



Fast and Succinct Population Protocols for Presburger Arithmetic

Philipp Czerner   

Department of Informatics, Technische Universität München, Germany

Roland Guttenberg  

Department of Informatics, Technische Universität München, Germany

Martin Helfrich   

Department of Informatics, Technische Universität München, Germany

Javier Esparza   

Department of Informatics, Technische Universität München, Germany

Abstract

In their 2006 seminal paper in Distributed Computing, Angluin et al. present a construction that, given any Presburger predicate as input, outputs a leaderless population protocol that decides the predicate. The protocol for a predicate of size m (when expressed as a Boolean combination of threshold and remainder predicates with coefficients in binary) runs in $\mathcal{O}(m \cdot n^2 \log n)$ expected number of interactions, which is almost optimal in n , the number of interacting agents. However, the number of states of the protocol is exponential in m . This is a problem for natural computing applications, where a state corresponds to a chemical species and it is difficult to implement protocols with many states. Blondin et al. described in STACS 2020 another construction that produces protocols with a polynomial number of states, but exponential expected number of interactions. We present a construction that produces protocols with $\mathcal{O}(m)$ states that run in expected $\mathcal{O}(m^7 \cdot n^2)$ interactions, optimal in n , for all inputs of size $\Omega(m)$. For this, we introduce population computers, a carefully crafted generalization of population protocols easier to program, and show that our computers for Presburger predicates can be translated into fast and succinct population protocols.

2012 ACM Subject Classification Theory of computation \rightarrow Distributed computing models

Keywords and phrases population protocols, fast, succinct, population computers

Digital Object Identifier 10.4230/LIPIcs.SAND.2022.11

Related Version *Full Version*: <https://arxiv.org/abs/2202.11601v2> [11]

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

Acknowledgements We thank the anonymous reviewers for many helpful remarks. In particular, one remark led to Lemma 11, which in turn led to a nicer formulation of Theorem 2, one of our main results.

1 Introduction

Population protocols [4, 5] are a model of computation in which indistinguishable, mobile finite-state agents, randomly interact in pairs to decide whether their initial configuration satisfies a given property, modelled as a predicate on the set of all configurations. The decision is taken by *stable consensus*; eventually all agents agree on whether the property holds or not, and never change their mind again. Population protocols are very close to chemical reaction networks, a model in which agents are molecules and interactions are chemical reactions.



© Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza; licensed under Creative Commons License CC-BY 4.0

1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022).

Editors: James Aspnes and Othon Michail; Article No. 11; pp. 11:1–11:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In a seminal paper, Angluin et al. proved that population protocols decide exactly the predicates definable in Presburger arithmetic (PA) [7]. One direction of the result is proved in [5] by means of a construction that takes as input a Presburger predicate and outputs a protocol that decides 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 $\vec{a} \cdot \vec{x} \geq c$ and *remainder* predicates of the form $\vec{a} \cdot \vec{x} \equiv_m c$, where \vec{a} is an integer vector, c and m are integers, and \equiv_m denotes congruence modulo m [14]. 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 two fundamental parameters of a protocol are the expected number of interactions until a stable consensus is reached, and the number of states of each agent. The expected number of interactions divided by the number of agents, also called the parallel execution time, is an adequate measure of the runtime of a protocol when interactions occur in parallel according to a Poisson process [6]. The number of states measures the complexity of an agent. In natural computing applications, where a state corresponds to a chemical species, it is difficult to implement protocols with many states.

Given a formula φ of QFPA, let m be the number of bits of the largest coefficient of φ in absolute value, and let s be the number of atomic formulas of φ , respectively. Let n be the number of agents participating in the protocol. The construction of [5] yields a protocol with $\mathcal{O}(s \cdot n^2 \log n)$ expected interactions. Observe that the protocol does not have a leader (an auxiliary agent helping the other agents to coordinate), and agents have a fixed number of states, independent of the size of the population. Under these assumptions, which are also the assumptions of this paper, every protocol for the majority predicate needs $\Omega(n^2)$ expected interactions [1], and so the construction is nearly optimal.² However, the number of states is $\Omega(2^{m+s})$, or $\Omega(2^{|\varphi|})$ in terms of the number $|\varphi|$ of bits needed to write φ with coefficients in binary. This is well beyond the only known lower bound, showing that for every construction there exist an infinite subset of predicates φ for which the construction produces protocols with $\Omega(|\varphi|^{1/4})$ states [9]. So the constructions of [5], and also those of [6, 3, 13], produce *fast* but *very large* protocols.

In [9, 8] Blondin et al. exhibit a construction that produces *succinct* protocols with $\mathcal{O}(\text{poly}(|\varphi|))$ states. However, they do not analyse their stabilisation time. We demonstrate that they run in $\Omega(2^n)$ expected interactions. Loosely speaking, the reason is the use of transitions that “revert” the effect of other transitions. This allows the protocol to “try out” different distributions of agents, retracing its steps until it hits the right one, but also makes it very slow. So [9, 8] produce *succinct* but *very slow* protocols.

Is it possible to produce protocols that are both *fast* and *succinct*? We give an affirmative answer. We present a construction that yields for every formula φ of QFPA a protocol with $\mathcal{O}(\text{poly}(|\varphi|))$ states and $\mathcal{O}(\text{poly}(|\varphi|) \cdot n^2)$ expected interactions. So our construction achieves optimal stabilisation time in n , and, at the same time, yields more succinct protocols than the construction of [8]. Moreover, for inputs of size $\Omega(|\varphi|)$ (a very mild constraint when agents are molecules), we obtain protocols with $\mathcal{O}(|\varphi|)$ states.

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

² If the model is extended by allowing a *leader* (and one considers the slightly weaker notion of convergence time), or the number of states of an agent is allowed to grow with the population size, $\mathcal{O}(n \cdot \text{polylog}(n))$ interactions can be achieved [6, 3, 2, 13, 12].

Our construction relies on *population computers*, a carefully crafted generalization of the population protocol model of [5]. Population computers extend population protocols in three ways. First, they can exhibit certain *k-way interactions* between more than two agents. Second, they have a more flexible *output condition*, defined by an arbitrary function that assigns an output to every subset of states, instead of to every state.³ Finally, population computers can use *helpers*: auxiliary agents that, like leaders, help regular agents to coordinate themselves but whose number, contrary to leaders, is not known *a priori*. We exhibit succinct population computers for all Presburger predicates in which every run is finite, and show how to translate such population computers into fast and succinct population protocols.

Organization of the paper. We give preliminary definitions in Section 2 and introduce population computers in Section 3. Section 4 gives an overview of the rest of the paper and summarises our main results. Section 5 describes why previous constructions were either not succinct or slow. Section 6 describes population computers for every Presburger predicate. Section 7 converts these computers into succinct population protocols. Section 8 shows that the resulting protocols are also fast.

An extended version of this paper, containing the details of the constructions and all proofs, can be found at [11]. It contains several appendices. Appendix A completes the proofs of Section 5. For the other appendices, there is no one-to-one correspondence to sections of the main paper, instead they are grouped by the construction they analyse. Appendix B concerns the construction of Section 6, but also analyses speed. The four parts of our conversion process are analysed separately, in Appendices C, D, E and F. Appendix G combines the previous to prove the complete conversion theorem. Appendix H summarises the definitions for our speed analyses, and Appendix I contains minor technical lemmata.

2 Preliminaries

Multisets. Let E be a finite set. A multiset over E is a mapping $E \rightarrow \mathbb{N}$, and \mathbb{N}^E denotes the set of all multisets over E . We sometimes write multisets using set-like notation, e.g. $\{a, 2 \cdot b\}$ denotes the multiset v such that $v(a) = 1$, $v(b) = 2$ and $v(e) = 0$ for every $e \in E \setminus \{a, b\}$. The empty multiset $\{\}$ is also denoted \emptyset .

For $E' \subseteq E$, $v(E') := \sum_{e \in E'} v(e)$ is the number of elements in v that are in E' . The *size* of $v \in \mathbb{N}^E$ is $|v| := v(E)$. The *support* of $v \in \mathbb{N}^E$ is the set $\text{supp}(v) := \{e \in E \mid v(e) > 0\}$. If $E \subseteq \mathbb{Z}$, then we let $\text{sum}(v) := \sum_{e \in E} e \cdot v(e)$ denote the sum of all the elements of v . Given $u, v \in \mathbb{N}^E$, $u + v$ and $u - v$ denote the multisets given by $(u + v)(e) := u(e) + v(e)$ and $(u - v)(e) := u(e) - v(e)$ for every $e \in E$. The latter is only defined if $u \geq v$.

Multiset rewriting transitions. A *multiset rewriting transition*, or just a *transition*, is a pair $(r, s) \in \mathbb{N}^E \times \mathbb{N}^E$, also written $r \mapsto s$. A transition $t = (r, s)$ is *enabled* at $v \in \mathbb{N}^E$ if $v \geq r$, and its *occurrence* leads to $v' := v - r + s$, denoted $v \rightarrow_t v'$. We call $v \rightarrow_t v'$ a *step*. The multiset v is *terminal* if it does not enable any transition. An *execution* is a finite or infinite sequence v_0, v_1, \dots of multisets such that $v \rightarrow_{t_1} v_1 \rightarrow_{t_2} \dots$ for some sequence t_1, t_2, \dots of transitions. A multiset v' is *reachable* from v if there is an execution v_0, v_1, \dots, v_k with $v_0 = v$ and $v_k = v'$; we also say that the execution *leads from v to v'* . An execution is a *run* if it is infinite or it is finite and its last multiset is terminal. A run v_0, v_1, \dots is *fair* if it is finite, or it is infinite and for every multiset v , if v is reachable from v_i for infinitely many $i \geq 0$, then $v = v_j$ for some $j \geq 0$.

³ Other output conventions for population protocols have been considered [10].

Presburger arithmetic. Angluin et al. proved that population protocols decide exactly the predicates $\mathbb{N}^k \rightarrow \{0, 1\}$ definable in Presburger arithmetic, the first-order theory of addition, which coincide with the *semilinear* predicates [14]. Using the quantifier elimination procedure of Presburger arithmetic, every Presburger predicate can be represented as a Boolean combination of *threshold* and *remainder* predicates. A predicate $\varphi : \mathbb{N}^v \rightarrow \{0, 1\}$ is a threshold predicate if $\varphi(x_1, \dots, x_v) = (\sum_{i=1}^v a_i x_i \geq c)$, where $a_1, \dots, a_v, c \in \mathbb{Z}$, and a remainder predicate if $\varphi(x_1, \dots, x_v) = (\sum_{i=1}^v a_i x_i \equiv_m c)$, where $a_1, \dots, a_v \in \mathbb{Z}$, $m \geq 1$, $c \in \{0, \dots, m-1\}$, and $a \equiv_m b$ denotes that a is congruent to b modulo m . We call the set of these formulas *quantifier-free Presburger arithmetic*, or QFPA. The *size* of a predicate is the minimal number of bits of a formula of QFPA representing it, with coefficients written in binary.

3 Population Computers

Population computers are a generalization of population protocols that allows us to give very concise descriptions of our protocols for Presburger predicates.

Syntax. A *population computer* is a tuple $\mathcal{P} = (Q, \delta, I, O, H)$, where:

- Q is a finite set of *states*. Multisets over Q are called *configurations*.
- $\delta \subseteq \mathbb{N}^Q \times \mathbb{N}^Q$ is a set of multiset rewriting transitions $r \mapsto s$ over Q such that $|r| = |s| \geq 2$ and $|\text{supp}(r)| \leq 2$. Further, we require that δ is a partial function, so $s_1 = s_2$ for all r, s_1, s_2 with $(r, s_1), (r, s_2) \in \delta$. A transition $r \mapsto s$ is *binary* if $|r| = 2$. We call a population computer *binary* if every transition binary.
- $I \subseteq Q$ is a set of *input states*. An *input* is a configuration C such that $\text{supp}(C) \subseteq \text{supp}(I)$.
- $O : 2^Q \rightarrow \{0, 1, \perp\}$ is an *output function*. The *output* of a configuration C is $O(\text{supp}(C))$. An output function O is a *consensus output* if there is a partition $Q = Q_0 \cup Q_1$ of Q such that $O(Q') = 0$ iff $Q' \subseteq Q_0$, $O(Q') = 1$ iff $Q' \subseteq Q_1$, and $O(Q') = \perp$ otherwise.
- $H \in \mathbb{N}^{Q \setminus I}$ is a multiset of *helper agents* or just *helpers*. A *helper configuration* is a configuration C such that $\text{supp}(C) \subseteq \text{supp}(H)$ and $C \geq H$.

Graphical notation. We visualise population computers as Petri nets (see e.g. Figure 3). Places (circles) and transitions (squares) represent respectively states and transitions. To visualise configurations, we draw agents as tokens (smaller filled circles).

Semantics. Intuitively, a population computer decides which output (0 or 1) corresponds to an input C_I as follows. It adds to the agents of C_I an arbitrary helper configuration C_H of agents to produce the initial configuration $C_I + C_H$. Then it starts the computation and lets it stabilise to configurations of output 1 or output 0. Formally, the *initial configurations* of \mathcal{P} for input C_I are all configurations of the form $C_I + C_H$ for some helper configuration C_H . A run $C_0 C_1 \dots$ *stabilises to* b if there exists an $i \geq 0$ such that $O(\text{supp}(C_i)) = b$ and C_i only reaches configurations C' with $O(\text{supp}(C')) = b$. An input C_I *has output* b if for every initial configuration $C_0 = C_I + C_H$, every fair run starting at C_0 stabilises to b . A population computer \mathcal{P} *decides* a predicate $\varphi : \mathbb{N}^I \rightarrow \{0, 1\}$ if every input C_I has output $\varphi(C_I)$.

Terminating and bounded computers. A population computer is *bounded* if no run starting at an initial configuration C is infinite, and *terminating* if no fair run starting at C is infinite. Observe that bounded population computers are terminating.

Size and adjusted size. Let $\mathcal{P} = (Q, \delta, I, O, H)$ be a population computer. We assume that O is described as a boolean circuit with $\text{size}(O)$ gates. For every transition $t = (r \mapsto s)$ let $|t| := |r|$. The *size* of \mathcal{P} is $\text{size}(\mathcal{P}) := |Q| + |H| + \text{size}(O) + \sum_{t \in \delta} |t|$. If \mathcal{P} is binary, then (as for population protocols) we do not count the transitions and define the *adjusted size* $\text{size}_2(\mathcal{P}) := |Q| + |H| + \text{size}(O)$. Observe that both the size of a transition and the size of the helper multiset are the number of elements, i.e. the size in unary, strengthening our later result about the existence of succinct population computers.

Population protocols. A population computer $\mathcal{P} = (Q, \delta, I, O, H)$ is a *population protocol* if it is binary, has no helpers ($H = \emptyset$), and O is a consensus output. It is easy to see that this definition coincides with the one of [5]. The speed of a binary population computer with no helpers, and so in particular of a population protocol, is defined as follows. We assume a probabilistic execution model in which at configuration C two agents are picked uniformly at random and execute a transition, if possible, moving to a configuration C' (by assumption they enable at most one transition). This is called an *interaction*. Repeating this process, we generate a *random execution* $C_0 C_1 \dots$. We say that the execution *stabilises* at time t if C_t reaches only configurations C' with $O(\text{supp}(C')) = O(\text{supp}(C_t))$, and we say that \mathcal{P} *decides* φ *within* T *interactions* if it decides φ and $\mathbb{E}(t) \leq T$. See e.g. [6] for more details.

Population computers vs. population protocols. Population computers generalise population protocols in three ways:

- They have non-binary transitions, but only those in which the interacting agents populate at most two states. For example, $\langle p, p, q \rangle \mapsto \langle p, q, o \rangle$ (which in the following is written simply as $p, p, q \mapsto p, q, o$) is allowed, but $p, q, o \mapsto p, p, q$ is not.
- They use a multiset H of auxiliary helper agents, but the addition of more helpers does not change the output of the computation. Intuitively, contrary to the case of leaders, agents do not know any upper bound on the number of helpers, and so the protocol cannot rely on this bound for correctness or speed.
- They have a more flexible output condition. Loosely speaking, population computers accept by stabilising the population to an accepting set of states, instead of to a set of accepting states.

4 Overview and Main Results

Given a predicate $\varphi \in QFPA$ over variables x_1, \dots, x_v , the rest of this paper shows how to construct a fast and succinct population protocol deciding φ . First, Section 5 gives an overview of previous constructions and explains why they are not fast or not succinct. Then we proceed in five steps:

1. Construct the predicate $\text{double}(\varphi) \in QFPA$ over variables $x_1, \dots, x_v, x'_1, \dots, x'_v$ by syntactically replacing every occurrence of x_i in φ by $x_i + 2x'_i$. For example, if $\varphi = (x - y \geq 0)$ then $\text{double}(\varphi) = (x + 2x' - y - 2y' \geq 0)$. Observe that $|\text{double}(\varphi)| \in \mathcal{O}(|\varphi|)$.
2. Construct a succinct bounded population computer \mathcal{P} deciding $\text{double}(\varphi)$.
3. Convert \mathcal{P} into a succinct population protocol \mathcal{P}' deciding φ for inputs of size $\Omega(|\varphi|)$.
4. Prove that \mathcal{P}' runs within $\mathcal{O}(n^3)$ interactions.
5. Use a refined running-time analysis to prove that \mathcal{P}' runs within $\mathcal{O}(n^2)$ interactions.

Section 6 constructs bounded population computers for all predicates $\varphi \in QFPA$. This allows us to conduct steps 1 and 2. More precisely, the section proves:

► **Theorem 1.** *For every predicate $\varphi \in QFPA$ there exists a bounded population computer of size $\mathcal{O}(|\varphi|)$ that decides φ .*

Section 7 proves the following conversion theorem (steps 3 and 4).

► **Theorem 2.** *Every bounded population computer of size m deciding $\text{double}(\varphi)$ can be converted into a terminating population protocol with $\mathcal{O}(m^2)$ states which decides φ in at most $\mathcal{O}(f(m)n^3)$ interactions for inputs of size $\Omega(m)$, for some function f .*

Section 8 introduces α -rapid population computers, where $\alpha \geq 1$ is a certain parameter, and uses a more detailed analysis to show that the population protocols of Theorem 2 are in fact smaller and faster (step 5):

► **Theorem 3.**

- (a) *The population computers constructed in Theorem 1 are $\mathcal{O}(|\varphi|^3)$ -rapid.*
- (b) *Every α -rapid population computer of size m deciding $\text{double}(\varphi)$ can be converted into a terminating population protocol with $\mathcal{O}(m)$ states that decides φ in $\mathcal{O}(\alpha m^4 n^2)$ interactions for inputs of size $\Omega(m)$.*

The restriction to inputs of size $\Omega(m)$ is very mild. Moreover, it can be lifted using a technique of [8], at the price of adding additional states (and at no cost regarding asymptotic speed, since the speed of the new protocol only changes for inputs of size $\mathcal{O}(m)$):

► **Corollary 4.** *For every $\varphi \in QFPA$ there exists a terminating population protocol with $\mathcal{O}(\text{poly}(|\varphi|))$ states that decides φ in $\mathcal{O}(f(|\varphi|)n^2)$ interactions, for a function f .*

It is known that the majority predicate can only be decided in $\Omega(n^2)$ interactions by population protocols [1], so — as a general construction — our result is optimal w.r.t. time. Regarding space, an $\Omega(|\varphi|^{1/4})$ lower bound was shown in [9], leaving a polynomial gap.

5 Previous Constructions: Angluin et al. and Blondin et al.

The population protocols for a quantifier free Presburger predicate φ constructed in [5] are not *succinct*, i.e. do not have $\mathcal{O}(|\varphi|^a)$ states for any constant a , and those of [8] are not *fast*, i.e. do not have speed $\mathcal{O}(|\varphi|^a n^b)$ for any constants a, b . We explain why with the help of some examples.

► **Example 5.** Consider the protocol of [5] for the predicate $\varphi = (x - y \geq 2^d)$. The states are the triples (ℓ, b, u) where $\ell \in \{A, P\}$, $b \in \{Y, N\}$ and $-2^d \leq u \leq 2^d$. Intuitively, ℓ indicates whether the agent is active (A) or passive (P), b indicates whether it currently believes that φ holds (Y) or not (N), and u is the agent's wealth, which can be negative. Agents for input x are initially in state $(A, N, 1)$, and agents for y in $(A, N, -1)$. If two passive agents meet their encounter has no effect. If at least one agent is active, then the result of the encounter is given by the transition $(*, *, u), (*, *, u') \mapsto (A, b, q), (P, b, r)$ where $b = Y$ if $u + u' \geq 2^d$ else N ; $q = \max(-2^d, \min(2^d, u + u'))$; and $r = (u + u') - q$. The protocol stabilises after $\mathcal{O}(n^2 \log n)$ expected interactions [5], but it has $2^{d+1} + 1$ states, exponentially many in $|\varphi| \in \Theta(d)$.

► **Example 6.** We give a protocol for $\varphi = (x - y \geq 2^d)$ with a polynomial number of states. This is essentially the protocol of [8]. We remove states and transitions from the protocol of Example 5, retaining only the states (ℓ, b, u) such that u is a power of 2, and some of the transitions involving these states:

$$\begin{aligned}
(*, *, 2^i), (*, *, 2^i) &\mapsto (A, N, 2^{i+1}), (P, N, 0) && \text{for every } 0 \leq i \leq d-2 \\
(*, *, 2^{d-1}), (*, *, 2^{d-1}) &\mapsto (A, Y, 2^d), (P, Y, 0) \\
(*, *, -2^i), (*, *, -2^i) &\mapsto (A, N, -2^{i+1}), (P, N, 0) && \text{for every } 0 \leq i \leq d-1 \\
(*, *, 2^i), (*, *, -2^i) &\mapsto (A, N, 0), (P, N, 0) && \text{for every } 0 \leq i \leq d-1
\end{aligned}$$

The protocol is not yet correct. For example, for $d = 1$ and the input $x = 2, y = 1$, the protocol can reach in one step the configuration in which the three agents (two x -agents and one y -agent) are in states $(A, Y, 2), (P, Y, 0), (A, N, -1)$, after which it gets stuck. In [8] this is solved by adding “reverse” transitions:

$$\begin{aligned}
(A, N, 2^{i+1}), (P, N, 0) &\mapsto (A, N, 2^i), (P, N, 2^i) && \text{for every } 0 \leq i \leq d-2 \\
(A, Y, 2^d), (P, Y, 0) &\mapsto (A, N, 2^{d-1}), (P, N, 2^{d-1}) \\
(A, N, -2^{i+1}), (P, N, 0) &\mapsto (A, N, -2^i), (A, N, -2^i) && \text{for every } 0 \leq i \leq d-1
\end{aligned}$$

The protocol has only $\Theta(d)$ states and transitions, but runs within $\Omega(n^{2^d-2})$ interactions. Consider the inputs x, y such that $x - y = 2^d$, and let $n := x + y$. Say that an agent is *positive* at a configuration if it has positive wealth at it. The protocol can only stabilise if it reaches a configuration with exactly one positive agent with wealth 2^d . Consider a configuration with $i < 2^d$ positive agents. The next configuration can have $i - 1, i, \text{ or } i + 1$ positive agents. The probability of $i + 1$ positive agents is $\Omega(1/n)$, while that of $i - 1$ positive agents is only $\mathcal{O}(1/n^2)$, and the expected number of interactions needed to go from 2^d positive agents to only 1 is $\Omega(n^{2^d-1})$ [11, Appendix A.1].

► **Example 7.** Given protocols $\mathcal{P}_1, \mathcal{P}_2$ with n_1 and n_2 states deciding predicates φ_1 and φ_2 , Angluin et al. construct in [5] a protocol \mathcal{P} for $\varphi_1 \wedge \varphi_2$ with $n_1 \cdot n_2$ states. It follows that the number of states of a protocol for $\varphi := \varphi_1 \wedge \dots \wedge \varphi_s$ grows exponentially in s , and so in $|\varphi|$. Blondin et al. give an alternative construction with polynomially many states [8, Section 5.3]. However, their construction contains transitions that, as in the previous example, reverse the effect of other transitions, and make the protocol very slow. The problem is already observed in the toy protocol with states q_1, q_2 and transitions $q_1, q_1 \mapsto q_2, q_2$ and $q_1, q_2 \mapsto q_1, q_1$. (Similar transitions are used in the initialisation of [8].) Starting with an even number $n \geq 2$ of agents in q_1 , eventually all agents move to q_2 and stay there, but the expected number of interactions is $\Omega(2^{n/10})$ [11, Appendix A.2].

6 Succinct Bounded Population Computers for Presburger Predicates

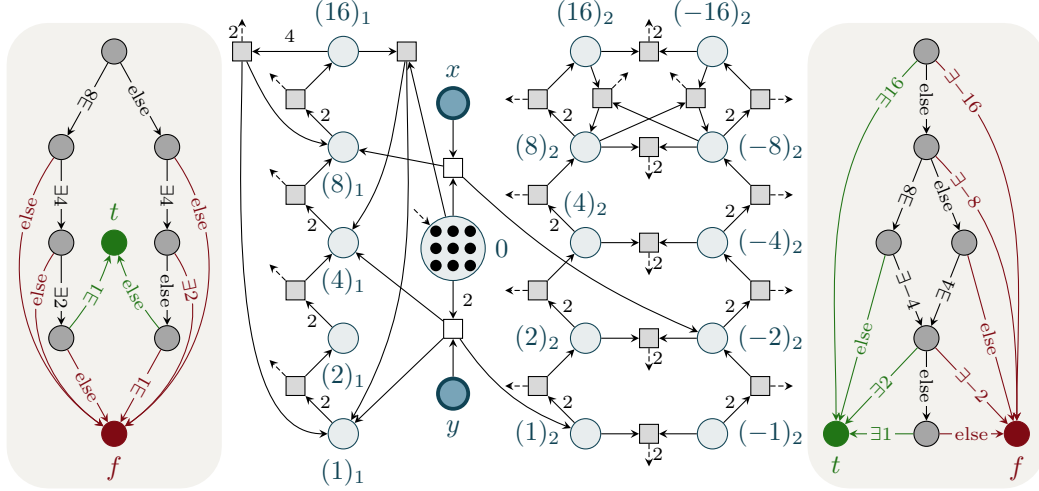
In Sections 6.1 and 6.2 we construct population computers for remainder and threshold predicates in which all coefficients are powers of two. We present the remainder case in detail, and sketch the threshold case. The generalization to arbitrary coefficients is achieved by means of a gadget very similar to the one we used to compute boolean combinations of predicates. This later gadget is presented in Section 6.3, and so we introduce the generalization there.

6.1 Population computers for remainder predicates

Let $Pow^+ = \{2^i \mid i \geq 0\}$ be the set of positive powers of 2.

We construct population computers \mathcal{P}_φ for remainder predicates $\varphi := \sum_{i=1}^v a_i x_i \equiv_m c$, where $a_i \in Pow^+ \cap \{0, \dots, m-1\}$ for every $1 \leq i \leq v$, $m \in \mathbb{N}$, and $c \in \{0, \dots, m-1\}$. We say that a finite multiset r over Pow^+ *represents* the residue $\text{rep}(r) := \text{sum}(r) \bmod m$. For example, if $m = 11$ then $r_{18} := \{2^3, 2^3, 2^1\}$ represents 7. Accordingly, we call the

multisets over Pow^+ representations. A representation of degree d only contains elements of $Pow_d^+ := \{2^d, 2^{d-1}, \dots, 2^0\}$. A representation r is a *support representation* if $r(x) \leq 1$ for every $x \in Pow^+$; so its represented value is completely determined by the support. For example, r_{18} is not a support representation of 7, but $\{2^5, 2^3\}$ is.



■ **Figure 1** (middle) Graphical Petri net representation (see Section 3) of population computer for the predicate $\varphi_1 \vee \varphi_2$ with $\varphi_1 = (8x + 5y \equiv_{11} 4)$ and $\varphi_2 = (y - 2x \geq 5)$. All dashed arrows implicitly lead to the reservoir state 0. It has 22 helpers although only 9 are drawn for space reasons. (left) decision diagram for output function of remainder predicate $8x + 5y \equiv_{11} 4$. It checks if the total value is 15 or 4. Starting at the top node of the diagram: if state 8 is populated, we move to the left child, otherwise to the right child; at the left child, if state 4 is populated we move to the right child, etc. (right) decision diagram for output function of threshold predicate $y - 2x \geq 5$.

We proceed to construct \mathcal{P}_φ . Let us give some intuition first. \mathcal{P}_φ has $Pow_d^+ \cup \{0\}$ as set of states. We extend the notion of representation to configurations by disregarding agents in state 0; a configuration is therefore a support representation if all states except 0 have at most one agent. The initial states of \mathcal{P}_φ are chosen so that every initial configuration for an input (x_1, \dots, x_v) is a representation of the residue $z := \sum_{i=1}^v a_i x_i \bmod m$. The transitions transform this initial representation of z into a support representation of z . Whether $z \equiv_m c$ holds or not depends only on the support of this representation, and the output function thus returns 1 for the supports satisfying $z \equiv_m c$, and 0 otherwise. Let us now formally describe \mathcal{P}_φ for $\varphi := \sum_{i=1}^v a_i x_i \equiv_m c$ where $a_i \in Pow^+ \cap \{0, \dots, m-1\}$.

States and initial states. Let $d := \lceil \log_2 m \rceil$. The set of states is $Q = Pow_d^+ \cup \{0\}$. The set of initial states is $I := \{a_1, \dots, a_v\}$. Observe that an input $C_I = \{x_1 \cdot a_1, \dots, x_v \cdot a_v\}$ is a representation of z , but not necessarily a support representation.

Transitions. Transitions ensure that non-support representations, i.e. representations with two or more agents in some state q , are transformed into representations of the same residue “closer” to a support representation. For $q \in 2^0, \dots, 2^{d-1}$ we introduce the transition:

$$2^i, 2^i \mapsto 2^{i+1}, 0 \quad \text{for } 0 \leq i \leq d-1 \quad \langle \text{combine} \rangle$$

For $q = 2^d$ we introduce a transition that replaces an agent in 2^d by a multiset of agents r with $\text{sum}(r) = 2^d - m$, preserving the residue. Let $b_d b_{d-1} \dots b_0$ be the binary encoding of $2^d - m$, and let $\{i_1, \dots, i_j\}$ be the positions such that $b_{i_1} = \dots = b_{i_j} = 1$. The transition is:

$$2^d, 0, \dots, 0 \mapsto 2^{i_1}, \dots, 2^{i_j} \quad \langle \text{modulo} \rangle$$

These transitions are enough, but we also add a transition that takes d agents in 2^d and replaces them by agents with sum $d \cdot 2^d \bmod m$. Intuitively, this makes the protocol faster. Let $b_d b_{d-1} \dots b_0$ and $\{i_1, \dots, i_j\}$ be as above, but for $d \cdot 2^d \bmod m$ instead of $2^d - m$.

$$2^d, \dots, 2^d \mapsto 2^{i_1}, \dots, 2^{i_j}, 0, \dots, 0 \quad \langle \text{fast modulo} \rangle$$

Helpers. We set $H := \lceil 3d \cdot 0 \rceil$, i.e. the computer initially places at least $3d$ helper agents in state 0. This makes sure one can always execute the next $\langle \text{modulo} \rangle$ or $\langle \text{fast modulo} \rangle$ transition: if no more agents can be combined, there are at most d agents in the states $2^0, \dots, 2^{d-1}$. Thus, there are at least $2d$ agents in the states 0 and 2^d , enabling one of these transitions. Observe that for every initial configuration $C_I + C_H$ we have $\text{sum}(C_I + C_H) = \text{sum}(C_I)$, and so, abusing language, every initial configuration for C_I is also a representation of z .

Output function. The computer eventually reaches a support configuration with at most one agent in every state except for 0. Thus, for every support set $S \subseteq Q$, we define $O(S) := 1$ if $\text{sum}(S) \equiv_m c$, and $O(S) = 0$ else. We show the existence of a small boolean circuit for the output function O in the proof of Lemma 8; this can be found in [11, Appendix B.1].

► **Lemma 8.** *Let $\varphi := \sum_{i=1}^v a_i x_i \equiv_m c$, where $a_i \in \{2^{d-1}, \dots, 2^1, 2^0\}$ for every $1 \leq i \leq v$ and $c \in \{0, \dots, m-1\}$ with $d := \lceil \log_2 m \rceil$. There is a bounded computer of size $\mathcal{O}(d)$ deciding φ .*

The left half of Figure 1 shows the population computer for $\varphi = (8x + 5y \equiv_{11} 4)$.

6.2 Population computers for threshold predicates

We sketch the construction of population computers \mathcal{P}_φ for threshold predicates $\varphi := \sum_{i=1}^v a_i x_i \geq c$, where $a_i \in \{2^j, 2^{-j} \mid j \geq 0\}$ for every $1 \leq i \leq v$ and $c \in \mathbb{N}$. As the construction is similar to the construction for remainder, we will focus on the differences and refer to [11, Appendix B.2] for details.

As for remainder, we work with representations that are multisets of powers of 2. However, they represent the sum of their elements (without modulo) and we allow both positive and negative powers of 2. Similar to the remainder construction, the computer transforms any representation into a *support representation* without changing the represented value. Then, the computer decides the predicate using only the support of that representation.

Again, there are $\langle \text{combine} \rangle$ transitions that allow agents with the same value to combine. Instead of modulo transitions, $\langle \text{cancel} \rangle$ transitions further simplify the representation: $2^i, -2^i \mapsto 0, 0$. Note that even after exhaustively applying $\langle \text{combine} \rangle$ and $\langle \text{cancel} \rangle$ there can still be many agents in 2^d or many agents in -2^d . This has two consequences:

- In the construction for general predicates of Section 6.3, we need that computers for remainder and threshold move most agents to state 0. In the remainder construction, all but a constant number of agents are moved to 0. In contrast, the threshold construction does not have this property. Thus, we do not design a single computer for a given threshold predicate φ but a family: one for every degree d larger than some minimum degree $d_0 \in \Omega(|\varphi|)$. Intuitively, larger degrees result in a larger fraction of agents in 0.

11:10 Fast and Succinct Population Protocols for Presburger Arithmetic

■ Assume we detect agents in 2^d (-2^d is analogous). If there are many, the predicate is true. However, if there is just one, then the represented value might be small, due to negative contributions $-2^0, \dots, -2^{d-1}$. We cannot distinguish the two cases, so we add transition $\langle \text{cancel 2nd highest} \rangle: 2^d, -2^{d-1} \mapsto 2^{d-1}, 0$. It ensures that agents cannot be present in both 2^d and -2^{d-1} ; therefore, an agent in 2^d certifies a value of at least 2^{d-1} . The right half of Figure 1 shows the population computer for $\varphi = (-2x + y \geq 5)$ with degree $d = 4$. [11, Appendix B.2] proves:

► **Lemma 9.** *Let $\varphi := \sum_{i=1}^v a_i x_i \geq c$, where $a_i \in \{2^j, 2^{-j} \mid j \geq 0\}$ for every $1 \leq i \leq v$. For every $d \geq \max\{\lceil \log_2 c \rceil + 1, \lceil \log_2 |a_1| \rceil, \dots, \lceil \log_2 |a_v| \rceil\}$ there is a bounded computer of size $\mathcal{O}(d)$ that decides φ .*

6.3 Population computers for all Presburger predicates

We present a construction that, given threshold or remainder predicates $\varphi_1, \dots, \varphi_s$, yields a population computer \mathcal{P} deciding an arbitrary given boolean combination $B(\varphi_1, \dots, \varphi_s)$ of $\varphi_1, \dots, \varphi_s$. We only sketch the construction, see [11, Appendix B.3] for details. We use the example $\varphi_1 = (y - 2x \geq 5)$, $\varphi_2 = (8x + 5y \equiv_{11} 4)$ and $B(\varphi_1, \varphi_2) = \varphi_1 \vee \varphi_2$. The result of the construction for this example is shown in Figure 1. The construction has 6 steps:

1. Rewrite Predicates. The constructions in Sections 6.1 and 6.2 only work for predicates where all coefficients are powers of 2. We transform each predicate φ_i into a new predicate φ'_i where all coefficients are decomposed into their powers of 2. In our example, $\varphi'_1 := \varphi_1$ because all coefficients are already powers of 2. However, $\varphi_2(x, y) = (8x + 5y \equiv_{11} 4)$ is rewritten as $\varphi'_2(x, y_1, y_2) := (8x + 4y_1 + 1y_2 \equiv_{11} 4)$ because $5 = 4 + 1$. Note that $\varphi_2(x, y) = \varphi'_2(x, y, y)$ holds for every $x, y \in \mathbb{N}$. Let r be the size of the largest split of a coefficient, i.e. $r = 2$ in the example.

2. Construct Subcomputers. For every $1 \leq i \leq s$, if φ_i is a remainder predicate, then let \mathcal{P}_i be the computer defined in Section 6.1. If φ_i is a threshold predicate, then let \mathcal{P}_i be the computer of Section 6.2, with $d = d_0 + \lceil \log_2 s \rceil$. We explain this choice of d in step 5.

3. Combine Subcomputers. Take the disjoint union of \mathcal{P}_i , but merging their 0 states. More precisely, rename all states $q \in Q_i$ to $(q)_i$, with the exception of state 0. Construct a computer with the union of all the renamed states and transitions. Figure 1 shows the Petri net representation of the computer so obtained for our example. We call the combined 0 state *reservoir* as it holds agents with no value that are needed for various tasks like input distribution.

4. Input Distribution. For each variable x_i add a corresponding new input state x_i . Then add a transition that takes an agent in state x_i and agents in 0 and distributes agents to the input states of the subcomputers that correspond to x_i . In our example, we add two states x and y and the transitions $x, 0 \mapsto (1)_1, (8)_2$ and $y, 0, 0 \mapsto (-2)_1, (4)_2, (1)_2$. The distribution for x needs one helper, because we need one agent in each subcomputer. The distribution for y needs two helpers, one for \mathcal{P}_1 and two for \mathcal{P}_2 , as $5y$ was split into $4y_1 + 1y_2$. This way, once the input states are empty, the correct value is distributed to each subcomputer. Crucially, this input distribution can be fast as it is not reversible.

5. Add Extra Helpers. In addition to all helpers from the subcomputers, add $r - 1$ more helpers to state 0. Intuitively, this allows to distribute the first input agent. Because of our choice for d in threshold subcomputers, each subcomputer returns most agents back to state 0. More precisely, for each distribution the number of agents that do not get returned to 0 only increases by at most $\frac{1}{s}$ (per subcomputer). So in total only one agent is “consumed” per distribution and enough agents are returned to 0 for the next distribution to occur. In our example, the agents that stay in each of the $s = 2$ subcomputers only increases by at most $\frac{1}{2}$ per distribution. (In fact, remainder subcomputers return all distributed agents.)

6. Combine Output. Note that we can still decide φ_i from the support of the states in the corresponding subcomputer \mathcal{P}_i . We compute the output for φ by combining the outputs of the subcomputers $\mathcal{P}_1, \dots, \mathcal{P}_s$ according to $B(\varphi_1, \dots, \varphi_s)$. In our example, we set the output to 1 if and only if the output of \mathcal{P}_1 or \mathcal{P}_2 is 1.

In [11, Appendix B.3], we show that this computer is succinct, correct and bounded:

► **Theorem 1.** *For every predicate $\varphi \in QFPA$ there exists a bounded population computer of size $\mathcal{O}(|\varphi|)$ that decides φ .*

7 Converting Population Computers to Population Protocols

In this section we prove Theorem 2. We proceed in four steps, which must be carried out in the given order. Section 7.1 converts any bounded computer \mathcal{P} for $\text{double}(\varphi)$ of size m into a *binary* bounded computer \mathcal{P}_1 with $\mathcal{O}(m^2)$ states. Section 7.2 converts \mathcal{P}_1 into a binary bounded computer \mathcal{P}_2 with a *marked consensus output function* (a notion defined in the section). Section 7.3 converts \mathcal{P}_2 into a binary bounded computer \mathcal{P}_3 for φ — not $\text{double}(\varphi)$ — with a marked consensus output function *and no helpers*. Section 7.4 shows that \mathcal{P}_3 runs within $\mathcal{O}(n^3)$ interactions. Finally, we convert \mathcal{P}_3 to a binary terminating (not necessarily bounded) computer \mathcal{P}_4 with a *normal consensus output* and no helpers, also running within $\mathcal{O}(n^3)$ interactions. This uses standard ideas; for space reasons it is described only in the full version at [11, Appendix F]. Similarly, the other conversions and results are only sketched, with details in [11].

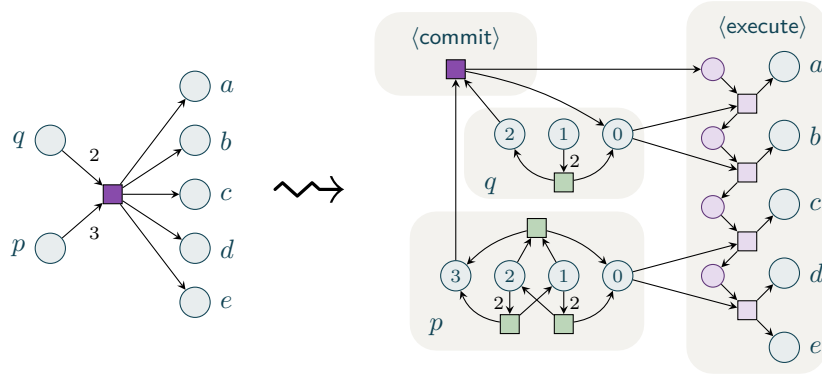
7.1 Removing multiway transitions

We transform a bounded population computer with k -way transitions $r \mapsto s$ such that $|\text{supp}(r)| \leq 2$ into a binary bounded population computer. Let us first explain why the construction introduced in [9, Lemma 3], which works for arbitrary transitions $r \mapsto s$, is too slow. In [9], the 3-way transition $t : q_1, q_2, q_3 \mapsto q'_1, q'_2, q'_3$ is simulated by the transitions

$$t_1 : q_1, q_2 \mapsto w, q_{12} \quad t_2 : q_{12}, q_3 \mapsto c_{12}, q'_3 \quad t_3 : q'_3, w \mapsto q'_1, q'_2 \quad \bar{t}_1 : w, q_{12} \mapsto q_1, q_2$$

Intuitively, the occurrence of t_1 indicates that two agents in q_1 and q_2 want to execute t , and are waiting for an agent in q_3 . If the agent arrives, then all three execute $t_2 t_3$, which takes them to q'_1, q'_2, q'_3 . Otherwise, the two agents must be able to return to q_1, q_2 to possibly execute other transitions. This is achieved by the “revert” transition \bar{t}_1 . The construction for a k -way transition has “revert” transitions $\bar{t}_1, \dots, \bar{t}_{k-2}$. As in Example 6 and Example 7, these transitions make the final protocol very slow.

We present a gadget without “revert” transitions that works for k -way transitions $r \mapsto s$ satisfying $|\text{supp}(r)| \leq 2$. Figure 2 illustrates it, using Petri net notation, for the 5-way transition $t : \{3p, 2q\} \mapsto \{a, b, c, d, e\}$. In the gadget, states p and q are split into $(p, 0), \dots, (p, 3)$



■ **Figure 2** Simulating the 5-way transition $\{3 \cdot p, 2 \cdot q \mapsto a, b, c, d, e\}$ by binary transitions.

and $(q, 0), \dots, (q, 2)$. Intuitively, an agent in (q, i) acts as representative for a group of i agents in state q . Agents in $(p, 3)$ and $(q, 2)$ commit to executing t by executing the binary transition $\langle \text{commit} \rangle$. After committing, they move to the states a, \dots, e together with the other members of the group, who are “waiting” in the states $(p, 0)$ and $(q, 0)$. Note that $\langle \text{commit} \rangle$ is binary because of the restriction $|\text{supp}(r)| \leq 2$ for multiway transitions.

To ensure correctness of the conversion, agents can commit to transitions if they represent more than the required amount. In this case, the initiating agents would commit to a transition and then elect representatives for the superfluous agents, before executing the transition. This requires additional intermediate states.

[11, Appendix C] formalises the gadget and proves its correctness and speed.

7.2 Converting output functions to marked-consensus output functions

We convert a computer with an arbitrary output function into another one with a *marked-consensus* output function. An output function is a *marked-consensus* output function if there are disjoint sets of states $Q_0, Q_1 \subseteq Q$ such that $O(S) := b$ if $S \cap Q_b \neq \emptyset$ and $S \cap Q_{1-b} = \emptyset$, for $b \in \{0, 1\}$, and $O(S) := \perp$ otherwise. Intuitively, for every $S \subseteq Q$ we have $O(S) = 1$ if all agents agree to avoid Q_0 (consensus), and at least one agent populates Q_1 (marked consensus). We only sketch the construction, a detailed description as well as a graphical example can be found in [11, Appendix D].

Our starting point is some bounded and binary computer $\mathcal{P} = (Q, \delta, I, O, H)$, e.g. as constructed in Section 7.1. Let (G, E) be a boolean circuit with only NAND-gates computing the output function O . We simulate \mathcal{P} by a computer \mathcal{P}' with a marked consensus output and $\mathcal{O}(|Q| + |G|)$ states. This result allows us to bound the number of states of \mathcal{P}' by applying well known results on the complexity of Boolean functions.

Intuitively, \mathcal{P}' consists of two processes running asynchronously in parallel. The first one is (essentially, see below) the computer \mathcal{P} itself. The second one is a gadget that simulates the execution of G on the support of the current configuration of \mathcal{P} . Whenever \mathcal{P} executes a transition, it raises a flag indicating that the gadget must be reset (for this, we duplicate each state $q \in Q$ into two states $(q, +)$ and $(q, -)$, indicating whether the flag is raised or lowered). Crucially, \mathcal{P} is bounded, and so it eventually performs a transition *for the last time*. This resets the gadget for the last time, after which the gadget simulates (G, E) on the support of the terminal configuration reached by \mathcal{P} .

The gadget is designed to be operated by one *state-helper* for each $q \in Q$, with set of states $Q_{\text{supp}}(q)$, and a *gate-helper* for each gate $g \in G$, with set of states $Q_{\text{gate}}(g)$, defined as follows:

- $Q_{\text{supp}}(q) := \{q\} \times \{0, 1, !\}$. These states indicate that q belongs/does not belong to the support of the current configuration (states $(q, 0)$ and $(q, 1)$), or that the output has changed from 0 to 1 (state $(q, !)$).
- $Q_{\text{gate}}(g) := \{g\} \times \{0, 1, \perp\}^3$ for each gate $g \in G$, storing the current values of the two inputs of the gate and its output. Uninitialised values are stored as \perp .

Recall that a population computer must also remain correct for a larger number of helpers. This is ensured by letting all helpers populating one of these sets, say $Q_{\text{supp}}(q)$, perform a leader election; whenever two helpers in states of $Q_{\text{supp}}(q)$ meet, one of them becomes a non-leader, and a flag requesting a complete reset of the gadget is raised. All resets are carried out by a *reset-helper* with set of states $Q_{\text{reset}} := \{0, \dots, |Q| + |G|\}$, initially in state 0. (Reset-helpers also carry out their own leader election!) Whenever a reset is triggered, the reset-helper contacts all other $|Q| + |G|$ helpers in round-robin fashion, asking them to reset the computation.

Eventually the original protocol \mathcal{P} has already reached a terminal configuration with some support Q_{term} , each set $Q_{\text{supp}}(q)$ and $Q_{\text{gate}}(g)$ is populated by exactly one helper, and all previous resets are terminated. From this moment on, \mathcal{P} never changes its configuration. The $|Q|$ state-helpers detect the support Q_{term} of the terminal configuration by means of transitions that move them to the states $Q_{\text{term}} \times \{1\}$ and $(Q \setminus Q_{\text{term}}) \times \{0\}$; the gate-helpers execute (G, E) on input Q' by means of transitions that move them to the states describing the correct inputs and outputs for each gate. State-helpers use $Q \times \{!\}$ as intermediate states, indicating that the circuit must recompute its output.

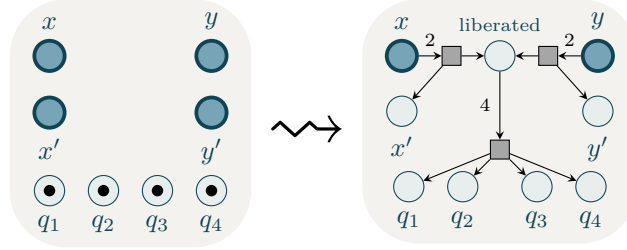
It remains to choose the sets Q_0 and Q_1 of states of the marked consensus output. We do it according to the output b of the output gate $g_{\text{out}} \in G$: Q_b is the set of states of $Q_{\text{gate}}(g_{\text{out}})$ corresponding to output b .

7.3 Removing helpers

We convert a bounded binary computer \mathcal{P} deciding the predicate $\text{double}(\varphi)$ over variables $x_1, \dots, x_k, x'_1, \dots, x'_k$ into a computer \mathcal{P}' with no helpers deciding φ over variables x_1, \dots, x_k . In [8], a protocol with helpers and set of states Q is converted into a protocol without helpers with states $Q \times Q$. We sketch a better construction that avoids the quadratic blowup. A detailed description can be found in [11, Appendix E].

Let us give some intuition first. All agents of an initial configuration of \mathcal{P}' are in input states. \mathcal{P}' simulates \mathcal{P} by *liberating* some of these agents and transforming them into helpers, without changing the output of the computation. For this, two agents in an input state x_i are allowed to interact, producing one agent in x'_i and one “liberated” agent, which can be used as a helper. This does not change the output of the computation, because $\text{double}(\varphi)(\dots, x_i, \dots, x'_i, \dots) = \text{double}(\varphi)(\dots, x_i - 2, \dots, x'_i + 1, \dots)$ holds by definition of $\text{double}(\varphi)$.

Figure 3 illustrates this idea. Assume \mathcal{P} has input states x, y, x', y' and helpers $H = \{q_1, q_2, q_3, q_4\}$, as shown on the left-hand side. Assume further that \mathcal{P} computes a predicate $\text{double}(\varphi)(x, y, x', y')$. The computer \mathcal{P}' is shown on the right of the figure. The additional transitions liberate agents, and send them to the helper states H . Observe that the initial states of \mathcal{P}' are only x and y . Let us see why \mathcal{P}' decides $\varphi(x, y)$. As the initial configuration of



■ **Figure 3** Illustration in graphical Petri net notation (see Section 3) of construction that removes helpers. Initial states are highlighted.

\mathcal{P}' for an input x, y puts no agents in x', y' , the computer \mathcal{P}' produces the same output on input x, y as \mathcal{P} on input $x, y, 0, 0$. Since \mathcal{P} decides $\text{double}(\varphi)$ and $\text{double}(\varphi)(x, y, 0, 0) = \varphi(x, y)$ by the definition of $\text{double}(\varphi)$, we are done. We make some remarks:

- \mathcal{P}' may liberate more agents than necessary to simulate the multiset H of helpers of \mathcal{P} . This is not an issue, because by definition additional helpers do not change the output of the computation.
- If the input is too small, \mathcal{P}' cannot liberate enough agents to simulate H . Therefore, the new computer only works for inputs of size $\Omega(|H|) = \Omega(|\varphi|)$.
- Even if the input is large enough, \mathcal{P}' might move agents out of input states before liberating enough helpers. However, the computers of Section 6 can only do this if there are enough helpers in the reservoir state (see point 3. in Section 6.3). Therefore, they always generate enough helpers when the input is large enough.

7.4 A $\mathcal{O}(n^3)$ bound on the expected interactions

We show that the computer obtained after the previous conversion runs within $\mathcal{O}(n^3)$ interactions. We sketch the main ideas; the details are in [11, Appendix G].

We introduce *potential functions* that assign to every configuration a positive *potential*, with the property that executing any transition strictly decreases the potential. Intuitively, every transition “makes progres”. We then prove two results: (1) under a mild condition, a computer has a potential function iff it is bounded, and (2) every binary computer with a potential function and no helpers, i.e. any bounded computer for which speed is defined, stabilises within $\mathcal{O}(n^3)$ interactions. This concludes the proof.

Fix a population computer $\mathcal{P} = (Q, \delta, I, O, H)$.

► **Definition 10.** A function $\Phi : \mathbb{N}^Q \rightarrow \mathbb{N}$ is linear if there exist weights $w : Q \rightarrow \mathbb{N}$ s.t. $\Phi(C) = \sum_{q \in Q} w(q)C(q)$ for every $C \in \mathbb{N}^Q$. We write $\Phi(q)$ instead of $w(q)$. A potential function (for \mathcal{P}) is a linear function Φ such that $\Phi(r) \geq \Phi(s) + |r| - 1$ for all $(r \mapsto s) \in \delta$.

Observe that k -way transitions reduce the potential by $k - 1$, binary transitions by 1. At this point, we consider only binary computers, but this distinction becomes relevant for the refined speed analysis.

If a population computer has a potential function, then every run executes at most $\mathcal{O}(n)$ transitions, and so the computer is bounded. Applying Farkas’ Lemma we can show that the converse holds for computers in which every state can be populated – a mild condition, since states that can never be populated can be deleted without changing the behaviour.

► **Lemma 11.** If \mathcal{P} has a reachable configuration C_q with $C_q(q) > 0$ for each $q \in Q$, then \mathcal{P} is bounded iff there is a potential function for \mathcal{P} .

Consider now a binary computer with a potential function and no helpers. At every non-terminal configuration, at least one (binary) transition is enabled. The probability that two agents chosen uniformly at random enable this transition is $\Omega(1/n^2)$, and so a transition occurs within $\mathcal{O}(n^2)$ interactions. Since the computer has a potential function, every run executes at most $\mathcal{O}(n)$ transitions, and so the computer stabilises within $\mathcal{O}(n^3)$ interactions.

The final step to produce a population protocol is to translate computers with marked-consensus output function into computers with standard consensus output function, while preserving the number of interactions. For space reasons this construction is presented in [11, Appendix F].

8 Rapid Population Computers: Proving a $\mathcal{O}(n^2)$ Bound

We refine our running-time analysis to show that the population protocols we have constructed actually stabilise within $\mathcal{O}(n^2)$ interactions. We continue to use potential functions, as introduced in Section 7.4, but improve our analysis as follows:

- We introduce *rapidly-decreasing* potential functions. Intuitively, their existence shows that progress is not only *possible*, but also *likely*. We prove that they certify stabilisation within $\mathcal{O}(n^2)$ interactions.
- We introduce *rapid* population computers, as computers with rapidly-decreasing potential functions that also satisfy some technical conditions. We convert rapid computers into protocols with $\mathcal{O}(|\varphi|)$ states, and show that the computers of Section 6 are rapid.

In order to define rapidly-decreasing potential functions, we need a notion of “probability to execute a transition” that generalises to multiway transitions and is preserved by our conversions. At a configuration C of a protocol, the probability of executing a binary transition $t = (p, q \mapsto p', q')$ is $C(q)C(p)/n(n-1)$. Intuitively, leaving out the normalisation factor $1/n(n-1)$, the transition has “speed” $C(q)C(p)$, proportional in the *product* of the number of agents in p and q . But for a multiway transition like $q, q, p \mapsto r_1, r_2, r_3$ the situation changes. If $C(q) = 2$, it does not matter how many agents are in p – the transition is always going to take $\Omega(n^2)$ interactions. We therefore define the speed of a transition as $\min\{C(q), C(p)\}^2$ instead of $C(q)C(p)$.

For the remainder of this section, let $\mathcal{P} = (Q, \delta, I, O, H)$ denote a population computer.

► **Definition 12.** Given a configuration $C \in \mathbb{N}^Q$ and some transition $t = (r \mapsto s) \in \delta$, we let $\text{tmin}_t(C) := \min\{C(q) : q \in \text{supp}(r)\}$. For a set of transitions $T \subseteq \delta$, we define $\text{speed}_T(C) := \sum_{t \in T} \text{tmin}_t(C)^2$, and write $\text{speed}(C) := \text{speed}_\delta(C)$ for convenience.

► **Definition 13.** Let Φ denote a potential function for \mathcal{P} and let $\alpha \geq 1$. We say that Φ is α -rapidly decreasing at a configuration C if $\text{speed}(C) \geq (\Phi(C) - \Phi(C_{\text{term}}))^2/\alpha$ for all terminal configurations C_{term} with $C \rightarrow C_{\text{term}}$.

We have not been able to find potential functions for the computers of Section 6 that are rapidly decreasing at every reachable configuration, only at reachable configurations with sufficiently many helpers, defined below. Fortunately, that is enough for our purposes.

► **Definition 14.** $C \in \mathbb{N}^Q$ is well-initialised if C is reachable and $C(I) + |H| \leq \frac{2}{3}n$.

Observe that an initial configuration C can only be well-initialised if $C(\text{supp}(H)) \in \Omega(C(I))$. We now define *rapid* population computers, and state the result of our improved analysis.

► **Definition 15.** \mathcal{P} is α -rapid if

1. it has a potential function Φ which is α -rapidly decreasing in well-initialised configurations,
2. every state of \mathcal{P} but one has at most 2 outgoing transitions,
3. all configurations in \mathbb{N}^I are terminal, and
4. for all transitions $t = (r \mapsto s)$, $q \in I$ we have $r(q) \leq 1$ and $s(q) = 0$.

► **Theorem 3.**

- (a) The population computers constructed in Theorem 1 are $\mathcal{O}(|\varphi|^3)$ -rapid.
- (b) Every α -rapid population computer of size m deciding $\text{double}(\varphi)$ can be converted into a terminating population protocol with $\mathcal{O}(m)$ states that decides φ in $\mathcal{O}(\alpha m^4 n^2)$ interactions for inputs of size $\Omega(m)$.

The detailed proofs can be found in the full version [11], in the following sections. The proof of (a) is given in Appendix B. For (b), we prove separate theorems for each conversion in Appendices C, D, E, and F. To achieve a tighter analysis of our conversions, we generalise the notion of potential function; this is described in Appendix H.

9 Conclusions

We have shown that every predicate φ of quantifier-free Presburger arithmetic has a population protocol with $\mathcal{O}(\text{poly}(|\varphi|))$ states and $\mathcal{O}(|\varphi|^7 \cdot n^2)$ expected number of interactions. If only inputs of size $\Omega(|\varphi|)$ matter, we give a protocol with $\mathcal{O}(|\varphi|)$ states and the same speed. The obvious point for further improvement is the $|\varphi|^7$ factor in the expected number of interactions.

Our construction is close to optimal. Indeed, for every construction there is an infinite family of predicates for which it yields protocols with $\Omega(|\varphi|^{1/4})$ states [9]; further, it is known that every protocol for the majority predicate requires in $\Omega(n^2)$ interactions.

References

- 1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *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.
- 3 Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. Fast and exact majority in population protocols. In *PODC*, pages 47–56. ACM, 2015.
- 4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC*, pages 290–299. ACM, 2004.
- 5 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Comput.*, 18(4):235–253, 2006.
- 6 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Comput.*, 21(3):183–199, 2008.
- 7 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Comput.*, 20(4):279–304, 2007.
- 8 Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. Succinct population protocols for Presburger arithmetic. In *STACS*, volume 154 of *LIPICs*, pages 40:1–40:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 9 Michael Blondin, Javier Esparza, and Stefan Jaax. Large flocks of small birds: On the minimal size of population protocols. In *STACS*, volume 96 of *LIPICs*, pages 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

- 10 Robert Brijder, David Doty, and David Soloveichik. Democratic, existential, and consensus-based output conventions in stable computation by chemical reaction networks. *Natural Computing*, 17(1):97–108, 2018.
- 11 Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. Fast and succinct population protocols for Presburger arithmetic, 2022. [arXiv:2202.11601v2](https://arxiv.org/abs/2202.11601v2).
- 12 David Doty, Mahsa Eftekhari, Leszek Gasieniec, Eric E. Severson, Grzegorz Stachowiak, and Przemyslaw Uznanski. Brief announcement: A time and space optimal stable population protocol solving exact majority. In *PODC*, pages 77–80. ACM, 2021.
- 13 Robert Elsässer and Tomasz Radzik. Recent results in population protocols for exact majority and leader election. *Bull. EATCS*, 126, 2018.
- 14 Christoph Haase. A survival guide to Presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018.