Succinct Population Protocols for Presburger Arithmetic

Michael Blondin
Département d'informatique, Université de Sherbrooke, Sherbrooke, Canada
michael.blondin@usherbrooke.ca

Javier Esparza
Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
esparza@in.tum.de

Blaise Genest
Univ Rennes, CNRS, IRISA, France
blaise.genest@irisa.fr

Martin Helfrich
Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
helfrich@in.tum.de

Stefan Jaax
Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
jaax@in.tum.de

Abstract

In [5], Angluin et al. proved that population protocols compute exactly the predicates definable in Presburger arithmetic (PA), the first-order theory of addition. As part of this result, they presented a procedure that translates any formula φ of quantifier-free PA with remainder predicates (which has the same expressive power as full PA) into a population protocol with $2^{\mathcal{O}(\text{poly}(|\varphi|))}$ states that computes φ. More precisely, the number of states of the protocol is exponential in both the bit length of the largest coefficient in the formula, and the number of nodes of its syntax tree.

In this paper, we prove that every formula φ of quantifier-free PA with remainder predicates is computable by a leaderless population protocol with $\mathcal{O}(\text{poly}(|\varphi|))$ states. Our proof is based on several new constructions, which may be of independent interest. Given a formula φ of quantifier-free PA with remainder predicates, a first construction produces a succinct protocol (with $\mathcal{O}(|\varphi|^3)$ leaders) that computes φ; this completes the work initiated in [8], where we constructed such protocols for a fragment of PA. For large enough inputs, we can get rid of these leaders. If the input is not large enough, then it is small, and we design another construction producing a succinct protocol with *one* leader that computes φ. Our last construction gets rid of this leader for small inputs.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Automata over infinite objects; Theory of computation → Logic and verification

Keywords and phrases Population protocols, Presburger arithmetic, state complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.40

Related Version A full version of this paper is available at [7], `https://arxiv.org/abs/1910.04600`.

Funding *Javier Esparza, Martin Helfrich, Stefan Jaax*: Supported by an ERC Advanced Grant (787367: PaVeS).
Michael Blondin: Supported by a Quebec–Bavaria project funded by the Fonds de recherche du Québec (FRQ), by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the Fonds de recherche du Québec – Nature et technologies (FRQNT).

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 40; pp. 40:1–40:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Population protocols [3, 4] are a model of distributed computation by indistinguishable, mobile finite-state agents, intensely investigated in recent years (see e.g. [2, 10]). Initially introduced to model networks of passively mobile sensors, they have also been applied to the analysis of chemical reactions under the name of chemical reaction networks (see e.g. [14]).

In a population protocol, a collection of agents, called a *population*, randomly interact in pairs to decide whether their initial configuration satisfies a given property, e.g. whether there are initially more agents in some state A than in some state B. Since agents are indistinguishable and finite-state, their configuration at any time moment is completely characterized by the mapping that assigns to each state the number of agents that currently populate it. A protocol is said to *compute a predicate* if for every initial configuration where the predicate holds, the agents eventually reach consensus 1, and they eventually reach consensus 0 otherwise.

In a seminal paper, Angluin et al. proved that population protocols compute exactly the predicates definable in Presburger arithmetic (PA) [5]. As part of the result, for every Presburger predicate Angluin et al. construct a leaderless protocol that computes it. The construction uses the quantifier elimination procedure for PA: every Presburger formula φ can be transformed into an equivalent boolean combination of *threshold predicates* of the form $\boldsymbol{\alpha} \cdot \boldsymbol{x} > \beta$ and *remainder predicates* of the form $\boldsymbol{\alpha} \cdot \boldsymbol{x} \equiv \beta \pmod{m}$, where $\boldsymbol{\alpha}$ is an integer vector, and β, m are integers [12]. Slightly abusing language, we call the set of these boolean combinations *quantifier-free Presburger arithmetic* (QFPA)[1]. Using that PA and QFPA have the same expressive power, Angluin et al. first construct protocols for all threshold and remainder predicates, and then show that the predicates computed by protocols are closed under negation and conjunction.

The construction of [5] is simple and elegant, but it produces large protocols. Given a formula φ of QFPA, let n be the number of bits of the largest coefficient of φ in absolute value, and let m be the number of atomic formulas of φ, respectively. The number of states of the protocols of [5] grows exponentially in both n and m. In terms of $|\varphi|$ (defined as the sum of the number of variables, n, and m) they have $\mathcal{O}(2^{\text{poly}(|\varphi|)})$ states. This raises the question of whether *succinct protocols* with $\mathcal{O}(\text{poly}(|\varphi|))$ states exist for every formula φ of QFPA. We give an affirmative answer by proving that every formula of QFPA has a succinct and leaderless protocol.

Succinct protocols are the state-complexity counterpart of *fast protocols*, defined as protocols running in polylogarithmic parallel time in the size of the population. Angluin et al. showed that every predicate has a fast protocol with a leader [6], but Alistarh et al., based on work by Doty and Soloveichik [9], proved that in the leaderless case some predicates need linear parallel time [1]. Our result shows that, unlike for time complexity, succinct protocols can be obtained for every QFPA formula in both the leaderless case and the case with leaders.

The proof of our result overcomes a number of obstacles. Designing succinct leaderless protocols is particularly hard for inputs with very few input agents, because there are less resources to simulate leaders. So we produce two completely different families of protocols, one for small inputs with $\mathcal{O}(|\varphi|^3)$ agents, and a second for large inputs with $\Omega(|\varphi|^3)$ agents, and combine them appropriately.

[1] Remainder predicates cannot be directly expressed in Presburger arithmetic without quantifiers.

Large inputs. The family for large inputs is based on our previous work [8]. However, in order to obtain leaderless protocols we need a new succinct construction for boolean combinations of atomic predicates. This obstacle is overcome by designing new protocols for threshold and remainder predicates that work under *reversible dynamic initialization*. Intuitively, agents are allowed to dynamically "enter" and "leave" the protocol through the initial states (dynamic initialization). Further, every interaction can be undone (reversibility), until a certain condition is met, after which the protocol converges to the correct output for the current input. We expect protocols with reversible dynamic initialization to prove useful in other contexts, since they allow a protocol designer to cope with "wrong" non-deterministic choices.

Small inputs. The family of protocols for small inputs is designed from scratch. We exploit that there are few inputs of small size. So it becomes possible to design one protocol for each possible size of the population, and combine them appropriately. Once the population size is fixed, it is possible to design agents that check if they have interacted with all other agents. This is used to simulate the *concatenation operator* of sequential programs, which allows for boolean combinations and succinct evaluation of linear combinations.

Relation to previous work. In [8], we designed succinct protocols with leaders for systems of linear equations. More precisely, we constructed a protocol with $\mathcal{O}((m+k)(n+\log m))$ states and $\mathcal{O}(m(n+\log m))$ leaders that computes a given predicate $A\boldsymbol{x} \geq \boldsymbol{c}$, where $A \in \mathbb{Z}^{m\times k}$ and n is the number of bits of the largest entry in A and $\boldsymbol{c}$, in absolute value. Representing $A\boldsymbol{x} \geq \boldsymbol{c}$ as a formula φ of QFPA, we obtain a protocol with $\mathcal{O}(|\varphi|^2)$ states and $\mathcal{O}(|\varphi|^2)$ leaders that computes φ. However, in [8] no succinct protocols for formulas with remainder predicates are given, and the paper makes extensive use of leaders.

Organization. Sections 2 and 3 introduce basic notation and definitions. Section 4 presents the main result. Sections 5 and 6 present the constructions of the protocols for large and small inputs, respectively. Section 7 presents conclusions. For space reasons, several proofs are only sketched. Detailed proofs are given in the full version of this paper [7].

2 Preliminaries

Notation. We write $\mathbb{Z}$ to denote the set of integers, $\mathbb{N}$ to denote the set of non negative integers $\{0,1,\ldots\}$, $[n]$ to denote $\{1,2,\ldots,n\}$, and $\mathbb{N}^E$ to denote the set of all multisets over E, i.e. unordered vectors with components labeled by E. The *size* of a multiset $\boldsymbol{v} \in \mathbb{N}^E$ is defined as $|\boldsymbol{v}| \stackrel{\text{def}}{=} \sum_{e\in E} \boldsymbol{v}(e)$. The set of all multisets over E with size $s \geq 0$ is $E^{\langle s\rangle} \stackrel{\text{def}}{=} \{\boldsymbol{v} \in \mathbb{N}^E : |\boldsymbol{v}| = s\}$. We sometimes write multisets using set-like notation, e.g. $\{\!\!\{a, 2\cdot b\}\!\!\}$ denotes the multiset $\boldsymbol{v}$ such that $\boldsymbol{v}(a) = 1$, $\boldsymbol{v}(b) = 2$ and $\boldsymbol{v}(e) = 0$ for every $e \in E \setminus \{a,b\}$. The empty multiset $\{\!\!\{\}\!\!\}$ is instead denoted $\mathbf{0}$ for readability. For every $\boldsymbol{u},\boldsymbol{v} \in \mathbb{N}^E$, we write $\boldsymbol{u} \geq \boldsymbol{v}$ if $\boldsymbol{u}(e) \geq \boldsymbol{v}(e)$ for every $e \in E$. Moreover, we write $\boldsymbol{u}+\boldsymbol{v}$ to denote the multiset $\boldsymbol{w} \in \mathbb{N}^E$ such that $\boldsymbol{w}(e) \stackrel{\text{def}}{=} \boldsymbol{u}(e) + \boldsymbol{v}(e)$ for every $e \in E$. The multiset $\boldsymbol{u} \ominus \boldsymbol{v}$ is defined analogously with $-$ instead of $+$, provided that $\boldsymbol{u} \geq \boldsymbol{v}$.

Presburger arithmetic. *Presburger arithmetic* (PA) is the first-order theory of $\mathbb{N}$ with addition, i.e. $\mathsf{FO}(\mathbb{N},+)$. For example, the PA formula $\psi(x,y,z) = \exists x' \exists z' (x = x'+x') \wedge (y = z+z') \wedge \neg(z' = 0)$ states that x is even and that $y > z$. It is well-known that for every formula

of PA there is an equivalent formula of quantifier-free Presburger arithmetic (QFPA) [13], the theory with syntax given by the grammar

$$\varphi(\boldsymbol{v}) ::= \boldsymbol{a} \cdot \boldsymbol{v} > b \mid \boldsymbol{a} \cdot \boldsymbol{v} \equiv_c b \mid \varphi(\boldsymbol{v}) \wedge \varphi(\boldsymbol{v}) \mid \varphi(\boldsymbol{v}) \vee \varphi(\boldsymbol{v}) \mid \neg\varphi(\boldsymbol{v})$$

where $\boldsymbol{a} \in \mathbb{Z}^X$, $b \in \mathbb{Z}$, $c \in \mathbb{N}_{\geq 2}$, and $\equiv_c$ denotes equality modulo c. For example, the formula $\psi(x, y, z)$ above is equivalent to $(x \equiv_2 0) \wedge (y - z \geq 1)$. Throughout the paper, we refer to any formula of QFPA, or the predicate $\mathbb{N}^X \to \{0, 1\}$ it denotes, as a *predicate*. Predicates of the form $\boldsymbol{a} \cdot \boldsymbol{v} > b$ and $\boldsymbol{a} \cdot \boldsymbol{v} \equiv_c b$ are *atomic*, and they are called *threshold* and *remainder* predicates respectively. The *max-norm* $\|\varphi\|$ of a predicate φ is the largest absolute value among all coefficients occurring within φ. The *length* $\mathrm{len}(\varphi)$ of a predicate φ is the number of boolean operators occurring within φ. The *bit length* of a predicate φ, over variables X, is defined as $|\varphi| \stackrel{\text{def}}{=} \mathrm{len}(\varphi) + \log\|\varphi\| + |X|$. We lift these definitions to sets of predicates in the natural way: given a finite set P of predicates, we define its *size* $\mathrm{size}(P)$ as the number of predicates in P, its *length* as $\mathrm{len}(P) \stackrel{\text{def}}{=} \sum_{\varphi \in P} \mathrm{len}(\varphi)$, its *norm* as $\|P\| \stackrel{\text{def}}{=} \max\{\|\varphi\| : \varphi \in P\}$, and its *bit length* as $|P| \stackrel{\text{def}}{=} \mathrm{size}(P) + \mathrm{len}(P) + \log\|P\| + |X|$. Note that $\mathrm{len}(P) = 0$ iff P only contains atomic predicates.

3 Population protocols

A *population protocol* is a tuple $\mathcal{P} = (Q, T, L, X, I, O)$ where

- Q is a finite set whose elements are called *states*;
- $T \subseteq \{(\boldsymbol{p}, \boldsymbol{q}) \in \mathbb{N}^Q \times \mathbb{N}^Q : |\boldsymbol{p}| = |\boldsymbol{q}|\}$ is a finite set of *transitions* containing the set $\{(\boldsymbol{p}, \boldsymbol{p}) : \boldsymbol{p} \in \mathbb{N}^Q, |\boldsymbol{p}| = 2\}$;
- $L \in \mathbb{N}^Q$ is the *leader multiset*;
- X is a finite set whose elements are called *input variables*;
- $I \colon X \to Q$ is the *input mapping*;
- $O \colon Q \to \{0, 1, \bot\}$ is the *output mapping*.

For readability, we often write $t \colon \boldsymbol{p} \mapsto \boldsymbol{q}$ to denote a transition $t = (\boldsymbol{p}, \boldsymbol{q})$. Given $\Delta \geq 2$, we say that t is Δ-*way* if $|\boldsymbol{p}| \leq \Delta$.

In the standard syntax of population protocols T is a subset of $\mathbb{N}^2 \times \mathbb{N}^2$, and $O \colon Q \to \{0, 1\}$. These differences are discussed at the end of this section.

Inputs and configurations. An *input* is a multiset $\boldsymbol{v} \in \mathbb{N}^X$ such that $|\boldsymbol{v}| \geq 2$, and a *configuration* is a multiset $C \in \mathbb{N}^Q$ such that $|C| \geq 2$. Intuitively, a configuration represents a population of agents where $C(q)$ denotes the number of agents in state q. The *initial configuration* $C_{\boldsymbol{v}}$ *for input* $\boldsymbol{v}$ is defined as $C_{\boldsymbol{v}} \stackrel{\text{def}}{=} L + \{\!\{\boldsymbol{v}(x) \cdot I(x) : x \in X\}\!\}$.

The *support* and *b-support* of a configuration C are respectively defined as $[\![C]\!] \stackrel{\text{def}}{=} \{q \in Q : C(q) > 0\}$ and $[\![C]\!]_b = \{q \in [\![C]\!] : O(q) = b\}$. The *output* of a configuration C is defined as $O(C) \stackrel{\text{def}}{=} b$ if $[\![C]\!]_b \neq \emptyset$ and $[\![C]\!]_{\neg b} = \emptyset$ for some $b \in \{0, 1\}$, and $O(C) \stackrel{\text{def}}{=} \bot$ otherwise. Loosely speaking, if $O(q) = \bot$ then agents in state q have no output, and a population has output $b \in \{0, 1\}$ if all agents *with output* have output b.

Executions. A transition $t = (\boldsymbol{p}, \boldsymbol{q})$ is *enabled* in a configuration C if $C \geq \boldsymbol{p}$, and *disabled* otherwise. Because of our assumption on T, every configuration enables at least one transition. If t is enabled in C, then it can be *fired* leading to configuration $C' \stackrel{\text{def}}{=} C \ominus \boldsymbol{p} + \boldsymbol{q}$, which we denote $C \xrightarrow{t} C'$. For every set of transitions S, we write $C \xrightarrow{S} C'$ if $C \xrightarrow{t} C'$ for some $t \in S$.

We denote the reflexive and transitive closure of $\xrightarrow{S}$ by $\xrightarrow{S^*}$. If S is the set of all transitions of the protocol under consideration, then we simply write $\to$ and $\xrightarrow{*}$.

An execution is a sequence of configurations $\sigma = C_0C_1\cdots$ such that $C_i \to C_{i+1}$ for every $i \in \mathbb{N}$. We write σ_i to denote configuration C_i. The *output* of an execution σ is defined as follows. If there exist $i \in \mathbb{N}$ and $b \in \{0,1\}$ such that $O(\sigma_i) = O(\sigma_{i+1}) = \cdots = b$, then $O(\sigma) \stackrel{\text{def}}{=} b$, and otherwise $O(\sigma) \stackrel{\text{def}}{=} \perp$.

Computations. An execution σ is *fair* if for every configuration D the following holds:

if $|\{i \in \mathbb{N} : \sigma_i \xrightarrow{*} D\}|$ is infinite, then $|\{i \in \mathbb{N} : \sigma_i = D\}|$ is infinite.

In other words, fairness ensures that an execution cannot avoid a configuration forever. We say that a population protocol *computes* a predicate $\varphi\colon \mathbb{N}^X \to \{0,1\}$ if for every $\boldsymbol{v} \in \mathbb{N}^X$ and every fair execution σ starting from $C_{\boldsymbol{v}}$, it is the case that $O(\sigma) = \varphi(\boldsymbol{v})$. Two protocols are *equivalent* if they compute the same predicate. It is known that population protocols compute precisely the Presburger-definable predicates [5, 11].

▶ **Example 1.** Let $\mathcal{P}_n = (Q, T, \mathbf{0}, \{x\}, I, O)$ be the protocol where $Q \stackrel{\text{def}}{=} \{0, 1, 2, 3, \ldots, 2^n\}$, $I(x) \stackrel{\text{def}}{=} 1$, $O(a) = 1 \iff a = 2^n$, and T contains a transition, for each $a, b \in Q$, of the form $(a, b) \mapsto (0, a+b)$ if $a+b < 2^n$, and $(a, b) \mapsto (2^n, 2^n)$ if $a+b \geq 2^n$. It is readily seen that $\mathcal{P}_n$ computes $\varphi(x) \stackrel{\text{def}}{=} (x \geq 2^n)$. Intuitively, each agent stores a number, initially 1. When two agents meet, one of them stores the sum of their values and the other one stores 0, with sums capping at 2^n. Once an agent reaches this cap, all agents eventually get converted to 2^n.

Now, consider the protocol $\mathcal{P}'_n = (Q', T', \mathbf{0}, \{x\}, I', O')$, where $Q' \stackrel{\text{def}}{=} \{0, 2^0, 2^1, \ldots, 2^n\}$, $I'(x) \stackrel{\text{def}}{=} 2^0$, $O'(a) = 1 \iff a = 2^n$, and T' contains a transition for each $0 \leq i < n$ of the form $(2^i, 2^i) \mapsto (0, 2^{i+1})$, and a transition for each $a \in Q'$ of the form $(a, 2^n) \mapsto (2^n, 2^n)$. Using similar arguments as above, it follows that $\mathcal{P}'_n$ also computes φ, but more succinctly: While $\mathcal{P}_n$ has $2^n + 1$ states, $\mathcal{P}'_n$ has only $n + 1$ states.

Types of protocols. A protocol $\mathcal{P} = (Q, T, L, X, I, O)$ is

- *leaderless* if $|L| = 0$, and *has* $|L|$ *leaders* otherwise;
- Δ*-way* if all its transitions are Δ-way;
- *simple* if there exist $\mathtt{f}, \mathtt{t} \in Q$ such that $O(\mathtt{f}) = 0$, $O(\mathtt{t}) = 1$ and $O(q) = \perp$ for every $q \in Q \setminus \{\mathtt{f}, \mathtt{t}\}$ (i.e., the output is determined by the number of agents in $\mathtt{f}$ and $\mathtt{t}$.)

Protocols with leaders and leaderless protocols compute the same predicates [5]. Every Δ-way protocol can be transformed into an equivalent 2-way protocol with a polynomial increase in the number of transitions [8]. Finally, every protocol can be transformed into an equivalent simple protocol with a polynomial increase in the number of states [7].

4 Main result

The main result of this paper is the following theorem:

▶ **Theorem 2.** *Every predicate φ of QFPA can be computed by a leaderless population protocol $\mathcal{P}$ with $\mathcal{O}(\text{poly}(|\varphi|))$ states. Moreover, $\mathcal{P}$ can be constructed in polynomial time.*

To prove Theorem 2, we first provide a construction that uses $\ell \in \mathcal{O}(|\varphi|^3)$ leaders. If there are at least $|\boldsymbol{v}| \geq \ell$ input agents $\boldsymbol{v}$ (*large inputs*), we will show how the protocol can be made leaderless by having agents encode both their state and the state of some leader. Otherwise, $|\boldsymbol{v}| < \ell$ (*small inputs*), and we will resort to a special construction, with a single

leader, that only works for populations of bounded size. We will show how the leader can be simulated collectively by the agents. Hence, we will construct succinct protocols computing φ for large and small inputs, respectively. Formally, we prove:

▶ **Lemma 3.** *Let φ be a predicate over variables X. There exist $\ell \in \mathcal{O}(|\varphi|^3)$ and leaderless protocols $\mathcal{P}_{\geq \ell}$ and $\mathcal{P}_{<\ell}$ with $\mathcal{O}(\text{poly}(|\varphi|))$ states such that:*

(a) *$\mathcal{P}_{\geq \ell}$ computes predicate $(|\boldsymbol{v}| \geq \ell) \to \varphi(\boldsymbol{v})$, and*

(b) *$\mathcal{P}_{<\ell}$ computes predicate $(|\boldsymbol{v}| < \ell) \to \varphi(\boldsymbol{v})$.*

Theorem 2 follows immediately from the lemma: it suffices to take the conjunction of both protocols, which only yields a quadratic blow-up on the number of states, using the classical product construction [3]. The rest of the paper is dedicated to proving Lemma 3. Parts (a) and (b) are shown in Sections 5 and 6, respectively.

In the remainder of the paper, whenever we claim the existence of some protocol $\mathcal{P}$, we also claim polynomial-time constructibility of $\mathcal{P}$ without mentioning it explicitly.

5 Succinct protocols for large populations

We show that, for every predicate φ, there exists a constant $\ell \in \mathcal{O}(|\varphi|^3)$ and a succinct protocol $\mathcal{P}_{\geq \ell}$ computing $(|\boldsymbol{v}| \geq \ell) \to \varphi(\boldsymbol{v})$. Throughout this section, we say that $n \in \mathbb{N}$ is *large* if $n \geq \ell$, and that a protocol *computes φ for large inputs* if it computes $(|\boldsymbol{v}| \geq \ell) \to \varphi(\boldsymbol{v})$.

We present the proof in a top-down manner, by means of a chain of statements of the form "$A \leftarrow B$, $B \leftarrow C$, $C \leftarrow D$, and D". Roughly speaking, and using notions that will be defined in the forthcoming subsections:

- Section 5.1 introduces protocols with helpers, a special class of protocols with leaders. The section shows: φ is computable for large inputs by a succinct leaderless protocol (A), if it is computable for large inputs by a succinct protocol with helpers (B).
- Section 5.2 defines protocols that simultaneously compute a set of predicates. The section proves: (B) holds if the set P of atomic predicates occurring within φ is simultaneously computable for large inputs by a succinct protocol with helpers (C).
- Section 5.3 introduces protocols with reversible dynamic initialization. The section shows: (C) holds if each atomic predicate of P is computable for large inputs by a succinct protocol with helpers and reversible dynamic initialization (D).
- Section 5.4 shows that (D) holds by exhibiting succinct protocols with helpers and reversible dynamic initialization that compute atomic predicates for large inputs.

5.1 From protocols with helpers to leaderless protocols

Intuitively, a protocol with helpers is a protocol with leaders satisfying an additional property: adding more leaders does not change the predicate computed by the protocol. Formally, let $\mathcal{P} = (Q, T, L, X, I, O)$ be a population protocol computing a predicate φ. We say that $\mathcal{P}$ is a protocol *with helpers* if for every $L' \succeq L$ the protocol $\mathcal{P}' = (Q, T, L', X, I, O)$ also computes φ, where $L' \succeq L \stackrel{\text{def}}{=} \forall q \in Q\colon (L'(q) = L(q) = 0 \vee L'(q) \geq L(q) > 0)$. If $|L| = \ell$, then we say that $\mathcal{P}$ is a protocol *with ℓ helpers*.

▶ **Theorem 4.** *Let $\mathcal{P} = (Q, T, L, X, I, O)$ be a Δ-way population protocol with ℓ-helpers computing some predicate φ. There exists a 2-way leaderless population protocol with $\mathcal{O}(\ell \cdot |X| + (\Delta \cdot |T| + |Q|)^2)$ states that computes $(|\boldsymbol{v}| \geq \ell) \to \varphi(\boldsymbol{v})$.*

Proof sketch. By [8, Lemma 3], $\mathcal{P}$ can be transformed into a 2-way population protocol (with helpers[2]) computing the same predicate φ, and with at most $|Q|+3\Delta\cdot|T|$ states. Thus, we assume $\mathcal{P}$ to be 2-way in the rest of the sketch.

For simplicity, assume $X=\{x\}$ and $L=\{\!\{3\cdot q, 5\cdot q'\}\!\}$; that is, $\mathcal{P}$ has 8 helpers, and initially 3 of them are in state q, and 5 are in q'. We describe a leaderless protocol $\mathcal{P}'$ that simulates $\mathcal{P}$ for every input $\boldsymbol{v}$ such that $|\boldsymbol{v}|\geq|L|=\ell$. Intuitively, $\mathcal{P}'$ runs in two phases:

- In the first phase each agent gets assigned a number between 1 and 8, ensuring that each number is assigned to at least one agent (this is the point at which the condition $|\boldsymbol{v}|\geq\ell$ is needed). At the end of the phase, each agent is in a state of the form (x,i), meaning that the agent initially represented one unit of input for variable x, and that it has been assigned number i. To achieve this, initially every agent is placed in state $(x,1)$. Transitions are of the form $\{\!\{(x,i),(x,i)\}\!\}\mapsto\{\!\{(x,i+1),(x,i)\}\!\}$ for every $1\leq i\leq 7$. The transitions guarantee that all but one agent is promoted to $(x,2)$, all but one to $(x,3)$, etc. In other words, one agent is "left behind" at each step.
- In the second phase, an agent's state is a multiset: agents in state (x,i) move to state $\{\!\{I(x),q\}\!\}$ if $1\leq i\leq 3$, and to state $\{\!\{I(x),q'\}\!\}$ if $4\leq i\leq 8$. Intuitively, after this move each agent has been assigned two jobs: simultaneously simulate a regular agent of $\mathcal{P}$ starting at state x, *and* a helper of L starting at state q or q'. Since in the first phase each number is assigned to at least one agent, $\mathcal{P}'$ has at least 3 agents simulating helpers in state q, and at least 5 agents simulating helpers in state q'. There may be many more helpers, but this is harmless, because, by definition, additional helpers do not change the computed predicate.

  The transitions of $\mathcal{P}'$ are designed according to this double role of the agents of $\mathcal{P}'$. More precisely, for all multisets $\boldsymbol{p},\boldsymbol{q},\boldsymbol{p}',\boldsymbol{q}'$ of size two, $\{\!\{\boldsymbol{p},\boldsymbol{q}\}\!\}\mapsto\{\!\{\boldsymbol{p}',\boldsymbol{q}'\}\!\}$ is a transition of $\mathcal{P}'$ iff $(\boldsymbol{p}+\boldsymbol{q})\to(\boldsymbol{p}'+\boldsymbol{q}')$ in $\mathcal{P}$. ◀

5.2 From multi-output protocols to protocols with helpers

A *k-output population protocol* is a tuple $\mathcal{Q}=(Q,T,L,X,I,O)$ where $O\colon[k]\times Q\to\{0,1,\perp\}$ and $\mathcal{Q}_i\stackrel{\text{def}}{=}(Q,T,L,X,I,O_i)$ is a population protocol for every $i\in[k]$, where O_i denotes the mapping such that $O_i(q)\stackrel{\text{def}}{=}O(i,q)$ for every $q\in Q$. Intuitively, since each $\mathcal{Q}_i$ only differs by its output mapping, $\mathcal{Q}$ can be seen as a single population protocol whose executions have k outputs. More formally, $\mathcal{Q}$ *computes* a set of predicates $P=\{\varphi_1,\varphi_2,\ldots,\varphi_k\}$ if $\mathcal{Q}_i$ computes φ_i for every $i\in[k]$. Furthermore, we say that $\mathcal{Q}$ is *simple* if $\mathcal{Q}_i$ is simple for every $i\in[k]$. Whenever the number k is irrelevant, we use the term *multi-output population protocol* instead of k-output population protocol.

▶ **Proposition 5.** *Assume that every finite set A of* atomic *predicates is computed by some $|A|$-way multi-output protocol with $\mathcal{O}(|A|^3)$ helpers and states, and $\mathcal{O}(|A|^5)$ transitions. Every QFPA predicate φ is computed by some simple $|\varphi|$-way protocol with $\mathcal{O}(|\varphi|^3)$ helpers and states, and $\mathcal{O}(|\varphi|^5)$ transitions.*

Proof sketch. Consider a binary tree decomposing the boolean operations of φ. We design a protocol for φ by induction on the height of the tree.

The case where the height is 0, and φ is atomic, is trivial. We sketch the induction step for the case where the root is labeled with $\wedge$, that is $\varphi=\varphi_1\wedge\varphi_2$, the other cases are similar. By induction hypothesis, we have simple protocols $\mathcal{P}_1,\mathcal{P}_2$ computing φ_1,φ_2,

[2] Lemma 3 of [8] deals with leaders and not the more specific case of helpers. Nonetheless, computation under helpers is preserved as the input mapping of $\mathcal{P}$ remains unchanged in the proof of the lemma.

respectively. Let $\mathtt{t}_j, \mathtt{f}_j$ be the output states of $\mathcal{P}_j$ for $j \in \{1,2\}$ such that $O_j(\mathtt{t}_j) = 1$ and $O_j(\mathtt{f}_j) = 0$. We add two new states $\mathtt{t}, \mathtt{f}$ (the output states of the new protocol) and an additional helper starting in state $\mathtt{f}$. To compute $\varphi_1 \wedge \varphi_2$ we add the following transitions for every $b_1 \in \{\mathtt{t}_1, \mathtt{f}_1\}, b_2 \in \{\mathtt{t}_2, \mathtt{f}_2\}$, and $b \in \{\mathtt{t}, \mathtt{f}\}$: $\wr b_1, b_2, b \wr \mapsto \wr b_1, b_2, \mathtt{t} \wr$ if $b_1 = \mathtt{t}_1 \wedge b_2 = \mathtt{t}_2$, and $\wr b_1, b_2, b \wr \mapsto \wr b_1, b_2, \mathtt{f} \wr$ otherwise. The additional helper computes the conjunction as desired. ◀

5.3 From reversible dynamic initialization to multi-output protocols

Let $P = \{\varphi_1, \ldots, \varphi_k\}$ be a set of $k \geq 2$ atomic predicates of arity $n \geq 1$ over a set $X = \{x_1, \ldots, x_n\}$ of variables. We construct a multi-output protocol $\mathcal{P}$ for P of size $\mathrm{poly}(|\varphi_1| + \cdots + |\varphi_k|)$.

Let $\mathcal{P}_1, \ldots, \mathcal{P}_k$ be protocols for $\varphi_1, \ldots, \varphi_k$. Observe that $\mathcal{P}$ cannot be a "product protocol" that executes $\mathcal{P}_1, \ldots, \mathcal{P}_k$ synchronously. Indeed, the states of such a $\mathcal{P}$ are tuples $(q_1, \ldots, q_k)$ of states of $\mathcal{P}_1, \ldots, \mathcal{P}_k$, and so $\mathcal{P}$ would have exponential size in k. Further, $\mathcal{P}$ cannot execute $\mathcal{P}_1, \ldots, \mathcal{P}_k$ asynchronously in parallel, because, given an input $\boldsymbol{x} \in \mathbb{N}^n$, it must dispatch $k \cdot \boldsymbol{x}$ agents ($\boldsymbol{x}$ to the input states of each $\mathcal{P}_j$), but it only has $\boldsymbol{x}$. Such a $\mathcal{P}$ would need $(k-1)|\boldsymbol{x}|$ helpers, which is not possible, because a protocol of size $\mathrm{poly}(|\varphi_1| + \cdots + |\varphi_k|)$ can only use $\mathrm{poly}(|\varphi_1| + \cdots + |\varphi_k|)$ helpers, whatever the input $\boldsymbol{x}$.

The solution is to use a more sophisticated parallel asynchronous computation. Consider two copies of inputs, denoted $\overline{X} = \{\overline{x}_1, \ldots, \overline{x}_n\}$ and $\underline{X} = \{\underline{x}_1, \ldots, \underline{x}_n\}$. For each predicate φ over X, consider predicate $\tilde{\varphi}$ over $\overline{X} \cup \underline{X}$ satisfying $\tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) = \varphi(k\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}})$ for every $(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) \in \mathbb{N}^{\overline{X} \cup \underline{X}}$. We obtain $\tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) = \varphi(\boldsymbol{x})$ whenever $k\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}} = \boldsymbol{x}$, e.g. for $\overline{\boldsymbol{x}} := \lfloor \frac{\boldsymbol{x}}{k} \rfloor$ and $\underline{\boldsymbol{x}} := \boldsymbol{x} \bmod k$. With this choice, $\mathcal{P}$ needs to dispatch a total of $k\,(|\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}}|) \leq |\boldsymbol{x}| + n \cdot (k-1)^2$ agents to compute $\tilde{\varphi}_1(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}), \ldots, \tilde{\varphi}_k(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}})$. That is, $n \cdot (k-1)^2$ helpers are sufficient to compute $\mathcal{P}$. Formally, we define $\tilde{\varphi}$ in the following way:

$$\text{For } \varphi(\boldsymbol{x}) = \left(\sum_{i=1}^{n} \alpha_i x_i > \beta\right), \text{ we define } \tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) := \left(\sum_{i=1}^{n} (k \cdot \alpha_i)\overline{x}_i + \alpha_i \underline{x}_i > \beta\right)$$

and similarly for modulo predicates. For instance, if $\varphi(x_1, x_2) = 3x_1 - 2x_2 > 6$ and $k = 4$, then $\tilde{\varphi}(\overline{x}_1, \underline{x}_1, \overline{x}_2, \underline{x}_2) = 12\overline{x}_1 + 3\underline{x}_1 - 8\overline{x}_2 - 2\underline{x}_2 > 6$. As required, $\tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) = \varphi(k\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}})$.

Let us now describe how the protocol $\mathcal{P}$ computes $\tilde{\varphi}_1(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}), \ldots, \tilde{\varphi}_k(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}})$. Let $\tilde{\mathcal{P}}_1, \ldots, \tilde{\mathcal{P}}_k$ be protocols computing $\tilde{\varphi}_1, \ldots, \tilde{\varphi}_k$. Let $X = \{\mathtt{x_1}, \ldots, \mathtt{x_n}\}$ be the input states of $\mathcal{P}$, and let $\overline{\mathtt{x}}_1^{\mathtt{j}}, \ldots, \overline{\mathtt{x}}_{\mathtt{n}}^{\mathtt{j}}$ and $\underline{\mathtt{x}}_1^{\mathtt{j}}, \ldots, \underline{\mathtt{x}}_{\mathtt{n}}^{\mathtt{j}}$ be the input states of $\tilde{\mathcal{P}}_j$ for every $1 \leq j \leq k$. Protocol $\mathcal{P}$ repeatedly chooses an index $1 \leq i \leq n$, and executes one of these two actions: (a) take k agents from $\mathtt{x_i}$, and dispatch them to $\overline{\mathtt{x}}_1^{\mathtt{1}}, \ldots, \overline{\mathtt{x}}_{\mathtt{i}}^{\mathtt{k}}$ (one agent to each state); or (b) take one agent from $\mathtt{x_i}$ and $(k-1)$ helpers, and dispatch them to $\underline{\mathtt{x}}_{\mathtt{i}}^{\mathtt{1}}, \ldots, \underline{\mathtt{x}}_{\mathtt{i}}^{\mathtt{k}}$. The index and the action are chosen nondeterministically. Notice that if for some input $\mathtt{x}_i$, all ℓ agents of $\mathtt{x_i}$ are dispatched, then $k\overline{\mathtt{x}}_{\mathtt{i}}^{\mathtt{j}} + \underline{\mathtt{x}}_{\mathtt{i}}^{\mathtt{j}} = \ell$ for all j. If all agents of $\mathtt{x_i}$ are dispatched for every $1 \leq i \leq n$, then we say that the *dispatch is correct*.

The problem is that, because of the nondeterminism, the dispatch may or may not be correct. Assume, e.g., that $k = 5$ and $n = 1$. Consider the input $x_1 = 17$, and assume that $\mathcal{P}$ has $n \cdot (k-1)^2 = 16$ helpers. $\mathcal{P}$ may correctly dispatch $\overline{x}_1 = \lfloor \frac{17}{5} \rfloor = 3$ agents to each of $\overline{\mathtt{x}}_{\mathtt{1}}^{\mathtt{1}}, \ldots, \overline{\mathtt{x}}_{\mathtt{5}}^{\mathtt{1}}$ and $\underline{x}_1 = (17 \bmod 5) = 2$ to each of $\underline{\mathtt{x}}_{\mathtt{1}}^{\mathtt{1}}, \ldots, \underline{\mathtt{x}}_{\mathtt{5}}^{\mathtt{1}}$; this gives a total of $(3+2) \cdot 5 = 25$ agents, consisting of the 17 agents for the input plus 8 helpers. However, it may also wrongly dispatch 2 agents to each of $\overline{\mathtt{x}}_{\mathtt{1}}^{\mathtt{1}}, \ldots, \overline{\mathtt{x}}_{\mathtt{5}}^{\mathtt{1}}$ and 4 agents to each of $\underline{\mathtt{x}}_{\mathtt{1}}^{\mathtt{1}}, \ldots, \underline{\mathtt{x}}_{\mathtt{5}}^{\mathtt{1}}$, with a total of $(2+4) \cdot 5 = 30$ agents, consisting of 14 input agents plus 16 helpers. In the second case, each $\mathcal{P}_j$ wrongly computes $\tilde{\varphi}_j(2,4) = \varphi_j(2 \cdot 5 + 4) = \varphi_j(14)$, instead of the correct value $\varphi_j(17)$.

To solve this problem we ensure that $\mathcal{P}$ can always recall agents already dispatched to $\tilde{\mathcal{P}}_1, \ldots, \tilde{\mathcal{P}}_k$ as long as the dispatch is not yet correct. This allows $\mathcal{P}$ to "try out" dispatches until it dispatches correctly, which eventually happens by fairness. For this we design $\mathcal{P}$ so that (i) the atomic protocols $\tilde{\mathcal{P}}_1, \ldots, \tilde{\mathcal{P}}_k$ can work with inputs agents that arrive over time (*dynamic initialization*), and (ii) $\tilde{\mathcal{P}}_1, \ldots, \tilde{\mathcal{P}}_k$ can always return to their initial configuration and send agents back to $\mathcal{P}$, unless the dispatch is correct (*reversibility*). To ensure that $\mathcal{P}$ stops redistributing after dispatching a correct distribution, it suffices to replace each reversing transition $\boldsymbol{p} \mapsto \boldsymbol{q}$ by transitions $\boldsymbol{p} + \wr \mathtt{x_i} \wr \mapsto \boldsymbol{q} + \wr \mathtt{x_i} \wr$, one for each $1 \leq i \leq n$: All these transitions become disabled when $\mathtt{x_1}, \ldots, \mathtt{x_n}$ are not populated.

Reversible dynamic initialization. Let us now formally introduce the class of *protocols with reversible dynamic initialization* that enjoys all properties needed for our construction. A simple protocol with *reversible dynamic initialization* (*RDI-protocol* for short) is a tuple $\mathcal{P} = (Q, T_\infty, T_\dagger, L, X, I, O)$, where $\mathcal{P}_\infty = (Q, T_\infty, L, X, I, O)$ is a simple population protocol, and $T_\dagger$ is the set of transitions making the system reversible, called the *RDI-transitions*.

Let $T \stackrel{\text{def}}{=} T_\infty \cup T_\dagger$, and let $\mathsf{In} \stackrel{\text{def}}{=} \{\mathsf{in}_x : x \in X\}$ and $\mathsf{Out} \stackrel{\text{def}}{=} \{\mathsf{out}_x : x \in X\}$ be the sets of *input* and *output transitions*, respectively, where $\mathsf{in}_x \stackrel{\text{def}}{=} (\mathbf{0}, \wr I(x) \wr)$ and $\mathsf{out}_x \stackrel{\text{def}}{=} (\wr I(x) \wr, \mathbf{0})$. An *initialization sequence* is a finite execution $\pi \in (T \cup \mathsf{In} \cup \mathsf{Out})^*$ from the *initial configuration* L' with $L' \succeq L$. The *effective input* of π is the vector $\boldsymbol{w}$ such that $\boldsymbol{w}(x) \stackrel{\text{def}}{=} |\pi|_{\mathsf{in}_x} - |\pi|_{\mathsf{out}_x}$ for every $x \in X$. Intuitively, a RDI-protocol starts with helpers only, and is dynamically initialized via the input and output transitions.

Let $\mathtt{f}, \mathtt{t} \in Q$ be the unique states of $\mathcal{P}$ with $O(\mathtt{f}) = 0$ and $O(\mathtt{t}) = 1$. For every configuration C, let $[C] \stackrel{\text{def}}{=} \{C' : C'(\mathtt{f}) + C'(\mathtt{t}) = C(\mathtt{f}) + C(\mathtt{t}) \text{ and } C'(q) = C(q) \text{ for all } q \in Q \setminus \{\mathtt{f}, \mathtt{t}\}\}$. Intuitively, all configurations $C' \in [C]$ are equivalent to C in all but the output states.

An RDI-protocol is required to be *reversible*, that is for every initialization sequence π with effective input $\boldsymbol{w}$, and such that $L' \xrightarrow{\pi} C$ for some $L' \succeq L$, the following holds:

- if $C \xrightarrow{T^*} D$ and $D' \in [D]$, then $D' \xrightarrow{T^*} C'$ for some $C' \in [C]$, and
- $C(I(x)) \leq \boldsymbol{w}(x)$ for all $x \in X$.

Intuitively, an RDI-protocol can never have more agents in an input state than the effective number of agents it received via the input and output transitions. Further, an RDI-protocol can always reverse all sequences that do not contain input or output transitions. This reversal does not involve the states $\mathtt{f}$ and $\mathtt{t}$, which have a special role as output states. Since RDI-protocols have a default output, we need to ensure that the default output state is populated when dynamic initialization ends, and reversal for $\mathtt{f}$ and $\mathtt{t}$ would prevent that.

An RDI-protocol $\mathcal{P}$ *computes* φ if for every initialization sequence π with effective input $\boldsymbol{w}$ such that $L' \xrightarrow{\pi} C$ for some $L' \succeq L$, the standard population protocol $\mathcal{P}_\infty$ computes $\varphi(\boldsymbol{w})$ from C (that is with $T_\dagger$ disabled). Intuitively, if the dynamic initialization terminates, the RDI-transitions $T_\dagger$ become disabled, and then the resulting standard protocol $\mathcal{P}_\infty$ converges to the output corresponding to the dynamically initialized input.

▶ **Theorem 6.** *Assume that for every atomic predicate φ, there exists a $|\varphi|$-way RDI-protocol with $\mathcal{O}(|\varphi|)$ helpers, $\mathcal{O}(|\varphi|^2)$ states and $\mathcal{O}(|\varphi|^3)$ transitions that computes φ. For every finite set P of atomic predicates, there exists a $|P|$-way simple* multi-output *protocol, with $\mathcal{O}(|P|^3)$ helpers and states, and $\mathcal{O}(|P|^5)$ transitions, that computes P.*

5.4 Atomic predicates under reversible dynamic initialization

Lastly, we show that atomic predicates are succinctly computable by RDI-protocols:

▶ **Theorem 7.** *Every atomic predicate φ over variables X can be computed by a simple $|\varphi|$-way population protocol with reversible dynamic initialization that has $\mathcal{O}(|\varphi|)$ helpers, $\mathcal{O}(|\varphi|^2)$ states, and $\mathcal{O}(|\varphi|^3)$ transitions.*

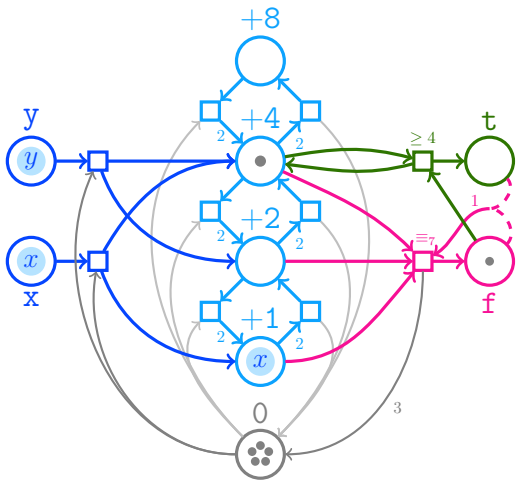

Figure 1 Partial representation of the protocol computing $5x + 6y \geq 4 \pmod 7$ as a Petri net, where places (circles), transitions (squares) and tokens (smaller filled circles) represent respectively states, transitions and agents. Non-helper agents remember their input variable (labeled here within tokens). The depicted configuration is obtained from input $x = 2$, $y = 1$ by firing the bottom leftmost transition (dark blue).

The protocols for arbitrary threshold and remainder predicates satisfying the conditions of Theorem 7, and their correctness proofs, are given in [7]. Note that the threshold protocol is very similar to the protocol for linear inequalities given in Section 6 of [8]. Thus, as an example, we will instead describe how to handle the remainder predicate $5x - y \equiv_7 4$. Note, that the predicate can be rewritten as $(5x + 6y \geq 4 \pmod 7) \wedge (5x + 6y \not\geq 5 \pmod 7)$. As we can handle negations and conjunctions separately in Section 5.2, we will now explain the protocol for $\varphi \stackrel{\text{def}}{=} 5x + 6y \geq 4 \pmod 7$. The protocol is partially depicted in Figure 1 using Petri net conventions for the graphical representation.

The protocol has an *input state* $\mathtt{x}$ for each variable $x \in X$, *output states* $\mathtt{f}$ and $\mathtt{t}$, a *neutral state* $\mathtt{0}$, and *numerical states* of the form $+2^{\mathtt{i}}$ for every $0 \leq i \leq n$, where n is the smallest number such that $2^n > \|\varphi\|$. Initially, (at least) one helper is set to $\mathtt{f}$ and (at least) $2n$ helpers set to $\mathtt{0}$. In order to compute $5x + 6y \geq 4 \pmod 7$ for $x := r$ and $y := s$, we initially place r and s agents in the states $\mathtt{x}$ and $\mathtt{y}$, i.e., the agents in state $\mathtt{x}$ encode the number r in unary, and similarly for $\mathtt{y}$. The blue transitions on the left of Figure 1 "convert" each agents in input states to a binary representation of their corresponding coefficient. In our example, agents in state $\mathtt{x}$ are converted to $\boldsymbol{a}(x) = 5 = 0101_2$ by putting one agent in $\mathtt{4}$ and another one in $\mathtt{1}$. Since two agents are needed to encode 5, the transition "recruits" one helper from state $\mathtt{0}$. Observe that, since the inputs can be arbitrarily large, but a protocol can only use a constant number of helpers, the protocol must reuse helpers in order to convert all agents in input states. This happens as follows. If two agents are in the same power of two, say $+2^{\mathtt{i}}$, then one of them can be "promoted" to $+2^{\mathtt{i}+1}$, while the other one moves to

state `0`, "liberating" one helper. This allows the agents to represent the overall value of all converted agents in the most efficient representation. That is, from any configuration, one can always reach a configuration where there is at most one agent in each place $2^0, \ldots, 2^{n-1}$, there are at most the number of agents converted from input places in place 2^n, and hence there are at least n agents in place 0, thus ready to convert some agent from the input place. Similar to promotions, "demotions" to smaller powers of two can also happen. Thus, the agents effectively shift through all possible binary representations of the overall value of all converted agents. The $\equiv_7$ transition in Figure 1 allows 3 agents in states `4`, `2` and `1` to "cancel out" by moving to state `0`, and it moves the output helper to `f`. Furthermore, there are RDI-transitions that allow to revert the effects of conversion and cancel transitions. These are not shown in Figure 1.

We have to show that this protocol computes φ under reversible dynamic initialization. First note, that while dynamic initialization has not terminated, all transitions have a corresponding reverse transition. Thus, it is always possible to return to wrong initial configurations. However, reversing the conversion transitions can create more agents in input states than the protocol effectively received. To forbid this, each input agent is "tagged" with its variable (see tokens in Figure 1). Therefore, in order to reverse a conversion transitions, the original input agent is needed. This implies, that the protocol is reversible.

Next, we need to argue that the protocol without the RDI-transitions computes φ once the dynamic initialization has terminated. The agents will shift through the binary representations of the overall value. Because of fairness, the $\equiv_7$ transition will eventually reduce the overall value to at most 6. There is a ≥ 4-transition which detects the case where the final value is at least 4 and moves the output helper from `f` to state `t`. Notice that whenever transition $\equiv_7$ occurs, we reset the output by moving the output helper to state `f`.

6 Succinct protocols for small populations

We show that for every predicate φ and constant $\ell = \mathcal{O}(|\varphi|^3)$, there exists a succinct protocol $\mathcal{P}_{<\ell}$ that computes the predicate $(|\boldsymbol{v}| < \ell) \to \varphi(\boldsymbol{v})$. In this case, we say that $\mathcal{P}_{<\ell}$ *computes* φ *for small inputs*. Further, we say that a number $n \in \mathbb{N}$ (resp. an input $\boldsymbol{v}$) is small with respect to φ if $n \leq \ell$ (resp. $|\boldsymbol{v}| \leq \ell$). We present the proof strategy in a top-down manner.

- Section 6.1 proves: There is a succinct leaderless protocol $\mathcal{P}$ that computes φ for small inputs (A), if for every small n some succinct protocol $\mathcal{P}_n$ computes φ for all inputs of size n (B). Intuitively, constructing a succinct protocol for all small inputs reduces to the simpler problem of constructing a succinct protocol for all small inputs of a fixed size.
- Section 6.2 introduces halting protocols. It shows: There is a succinct protocol that computes φ for inputs of size n, if for every *atomic* predicate ψ of φ some halting succinct protocol computes ψ for inputs of size n (C). Thus, constructing protocols for arbitrary predicates reduces to constructing *halting* protocols for atomic predicates.
- Section 6.3 proves (C). Given a threshold or remainder predicate φ and a small n, it shows how to construct a succinct halting protocol that computes φ for inputs of size n.

6.1 From fixed-sized protocols with one leader to leaderless protocols

We now define when a population protocol computes a predicate *for inputs of a fixed size*. Intuitively, it should compute the correct value for every initial configurations of this size; for inputs of other sizes, the protocol may converge to the wrong result, or may not converge.

▶ **Definition 8.** *Let φ be a predicate and let $i \geq 2$. A protocol $\mathcal{P}$* computes *φ for inputs of size i, denoted "$\mathcal{P}$ computes $(\varphi \mid i)$", if for every input $\boldsymbol{v}$ of size i, every fair execution of $\mathcal{P}$ starting at $C_{\boldsymbol{v}}$ stabilizes to $\varphi(\boldsymbol{v})$.*

We show that if, for each small number i, some succinct protocol computes $(\varphi \mid i)$, then there is a single succinct protocol that computes φ for all small inputs.

▶ **Theorem 9.** *Let φ be a predicate over a set of variables X, and let $\ell \in \mathbb{N}$. Assume that for every $i \in \{2, 3, \ldots, \ell - 1\}$, there exists a protocol with at most one leader and at most m states that computes $(\varphi \mid i)$. Then, there is a leaderless population protocol with $\mathcal{O}(\ell^4 \cdot m^2 \cdot |X|^3)$ states that computes $(\boldsymbol{x} < \ell) \to \varphi(\boldsymbol{x})$.*

Proof sketch. Fix a predicate φ and $\ell \in \mathbb{N}$. For every $2 \leq i < \ell$, let $\mathcal{P}_i$ be a protocol computing $(\varphi \mid i)$. We describe the protocol $\mathcal{P} = (Q, T, X, I, O)$ that computes $(\boldsymbol{x} \geq \ell) \vee \varphi(\boldsymbol{x}) \equiv (\boldsymbol{x} < \ell) \to \varphi(\boldsymbol{x})$. The input mapping I is the identity. During the computation, agents never forget their initial state – that is, all successor states of an agent are annotated with their initial state. The protocol initially performs a leader election. Each provisional leader stores how many agents it has "knocked out" during the leader election in a counter from 0 to $\ell - 1$. After increasing the counter to a given value $i < \ell$, it resets the state of i agents and itself to the corresponding initial state of $\mathcal{P}_{i+1}$, annotated with X, and initiates a simulation of $\mathcal{P}_{i+1}$. When the counter of an agent reaches $\ell - 1$, the agent knows that the population size must be $\geq \ell$, and turns the population into a permanent 1-consensus. Now, if the population size i is smaller than ℓ, then eventually a leader gets elected who resets the population to the initial population of $\mathcal{P}_i$. Since $\mathcal{P}_i$ computes $(\varphi \mid i)$, the simulation of $\mathcal{P}_i$ eventually yields the correct output. ◀

6.2 Computing boolean combinations of predicates for fixed-size inputs

We want to produce a population protocol $\mathcal{P}$ for a boolean combination φ of atomic predicates $(\varphi_i)_{i \in [k]}$ for which we have population protocols $(\mathcal{P}_i)_{i \in [k]}$. As in Section 5.3, we cannot use a standard "product protocol" that executes $\mathcal{P}_1, \ldots, \mathcal{P}_k$ synchronously because the number of states would be exponential in k. Instead, we want to simulate the *concatenation* of $(\mathcal{P}_i)_{i \in [k]}$. However, this is only possible if for all $i \in [k]$, the executions of $\mathcal{P}_i$ eventually "halt", i.e. some agents are eventually certain that the output of the protocol will not change anymore, which is not the case in general population protocols. For this reason we restrict our attention to "halting" protocols.

▶ **Definition 10.** *Let $\mathcal{P}$ be a simple protocol with output states* `f` *and* `t`*. We say that $\mathcal{P}$ is a* halting protocol *if every configuration C reachable from an initial configuration satisfies:*

- $C(\mathtt{f}) = 0 \vee C(\mathtt{t}) = 0$,
- $C \xrightarrow{*} C' \wedge C(q) > 0 \Rightarrow C'(q) > 0$ *for every $q \in \{\mathtt{f}, \mathtt{t}\}$ and every configuration C'.*

Intuitively, a halting protocol is a simple protocol in which states `f` and `t` behave like "final states": If an agent reaches $q \in \{\mathtt{f}, \mathtt{t}\}$, then the agent stays in q forever. In other words, the protocol reaches consensus 0 (resp. 1) iff an agent ever reaches `f` (resp. `t`).

▶ **Theorem 11.** *Let $k, i \in \mathbb{N}$. Let φ be a boolean combination of atomic predicates $(\varphi_j)_{j \in [k]}$. Assume that for every $j \in [k]$, there is a simple halting protocol $\mathcal{P}_j = (Q_j, L_j, X, T_j, I_j, O_j)$ with one leader computing $(\varphi_j \mid i)$. Then there exists a simple halting protocol $\mathcal{P}$ that computes $(\varphi \mid i)$, with one leader and $\mathcal{O}(|X| \cdot (\mathrm{len}(\varphi) + |Q_1| + \ldots + |Q_k|))$ states.*

Proof sketch. We only sketch the construction for $\varphi = \varphi_1 \wedge \varphi_2$. The main intuition is that, since $\mathcal{P}_1$ and $\mathcal{P}_2$ are halting, we can construct a protocol that, given an input $\boldsymbol{v}$, first simulates $\mathcal{P}_1$ on $\boldsymbol{v}$, and, after $\mathcal{P}_1$ halts, either halts if $\mathcal{P}_1$ converges to 0, or simulates $\mathcal{P}_2$ on $\boldsymbol{v}$ if $\mathcal{P}_1$ converges to 1. Each agent remembers in its state the input variable it corresponds to, in order to simulate $\mathcal{P}_2$ on $\boldsymbol{v}$. ◀

6.3 Computing atomic predicates for fixed-size inputs

We describe a halting protocol that computes a given threshold predicate for fixed-size inputs.

▶ **Theorem 12.** *Let $\varphi(\boldsymbol{x}, \boldsymbol{y}) \stackrel{def}{=} \boldsymbol{\alpha} \cdot \boldsymbol{x} - \boldsymbol{\beta} \cdot \boldsymbol{y} > 0$. For every $i \in \mathbb{N}$, there exists a halting protocol with one leader and $\mathcal{O}(i^2(|\varphi| + \log i)^3)$ states that computes $(\varphi \mid i)$.*

We first describe a sequential algorithm Greater-Sum$(\boldsymbol{x}, \boldsymbol{y})$, that for every input $\boldsymbol{x}, \boldsymbol{y}$ satisfying $|\boldsymbol{x}| + |\boldsymbol{y}| = i$ decides whether $\boldsymbol{\alpha} \cdot \boldsymbol{x} - \boldsymbol{\beta} \cdot \boldsymbol{y} > 0$ holds. Then we simulate Greater-Sum by means of a halting protocol with i agents.

Since each agent can only have $\mathcal{O}(\log i + \log |\varphi|)$ bits of memory (the logarithm of the number of states), Greater-Sum must use at most $\mathcal{O}(i \cdot (\log i + \log |\varphi|))$ bits of memory, otherwise it cannot be simulated by the agents. Because of this requirement, Greater-Sum cannot just compute, store, and then compare $\boldsymbol{\alpha} \cdot \boldsymbol{x}$ and $\boldsymbol{\beta} \cdot \boldsymbol{y}$; this uses too much memory.

Greater-Sum calls procedures $Probe_1(j)$ and $Probe_2(j)$ that return the j-th bits of $\boldsymbol{\alpha x}$ and $\boldsymbol{\beta y}$, respectively, where $j = 1$ is the most significant bit. Since $|\boldsymbol{x}| \leq i$, and the largest constant in $\boldsymbol{\alpha}$ is at most $||\varphi||$, we have $\boldsymbol{\alpha} \cdot \boldsymbol{x} \leq i \cdot ||\varphi||$, and so $\boldsymbol{\alpha} \cdot \boldsymbol{x}$ has at most $m \stackrel{\text{def}}{=} |\varphi| + \lfloor \log(i) \rfloor + 1$ bits, and the same holds for $\boldsymbol{\beta y}$. So we have $1 \leq j \leq m$. Let us first describe Greater-Sum, and then $Probe_1(j)$; the procedure $Probe_2(j)$ is similar.

Greater-Sum$(\boldsymbol{x}, \boldsymbol{y})$ loops through $j = 1, \ldots, m$. For each j, it calls $Probe_1(j)$ and $Probe_2(j)$. If $Probe_1(j) > Probe_2(j)$, then it answers $\varphi(\boldsymbol{x}, \boldsymbol{y}) = 1$, otherwise it moves to $j + 1$. If Greater-Sum reaches the end of the loop, then it answers $\varphi(\boldsymbol{x}, \boldsymbol{y}) = 0$. Observe that Greater-Sum only needs to store the current value of j and the bits returned by $Probe_1(j)$ and $Probe_2(j)$. Since $j \leq m$, Greater-Sum only needs $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory.

$Probe_1(j)$ uses a decreasing counter $k = m, \ldots, j$ to successively compute the bits $b_1(k)$ of $\boldsymbol{\alpha} \cdot \boldsymbol{x}$, starting at the least significant bit. To compute $b_1(k)$, the procedure stores the carry $c_k \leq i$ of the computation of $b_1(k+1)$; it then computes the sum $s_k := c_k + \boldsymbol{\alpha}(k) \cdot \boldsymbol{x}$ (where $\boldsymbol{\alpha}(k)$ is the k-th vector of bits of $\boldsymbol{\alpha}$), and sets $b_k := s_k \bmod 2$ and $c_{k-1} := s_k \div 2$. The procedure needs $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory for counter k, $\log(i) + 1$ bits for encoding s_k, and $\mathcal{O}(\log(i))$ bits for encoding c_k. So it only uses $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory.

Let us now simulate Greater-Sum$(\boldsymbol{x}, \boldsymbol{y})$ by a halting protocol with one leader agent. Intuitively, the protocol proceeds in rounds corresponding to the counter k. The leader stores in its state the value j and the current values of the program counter, of counter k, and of variables b_k, s_k, and c_k. The crucial part is the implementation of the instruction $s_k := c_k + \boldsymbol{\alpha}(k) \cdot \boldsymbol{x}$ of $Probe_1(j)$. In each round, the leader adds input agents one by one. As the protocol only needs to work for populations with i agents, it is possible for each agent to know if it already interacted with the leader in this round, and for the leader to count the number of agents it has interacted with this round, until it reaches i to start the next round.

7 Conclusion and further work

We have proved that every predicate φ of quantifier-free Presburger arithmetic (QFPA) is computed by a leaderless protocol with poly$(|\varphi|)$ states. Further, the protocol can be computed in polynomial time. The number of states of previous constructions was exponential

both in the bit-length of the coefficients of φ, and in the number of occurrences of boolean connectives. Since QFPA and PA have the same expressive power, every computable predicate has a succinct leaderless protocol. This result completes the work initiated in [8], which also constructed succinct protocols, but only for some predicates, and with the help of leaders.

It is known that protocols with leaders can be exponentially faster than leaderless protocols. Indeed, every QFPA predicate is computed by a protocol with leaders whose expected time to consensus is polylogarithmic in the number of agents [6], while every leaderless protocol for the majority predicate needs at least linear time in the number of agents [1]. Our result shows that, if there is also an exponential gap in state-complexity, then it must be because some family of predicates have protocols with leaders of logarithmic size, while all leaderless families need polynomially many states. The existence of such a family is an open problem.

The question of whether protocols with $\text{poly}(|\varphi|)$ states exist for every PA formula φ, possibly with quantifiers, also remains open. However, it is easy to prove that no algorithm for the construction of protocols from PA formulas runs in time $2^{p(n)}$ for any polynomial p.

▶ **Theorem 13.** *For every polynomial p, every algorithm that accepts a formula φ of PA as input, and returns a population protocol computing φ, runs in time $2^{\omega(p(|\varphi|))}$.*

Therefore, if PA also has succinct protocols, then they are very hard to find.

Our succinct protocols for QFPA have slow convergence (in the usual parallel time model, see e.g. [2]), since they often rely on exhaustive exploration of a number of alternatives, until the right one is eventually hit. The question of whether every QFPA predicate has a succinct *and* fast protocol is very challenging, and we leave it open for future research.

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *Proc. Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2560–2579. SIAM, 2017.

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 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 290–299, 2004. doi:10.1145/1011767.1011810.

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

5 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 292–299, 2006. doi:10.1145/1146381.1146425.

6 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, 2008.

7 Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. Succinct population protocols for Presburger arithmetic, 2019. arXiv:1910.04600.

8 Michael Blondin, Javier Esparza, and Stefan Jaax. Large flocks of small birds: On the minimal size of population protocols. In *Proc. 35th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 16:1–16:14, 2018. doi:10.4230/LIPIcs.STACS.2018.16.

9 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018.

10 Robert Elsässer and Tomasz Radzik. Recent results in population protocols for exact majority and leader election. *Bulletin of the EATCS*, 126, 2018.

11 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.

12 Christoph Haase. A survival guide to Presburger arithmetic. *SIGLOG News*, 5(3):67–82, 2018.

13 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes rendus du I^{er} Congrès des mathématiciens des pays slaves*, pages 192–201, 1929.

14 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008.