of states has not yet been determined. While it is known that protocols with $o(\log n)$ number of states can only compute semilinear predicates [6,14] and with $f(n) \in \Omega(n)$ states the expressive power is $\mathsf{UNSPACE}(n \log f(n))$ [14], i.e. predicates which can be decided in $\mathsf{NSPACE}(n \log f(n))$, when the input is encoded in unary, the important case of having logarithmically many states is unknown. To the best of our knowledge, the only research in this direction is [12], where the expressive power is characterised for polylog(n) number of states for a similar model – not population protocols themselves. Their results do not lead to a complete characterization for $\Theta(\log n)$ states since their construction is slightly too space-inefficient, simulating a $\log\log n$-space TM by approximately $\log^2 n$ space protocols.

In this paper, we resolve this gap by proving that for functions $f(n) \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$, where $\varepsilon > 0$, we have $\mathsf{UPP}(f(n)) = \mathsf{UNSPACE}(f(n) \cdot \log n)$, i.e. predicates computable by population protocols using $\mathcal{O}(f(n))$ number of states are exactly the predicates computable by a non-deterministic Turing machine using $\mathcal{O}(f(n) \cdot \log n)$ space with the input encoded in unary. The "U" in $\mathsf{UPP}(f(n))$ stands for *uniform*: Modern population protocol literature distinguishes between uniform and non-uniform protocols. In a non-uniform protocol, a different protocol is allowed to be used for every population size. While we have stated the expressive power for uniform protocols here, our complexity characterization also holds for non-uniform population protocols.

Our results complete the picture of the expressive power of uniform protocols: For $o(\log n)$ only semilinear predicates can be computed (open for non-uniform), for a class of *reasonable* functions $f(n) \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$, which contains most practically relevant functions[1] we have $\mathsf{UNSPACE}(\log(n) \cdot f(n))$ by our results, and for $f \in \Omega(n)$ we have $\mathsf{UNSPACE}(n \cdot \log f(n))$. (A slight gap between $\mathcal{O}(n^{1-\varepsilon})$ and $\Omega(n)$ remains.)

Main Contribution. The technically most involved part of our result is the lower bound, i.e. constructing a $\mathcal{O}(f(n))$ space uniform population protocol simulating a $\mathcal{O}(f(n) \log n)$ space Turing machine, or – equivalently [23], and used in our construction – simulating a $\mathcal{O}(2^{f(n) \log n})$-bounded counter machine. Let us briefly illustrate the main techniques and difficulties towards this result. In a nutshell, the crucial difference between $o(n)$ and $\Omega(n)$ states is the ability to assign unique identifiers to agents, and to store the population size n in a single agent. In our construction, therefore, we must distribute the value of n over multiple agents, and they must collaborate to compute operations involving it. We also introduce a novel approach for encoding the counters of the counter machine, as those described in previous publications such as [5] and [12] cannot encode large enough numbers for our purposes.

Overview. The paper is structured as follows: In Section 2 we give preliminaries and define population protocols. Section 3 briefly states our main result and prove the lower bound for weakly uniform population protocols. The proof of the matching upper bound (even for uniform protocols) is presented in Section 4.

[1] This will be clarified in the next section.

2 Preliminaries

We let $\mathbb{N}$ denote the set of natural numbers including 0 and let $\mathbb{Z}$ denote the set of integers. We write $\log n$ for the binary logarithm $\log_2 n$.

A multiset over a set Q is a multiplicity function $f\colon Q \to \mathbb{N}$, which maps every $q \in Q$ to its number of occurances in the multiset f. We denote multisets using a set notation with multiplicities, i.e. $\{f(q_1) \cdot q_1, \dots, f(q_m) \cdot q_m\}$. We define addition $f + f'$ on multisets via $(f + f')(q) = f(q) + f'(q)$ for all $q \in Q$. Multisets are compared via inclusion, defined as $f \subseteq f' \iff f(q) \leq f'(q)$ for all $q \in Q$. If $f \subseteq f'$, then subtraction $f' - f$ is defined via $(f' - f)(q) = f'(q) - f(q)$ for all $q \in Q$. The number of elements of f is denoted $|f|$ and defined as $\sum_{f(q)\neq 0} f(q)$ if only finitely many $q \in Q$ fulfill $f(q) \neq 0$, and $|f| = \infty$ otherwise. Elements q of Q are identified with the multiset $\{1 \cdot q\}$. The set of all *finite* multisets over Q is denoted $\mathbb{N}^Q$. Given a function $g\colon A \to B$, its extension $\hat{g}$ to finite multisets is $\hat{g}\colon \mathbb{N}^A \to \mathbb{N}^B, \hat{g}(f) = \sum_{f(a)\neq 0} f(a) \cdot \{g(a)\}$.

▶ **Definition 1.** *A* protocol scheme $\mathcal{P}$ *is a 5-tuple* $(Q, \Sigma, \delta, I, O)$ *of*

- *a (not necessarily finite) set of states* Q*,*
- *a finite input alphabet* Σ*,*
- *a transition function* $\delta : Q \times Q \to Q \times Q$*,*
- *an input mapping* $I\colon \Sigma \to Q$*,*
- *an output mapping* $O\colon Q \to \{0, 1\}$*.*

A configuration of $\mathcal{P}$ is a *finite* multiset $C \in \mathbb{N}^Q$. A step $C \to C'$ in $\mathcal{P}$ consists of choosing a multiset $\{q_1, q_2\} \subseteq C$ and replacing $\{q_1, q_2\}$ by $\{\delta(q_1, q_2)\}$ or $\{\delta(q_2, q_1)\}$, i.e. $C' = (C - \{q_1, q_2\} + \{\delta(q_1, q_2)\})$. The intuition is that the configuration describes for every q the number of agents in q, and a step consists of an agent in q_1 exchanging messages with q_2, upon which these two agents change into the states $\delta(q_1, q_2)$. Observe that the transition function δ distinguishes between the initiator of the exchange and the responder, while in the configuration all agents are anonymous. The number of agents is denoted $n := |C|$.

We write $\to^*$ for the reflexive and transitive closure of $\to$, and say that a configuration C' is reachable from C if $C \to^* C'$. A configuration C is initial if there exists a multiset $w \in \mathbb{N}^\Sigma$ such that $\hat{I}(w) = C$. In that case C is the initial configuration for input w.

A configuration C is a b-consensus for $b \in \{0, 1\}$ if $O(q) = b$ for all q such that $C(q) \neq 0$, i.e. if every state which occurs in the configuration has output b. A configuration C is stable with output b if every configuration C' reachable from C is a b-consensus.

A run ρ is an infinite sequence of configurations $\rho = (C_0, C_1, \dots)$ such that $C_i \to C_{i+1}$ for all $i \in \mathbb{N}$. A run is fair if for all configurations C which occur infinitely often in ρ, i.e. such that there are infinitely many i with $C_i = C$, also every configuration C' reachable from C occurs infinitely often in ρ. A run has output b if some configuration C_i along the run is stable with output b (and hence all C_j for $j \geq i$ are also stable with output b).

An input $w \in \mathbb{N}^\Sigma$ has output b if every fair run starting at its corresponding initial configuration $\hat{I}(w)$ has output b. The protocol scheme $\mathcal{P}$ computes a predicate if every input w has some output. In that case the computed predicate is the mapping $\mathbb{N}^\Sigma \to \{0, 1\}$, which maps w to the output of $\hat{I}(w)$.

▶ **Example 2.** Consider $Q := \{0\} \cup \{2^i \mid i \in \mathbb{N}\}$, and define $\delta(2^i, 2^i) = (2^{i+1}, 0)$, otherwise δ is the identity function. Let $\Sigma = \{x\}$, and let $x \mapsto 2^0$ be the input mapping. Then a configuration is initial if every agent is in state 2^0. Intuitively this protocol will eventually end up with the binary representation of the number of agents. Namely each transition preserves the total sum of all agents' values, and every actual transition (which does not simply leave the agents the same) causes an agent to enter 0, so this protocol in fact always reaches a terminal configuration. For example if we start this protocol with 22 agents we will eventually reach the stable configuration $\{1 \cdot 2^1, 1 \cdot 2^2, 1 \cdot 2^4, 19 \cdot 0\}$, which corresponds to the binary encoding of $22 = 10110_2$.

We now define the state complexity of a protocol scheme. A state $q \in Q$ is *coverable* from some initial configuration C_0 if there exists a configuration C reachable from C_0 which fulfills $C(q) > 0$. The state complexity $S(n)$ of $\mathcal{P}$ for n agents is the number of states $q \in Q$ which are coverable from some initial configuration with n agents.

▶ **Example 3.** In the scheme of Example 2, let C_n be the unique initial configuration with n agents, i.e. $C_n(2^0) = n$ and $C_n(q) = 0$ otherwise. For $n \geq 2$, the states coverable from C_n are exactly $\{0\} \cup \{2^i \mid i \leq \log n\}$. Hence the state complexity is $S(n) = \lfloor \log n \rfloor + 2$.

As defined so far, protocol schemes are not necessarily computable. Hence actual population protocols require some uniformity condition, and that $S(n)$ is finite for all n.

▶ **Definition 4.** *A* uniform population protocol $\mathcal{P} = (Q, \Sigma, \delta, I, O)$ *is a protocol scheme s.t. 1) the space complexity $S(n) \neq \infty$ for all $n \in \mathbb{N}$ and 2) there is a representation of states as binary strings and linear space Turing-machines (TMs) M_δ, M_I, M_O, where*

1. *M_δ: Given (the representation of) two states q_1, q_2, M_δ outputs $\delta(q_1, q_2)$.*
2. *M_I: Given multiset w, M_I outputs a representation of $\hat{I}(w)$.*
3. *M_O: Given a state q and $b \in \{0,1\}$, M_O checks whether $O(q) = b$.*

We remark that "linear space" then in terms of our n, the number of agents, is $\mathcal{O}(\log S(n))$ space (since the input of the machine is a representation of a state).

In the literature on uniform population protocols, e.g. [13–15,19], often agents are defined as TMs and states hence automatically assumed to be represented as binary strings. We avoid talking about the exact implementation of a protocol via TMs because it introduces an additional logarithm in the number of states and potentially confuses the reader, while most examples are clearly computable.

▶ **Example 5.** In the protocol scheme of Example 2 we represent states by the binary representation of the exponent. Clearly incrementing natural numbers or setting the number to a fixed value are possible by a linear space TM, hence this is a uniform population protocol.

Next we define a more general class of population protocols, which we call weakly uniform. This class includes all known population protocols, and our results also hold for this class, which shows that having a different protocol for every n does not strengthen the model.

▶ **Definition 6.** *A* finite population protocol *is a protocol scheme with a finite set Q.*

A population protocol $\mathcal{P}$ *is an infinite family $(\mathcal{P}_n)_{n \in \mathbb{N}} = (Q_n, \Sigma, \delta_n, I_n, O_n)_n$ of finite population protocols. The state complexity for inputs of size n is $S(n) := |Q_n|$.*

$\mathcal{P}$ *is* weakly uniform *if there exist TMs M_δ, M_I, M_O using $\mathcal{O}(S(n))$ space which:*

1. *M_δ: Given two states q_1, q_2 and $n \in \mathbb{N}$ in unary, M_δ outputs $\delta_n(q_1, q_2)$.*
2. *M_I: Given multiset w with n elements, M_I outputs a representation of $\hat{I_n}(w)$.*
3. *M_O: Given a state q, $b \in \{0,1\}$ and $n \in \mathbb{N}$ in unary, M_O checks whether $O_n(q) = b$.*

The configurations of $\mathcal{P}$ with n agents are exactly the configurations of $\mathcal{P}_n$ with n agents, and accordingly the semantics of steps, runs and acceptance are inherited from $\mathcal{P}_n$.

The protocol for a given population size n is allowed to differ completely from the protocol for $n-1$ agents, as long as TMs are still able to evaluate transitions, input and output. Usually this is not fully utilised, with the most common case of a non-uniform protocol being that $\log n$ is encoded into the transition function [18].

Clearly uniform population protocols are weakly uniform. Namely let $\mathcal{P} = (Q, \Sigma, \delta, I, O)$ be a protocol scheme. Then for every $n \in \mathbb{N}$ we let Q_n be the set of states coverable by some initial configuration with n agents, similar to the definition of state complexity, and define $\mathcal{P}_n := (Q_n, \Sigma, \delta_n|_{Q_n^2}, I, O|_{Q_n})$, where $f|_A$ is the restriction of f to inputs in A. This protocol family computes the same predicate, and is weakly-uniform with the same state complexity.

Next we define the complexity classes for our main result. Let $f \colon \mathbb{N} \to \mathbb{N}$ be a function. f is space-constructible if there exists a TM M which computes f using $\mathcal{O}(f(n))$ space. Given a space-constructible function $f \colon \mathbb{N} \to \mathbb{N}$, we denote by $\mathsf{NSPACE}(f(n))$ the class of predicates computable by a non-deterministic Turing-machine in $\mathcal{O}(f(n))$ space. Similarly, let $\mathsf{UPP}(f(n))$ be the class of predicates computable by uniform population protocols with $\mathcal{O}(f(n))$ space, and $\mathsf{WUPP}(f(n))$ be the class of predicates computable by weakly-uniform population protocols with $\mathcal{O}(f(n))$ space.

Population protocols decide predicates on multisets $w \in \mathbb{N}^\Sigma$, or equivalently predicates on $\mathbb{N}^k$ for $k = |\Sigma|$. In order to compare the complexity classes defined on predicates with those defined on languages over an alphabet we define the unary encoding of a predicate $\varphi \colon \mathbb{N}^k \longrightarrow \{0, 1\}$ as the language $L_\varphi := \{1^{x_1}\#1^{x_2}\# \cdots \#1^{x_k} \mid \varphi(x_1, x_2, \ldots, x_k) = 1\}$. For any complexity class $\mathcal{C}$ we can now define $\mathsf{UENC}(\mathcal{C}) := \left\{\varphi \colon \mathbb{N}^k \longrightarrow \{0, 1\} \mid k \in \mathbb{N}, L_\varphi \in \mathcal{C}\right\}$ as the class of predicates whose unary encoding lies in $\mathcal{C}$. More specificaly we define $\mathsf{UNSPACE}(f(n)) := \mathsf{UENC}(\mathsf{NSPACE}(f(n)))$ [2].

3 Main Result

We give a characterisation for the expressive power of both uniform and weakly uniform population protocols with $f(n)$ states, where $f \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$, for some $\varepsilon > 0$. For technical reasons, we must place two limitations on $f(n)$:

1. $f(n) = g(\lfloor \log n \rfloor)$ for some $g : \mathbb{N} \to \mathbb{N}$, i.e. f is computable knowing only $\lfloor \log n \rfloor$.
2. $f(n)$ is space-constructible, i.e. the function f can be computed in $\mathsf{SPACE}(f(n))$, and
3. $f(n)$ is monotonically increasing.

All practically relevant functions fulfil these properties. For the first, we remark that “usually” $f(n) \in \Theta(f(2^{\lfloor \log n \rfloor}))$.[3] For example, while $\sqrt{n}$ is not computable from $\lfloor \log n \rfloor$, we can instead use $\sqrt{2^{\lfloor \log n \rfloor}}$, which is asymptotically equivalent.

In the remainder of this paper, a function f with these properties is called *reasonable.*

Our bound applies to uniform and weakly uniform protocols. As mentioned in the previous section, the latter includes, to the best of our knowledge, all non-uniform constructions from the literature.

▶ **Theorem 7.** *Let $\varepsilon > 0$ and let $f \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$ be reasonable. Then*

$$\mathsf{UPP}(f(n)) = \mathsf{WUPP}(f(n)) = \mathsf{UNSPACE}(f(n) \cdot \log n).$$

Proof. This will follow directly from the upper and lower bounds given by Proposition 8 and Theorem 9. ◀

In particular, we have $\mathsf{UPP}(\log n) = \mathsf{WUPP}(\log n) = \mathsf{UNSPACE}(\log^2 n)$.

[2] Previous work has instead used the complexity class $\mathsf{SNSPACE}(f(n))$ consisting of the symmetric languages (i.e. languages closed under permutation) over the alphabet Σ in $\mathsf{NSPACE}(f(n))$ to reflect that the agents in a population protocol are unordered. We find it more intuitive to think about a unary encoding with separators, but languages with either encoding can be polynomially reduced to the other.

[3] The exceptions are plateau functions with large jumps.

▶ **Proposition 8.** *Let $\varepsilon > 0$ and let $f \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$ be space-constructible. Then*

$$\mathsf{UPP}(f(n)) \subseteq \mathsf{WUPP}(f(n)) \subseteq \mathsf{UNSPACE}(f(n)\log n).$$

Proof. $\mathsf{UPP}(f(n)) \subseteq \mathsf{WUPP}(f(n))$ follows since uniform protocols are also weakly-uniform.

Hence let $(\mathcal{P}_n)_n = (Q_n, \Sigma, \delta_n, I_n, O_n)_n$ be a weakly uniform population protocol computing a predicate φ. We have to show that there exists a TM $M \in \mathsf{NSPACE}(f(n)\log n)$ computing φ, when given the input in unary. We employ a similar argument as in the proof of the upper bound in [11]: First observe that a configuration of $\mathcal{P}_n$ with n agents can be described by $|Q_n| \in \mathcal{O}(f(n))$ many numbers up to n, i.e. can be stored using $\mathcal{O}(f(n)\log n)$ bits. Namely one can store the number of agents per state $q \in Q_n$. The encoding of the initial configuration can easily be calculated by simply counting the ones on the input tape corresponding to each initial state.

Since f is space-constructible, $f(n)\log n$ is space-constructible as well. By the Immerman-Szelepcsényi theorem we have $\mathsf{NSPACE}(f(n)\log n) = \mathsf{coNSPACE}(f(n)\log n)$.

Since the population protocol $(\mathcal{P}_n)_n$ computes a predicate, either every fair run starting from the initial configuration $\hat{I}(w)$ accepts or every fair run rejects. M has to determine which of these is the case. In fact, because every fair run has the same output, we claim that some configuration C reachable from $\hat{I}(w)$ is stable for output 1 if and only if $\hat{I}(w)$ is accepted. By definition, an accepting run visits a configuration stable for output 1, proving one direction, and for the other direction construct a fair run $\rho\colon \hat{I}(w) \rightarrow^* C \rightarrow \dots$ by extending $\hat{I} \rightarrow^* C$ in a fair way. This run is accepting, and hence also every other fair run is.

We hence construct M as follows: M applies M_I to obtain a representation of the initial configuration $\hat{I}(w)$. It guesses a configuration C, and checks using repeatedly M_δ that C is reachable from $\hat{I}(w)$. It remains to check that C is stable with output 1. A configuration is not stable for output 1 if and only if some configuration C' reachable from C contains an agent with output 0. Therefore non-stability can be checked in $\mathsf{NSPACE}(f(n)\log n)$ by guessing C', checking using M_O that C' is not a 1-consensus and checking reachability. By Immerman-Szelepcsényi hence also stability is decidable in $\mathsf{NSPACE}(f(n)\log n)$. ◀

4 Lower Bound

In this section, we prove the following.

▶ **Theorem 9.** *Let $\varepsilon > 0$ and let $f \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$ be reasonable. Then*

$$\mathsf{UNSPACE}(f(n)\log n) \subseteq \mathsf{UPP}(f(n)).$$

To do this we first fix a reasonable function $f \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$ and a predicate $\varphi\colon \mathbb{N}^\Sigma \longrightarrow \{0,1\}$ with $\varphi \in \mathsf{UNSPACE}(f(n)\log n)$. With a slight modification to the classic 3-counter simulation of a $f(n)$ space-bounded Turing machine described in [23], we obtain a counter machine $\mathcal{CM}$ with $|\Sigma|$ input registers and 3 computation registers that decides φ using $\mathcal{O}\left(2^{f(n)\log n}\right)$ space.

We now construct a population protocol $\mathcal{P} = (Q, \Sigma, \delta, I, O)$ simulating $\mathcal{CM}$. There are three main difficulties involved in this construction:

Firstly, in order to sequence multiple operations such that an operation only starts once the previous one has finished, we need a way of performing a *zero-check*, i.e. detecting whether an agent with a certain state exists. We achieve this by counting the number of agents using a binary encoding similar as to [12]. By keeping track of the agents already seen in an additional counter, we can then perform loops over all agents, and so detect absence of a certain state.

Secondly, we need to encode the counters of $\mathcal{CM}$, which can hold values up to $\mathcal{O}\left(2^{f(n)\log n}\right)$. The two counter encodings described in existing literature of either counting in unary the number of agents in a special state [5], or the binary encoding of [12] both cannot encode numbers this large. We improve on the binary encoding by using *digits* in a higher base of $\Theta\left(\frac{n}{f(n)}\right)$ and counting in unary within each digit. Manipulating these digits makes heavy use of the looping construct mentioned above.

The final problem, which is inherent to all population protocols, is that an arbitrary number of agents may not participate in any interactions for an arbitrary long amount of time. These errors are detected at some point, but this can happen arbitrarily late. At that point, we solve this by providing a way for the simulation to re-initialize itself.

The protocol consists of multiple phases:

1. We count the number of agents and initialize the additional counters to zero. This process is detailed in Section 4.2. Section 4.3 describes how the counters are manipulated, and Section 4.4 presents a macro for looping over all agents using the counters for bookkeeping.
2. We set up the *digits*, which encode the counters of $\mathcal{CM}$. This phase is described in Section 4.6.
3. The instructions of $\mathcal{CM}$ are simulated. This is described in Section 4.7.

As a technical aside, as is usual we assume that the protocol is started with a sufficient number of agents (i.e. exceeding some constant). We argue in the proof of Theorem 9 why this is not a problem.

4.1 State Space

The states will be of the form $Q = \mathbb{N} \times 2^F$ for a finite set F of *flags*. A state $(q, S) \in Q$ has *level* q. We defer precise definitions until they become relevant.

Notation. To compactly denote sets of states characterised by flags, we, for example, write (i, Ldr_0) for the set of all level i states which do not include the flag Ldr. In particular, this notation avoids mentioning other flags.

Formally, we write $(i, X^{(1)}_{b_1}, ..., X^{(k)}_{b_k})$, where $X^{(1)}, ..., X^{(k)} \in F$ and $b_1, ..., b_k \in \{0, 1\}$ to refer to the set of all states (i, S) where $S \subseteq F$ fulfils $X^{(j)} \in S \Leftrightarrow b_j = 1$ for all $j = 1, ..., k$.

On the right-hand side of a transition, we use the same notation with a different meaning: it refers to the state where flags $X^{(1)}, ..., X^{(k)}$ are as given, and all other flags match the corresponding state that initiated the transition. I.e. similar to an assignment command, the mentioned values are set while leaving other flags the same as before.

We also use $*$ as wildcard. On the left-hand side of a transition, it matches anything, and on the right-hand side, it refers to the same value as the corresponding element of the left-hand side. For example, the transition

$$(i, \mathsf{Ex}_1), (*, \mathsf{Ex}_1) \mapsto (i+1), (*, \mathsf{Ex}_0) \qquad \text{for } i \in \mathbb{N} \qquad \langle\mathsf{example}\rangle$$

means that any two agents with flag Ex can interact. The first moves to the next level (with flags unchanged), while the second removes the Ex flag (and leaves its level unchanged).

Sometimes, we want to refer to groups of flags at once, and we write S_b for $S = \{X^{(1)}, ..., X^{(k)}\} \subseteq F, b \in \{0, 1\}$ instead of $X^{(1)}_b, ..., X^{(k)}_b$.

4.2 Initialisation

Our first goal is to reach a configuration with one leader at level $l_n := \lfloor \log n \rfloor$, with $l_n + 1$ agents each storing one bit of the binary representation of n, and all other agents "ready to be reset". Let $b_{l_n} ... b_0$ be the binary representation of n. Formally we want the leader in state $(l_n, \mathsf{Ldr}_1, \mathsf{I}_1)$, exactly one counter agent in $(j, \mathsf{Ctr}_1, \mathsf{N}_{b_j})$ for each $j \leq l_n = \lceil \log n \rceil$, and all other agents in states $(*, \mathsf{Ldr}_0, \mathsf{Ctr}_0)$. The flags $\mathsf{Ldr}, \mathsf{Ctr}, \mathsf{Free} \in F$ indicate whether the agent is currently a *leader*, a *counter agent*, or *free*, respectively (these are exclusive). Additionally, $\mathsf{N}, \mathsf{I} \in F$, where N indicates whether the bit of the counter is set, and I whether the leader should perform initialisation.

Regarding the input we define $I(X) := (0, \{\mathsf{Ctr}, \mathsf{N}, X\})$ for $X \in \Sigma$.

In the counter, the agents perform usual bitwise increments as in Example 2, though now expressed in terms of the exponent i, and we have to leave one agent in every bit.

$$\begin{aligned} &(i, \mathsf{Ctr}_1, \mathsf{N}_1), (i, \mathsf{Ctr}_1, \mathsf{N}_1) \mapsto (i+1), (i, \mathsf{N}_0) && \text{for } i \in \mathbb{N} \\ &(i, \mathsf{Ctr}_1, \mathsf{N}_a), (i, \mathsf{Ctr}_1, \mathsf{N}_b) \mapsto (i, \mathsf{N}_{a+b}), (i, \mathsf{Ctr}_0, \mathsf{Ldr}_1, \mathsf{I}_1) && \text{for } i \in \mathbb{N}, a+b \leq 1 \end{aligned} \qquad \langle \mathsf{counter} \rangle$$

This uses the compact notation for transitions introduced above. Consider the first line. If two agents with value i are both responsible for the counter and have their N flag set to 1, then, regardless of any other flags, the outcome is as follows: The first agent increments i (leaving every flag unchanged), and the second agents sets N to 0, again leaving the rest as is.

For the second line, if – in the same type of encounter – at most one of the two bits N_a and N_b was set, then one of the agents unsets his counter flag and becomes a leader with I flag set to 1.

This is the way for agents to originally set the leader flag. Since we want to have only one leader, we execute a leader election subprotocol. Every time a leader is eliminated, it moves into Free, and the remaining leader re-initialises.

$$\begin{aligned} &(i, \mathsf{Ldr}_1), (j, \mathsf{Ldr}_1) \mapsto (i, \mathsf{I}_1), (0, \mathsf{Ldr}_0, \mathsf{Free}_1) && \text{for } i, j \in \mathbb{N}, i \geq j \\ &(i, \mathsf{Ldr}_1), (j, \mathsf{Ctr}_1) \mapsto (j, \mathsf{I}_1), (j) && \text{for } i, j \in \mathbb{N}, i < j \end{aligned} \qquad \langle \mathsf{leader} \rangle$$

The second line causes the leader to eventually point to the most significant bit of n.

Let $\delta_{\text{init}} := \langle \mathsf{counter} \rangle \cup \langle \mathsf{leader} \rangle$. For the following proof, as well as later sections, it will be convenient to denote the value of the counter. Given a configuration C and $X \in F$ we write $\text{val}(C, X) := \sum_{i \in \mathbb{N}} 2^i C((i, \mathsf{Ctr}_1, X_1))$. For example, the goal of the initialisation is to ensure $\text{val}(C, \mathsf{N}) = n$ at all times.

We say that a configuration is *initialised*, if it has

(1) exactly one agent in $(l_n, \mathsf{Ldr}_1, \mathsf{Ctr}_0)$,
(2) exactly one agent in $(i, \mathsf{Ctr}_1, \mathsf{N}_{b_i})$, for $i = 0, ..., l_n$ and b_i the i-th bit of n, and
(3) all other agents in $(*, \mathsf{Ldr}_0, \mathsf{Ctr}_0)$.

▶ **Lemma 10.** *Assume that each transition $t \in \delta \setminus \delta_{\text{init}}$ leaves flags $\mathsf{Ldr}, \mathsf{Ctr}, \mathsf{N}$ unchanged, and does not affect levels of agents with the Ldr or Ctr flag. $\mathcal{P}$ eventually reaches an initialised configuration with an agent in $(l_n, \mathsf{Ldr}_1, \mathsf{I}_1)$, and will remain in an initialised configuration.*

Proof. We will show that eventually such a configuration is reached via a $\langle \mathsf{leader} \rangle$ transition. Since transitions in δ_{init} observe only flags $\mathsf{Ldr}, \mathsf{Ctr}, \mathsf{N}$ and levels of leader and counter agents, which by assumption no other transition can change, we can disregard all transitions in $\delta \setminus \delta_{\text{init}}$ for the purposes of this proof.

We have that $\text{val}(C, \mathsf{N})$ is invariant in all reachable configurations C, as no transition changes its value. Further, in an initial configuration we have $\text{val}(C, \mathsf{N}) = 2^0|C| = n$. Hence the level of any agent with flags Ctr and N is at most l_n.

Furthermore, let n_i denote the number of counter agents at level i. Then $(n_0, ...)$ decreases lexicographically with every $\langle \mathsf{counter} \rangle$ transition. As $\langle \mathsf{counter} \rangle$ is enabled as long as we have two counter agents on the same level, eventually we will have exactly one agent in (i, Ctr_1) for every $i = 0, ..., l_n$, and by the invariant $\text{val}(C, \mathsf{N})$ the N flag corresponds to the binary representation of n, proving (2).

Therefore eventually no more leaders are created and transition $\langle\mathsf{leader}\rangle$ leaves exactly one leader. All other agents are then necessarily in states $(*, \mathsf{Ldr}_0, \mathsf{Ctr}_0)$, proving (3). Once the last $\langle\mathsf{leader}\rangle$ transition occurs, flag I is set on the leader and it has level l_n, showing (1). ◀

4.3 The Counter

We created a counter during initialisation, which now contains the precise number of agents. To perform arithmetic on this counter, we designate a helper agent that executes one operation at a time. This agent uses flags $F_{\text{counter}} := \{\mathsf{Clr}, \mathsf{Incr}, \mathsf{Cmp}, \mathsf{Swap}, \mathsf{Done}\}$ to store the operation it is currently executing, and it uses its level to iterate over the bits of the counter. Formally, we say that an agent is a *(counter) helper*, if it has one of the flags in F_{counter}.

The value stored in the counter using the N flag is immutable (to satisfy the assumptions of Lemma 10), so we use flags A, B to store two additional values in the counter agents.

The first operation clears the value in A, i.e. sets it to zero.

$$\begin{aligned} (i, \mathsf{Clr}_1), (i, \mathsf{Ctr}_1) &\mapsto (i+1), (i, \mathsf{A}_0) && \text{for } i \in \mathbb{N} \\ (i+1, \mathsf{Clr}_1), (i, \mathsf{Ldr}_1) &\mapsto (0, \mathsf{Clr}_0, \mathsf{Done}_1), (i) \end{aligned} \qquad \langle\mathsf{clear}\rangle$$

It iterates over each bit using the level. To detect that the end has been reached, the helper communicates with the leader, which always has level l_n.

To access the value stored in B, we create an operation that swaps it with A. It proceeds in much the same way.

$$\begin{aligned} (i, \mathsf{Swap}_1), (i, \mathsf{Ctr}_1, \mathsf{A}_a, \mathsf{B}_b) &\mapsto (i+1), (i, \mathsf{A}_b, \mathsf{B}_a) && \text{for } i \in \mathbb{N}, a, b \in \{0,1\} \\ (i+1, \mathsf{Swap}_1), (i, \mathsf{Ldr}_1) &\mapsto (0, \mathsf{Swap}_0, \mathsf{Done}_1), (i) \end{aligned} \qquad \langle\mathsf{swap}\rangle$$

Incrementing is slightly more involved, but only because we do multiple things: we increase the value in A by 1, and then compare it with N. If they match, the value of A is cleared and the helper sets flag R to indicate whether this happened.

$$\begin{aligned} (i, \mathsf{Incr}_1), (i, \mathsf{Ctr}_1, \mathsf{A}_1) &\mapsto (i+1), (i, \mathsf{A}_0) && \text{for } i \in \mathbb{N} \\ (i, \mathsf{Incr}_1), (i, \mathsf{Ctr}_1, \mathsf{A}_0) &\mapsto (0, \mathsf{Incr}_0, \mathsf{Cmp}_1), (i, \mathsf{A}_1) && \text{for } i \in \mathbb{N} \\ (i, \mathsf{Cmp}_1), (i, \mathsf{Ctr}_1, \mathsf{A}_a, \mathsf{N}_a) &\mapsto (i+1), (i) && \text{for } i \in \mathbb{N}, a \in \{0,1\} \\ (i, \mathsf{Cmp}_1), (i, \mathsf{Ctr}_1, \mathsf{A}_a, \mathsf{N}_{1-a}) &\mapsto (0, \mathsf{Cmp}_0, \mathsf{Done}_1, \mathsf{R}_0), (i) && \text{for } i \in \mathbb{N}, a \in \{0,1\} \\ (i+1, \mathsf{Cmp}_1), (i, \mathsf{Ldr}_1) &\mapsto (0, \mathsf{Cmp}_0, \mathsf{Clr}_1, \mathsf{R}_1), (i) && \text{for } i \in \mathbb{N} \end{aligned} \qquad \langle\mathsf{incr}\rangle$$

Let $\delta_{\text{counter}} := \langle\mathsf{clear}\rangle \cup \langle\mathsf{swap}\rangle \cup \langle\mathsf{incr}\rangle$.

▶ **Observation 11.** *Let C denote an initialised configuration with exactly one counter helper in state $(0, S)$. If only transitions in δ_{counter} are executed, C eventually reaches a configuration C' with*

(1) *exactly one counter helper in state $(0, S')$, where $S' \cap F_{\text{counter}} = \{\mathsf{Done}\}$,*
(2) $\operatorname{val}(C', \mathsf{A}) = 0$*, if* $\mathsf{Clr} \in S$*,*
(3) $\operatorname{val}(C', \mathsf{A}) = \operatorname{val}(C, \mathsf{B})$*, and* $\operatorname{val}(C', \mathsf{B}) = \operatorname{val}(C, \mathsf{A})$*, if* $\mathsf{Swap} \in S$*,*
(4) $\operatorname{val}(C', \mathsf{A}) = \operatorname{val}(C, \mathsf{A}) + 1$ *and* $\mathsf{R} \notin S'$*, if* $\mathsf{Incr} \in S$ *and* $\operatorname{val}(C, \mathsf{A}) + 1 < \operatorname{val}(C, \mathsf{N})$*,*
(5) $\operatorname{val}(C', \mathsf{A}) = 0$ *and* $\mathsf{R} \in S'$*, if* $\mathsf{Incr} \in S$ *and* $\operatorname{val}(C, \mathsf{A}) + 1 = \operatorname{val}(C, \mathsf{N})$*.*

In cases (2), (4), and (5), we also have $\operatorname{val}(C', \mathsf{B}) = \operatorname{val}(C, \mathsf{B})$*.*

Proof. Each operation iterates through the bits of the counter and performs the operations according to the above specification. Once the helper reaches level $l_n + 1$, we use Lemma 10 to deduce the existence of a leader at level l_n, causing the helper to move to Done. We also remark that the increment operation cannot overflow, as (by specification) $\mathrm{val}(C, \mathsf{A}) + 1 \leq \mathrm{val}(C, \mathsf{N})$. ◀

4.4 Loops

A common pattern is to iterate over all agents. To this end, we implement a loop functionality, which causes a loop body to be executed precisely $n - 1$ times.

$$\begin{aligned}(*, \mathsf{Loop}_1, \mathsf{Body}_0), (*, \mathsf{Done}_1) &\mapsto (*, \mathsf{Loop}_0, \mathsf{LoopA}_1), (0, \mathsf{Done}_0, \mathsf{Incr}_1)\\ (*, \mathsf{LoopA}_1), (*, \mathsf{Done}_1, \mathsf{R}_0) &\mapsto (*, \mathsf{LoopA}_0, \mathsf{Loop}_1, \mathsf{Body}_1), (*) \qquad \langle\mathsf{loop}\rangle\\ (*, \mathsf{LoopA}_1), (*, \mathsf{Done}_1, \mathsf{R}_1) &\mapsto (*, \mathsf{LoopA}_0, \mathsf{End}_1), (*)\end{aligned}$$

This transition is to be understood as a template. Any agent can set flag Loop, and $\langle\mathsf{loop}\rangle$ will then interact with the counter, and set flag Body. The agent must then execute another transition removing flag Body, to commence another iteration of the loop. At some point, $\langle\mathsf{loop}\rangle$ will instead indicate that the loop is finished, by setting flag End.

4.5 Cleanup

After the initialisation of Section 4.2, most agents are in some state in $(*, \mathsf{Ldr}_0, \mathsf{Ctr}_0)$. We now want to move all of them into state $(0, \{\mathsf{Free}\})$, and move the leader to $(l_n, \{\mathsf{Ldr}, \mathsf{Start}\})$. (For intuitive explanations we sometimes elide, as here, the flags corresponding to the input Σ, but the transitions take care to not inadvertently clear them.)

During the cleanup, we need one helper agent to perform operations on the counter. The leader will appoint one such agent and mark it using Q. However, it is unavoidable that sometimes such an agent may already exist. Therefore, any counter helper can cause the leader to reset, and during a reset the leader moves any such agents to $(0, \{\mathsf{Free}, \mathsf{T}\})$. Additionally, while resetting the leader sets flag T on any agent it encounters.

$$\begin{aligned}(*, \mathsf{Ldr}_1, \mathsf{I}_0), (*, \mathsf{Q}_1) &\mapsto (*, \mathsf{I}_1), (*)\\ (*, \mathsf{Ldr}_1, \mathsf{I}_1), (*, \mathsf{Ldr}_0, \mathsf{Ctr}_0) &\mapsto (*), (0, (F \setminus \Sigma)_0, \mathsf{Free}_1, \mathsf{T}_1) \qquad \langle\mathsf{reset}\rangle\\ (*, \mathsf{Ldr}_1, \mathsf{I}_1), (*, \mathsf{Ctr}_1, \mathsf{T}_0) &\mapsto (*), (*, \mathsf{T}_1)\end{aligned}$$

For the actual cleanup, the leader first appoints one free agent as helper, then uses the loop template from the previous section to iterate over all agents. Free agents are moved to $(0, \{\mathsf{Free}\})$, and all other agents are left as-is. At the end of the loop, the helper is moved as well, and the leader enters Start, indicating that cleanup is complete. The following transition $\langle\mathsf{cleanup}\rangle$ part 1 is the only transition which unsets the I flag.

$$\begin{aligned}(*, \mathsf{Ldr}_1, \mathsf{I}_1), (*, \mathsf{Free}_1) &\mapsto \begin{matrix}(*, (F \setminus \Sigma)_0, \mathsf{Ldr}_1, \mathsf{Loop}_1),\\ (0, \mathsf{Free}_0, \mathsf{Clr}_1, \mathsf{T}_1, \mathsf{Q}_1)\end{matrix}\\ (*, \mathsf{Ldr}_1, \mathsf{Body}_1, \mathsf{Start}_0), (*, \mathsf{T}_1) &\mapsto (*, \mathsf{Body}_0), (*, \mathsf{T}_0) \qquad \langle\mathsf{cleanup}\rangle\\ (*, \mathsf{Ldr}_1, \mathsf{End}_1, \mathsf{Start}_0), (*, \mathsf{Done}_1) &\mapsto (*, \mathsf{End}_0, \mathsf{Start}_1), (0, (F \setminus \Sigma)_0, \mathsf{Free}_1)\end{aligned}$$

Now we are ready to prove that eventually the protocol reaches a “clean” configuration as in the following lemma. Let $\delta_{\text{cleanup}} := \langle\mathsf{clear}\rangle \cup \langle\mathsf{swap}\rangle \cup \langle\mathsf{incr}\rangle \cup \langle\mathsf{loop}\rangle \cup \langle\mathsf{reset}\rangle \cup \langle\mathsf{cleanup}\rangle$.

▶ **Lemma 12.** *Assume that the assumptions of Lemma 10 hold, and that every transition in* $\delta \setminus (\delta_{\text{init}} \cup \delta_{\text{cleanup}})$

(a) *does not change* I *or* Start*,*

(b) *does not reduce the number of counter helpers,*

(c) *does not use any free agent or counter helper with* T *set,*

(d) *does not use any agent with* Ctr *set, and*

(e) *does not only use a counter helper or agents in* $(*, \mathsf{Free}_1)$ *or* $(*, \mathsf{Start}_0)$*.*

Then $\mathcal{P}$ *eventually reaches an initialised configuration with*

(1) *exactly one agent in* $(l_n, \{\mathsf{Ldr}, \mathsf{Start}\})$ *and* $l_n + 1$ *agents in* $(*, \mathsf{Ctr}_1)$*, and*

(2) *all other agents in* $(0, S \cup \{\mathsf{Free}\})$ *for* $S \subseteq \Sigma$*, i.e. only* Free *and input flags are set.*

Proof. Let $\mathcal{C}$ denote the set of initialised configurations with in agent in $(*, \mathsf{Ldr}_1, \mathsf{I}_1)$.

Lemma 10 guarantees that we reach a configuration $C_1 \in \mathcal{C}$. As stated there, all configurations reachable from C_1 are initialised. We start by arguing that C_1 reaches a configuration C_2 with exactly one counter helper and one leader with I unset.

First, we note that it is *possible* to reach such a C_2, by executing line 2 of $\langle\mathsf{reset}\rangle$ to remove all counter helpers, and then executing the first line of $\langle\mathsf{cleanup}\rangle$ to create one counter helper and unset I. So any fair run from C_1 that does not reach such a C_2 must avoid configurations in $\mathcal{C}$ eventually. (If it visited $\mathcal{C}$ infinitely often, by fairness it would have to reach C_2 at some point.)

So we now assume that C_1 is the last configuration in $\mathcal{C}$ on that run. The only possibility to leave $\mathcal{C}$ is to have the leader clear I, which by assumption (a) can only be done in $\langle\mathsf{cleanup}\rangle$. This transition creates a counter helper; since we do not reach C_2 we thus must have multiple such helpers.

By assumption (b), the number of counter helpers can only be reduced by a transition in $\delta_{\text{init}} \cup \delta_{\text{cleanup}}$. Inspecting these transitions, the only candidates are line 2 of $\langle\mathsf{reset}\rangle$ and line 3 of $\langle\mathsf{cleanup}\rangle$. The former is only enabled at configurations in $\mathcal{C}$. The latter reduces the number of counter helpers by 1 and sets flag Start on the leader. This flag, by assumption (a), cannot be cleared by any transition other than line 1 of $\langle\mathsf{cleanup}\rangle$. (Note that line 3 modifies an agent that is not the leader, and there is only one leader since we are operating within initialised configurations.)

Since Start prevents further reductions in the number of counter helpers, at least one such helper remains. Therefore, it is possible to execute the first line of $\langle\mathsf{reset}\rangle$ and move back to $\mathcal{C}$. By fairness, this happens eventually, contradicting our assumption that $\mathcal{C}$ is visited finitely often and C_2 not reached, proving our first claim.

Reaching such a C_2 must be done by line 1 of $\langle\mathsf{cleanup}\rangle$ (since no other transition clears I), which clears the counter and initiates a loop. As we have argued, C_2 is initialised and has exactly one counter helper. We now show that all fair runs from C_2 either reach $\mathcal{C}$ or a configuration C_3 fulfilling conditions (1-2).

By assumption (d), transitions outside of $\delta_{\text{init}} \cup \delta_{\text{cleanup}}$ do not interact with the counter, and by (c) cannot interact with the counter helper (since it has T set). The only transitions involving the counter helper in a state other than Done are δ_{counter} and the first line of $\langle\mathsf{reset}\rangle$. Since the latter moves to a configuration in $\mathcal{C}$, we may assume wlog that it does not occur.

Inspecting $\langle\mathsf{loop}\rangle$, line 2 of $\langle\mathsf{cleanup}\rangle$ is only enabled when the counter helper is in Done. Similarly for line 3 of $\langle\mathsf{cleanup}\rangle$. So when we move the helper to another state, we can apply Observation 11 and conclude that it performs its operation correctly. (Transitions outside of δ_{counter} may be executed, but cannot affect either the counter or the counter helper.)

This means that line 3 of $\langle\mathsf{cleanup}\rangle$ is only executed once line 2 has run exactly $n-1$ times. If $C_2(*, \mathsf{Ldr}_0, \mathsf{T}_1) < n-1$, this is not possible, and we go back to $\mathcal{C}$ eventually using line 1 of $\langle\mathsf{reset}\rangle$. Otherwise, T is set on all non-leader agents and we claim that it was set by lines 2-3 of $\langle\mathsf{reset}\rangle$. Namely note that by assumption (c) no transition other than $\langle\mathsf{cleanup}\rangle$ may use the agents in $(*, \mathsf{Free}_1, \mathsf{T}_1)$ at all, and by assumption (e) no transition may be initiated using only the counter helper, the free agents, and the leader without Start.

In that case, all agents with T set must result from lines 2-3 of $\langle\mathsf{reset}\rangle$. Since it resets the (non-input) flags of all non-free agents, the leader will execute line 2 of $\langle\mathsf{cleanup}\rangle$ precisely $n-1$ times, and then execute line 3 once, moving to the desired configuration. ◀

4.6 Digits

Let g be the function, such that $f(x) = g(\lfloor \log x \rfloor)$ for all x. For the simulation of $\mathcal{CM}$, we organise the agents into $f(n) = g(l_n)$ many "digits", which are counters that count up to (roughly) $n/g(l_n)$. They do not work by storing the bits individually, as for the counters of the previous section, but instead digit i is stored by having the appropriate number of agents in state $(i, \mathsf{Digit}_1, \mathsf{N}_1)$.

Overall, the goal is to simulate registers by using multiple digits. For example, consider k digits, where digit i can store a number in $0, ..., n_i - 1$, and currently stores d_i. Then the number stored by this group of digits would be $\sum_{i=1}^{k}(n_1 \cdot ... \cdot n_{i-1})d_i$. This is a generalization of standard base b number systems to allow every digit to have a different base n_i.

In the previous sections, we have made use of a helper agents that could autonomously execute certain tasks (e.g. interacting with the counter). We will continue in this vein and designate a new agent for each task.

We start by distributing the free agents into the $g(l_n)$ digits. This happens in a simple round-robin fashion.

$$\begin{aligned}
(*, \mathsf{Dist}_1), * &\mapsto (0, \mathsf{Dist}_0, \mathsf{DistA}_1, \mathsf{Loop}_1), * \\
(i, \mathsf{DistA}_1, \mathsf{Body}_1), (0, \mathsf{Free}_1, \mathsf{V}_0) &\mapsto (i{-}1, \mathsf{Body}_0), (i, \mathsf{Free}_0, \mathsf{Digit}_1, \mathsf{V}_1) \quad \text{for } i > 0 \\
(*, \mathsf{DistA}_1, \mathsf{Body}_1), (*, \mathsf{Free}_0, \mathsf{V}_0, \mathsf{T}_0) &\mapsto (*, \mathsf{Body}_0), (*, \mathsf{V}_1) \qquad\qquad \langle\mathsf{dist}\rangle \\
(0, \mathsf{DistA}_1), (i, \mathsf{Ldr}_1) &\mapsto (g(i), *), (i) \quad \text{for } i \in \mathbb{N} \\
(*, \mathsf{DistA}_1, \mathsf{End}_1), * &\mapsto (*, \mathsf{DistA}_0, \mathsf{End}_0, \mathsf{DistDone}_1), *
\end{aligned}$$

We use a new flag V to mark agents that have already been seen. This ensures that all available agents are distributed. The restriction to T_0 is necessary to satisfy the assumptions of Lemma 12 – but once the cleanup has successfully completed, no agents will have T set.

Now we implement arithmetic operations on the digits. First, we give a subroutine to detect whether a digit is full (or empty). For the following transition, let $i, j \in \mathbb{N}$, $a, b \in \{0, 1\}$ and $M \subseteq F$, with $(j, M) \notin (i, \mathsf{Digit}_1, \mathsf{M}_a)$ and $(j, M) \in (*, \mathsf{U}_b)$.

$$\begin{aligned}
(*, \mathsf{Det}_1), * &\mapsto (*, \mathsf{Det}_0, \mathsf{DetA}_1, \mathsf{Loop}_1, \mathsf{R}_0), * \\
\begin{matrix}(i, \mathsf{DetA}_1, \mathsf{Body}_1, \mathsf{M}_a, \mathsf{U}_b), \\ (i, \mathsf{Digit}_1, \mathsf{M}_a, \mathsf{U}_b)\end{matrix} &\mapsto (i, \mathsf{Body}_0, \mathsf{R}_1), (*, \mathsf{U}_{1-b}) \qquad\qquad \langle\mathsf{detect}\rangle \\
(i, \mathsf{DetA}_1, \mathsf{Body}_1, \mathsf{M}_a, \mathsf{U}_b), (j, M) &\mapsto (i, \mathsf{Body}_0), (j, \mathsf{U}_{1-b}) \\
(i, \mathsf{DetA}_1, \mathsf{End}_1, \mathsf{U}_b), * &\mapsto (i{+}1, \mathsf{DetA}_0, \mathsf{DetDone}_1, \mathsf{End}_0, \mathsf{U}_{1-b}), *
\end{aligned}$$

This is slightly more involved. Similar to before, we mark agents that have been counted (this time using U). To avoid having to do a second loop which resets U, we instead alternate between U_0 and U_1 every time $\langle\mathsf{detect}\rangle$ is executed. In each iteration, we count agents by setting U to the opposite of the value stored in the digit helper. After the loop has completed, the digit helper then flips its own U flag.

To use this routine on digit i, we move an agent into $(i, \mathsf{Det}_1, \mathsf{M}_b)$, where b indicates whether we want to check that the digit is not empty ($b = 1$) or not full ($b = 0$). The output is returned using the R flag. (For technical reasons, the agent ends in level $i + 1$ – this will be useful when checking multiple digits.)

There are two ways to change the value of a digit $i \in \mathbb{N}$: incrementing and decrementing. Both are analogous, so we only describe the former. The process is straightforward: we check whether digit i is already full; if it is not, we move an agent from $(i, \mathsf{Digit}_1, \mathsf{M}_0)$ to $(i, \mathsf{Digit}_1, \mathsf{M}_1)$. Otherwise the digit overflows; we have to set it to 0 and increment digit $i + 1$. (This is simply adding 1 to a number represented using multiple digits in some base.)

Similar to before, let $i, j \in \mathbb{N}$, $b \in \{0, 1\}$ and $M \subseteq F$, with $(j, M) \notin (i, \mathsf{Digit}_1)$ and $(j, M) \in (*, \mathsf{W}_b)$.

$$\begin{aligned}
(i, \mathsf{DigIncr}_1), (*, \mathsf{DetDone}_1) &\mapsto \begin{array}{l}(i, \mathsf{DigIncr}_0, \mathsf{DigIncrA}_1),\\ (i, \mathsf{DetDone}_0, \mathsf{Det}_1, \mathsf{M}_0)\end{array}\\
(*, \mathsf{DigIncrA}_1), (*, \mathsf{DetDone}_1, \mathsf{R}_1) &\mapsto (*, \mathsf{DigIncrA}_0, \mathsf{DigIncrB}_1), (*)\\
(i, \mathsf{DigIncrB}_1), (i, \mathsf{Digit}_1, \mathsf{M}_0) &\mapsto (0, \mathsf{DigIncrB}_0, \mathsf{DigDone}_1), (*, \mathsf{M}_1)\\
(*, \mathsf{DigIncrA}_1), (*, \mathsf{DetDone}_1, \mathsf{R}_0) &\mapsto (*, \mathsf{DigIncrA}_0, \mathsf{DigIncrC}_1, \mathsf{Loop}_1), (*)\\
(i, \mathsf{DigIncrC}_1, \mathsf{Body}_1, \mathsf{W}_b), (i, \mathsf{Digit}_1, \mathsf{W}_b) &\mapsto (i, \mathsf{Body}_0), (i, \mathsf{W}_{1-b}, \mathsf{M}_0)\\
(i, \mathsf{DigIncrC}_1, \mathsf{Body}_1, \mathsf{W}_b), (j, M) &\mapsto (i, \mathsf{Body}_0), (i, \mathsf{W}_{1-b})\\
(i, \mathsf{DigIncrC}_1, \mathsf{End}_1, \mathsf{W}_b), * &\mapsto (i+1, \mathsf{DigIncrC}_0, \mathsf{End}_0, \mathsf{DigIncr}_1, \mathsf{W}_{1-b}), *
\end{aligned} \qquad \langle\mathsf{digit}\rangle$$

We define transitions for $\mathsf{DigDecr}$ analogously.

4.7 Counter Machine

In this section, we describe a subprocess that simulates instructions of $\mathcal{CM}$, using the digits of the previous section. Each of the $|\Sigma|+3$ registers is simulated by $K = g(l_n)/(|\Sigma|+3)$ digits. We write $\nu_{l_n}(r)$ for the function that maps each register r to its first digit. In particular, r is then simulated by digits $\nu_{l_n}(r), ..., \nu_{l_n}(r) + K - 1$. Formally, we have $\nu_{l_n}(r) := 1 + K(r - 1)$.

4.7.1 Input

Initially, each agent holds one input in Σ. We need to initialise the $|\Sigma|$ input registers of $\mathcal{CM}$ accordingly. We use a loop to make sure that all agents have been moved. However, both the loop and incrementing the digit use the counter stored in A by the Ctr agents; therefore, we swap A and B to switch between them.

Let $X, Y \in \Sigma$ denote inputs, where X is stored in digits $r, ..., r + K - 1$.

$$\begin{aligned}
(*, \mathsf{Inp}_1, X_1, \mathsf{O}_0), (*, \mathsf{Done}_1) &\mapsto (*, \mathsf{Inp}_0, \mathsf{InpA}_1), (0, \mathsf{Swap}_1)\\
(*, \mathsf{InpA}_1, X_1, \mathsf{O}_0), (*, \mathsf{DigDone}_1) &\mapsto (*, \mathsf{InpA}_0, \mathsf{InpB}_1, X_0, \mathsf{O}_1), (r, \mathsf{DigIncr}_1)\\
(*, \mathsf{InpB}_1), (*, \mathsf{DigDone}_1) &\mapsto (*, \mathsf{InpB}_0, \mathsf{InpC}_1), (*)\\
(*, \mathsf{InpC}_1), (*, \mathsf{Done}_1) &\mapsto (*, \mathsf{InpC}_0, \mathsf{Inp}_1, \mathsf{Loop}_1, \mathsf{Body}_0), (*, \mathsf{Swap}_1)\\
(*, \mathsf{Inp}_1, \mathsf{Body}_1, X_1, \mathsf{O}_1), (*, Y_1, \mathsf{O}_0) &\mapsto (*, X_0, Y_1, \mathsf{O}_0), (*, X_1, Y_0, \mathsf{O}_1)\\
(*, \mathsf{Inp}_1, \mathsf{End}_1, \Sigma_0), * &\mapsto (*, \mathsf{InpDone}), *
\end{aligned} \qquad \langle\mathsf{input}\rangle$$

There are two considerations complicating the implementation of $\langle\mathsf{input}\rangle$. First, the agent in Inp must count its own input. Second, the overall amount of input flags in the population must not change. We ensure the latter by marking agents with O (instead of e.g. consuming the input) and exchanging input flags (second to last line).

4.7.2 Simulating Instructions

Finally, we can start simulating the instructions of the counter machine. There are two types of instructions. Incr instructions increment a register and then go nondeterministically to one of two instructions. Decr instructions decrement a register and go to one of two instructions, depending on whether the resulting value is zero. The counter machine accepts by reaching the last instruction. We make the following assumptions on the behaviour of the counter machine:

(P1) No increment that would cause an overflow is performed, nor is a decrement on an empty register.

(P2) If it is possible to accept from the initial configuration, every fair run will accept eventually.

(P3) Once reaching the final instruction, the counter machine loops and remains there.

Let $\mathcal{L}_1, ..., \mathcal{L}_l$ denote the instructions of the counter machine. The subprocess simulating the machine is led by the agent with flag CM; it stores the current instruction using flag IP^s, with $s \in \{1, ..., l\}$. Fix some instruction $\mathcal{L}_s = (\text{op}, r, s_0, s_1) \in \{\mathsf{Incr}, \mathsf{Decr}\} \times \{1, ..., |\Sigma| + 3\} \times \{1, ..., l\}^2$. If op $= \mathsf{Incr}$, we increment counter $\nu_i(r)$ and move nondeterministically to instruction s_0 or s_1. Let $i \in \mathbb{N}, b \in \{0, 1\}$.

$$(i, \mathsf{CM}_1, \mathsf{IP}^s_1), (*, \mathsf{DigDone}_1) \mapsto (i, \mathsf{IP}^s_0, \mathsf{IP}^{s_b}_1), (\nu_i(r), \mathsf{DigDone}_0, \mathsf{DigIncr}_1) \qquad \langle\textsf{cm-incr}\rangle$$

We remark that the digits have no concept of being grouped into registers – if digit i overflows during an increment, the digit helper moves on to the next digit, even if it "belongs" to a different register. For our purposes, this is not a problem, since property (P1) ensures that the last digit of a register never overflows.

If op $= \mathsf{Decr}$, we decrement counter $\nu_i(r)$ and check whether it is zero. If so, we move to s_1, else to s_0.

$$\begin{aligned}
(i, \mathsf{CM}_1, \mathsf{IP}^s_1), (*, \mathsf{DigDone}_1) &\mapsto \begin{array}{c}(i, \mathsf{IP}^s_0, \mathsf{IPA}^s_1),\\ (\nu_i(r), \mathsf{DigDone}_0, \mathsf{DigDecr}_1)\end{array}\\
(i, \mathsf{CM}_1, \mathsf{IPA}^s_1), (*, \mathsf{DigDone}_1) &\mapsto (i, \mathsf{IPA}^s_0, \mathsf{IPB}^s_1), (*)\\
(i, \mathsf{CM}_1, \mathsf{IPB}^s_1), (*, \mathsf{DetDone}_1) &\mapsto (i, \mathsf{IPB}^s_0, \mathsf{IPC}^s_1), (\nu_i(r), \mathsf{R}_0)\\
(i, \mathsf{CM}_1, \mathsf{IPC}^s_1), (j, \mathsf{DetDone}_1, \mathsf{R}_0) &\mapsto (i), (j, \mathsf{DetDone}_0, \mathsf{Det}_1, \mathsf{M}_1)\\
&\qquad\qquad\qquad \text{for } j < \nu_i(r{+}1)\\
(i, \mathsf{CM}_1, \mathsf{IPC}^s_1), (\nu_i(r{+}1), \mathsf{DetDone}_1, \mathsf{R}_0) &\mapsto (i, \mathsf{IPC}^s_0, \mathsf{IP}^{s_0}_1), (*)\\
(i, \mathsf{CM}_1, \mathsf{IPC}^s_1), (*, \mathsf{DetDone}_1, \mathsf{R}_1) &\mapsto (i, \mathsf{IPC}^s_0, \mathsf{IP}^{s_1}_1), (*)
\end{aligned} \qquad \langle\textsf{cm-decr}\rangle$$

4.7.3 Output

For the population protocol to have an output, we do a standard output broadcast. The agent simulating the counter machine outputs 1 once the machine has reached the last instruction, and 0 otherwise. All other agents copy that output.

$$\begin{aligned}
(*, \mathsf{CM}_l), * &\mapsto (*, \mathsf{Output}_1), *\\
(*, \mathsf{CM}_1, \mathsf{Output}_b), (*) &\mapsto (*), (*, \mathsf{Output}_b) \quad \text{for } b \in \{0, 1\}
\end{aligned} \qquad \langle\textsf{output}\rangle$$

And $O((q, S)) := 1$ if $\mathsf{Output} \in S$, else $O((q, S)) := 0$.

4.7.4 Starting the Simulation

All that remains is initialising the above subprocesses. After cleanup, there will be one unique leader in Start (Lemma 12). It creates the subprocesses for the counter and the digits. Then it starts the subprocess that distributes the agents in to the digits. Once that is finished, the leader starts the initialisation of the input registers, and after that, finally starts the counter machine simulation.

$$\begin{aligned}(*, \mathsf{Start}_1, \mathsf{Go}_0), (*, \mathsf{Free}_1) &\mapsto (*, \mathsf{Go}_1, \mathsf{GoA}_1), (*, \mathsf{Free}_0, \mathsf{Done}_1)\\ (*, \mathsf{Start}_1, \mathsf{GoA}_1), (*, \mathsf{Free}_1) &\mapsto (*, \mathsf{GoA}_0, \mathsf{GoB}_1), (0, \mathsf{Free}_0, \mathsf{DetDone}_1)\\ (*, \mathsf{Start}_1, \mathsf{GoB}_1), (*, \mathsf{Free}_1) &\mapsto (*, \mathsf{GoB}_0, \mathsf{GoC}_1), (0, \mathsf{Free}_0, \mathsf{DigDone}_1)\\ (*, \mathsf{Start}_1, \mathsf{GoC}_1), (*, \mathsf{Free}_1) &\mapsto (*, \mathsf{GoC}_0, \mathsf{GoD}_1), (0, \mathsf{Free}_0, \mathsf{Dist}_1)\\ (*, \mathsf{Start}_1, \mathsf{GoD}_1), (*, \mathsf{DistDone}_1) &\mapsto (*), (0, \mathsf{DistDone}_0, \mathsf{Inp}_1)\\ (*, \mathsf{Start}_1, \mathsf{GoD}_1), (*, \mathsf{InpDone}_1) &\mapsto (*), (0, \mathsf{InpDone}_0, \mathsf{CM}_1, \mathsf{IP}^1)\end{aligned} \qquad \langle\mathsf{go}\rangle$$

This finally allows us to prove Lemma 9:

▶ **Theorem 9.** *Let* $\varepsilon > 0$ *and let* $f \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$ *be reasonable. Then*

$$\mathsf{UNSPACE}(f(n)\log n) \subseteq \mathsf{UPP}(f(n)).$$

Proof. Let $\varphi \in \mathsf{NSPACE}(f(n)\log n)$ denote a predicate, where $\varphi : \mathbb{N}^\Sigma \to \{0,1\}$. Then there is a $2^{cf(n)\log n}$-bounded counter machine $\mathcal{CM}$ deciding φ, for some $c \in \mathbb{N}$, using $\Gamma := \Sigma + 3$ registers. (The three additional counters are usually used to store the tape left of the head, right of the head, and as a temporary area to perform multiplication and division by constants.)

We may assume that $\mathcal{CM}$ never exceeds its bounds (ensuring (P1)). Further, we can assume that $\mathcal{CM}$ stores its inputs in some fashion and may nondeterministically restart, as long as it has not accepted. This yields (P2). Property (P3) can easily be achieved by a syntactic modification.

Furthermore, it is enough to show that our uniform population protocol $\mathcal{P}$ is correct for all inputs $\geq n_0$ for some constant n_0 by possibly taking a product with an $\mathcal{O}(1)$ states population protocol computing φ for small inputs.

We argue that there is a constant β, s.t. the construction from Section 4 can simulate Γ registers that are $2^{cf(n)\log n}$-bounded, using $g(l_n) := \beta f(2^{l_n})$ digits in total. Each digit has at least $(n - l_n - 5)/g(l_n) - 1$ agents and there are $\beta f(2^{l_n})/\Gamma$ digits per register. Taking the logarithm, we obtain

$$\log\Big(\big(\frac{n - l_n - 5}{\beta f(2^{l_n})} - 1\big)^{\beta f(2^{l_n})/\Gamma}\Big) \geq \frac{\beta f(n)}{\Gamma}\log\Big(\frac{n}{2\beta \cdot 2f(n)} - 1\Big) \geq \frac{\beta f(n)}{\Gamma}\log\Big(\frac{n^\varepsilon}{d\beta} - 1\Big)$$

where $d \in \mathbb{N}$ is a constant s.t. $f(n) \leq dn^{1-\varepsilon}$. We can further lower-bound this by $\varepsilon\beta/\Gamma \cdot f(n)\log n - \mathcal{O}(1)$. Choosing a suitably large constant β, this is at least $cf(n)\log n$, as desired.

It remains to argue that our construction is correct. Using lemmas 10 and 12, we know that the protocol eventually reaches a configuration with exactly one leader, a counter initialised to n, and all other agents in a well-defined state. Afterwards, at each step at most one agent can execute a transition, and correctness follows from careful inspection of the transitions defined above. ◀

5 Conclusion

We have characterised the expressive power of population protocols with $f \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$ states. This closes the gap left open by prior research for uniform protocols, and gives the complexity for protocols with $\Theta(\log n)$ or $\Theta(\operatorname{polylog} n)$ states – the most common constructions in the literature. Our characterisation applies to both uniform and non-uniform protocols.

The upper bound uses the Immerman-Szelepcsényi theorem to argue that a nondeterministic space-bounded Turing machine can simulate the protocol and determine whether it has stabilised. Similar arguments can be found in the literature [11].

Our construction is more involved. It uses the standard idea of determining the total number of agents and then performing zero-checks, i.e. checking whether a state is absent by iterating over all agents. Using zero-checks, it is straightforward to simulate counter-machines. There are two main difficulties: First, with only $\mathcal{O}(\log n)$ states, no single agent can store n. Instead, we have to distribute that information over multiple agents (namely those with flag Ctr), and those agents must collaborate to perform computations on that number. Second, it is neither sufficient to use a constant number of counters with n agents, nor to use $f(n)$ counters with constant number of agents (i.e. bits). We must do both at the same time, which results in the Digit agents. This is one main point where our construction improves upon [12] and prevents the loss of log factors.

We have focused on the expressive power of protocols that can run for an arbitrary amount of time. However, time-complexity plays an important role, and many constructions in the literature focus on being fast. Does limitting the running time affect the expressive power? We conjecture that such protocols can be modelled well by randomised, space-bounded Turing machines, but it is unclear whether one can obtain a characterisation in that case.

One important result about constant-state population protocols is the decidability of the verification problem [22] – a natural question is whether this result can be extended to, e.g. protocols with $\Theta(\log n)$ states. Unfortunately, our characterisation answers this question in the negative. This does open the question of whether there exist subclasses that exclude our construction (and may, therefore, have a decidable verification problem), but include known constructions from the literature for e.g. the majority predicate.

Finally, one gap remains for *non-uniform* (or weakly uniform) protocols with $o(\log n)$ states. In particular, is it possible to decide a non-semilinear predicate with $o(\log n)$ states? We conjecture $\mathsf{UNL} := \mathsf{UENC}(\mathsf{NL}) \subseteq \mathsf{WUPP}(\log\log n)$, i.e. there is a (non-uniform) population protocol with $\mathcal{O}(\log\log n)$ states for every predicate in UNL, in particular for $x \cdot y = z$ or for deciding whether a given input x is a prime number.

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *SODA 2017*, pages 2560–2579. SIAM, 2017. doi:10.1137/1.9781611974782.169.

2 Dan Alistarh and Rati Gelashvili. Recent algorithmic advances in population protocols. *SIGACT News*, 49(3):63–73, 2018. doi:10.1145/3289137.3289150.

3 Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. Fast and exact majority in population protocols. In *PODC*, pages 47–56. ACM, 2015. doi:10.1145/2767386.2767429.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC 2004*, pages 290–299. ACM, 2004. doi:10.1145/1011767.1011810.

5 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. In *DISC*, volume 4167 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2006. doi:10.1007/11864219_5.

6 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Comput.*, 20(4):279–304, 2007. doi:10.1007/S00446-007-0040-2.

7 Amanda Belleville, David Doty, and David Soloveichik. Hardness of computing and approximating predicates and functions with leaderless population protocols. In *ICALP*, volume 80 of *LIPIcs*, pages 141:1–141:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.ICALP.2017.141.

8 Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. Time-space trade-offs in population protocols for the majority problem. *Distributed Comput.*, 34(2):91–111, 2021. doi:10.1007/S00446-020-00385-0.

9 Petra Berenbrink, George Giakkoupis, and Peter Kling. Optimal time and space leader election in population protocols. In *STOC*, pages 119–129. ACM, 2020. doi:10.1145/3357713.3384312.

10 Petra Berenbrink, Dominik Kaaser, and Tomasz Radzik. On counting the population size. In *PODC*, pages 43–52. ACM, 2019. doi:10.1145/3293611.3331631.

11 Michael Blondin, Javier Esparza, and Stefan Jaax. Expressive power of broadcast consensus protocols. In *CONCUR*, volume 140 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.CONCUR.2019.31.

12 Olivier Bournez, Johanne Cohen, and Mikaël Rabie. Homonym population protocols. *Theory Comput. Syst.*, 62(5):1318–1346, 2018. doi:10.1007/S00224-017-9833-2.

13 Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. Passively mobile communicating logarithmic space machines. *CoRR*, abs/1004.3395, 2010. arXiv:1004.3395.

14 Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. Passively mobile communicating machines that use restricted space. *Theor. Comput. Sci.*, 412(46):6469–6483, 2011. doi:10.1016/J.TCS.2011.07.001.

15 David Doty and Mahsa Eftekhari. Efficient size estimation and impossibility of termination in uniform dense population protocols. In *PODC*, pages 34–42. ACM, 2019. doi:10.1145/3293611.3331627.

16 David Doty and Mahsa Eftekhari. A survey of size counting in population protocols. *Theor. Comput. Sci.*, 894:91–102, 2021. doi:10.1016/J.TCS.2021.08.038.

17 David Doty and Mahsa Eftekhari. Dynamic size counting in population protocols. In *SAND*, volume 221 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SAND.2022.13.

18 David Doty, Mahsa Eftekhari, Leszek Gasieniec, Eric E. Severson, Przemyslaw Uznanski, and Grzegorz Stachowiak. A time and space optimal stable population protocol solving exact majority. In *FOCS*, pages 1044–1055. IEEE, 2021. doi:10.1109/FOCS52979.2021.00104.

19 David Doty, Mahsa Eftekhari, Othon Michail, Paul G. Spirakis, and Michail Theofilatos. Brief announcement: Exact size counting in uniform population protocols in nearly logarithmic time. In *DISC*, volume 121 of *LIPIcs*, pages 46:1–46:3. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.DISC.2018.46.

20 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. In *DISC*, volume 9363 of *Lecture Notes in Computer Science*, pages 602–616. Springer, 2015. doi:10.1007/978-3-662-48653-5_40.

21 Robert Elsässer and Tomasz Radzik. Recent results in population protocols for exact majority and leader election. *Bull. EATCS*, 126, 2018. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/549/546.

22 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, 54(2):191–215, 2017. doi:10.1007/S00236-016-0272-3.

23 Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter machines and counter languages. *Math. Syst. Theory*, 2(3):265–283, 1968. doi:10.1007/BF01694011.

24 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Nat. Comput.*, 7(4):615–633, 2008. doi:10.1007/S11047-008-9067-Y.