Technical University of Munich
TUM School of Computation, Information and Technology

Computational Limits of Population Protocols: Space Complexity, Robustness and Expressive Power

Philipp Czerner

Complete reprint of the dissertation approved by the TUM School of Computation, Information and Technology of the Technical University of Munich for the award of the

Doktor der Naturwissenschaften (Dr. rer. nat.)

Chair:

Prof. Dr. Harald Räcke

Examiners:

1. Prof. Dr. Francisco Javier Esparza Estaun
2. Prof. Dr. Helmut Seidl
3. Prof. Dr. Joël Ouaknine

The dissertation was submitted to the Technical University of Munich on ??.??.???? and accepted by the TUM School of Computation, Information and Technology on ??.??.????.

Abstract

Population protocols are a model of distributed computation where a large number of agents interact. These agents have limited computational capabilities; they have no identities and are finite-state. They interact pairwise and stochastically. The goal of the computation is to decide a global property of the initial configuration. The output is given by stable consensus: At each point, every agent has an opinion on the output. This opinion can change any number of times, but eventually all agents must have the same opinion, and it must not be possible for any agent to change their opinion.

We analyse the space complexity of population protocols, which is the smallest number of states necessary to decide a given family of predicates. We give the first general lower bound on the family of flock-of-birds predicates $\varphi_k(x) \Leftrightarrow x \geq k$, and then give a matching construction, proving that the bound is tight. We also give a construction for arbitrary Presburger predicates, which results in protocols that are both succinct and have optimal time complexity.

In addition to making protocols more succinct, we also investigate fault tolerance. We show that our succinct construction for flock-of-birds predicates is almost self-stabilising, meaning that, essentially, it still works in a model with an adversary that can add agents to the computation in arbitrary states. We also investigate robustness of protocols against an adversary that can remove agents during the computation and show that any threshold or modulo predicate can be decided in this model.

We also consider variants of population protocols, as well as related models. We consider population protocols where the number of states is allowed to grow with the number of agents n, and characterise their expressive power when using $\Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$ states, for any $\varepsilon > 0$.

In broadcast consensus protocols, population protocols are extended with a broadcast transition. We determine their expressive power when limited to a polynomial number of interactions. Finally, we consider a related family of models where finite-state agents interact on a graph and determine their expressive power when the graph is chosen by an adversary.

Acknowledgements

I thank Javier Esparza; Martin Helfrich, Roland Guttenberg, Christoph Welzel-Mohr, Valentin Krasotin; Rupak Majumdar, Jérôme Leroux, Stefan Jaax, Eszter Couillard, Benno Lossin, Vincent Fischer, Julie Cailler; Harald Räcke, Jürgen Dix, Johannes Brasche; Felix Czerner and Peter Czerner.

Contents

1. Introduction

> Complex systems can sometimes behave in ways that are entirely unpredictable.
>
> *Lieutenant Commander Data,*
> *Star Trek: The Next Generation*

If air inside a balloon is heated, the balloon will expand. This behaviour is predicted by the ideal gas law: the pressure exerted by a gas is proportional to its temperature divided by its volume. These properties, temperature, volume and pressure, are *emergent* — they are collective properties of all molecules making up the gas, not of any individual one. Indeed, one would find it nigh-impossible to understand the behaviour of a balloon by only considering trajectories of individual molecules.

Similarly, many dynamics shaping our world arise from complex interactions between a large number of simple parts. It is crucial that we understand, and are able to design, such emergent behaviours.

Various approaches attempt to tackle these issues. In this thesis, we shall walk a path less trodden, and consider systems through the lens of *computation*. While molecules in the air interact aimlessly, many other systems are designed, or have evolved, to serve a purpose.

- Chemical reactions use specially selected or even synthesised substances to elicit a desired behaviour. For example, one might wish to detect the presence or absence of a certain kind of molecule, and perform a reaction that indicates the results, by assuming a particular colour.
- In a biological system, cells cooperate to form larger organisms. To do this, they must communicate important information, such as availability of certain nutrients.
- Computers clearly perform computation. While many networks consist of parts much more complex than a single molecule, many others obey severe resource constraints, limiting the computational power of their parts. Examples include mobile sensor networks, or passively powered devices such as smart cards.

By considering complex systems as processes of computation, we are able to employ tools from theoretical computer science to understand how powerful they are, and where their limits lie.

Weak models of distributed computing. In this thesis, I consider and analyse multiple theoretical models of distributed computation. These models are simplified — their aim

is not to model specific real-world systems, but rather to abstract and generalise the essential aspects, to understand the underlying dynamics. The following key properties are shared amongst them, and distinguish them from the wider study of distributed computing.

- **Many participants.** Computer networks often consist of merely a few hundred participants, with the largest having perhaps a few billion, or tens of billions. In contrast, a chemical reaction easily involves 10^{23} molecules.
- **Participants are limited.** In distributed computing, one often assumes the ability for all participants to store data and perform arbitrary computations on it. In contrast, here we assume strict limits on their computational capabilities. They have only finitely many states, and cannot store additional information.
- **Stochastic interactions.** Participants cannot choose when and with whom to communicate. Consider e.g. a chemical reaction, where molecules collide randomly. Analogously, we assume that communication happens according to a stochastic process.
- **No identities.** Two agents in the same state are indistinguishable in principle, just like two molecules of the same kind.
- **Global input.** The input is a *global* property of the system, meaning that the information is shared across all participants. For example, the input could be the total number of participants (which is initially not known by any individual).
- **Computation by consensus.** At each point every participant stores its current output. This output may change throughout the computation, but eventually all participants must agree on one output, which must not change thereafter.

Population protocols. To make the above more concrete, let us consider the model of *population protocols* [5], which has enjoyed immense popularity in the distributed computing community. It is also the main focus of this thesis, occupying all but the last section. I will only give a brief introduction here, a formal definition can be found in Chapter 3.

We refer to the individual participants of the computation as *agents*, and to the group of all agents in the computation as *population*. An agent is in one of finitely many states, and has no other properties. For our first example, let $Q := \{\mathsf{A}, \mathsf{B}\}$ denote the set of states. Since an agent is described by only its state (in other words, two agents in the same state are indistinguishable), all we need to describe a *configuration*, i.e. the state of the computation at a given point in time, is the number of agents in each state. For example, consider a configuration $C = 5 \cdot \mathsf{A} + 7 \cdot \mathsf{B}$, i.e. in configuration C there are 5 agents in state A and 7 agents in state B.

The computation proceeds via pairwise interactions. To execute one step, we pick two (distinct) agents uniformly at random, yielding a pair of states. A transition function then determines the result of the interaction, which is also a pair of states. In particular, the total number of agents does not change. Continuing the example, we choose transitions $\delta := \{(\mathsf{A}, \mathsf{B}) \mapsto (\mathsf{B}, \mathsf{B})\}$, i.e. if two agents in states A and B meet, respectively, the former

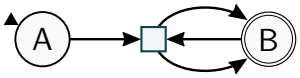


Figure 1.1.: The first example protocol using graphical notation (see Section 3.1.1).

moves to state B as well. (Implicitly, we assume that all other interactions have no effect.) This transition is illustrated in Figure 1.1.

If this interaction occurs in the configuration $C = 5 \cdot \mathsf{A} + 7 \cdot \mathsf{B}$ from above, we end up in a configuration with 4 agents in state A and 8 agents in state B. Continuing to execute interactions eventually leads to a configuration where all 12 agents are in B. Having interactions occur sequentially is a convenient viewpoint for analysis, but we usually assume that a linear number of interactions (w.r.t. the number of agents) occur independently in parallel. This does not affect the behaviour of the protocol, but it does affect the speed — we say that a computation that takes T interactions runs in *parallel time* $\mathcal{O}(T/n)$, where n is the total number of agents.

Finally, we must define input and output. For the former, we designate some states as *input states*. The initial configuration consists only of agents in those states; their amounts are the input to the protocol. For the output, there is an *output function* assigning each state either 0 or 1. In the example, let us choose all states as inputs, and let the output function be $O := \{\mathsf{A} \mapsto 0, \mathsf{B} \mapsto 1\}$.

A protocol computes a predicate if from any input configuration it will eventually reach a *stable consensus*, that is, a configuration where all agents have the same output, and no agent can reach a state with a different output. The protocol above decides the predicate $\varphi(A, B) \Leftrightarrow B \geq 1$. If all agents are in state A, the configuration cannot change, while in any configuration with at least one agent in state B eventually all agents in state A will move to B as well.

Research questions. In this thesis, I investigate the computational limits of population protocols and related models. The lines of inquiry comprise the following:

- **Space complexity.** Given a predicate φ, what is the smallest population protocol that decides φ?
- **Time-space tradeoffs.** Are small population protocols necessarily slow?
- **Robustness.** Is it possible to construct fault-tolerant population protocols — at least for some fault models?
- **Model variants.** How do different parameters (such as varying the transition rule, or locating agents in a graph) affect the computational power of the model?

1.1. Prior Work

The population protocol model was introduced by Angluin, Aspnes, Diamadi, Fischer and Peralta in 2004 [5, 6], and they proved that it is possible to decide every Presburger (or semilinear) predicate in this model. Subsequently, Angluin, Aspnes, Eisenstat and

Ruppert [8] showed that the decision power of population protocols is precisely the Presburger predicates, i.e. every predicate decided by a population protocol is Presburger.

In the two decades since the model was introduced, many results on both the basic model as well as variants of it were obtained.

In the basic model, one can analyse the space complexity of some predicate φ, i.e. the size of the smallest protocol deciding φ. Of particular interest is the asymptotic space complexity of various families of predicates. This means that one investigates how quickly the protocol must grow to handle predicates of increasing complexity. (This is distinct from investigations in the non-constant state space model, described below, where a single predicate — usually majority — is fixed, and the growth is w.r.t. the size of the population.)

The simplest family that has been investigated in this context are the *flock-of-birds predicates*, i.e. predicates of the form $\varphi_k(x) \Leftrightarrow x \geq k$, where $k \in \mathbb{N}$ is a fixed parameter. Their name comes from the original motivation for population protocols — modelling mobile sensors, e.g. affixed to birds. The well-known flock-of-birds problem is for the sensors to reach a consensus on whether at least k ill birds are part of the flock.[1]

It is known that all flock-of-birds predicates can be decided by population protocols with $\mathcal{O}(\log k)$ states [15]. If one extends the model with a leader, i.e. a unique designated agent that can coordinate the computation, it is even possible to use only $\mathcal{O}(\log \log k)$ states [15], an exponential advantage. No upper bounds are known for this problem.

This result was subsequently generalised, showing that for *any* Presburger predicate φ (i.e. any predicate that can be decided by a population protocol) there is a *succinct* protocol deciding φ. More precisely, a protocol with $\mathcal{O}(\text{poly}|\varphi|)$ states, where $|\varphi|$ is the length of the quantifier-free Presburger representation of φ, with coefficients in binary.

Non-constant state space. The most popular extension of population protocols — perhaps more popular than the basic model even — allows the state space to grow slightly with the number of agents n in the computation. When constructing protocols, one tries to minimise both the running time of the protocol, ideally to $\mathcal{O}(\text{polylog}\, n)$ parallel time, and the growth of the state space, ideally to $\mathcal{O}(\text{polylog}\, n)$ (though some constructions even use only $\mathcal{O}(\log \log n)$ states).

The main motivation for the non-constant state space model is its ability to decide many important predicates in polylogarithmic parallel time. This is not known to be possible in the basic (i.e. constant state) model, and (under some technical assumptions) it is proven to be impossible.

Two tasks are of particular interest: the *majority* predicate, i.e. constructing a protocol that decides the predicate $\varphi(A, B) \Leftrightarrow A \leq B$, and the task of *leader election*. For the latter, the protocol must eventually reach a configuration where precisely one agent is in a designated leader state. While not a predicate, leader election is an important subtask — in particular, as will be discussed below, population protocols with a leader can quickly compute arbitrary Presburger predicates.

In the non-constant state space model, lower bounds [1, 2] have been obtained for

[1] I remark that this problem would be better modelled by the predicate $\varphi(x, y) \Leftrightarrow x \geq k$, though in this context it does not make a difference.

both majority and leader election, relating the growth of the state space to the time complexity. In particular, their results also extend to the basic model, showing that e.g. the majority predicate requires $\Omega(n/\operatorname{polylog} n)$ parallel time (under some technical conditions). These results extend to "most" predicates, in particular to all predicates which are not constant for sufficiently large inputs [9].

Subsequently, many constructions have explored the boundary between time and space for both majority and leader election [1, 13, 2, 11, 41, 10, 12, 36] (see also [3, 37]).

The decision power of this model remains largely open. Consider protocols with $f(n)$ states, where $f : \mathbb{N} \to \mathbb{N}$ is a function (obeying some mild technical restrictions). It is known that $f \in \Omega(n)$ implies that all predicates in $\mathsf{SNSPACE}(n \log f(n))$ can be decided, i.e. the predicates decidable by a nondeterministic Turing machine with $\mathcal{O}(n \log f(n))$ space when the input is encoded in unary [22], while $f \in o(\log n)$ implies that only Presburger predicates can be computed [8, 22], at least for certain subclasses of protocols[2].

Model variants. As mentioned above, one variant considers the introduction of a *leader*, a unique agent in a specially designated state that is part of the input configuration and can help to synchronise the computation. This does not extend the decision power of the model, but it enables exponentially faster computation [7].

In *graph population protocols* the communication structure is changed. While in the basic model any two agents may interact, instead one executes the protocol on a connected undirected graph G. The nodes of G correspond to the agents, and only adjacent agents may interact. If the graph is chosen adversarially, the model can still decide precisely the semi-linear predicates [6], though on specific classes of graphs the model becomes much more powerful (e.g. on a line one can easily simulate a linear-space Turing machine). Arbitrary protocols in the basic model can be converted to work on an adversarially chosen graph, with time and space overhead bounded by the conductance of the graph [4].

Many related models also consider finite-state agents located on a graph, e.g. [23, 43, 38], see also [48]. In [40], the authors identify four parameters along which such models commonly differ. Combining these four parameters, they construct 24 variants of such models, which collapse into 7 classes based on their expressive power.

Another way to modify the model is to add additional data to the agents. In *community population protocols* [42], as well as the related *homonym population protocols* [18], agents have identities and can store a constant number of identities of other agents they interact with. Roughly speaking, these models are much stronger than population protocols in terms of expressive power.

Population protocols with unordered data give each agent an additional piece of immutable data from some infinite set, but transitions may only check whether the data from the two agents are equal, i.e. the transition function has the signature $\delta : Q^2 \times \{0, 1\} \to Q^2$ and remains finite [17]. The expressive power of this model is open; however, it is known that the verification problem is undecidable [52], suggesting that the model is strictly more powerful than population protocols, where verification is decidable [39].

Finally, one can also add additional types of transitions. In [47], the model is extended with *absence detectors*, which allow an agent to determine whether a state is currently

[2]This is related to uniformity. A detailed explanation can be found in Section 6.2.

occupied by some agent. This extends the power to $\mathsf{SNSPACE}(\log n)$. In a similar vein, the model of *broadcast consensus protocols* can add *broadcast transitions*, allowing one agent to issue a broadcast that all other agents react to [16]. This also increases the expressive power to $\mathsf{SNSPACE}(\log n)$.

Fault-tolerance. The basic model assumes that all agents work as intended, and agents neither appear nor disappear throughout the computation. Relaxing this assumption leads one to consider different notions of fault-tolerance.

In [35], the authors allow for two kinds of failures: agents may disappear, or change state arbitrarily. However, the total number of failures is bounded by a constant, and the actual number of failures must be low enough to not change the output of the initial configuration. For example, if given the majority predicate $\varphi(A, B) \Leftrightarrow A \leq B$, only $|A - B|$ failures may occur. Under these assumptions, the expressive power is not affected and all Presburger predicates can be computed.

Robustness has also been considered in variants of the population protocol model. In *community population protocols*, it is possible to tolerate a constant number of *Byzantine* failures [42], i.e. agents that change their states arbitrarily often and are adversarial.

Self-stabilisation is a strong notion of fault-tolerance considered primarily in the non-constant state space model. It states that the protocol stabilises to the correct output regardless of the input configuration. It is known that this requires $\Omega(n)$ states (in particular, it is impossible in the basic model) [21], and constructions using $\mathcal{O}(n)$ states exist [21, 20].

1.2. Contributions

I briefly outline the main contributions of this thesis. Detailed descriptions can be found in the corresponding chapters.

State complexity of flock-of-birds predicates. Consider the flock-of-birds predicates $\varphi_k(x) \Leftrightarrow x \geq k$. The state complexity of φ_k is the size of the smallest population protocol deciding φ_k. In other words, we ask how many states a protocol needs to count to k — a fundamental question about the power of the population protocol model.

In 2018, Blondin, Esparza and Jaax first considered this problem. While they provided some upper bounds, they were unable to show any *general* lower bound, i.e. a lower bound that holds for every k. This problem remained open until 2021, where we proved that any protocol for φ_k needs $\Omega(\log \log \log k)$ states [26]. We later improved this bound to $\Omega(\log \log k)$ [29]. This is described in Section 4.2.

This leaves a gap to the best known construction with $\mathcal{O}(\log k)$ states. Contrary to prevailing opinion, I prove that the lower bound is tight, by describing a $\mathcal{O}(\log \log k)$ construction in the basic model [25], described here in Section 4.3. To go beyond $\mathcal{O}(\log k)$ states, this construction needs to have certain properties that are unusual for constructions in the literature — in particular, to the best of my knowledge it is the first construction for flock-of-birds predicates that is not 1-aware, and the first for any predicate that is intrinsically not silent. (Roughly speaking, a protocol is *1-aware* if for any accepted input at some point some agent "knows" that the protocol will accept. A

protocol is *silent* if computation eventually ceases and no agent changes state after that point.)

Due to these properties, the protocol exhibits certain fault-tolerance properties. In particular, it is *almost self-stabilising*, meaning that it stabilises to the correct output for any input configuration where a small number of agents start in the initial state. I detail the fault-tolerance properties of this protocol in Section 5.1.

Arbitrary Predicates. Blondin, Esparza, Genest, Helfrich and Jaax showed in 2020 that succinct protocols exist for arbitrary Presburger predicates [14]. (Succinct means that the protocol decides a predicate φ with $\mathcal{O}(\text{poly}|\varphi|)$ states, where φ is represented as quantifier-free Presburger formula.) However, their construction produces protocols that are slow — they can take exponential parallel time to reach a consensus.

We improve upon their result in 2022, proving that it is possible to construct succinct protocols that run within linear parallel time [32, 33]. This is described in Section 4.4. This time bound is optimal, since e.g. the majority predicate requires at least $\Omega(n)$ parallel time [1, 9].

As a corollary, we also obtain protocols of size $\mathcal{O}(|\varphi|)$, i.e. linear instead of polynomial, but they work only under the assumption that the number of agents is not too low. More precisely, the protocols assume that $n \in \Omega(|\varphi|)$ agents are present — a mild assumption in the context of natural computing.

Robustness. Similar to [35], we consider a model of fault-tolerance where agents may disappear during the computation. As in their model, we restrict the number of disappearances such that the output of the initial configuration cannot be changed by removing that many agents. However, we do not bound it by a constant, so a single protocol has to work for a potentially unbounded number of errors. We call a protocol *robust* if it works in this model, i.e. a protocol that can tolerate disappearing agents.

We show that it is possible to construct robust protocols for arbitrary *threshold predicates* [45], described in Section 5.2. These are predicates of the form $\varphi(x_1, ..., x_l) \Leftrightarrow a_1x_1 + ... + a_lx_l \geq t$, where $a_1, ..., a_l, t \in \mathbb{Z}$. While this is an important subclass that covers predicates such as flock-of-birds predicates and majority, it remains open whether all Presburger predicates can be computed in this model.

Other models. As mentioned above, the model of broadcast consensus protocols extends population protocol by adding broadcast transitions. Previously, their expressive power was determined to be precisely $\mathsf{SNSPACE}(\log n)$ [16].

We consider their running time and show that any Presburger predicate can be computed in parallel time $\mathcal{O}(\log n)$, which is optimal [34]. Moreover, any randomised Turing machine can be simulated with an overhead of parallel time $\mathcal{O}(\log n)$ per step of the Turing machine. This is described in Section 6.3.

In [40], Esparza and Reiter introduce a variety of models where agents are located on nodes of a graph. They analyse their expressive power when both the initial states of the agents and the graph structure itself form the input. Out of the 24 variants they define, they identify 7 different equivalence classes, which they separate. However, they make no statements about the precise expressive power of each of the 7 classes.

We consider the expressive power of their model under the assumption that both

the graph and the locations of the agents on the graph are adversarially chosen. As in population protocols, the input is then a multiset of input states and the protocol decides a predicate.

With these assumptions, we precisely characterise the expressive power of all 7 classes [31]. We also strengthen the model by restricting the degree of the graph to be bounded by a constant, a natural limitation in biological systems, and precisely characterise 6 of the 7 resulting classes, while bounding the 7th. Details can be found in Section 6.4.

1.2.1. Other contributions

The following results were also obtained during my doctoral studies, but they are not included in this thesis. Nevertheless, I shall give a brief description here.

Interactive protocols. Automated reasoning tools, such as SAT-solvers, are both increasingly complex and increasingly relied upon to ensure the correctness of safety-critical systems. It is, therefore, paramount that we are able to check the output of such tools. To this end, *certification* is used: in addition to the result (i.e. satisfiable or unsatisfiable), the tool outputs a certificate, which an independent checker can verify. If the certificate is valid, the output of the tool must be correct.

However, certifying that, say, a SAT-instance is unsatisfiable requires certificates of exponential length[3]. This is a problem also in practice — certificates can be many TiB in size, and verifying them takes asymptotically the same amount of computing power as solving the instance.

One possible solution is suggested by the $\mathsf{IP} = \mathsf{PSPACE}$ theorem, a famous breakthrough result in complexity theory [51]. It considers a model with two parties:

- *Verifier*, running a polynomially time-bounded, randomised algorithm, and
- *Prover*, which is computationally unbounded, but untrusted.

While Prover can simply solve the instance and tell Verifier the output, Verifier cannot trust Prover. Instead, Verifier sends a series of challenges to Prover to determine the veracity of Prover's claims. In particular, it must be impossible for Prover to convince Verifier of a false output (except for some negligible failure probability).

The $\mathsf{IP} = \mathsf{PSPACE}$ theorem then states that every problem in PSPACE, which contains most practically relevant automated reasoning tasks, can be certified by such an interactive protocol.

Note that Verifier is polynomially time-bounded, while solving the instance takes exponential time[4].

Though this seems promising, there is one major obstacle to applying such an approach in practice: the algorithm for Prover proposed in [51] is not feasible to implement, since it iterates through all possible assignments. Solving practical SAT-instances requires

[3]Unless $\mathsf{NP} = \mathsf{coNP}$, which is widely believed to be false.

[4]Or at least superpolynomial, unless $\mathsf{P} = \mathsf{PSPACE}$, which is very widely believed to be false.

the use of clever heuristics which, while still exponential in the worst case, significantly outperform brute-force methods.

We therefore consider binary decision diagrams (BDDs) — an approach that is still widely used in practice for many different automated reasoning problems. We show that it is possible to adapt existing BDD algorithms to yield an efficient implementation of Prover [24]. In particular, our algorithm for Prover is guaranteed to take only constant-factor overhead for both solving the instance and responding to Verifier's challenges, compared to simply solving the instance using the standard BDD algorithm.

This algorithm relies on a theoretical statement connecting the intermediate BDDs computed in the standard BDD algorithm to the finite-field polynomials that are required in the $\mathsf{IP} = \mathsf{PSPACE}$ theorem.

We also implement our approach. In our measurements, the constant-factor overhead is low (roughly 3), and the certification time is much faster than state-of-the-art approaches (roughly 300 times faster, and improving with increasing instance size).

In follow-up work [27], we formalise the notion of *competitive* interactive protocols, where Prover has low overhead compared to a given algorithm, and give a general framework to construct such protocols. We also show that within the framework, an interactive protocol competitive with the Davis-Putnam resolution procedure can be constructed.

Regular abstraction frameworks. In regular model checking, one is given a *regular transition system* (C_I, δ), where $C_I \subseteq \Sigma^*$ is a regular language over some alphabet Σ, and $\delta \subseteq \Sigma^* \times \Sigma^*$ is a regular relation over Σ. In particular, both are represented by finite automata. This induces a transition system where the configurations are the words over Σ, C_I are the initial configurations, and δ determines the step relation.

An important property is *safety*: given a regular language $C_U \subseteq \Sigma^*$ of *unsafe* configurations, decide whether it is possible to reach any configuration in C_U in the system (C_I, δ).

It is easy to see that this problem is undecidable. Still, in practice many instances can be solved with the help of *inductive invariants*, i.e. a set $S \subseteq \Sigma^*$ with $C_I \subseteq S$ and $\delta(S) \subseteq S$. However, depending on the instance different kinds of invariants might be necessary to prove safety.

We propose a general approach and introduce the notion of *regular abstraction frameworks (RAFs)* [28]. An RAF is simply a regular relation $\mathcal{V} \subseteq \Gamma^* \times \Sigma^*$, where Γ is some other alphabet. Each word $w \in \Gamma^*$ represents a potential invariant $\mathcal{V}(w)$. In this manner, the RAF $\mathcal{V}$ encodes infinitely many possible invariants. We also develop an algorithm that determines the inductive invariants of the RAF and determines whether these are strong enough to prove safety of the regular transition system.

1.3. Publication Index

The following publications are part of this thesis. Core publications are marked with the corresponding factor, and the first author(s) of each paper are highlighted with an asterisk.

A [core, 1] Philipp Czerner*, Javier Esparza, Jérôme Leroux.
"Lower bounds on the state complexity of population protocols".
Distributed Computing, 2023 [29].

B [core, 1] Philipp Czerner*.
"Breaking Through the $\Omega(n)$-Space Barrier: Population Protocols Decide Double-Exponential Thresholds".
DISC, 2024 [25].

C [core, 0.5] Philipp Czerner*, Roland Guttenberg*, Martin Helfrich, Javier Esparza.
"Fast and Succinct Population Protocols for Presburger Arithmetic".
Journal of Computer and System Sciences, 2024 [33].

D Benno Lossin*, Philipp Czerner, Javier Esparza, Roland Guttenberg, Tobias Prehn.
"The Black Ninjas and the Sniper: On Robust Population Protocols".
Principles of Verification: Cycling the Probabilistic Landscape, Part III, 2024 [45].

E Philipp Czerner, Vincent Fischer*, Roland Guttenberg.
"Brief Announcement: The Expressive Power of Uniform Population Protocols with Logarithmic Space".
DISC, 2024 [30].

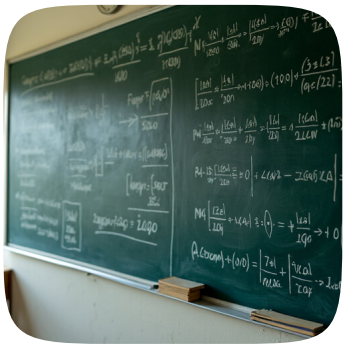
F [core, 1] Philipp Czerner*, Stefan Jaax.
"Running Time Analysis of Broadcast Consensus Protocols".
FOSSACS, 2021 [34].

G [core, 0.5] Philipp Czerner*, Roland Guttenberg*, Martin Helfrich, Javier Esparza.
"Decision Power of Weak Asynchronous Models of Distributed Computing".
PODC, 2021 [31].

All of the above publications are in English and have been published in international, peer-reviewed conferences or journals. They are included in the appendix. All, except for E, are full papers.

Publications A, B and C are covered in Chapter 4, publication D in Chapter 5, and publications E, F and G in Chapter 6.

2. Preliminaries



> You'll need to know chess notation before you can read it.
>
> *Mr. Shaibel, The Queen's Gambit*

I define some basic notations used throughout the thesis.

We write $\mathbb{N} := \{0, 1, 2, ...\}$ for the natural numbers. Given two sets A, B, we write B^A for the set of functions $f : A \to B$.

Multisets. We refer to elements of $\mathbb{N}^A$ as *multisets (over A)*. Given two multisets $C, D \in \mathbb{N}^A$ we write $C + D$ for the multiset defined as $(C + D)(x) := C(x) + D(x)$ for all $x \in A$, and $C \cap D$ is the multiset with $(C \cap D)(x) := \min\{C(x), D(x)\}$ for $x \in A$. We write $C \leq D$ if $C(x) \leq D(x)$ for all x. If $C \leq D$ holds, we write $D - C$ for the multiset fulfilling $(D - C)(x) = D(x) - C(x)$ for all x. We also write $C \leq k$ for $k \in \mathbb{N}$ if $C(x) \leq k$ for all $x \in A$, and use $\geq$ analogously to $\leq$. Given $k \in \mathbb{N}$, we define $k \cdot C$ via $(k \cdot C)(x) = k \cdot C(x)$ for $x \in A$.

We sometimes abuse notation and, for $x \in A$, write simply x to refer to the multiset C with $C(x) := 1$ and $C(y) := 0$ for $y \in A \setminus \{x\}$. If $A \subseteq B$ and $C \in \mathbb{N}^A$, we also implicitly consider C as a multiset $C \in \mathbb{N}^B$ by extending with zero, i.e. $C(x) := 0$ for $x \in B \setminus A$. Rarely, we will also consider a set $S \subseteq A$ implicitly as the multiset given by its indicator function, i.e. the multiset C with $C(x) := 1$ for $x \in S$ and $C(x) := 0$ otherwise.

For a set $S \subseteq A$ we write $C(S) := \sum_{x \in S} C(x)$ for the number of times an element of S appears in A. Given $f : A \to B$, we set $f(C) \in \mathbb{N}^B$ to the multiset with $(f(C))(y) := \sum_{f(x)=y} C(x)$ for $x \in A$. The *size* of a multiset $C \in \mathbb{N}^A$ is $|C| := C(A) = \sum_{x \in A} C(x)$. Its *support* is $[\![C]\!] := \{x \in A : C(x) > 0\}$.

Predicates. In this thesis we only consider predicates over multisets, i.e. predicates of the form $\varphi : \mathbb{N}^A \to \{0, 1\}$ for some finite A. We sometimes consider φ as a function with $|A|$ arguments over $\mathbb{N}$ instead. For example, given $A := \{x, y\}$ we would write $\varphi(x, y) \Leftrightarrow x \leq y$ for $x, y \in \mathbb{N}$ instead of $\varphi(C) \Leftrightarrow C(x) \leq C(y)$ for $C \in \mathbb{N}^A$.

Graphs. A *(simple) undirected graph* is a pair $G = (V, E)$ where V is a finite set of *nodes* and $E \subseteq \{\{u, v\} : u, v \in V, u \neq v\}$ is a set of *edges*. The *neighbourhood* of a node $v \in V$ is the set $\Gamma(v) := \{u : \{u, v\} \in E\}$. We extend this to sets $S \subseteq V$ as $\Gamma(S) := \bigcup_{v \in S} \Gamma(v)$. A graph is *connected* if $\Gamma(S) = S$ implies $S \in \{\emptyset, V\}$.

Digital readers. Please note that most named mathematical objects are clickable links to their definitions. They are not coloured, to reduce distraction.

3. Population Protocols

> I just wanted to see how they would interact and function in tandem. You see, in my experiment, I had proposed a theory that by working together they could combine their skills and increase their usefulness.
>
> *Wesley Crusher,*
> *Star Trek: The Next Generation*

In this chapter, I introduce the formal definition of population protocols and give a few examples.

3.1. The Model

Definition 1. *A* population protocol *is a tuple* (Q, δ, I, O)*, where*

- Q *is a finite set of* states*,*
- $\delta \subseteq Q^2 \times Q^2$ *is a set of* transitions*,*
- $I \subseteq Q$ *are the* input states*, and*
- $O : Q \to \{0, 1\}$ *is an* output mapping*.*

We will use the simple protocol $\mathcal{P}_{\text{maj}} := (Q, \delta, I, O)$ for the majority predicate $\varphi(A, B) \Leftrightarrow A \leq B$ as an example. We set $Q := \{A, B, a, b\}$, $I := \{A, B\}$ and $O(q) := 1$ for $q \in \{B, b\}$ and $O(q) := 0$ otherwise.

The idea of the protocol is that each agent has an opinion regarding the output — states A, a want to reject, and states B, b want to accept (this is reflected in the choice of O). To come to a consensus, it must be possible for agents to convince other agents of their opinion. To that end, we separate agents into "active" agents in states A, B, and "passive" agents in a, b.

When two active agents with different opinions meet, they keep their opinions but become passive. This is effected by the following transition:

$$A, B \mapsto a, b \qquad \langle \mathsf{cancel} \rangle$$

For transitions, we write $(p, q \mapsto p', q')$ instead of $((p, q), (p', q'))$, where $p, q, p', q' \in Q$.

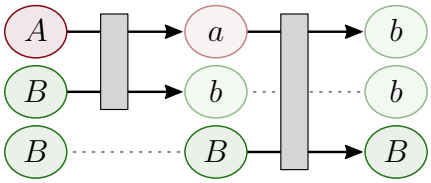


Figure 3.1.: A run of $\mathcal{P}_{\text{maj}}$.

We then allow passive agents to change their opinion, which happens whenever they meet an active agent. In this case, the passive agent adopts the opinion of the active agent.

$$A, b \mapsto A, a$$
$$B, a \mapsto B, b \qquad \langle \textsf{convince} \rangle$$

The resulting protocol is almost correct. For example, if we run the protocol starting from the configuration $C = A + 2 \cdot B$, we get the following execution:

$$A + 2 \cdot B \to_{\langle \textsf{cancel} \rangle} a + b + B \to_{\langle \textsf{convince} \rangle} 2 \cdot b + B$$

The latter configuration then repeats infinitely often. Since it is a 1-consensus, the protocol accepts. This run is also illustrated in Figure 3.1.

However, the protocol does not work correctly in the case of a tie, as eventually all agents will become passive and no consensus can be reached. Since we want to accept that case, we allow the passive accepting agents to convince passive rejecting agents.

$$a, b \mapsto b, b \qquad \langle \textsf{tiebreak} \rangle$$

This completes the description of $\mathcal{P}_{\text{maj}}$; as we will show later, the protocol is correct. But first, we must formally define how a population protocol is executed, and what it means for it to be correct.

Definition 2. *Let $\mathcal{P} = (Q, \delta, I, O)$ denote a population protocol. We refer to elements of $\mathbb{N}^Q$ as* configurations*, and to elements of $\mathbb{N}^I$ as* initial configurations*. A configuration C with $[\![C]\!] \subseteq O^{-1}(b)$ is a b-*consensus*, with $b \in \{0, 1\}$.*

Let $t = (q, p \mapsto q', p') \in \delta$ denote a transition and $C, C' \in \mathbb{N}^Q$ configurations. If $C \geq q + p$ then t is enabled *at C and we write $C \to_t C'$, if $C' = C - q - p + p' + q'$. We write $C \to C'$ if there are $t_1, ..., t_k \in \delta$ with $C \to_{t_1} ... \to_{t_k} C'$ (allowing $k = 0$, in which case $C' = C$) and say that C' is* reachable *from C.*

We say that C is terminal *if $C \to C'$ implies $C = C'$.*

Executing a single transition $p, q \mapsto p', q'$ is simple. One must ensure that agents in states p, q are present (otherwise the transition may not be executed), then remove those two agents, and finally add two agents in states p', q'. To run the protocol, we execute arbitrary transitions infinitely often.

Definition 3. *Let $\sigma = (C_i)_i$ denote a (finite or infinite) sequence of configurations. We say that σ is an* execution*, if for all i we have $C_i \to_t C_{i+1}$ or $C_i = C_{i+1}$. We write $\inf(\sigma) := \bigcap_i \{C_i, C_{i+1}, ...\}$ for the configurations appearing infinitely often in σ and call σ a* run*, if it is an infinite execution and $\inf(\sigma)$ is closed under $\to$.*

Intuitively, we model the interactions of a population protocol by having one Poisson process for each pair of agents, with the same constant rate. Equivalently, one can

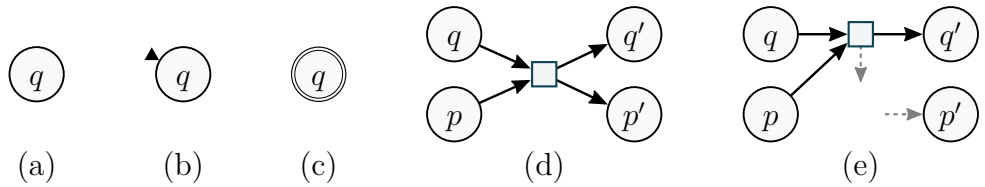


Figure 3.2.: Graphical notation. (a) A state. (b) An initial state. (c) A state with output 1. (d) A transition $(q, p \mapsto q', p')$. (e) Same transition with alternative notation.

consider transitions to occur in sequence, and choose the next interaction by picking two distinct agents uniformly at random. Since in the Poisson process model we execute $\Theta(n)$ interactions within one unit of time (n being the number of agents), we measure the time complexity in terms of *parallel time*, i.e. the number of interactions divided by n.

However, when we are not interested in the time complexity of a population protocol, we can simplify the model further, as in the definition above. The stochastic scheduler (i.e. picking two agents uniformly at random) is replaced by a fairness condition: in any run σ the set $\inf(\sigma)$ is closed under reachability.

This fairness condition is sometimes referred to as "strong fairness". It ensures that every configuration that can be reached infinitely often is reached infinitely often.

Definition 4. *Let $\mathcal{P} = (Q, \delta, I, O)$ denote a population protocol and $\varphi : \mathbb{N}^I \to \{0, 1\}$ a predicate. A run $\sigma = C_0 C_1 \ldots$* stabilises *to $b \in \{0, 1\}$, if $\inf(\sigma)$ contains only b-consensuses.*

We say that $\mathcal{P}$ decides *φ, if for all $C \in \mathbb{N}^I$ any run starting from C stabilises to $\varphi(C)$.*

3.1.1. Graphical notation

We visualise population protocols using the graphical notation from Petri nets. States are drawn as ellipses, transitions are drawn as rectangles, and they are connected by arcs. More precisely, given a population protocol $\mathcal{P} = (Q, \delta, I, O)$, let $t = (q, p \mapsto q', p') \in \delta$ denote a transition. Then we draw arcs $(q, t), (p, t), (t, q'), (t, p')$.

Initial states are highlighted by a triangle next to the state, and states q with $O(q) = 1$ have a double border.

Sometimes, we have many arcs to the same state and, instead of drawing them, draw outgoing and incoming dashed arcs that are not connected.

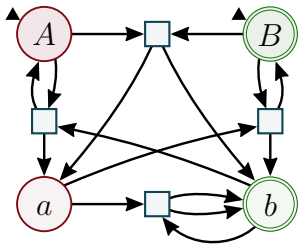


Figure 3.3.: $\mathcal{P}_{\text{maj}}$.

The graphical notation is illustrated in Figure 3.2, and the protocol $\mathcal{P}_{\text{maj}}$ is drawn in Figure 3.3.

3.1.2. Proving correctness

In the sequel, we will prove correctness of various population protocols. It is thus instructive to look at a correctness proof of the simple protocol $\mathcal{P}_{\text{maj}}$ here which both

illustrates the definitions above, and demonstrates important techniques that also feature in the proofs of more complicated protocols.

- We use *invariants* to relate reachable configurations to the corresponding initial configurations. Roughly, this is a function $\mathrm{val} : \mathbb{N}^Q \to S$, for some set S, which remains unchanged by executing transitions, i.e. $C \to C'$ implies $\mathrm{val}(C) = \mathrm{val}(C')$ for all configurations $C \in \mathbb{N}^Q$. Crucially, this can be checked locally, by simply considering all possible transitions in isolation.

  Usually, our invariants will assign a numeric *value* to a configuration, so we have $S = \mathbb{Z}$.

- To show that a protocol terminates, we use *potential functions*. These are functions $\mathrm{pot} : \mathbb{N}^Q \to \mathbb{N}$ which can never increase by executing a transition, so $\mathrm{pot}(C) \geq \mathrm{pot}(C')$ for $C \to C'$. Again, this can be checked locally.

  If we have two configurations C, C' with $C \to C'$ and $\mathrm{pot}(C) > \mathrm{pot}(C')$, then we know $C' \not\to C$. But we get even more: Let σ denote a run, so (by Definition 3) we know that $\inf(\sigma)$ is closed under $\to$. From this we derive $C \notin \inf(\sigma)$, as $C \in \inf(\sigma)$ would imply $C' \in \inf(\sigma)$, but since $C' \not\to C$ we cannot visit both infinitely often. In other words, we will eventually reach a configuration from which the potential cannot be reduced further.

Now, let us prove that $\mathcal{P}_{\mathrm{maj}}$ does indeed decide the majority predicate.

Lemma 5. *$\mathcal{P}_{\mathrm{maj}}$ decides $\varphi(A, B) \Leftrightarrow A \leq B$.*

Proof. Let $(Q, \delta, I, O) := \mathcal{P}_{\mathrm{maj}}$, and let $C_0 \in \mathbb{N}^I$ be arbitrary, and let $\sigma = C_0 C_1 ...$ denote a run from C_0. Following Definition 4 we have to show that σ stabilises to $\varphi(C_0)$. Again by definition this means that every configuration $C \in \inf(\sigma)$ is a $\varphi(C_0)$-consensus. Before we show that, we make two observations.

First, we note that $\mathrm{val}(C_i) := C_i(B) - C_i(A)$ is an invariant, so it remains constant (i.e. does not depend on i). We can verify this claim by inspecting the transitions: $\langle \mathsf{cancel} \rangle$ reduces both $C_i(B)$ and $C_i(A)$ by 1, while $\langle \mathsf{convince} \rangle$ and $\langle \mathsf{tiebreak} \rangle$ leave either unchanged.

Second, we can observe that the value of $\mathrm{pot}(C_i) := C_i(A) + C_i(B)$ cannot increase, again by inspecting all transitions.

Now, pick some configuration $C \in \inf(\sigma)$. Our goal is to show that C is a $\varphi(C_0)$-consensus. We consider the following five cases. The first three deal with the situation that C is not a consensus, the fourth considers the special case of a consensus consisting of only a. These four will lead to a contradiction. The last case then handles the remaining consensuses.

Case 1: $C(A) > 0$**,** $C(B) > 0$**.** Then transition $\langle \mathsf{cancel} \rangle$ is enabled at C, and with $C' := C - A - B + a + b$ we have $C \to C'$. However, $\mathrm{pot}(C) > \mathrm{pot}(C')$, so $C' \not\to C$, contradicting the fact that $\inf(\sigma)$ is closed under $\to$. Hence this case cannot occur.

Case 2: $C(A) > 0$**,** $C(B) = 0$**,** $C(b) > 0$**.** Similarly to the previous case, we can now execute transition $\langle \mathsf{convince} \rangle$ to move to configuration $C - b + a$. By executing that transition $C(b)$ times, we move to $C' := C - C(b) \cdot b + C(b) \cdot a$. In particular, in C' all

agents are in states A or a and no transition can be executed at C'. Hence $C' \not\to C$; a contradiction.

Case 3: $C(A) = 0$**,** $C(B) + C(b) > 0$**,** $C(a) > 0$**.** Now we can execute $\langle \mathsf{tiebreak} \rangle$ for $C(a)$ times, and reach a terminal configuration $C' \neq C$, another contradiction.

Case 4: $[\![C]\!] = \{a\}$**.** Let i be maximal such that $C_i(A) + C_i(B) > 0$. (Since it holds for $i = 0$, such an i must exist.) In particular, we have $C_{i+1}(A) + C_{i+1}(B) = 0$, so C_{i+1} results from C_i after executing $\langle \mathsf{cancel} \rangle$. Hence $[\![C_{i+1}]\!] = \{a, b\}$. After step i, $\langle \mathsf{cancel} \rangle$ is never enabled (since, e.g. $\mathrm{pot}(C_j) = 0$ for $j > i$), so the number of agents in state a can never increase. This contradicts $C_j \to C$.

Case 5: C is a β-consensus, $[\![C]\!] \neq \{a\}$**.** We get

$$
\begin{aligned}
& \beta = 1 \\
\Leftrightarrow\ & \mathrm{val}(C) \geq 0 && (C \text{ is a consensus and } [\![C]\!] \neq \{a\}) \\
\Leftrightarrow\ & \mathrm{val}(C_0) \geq 0 && (\text{val is constant}) \\
\Leftrightarrow\ & C_0(A) \leq C_0(B) && (\text{definition of val}) \\
\Leftrightarrow\ & \varphi(C_0)
\end{aligned}
$$

□

3.2. Basic Properties

Let us collect some basic technical properties, which will become useful later. Fix some population protocol $\mathcal{P} = (Q, \delta, I, O)$.

First, we are going to show that the reachability relation is *monotonic*, i.e. adding agents to a configuration does not limit which transitions can be executed.

Lemma 6. *Let $C, C', D, D' \in \mathbb{N}^Q$ with $C \to C'$ and $D \to D'$. Then $C + D \to C' + D'$.*

Proof. It suffices to prove the special case $D = D'$, since one obtains the general case by applying the statement twice.

By there are transitions $t_1, ..., t_k \in \delta$ with $C \to_{t_1} ... \to_{t_k} C'$. We proceed by induction on k. In the base case $k = 0$, $C = C'$ and the statement is trivial.

Otherwise, let C'' such that $C \to_{t_1} ... \to_{t_k} C'' \to_{t_{k+1}} C'$, and let $t_{k+1} = (p, q \mapsto p', q')$. By induction hypothesis, $C + D \to C'' + D$. Since $C'' \geq p + q$, we also have $C'' + D \geq p + q$. Moreover, $C' = C'' - p - q + p' + q'$ implies $C' + D = C'' + D - p - q + p' + q'$ and thus $C'' + D \to C' + D$. □

The next lemma shows that a run exists from any configuration. Actually, it is even true that if one executes transitions uniformly at random (from the set of enabled transitions, say), then the resulting infinite execution will be a run with probability 1.

Lemma 7. *Let $C_0 \in \mathbb{N}^Q$. Then there is some sequence $\sigma = C_0 C_1 ...$ with $C_1, ... \in \mathbb{N}^Q$, such that σ is a run.*

Proof. To make the following more convenient, we write $C \to_1 C'$ for configurations C, C' if either $C = C'$ or there is a $t \in \delta$ with $C \to_t C'$.

We choose the sequence σ as follows. At index i, let $S := \{C \in \mathbb{N}^Q : C_i \to_1 C\}$. We observe that S is finite and nonempty (since $C_i \in S$).

Let $f : \mathbb{N}^Q \to \mathbb{N}$ be defined as $f(C) := \max(\{0\} \cup \{j \leq i : C_j = C\})$ for all C. Intuitively, $f(C)$ is the last index i at which configuration C appears so far.

Pick some configuration $C \in S$ where $f(C)$ is minimal and set $C_{i+1} := C$. We repeat the above procedure to generate an infinite sequence. Clearly, the resulting sequence σ is an execution.

Assume for contradiction that σ is not a run. Then $\inf(\sigma)$ is not closed under $\to$. In particular, there exist $C, C' \in \mathbb{N}^Q$ with $C \to C'$, but $C \in \inf(\sigma)$ and $C' \notin \inf(\sigma)$. Due to $C \to C'$, there is a sequence $D_1, ..., D_k$ with $D_1 = C$, $D_k = C'$ and $D_1 \to_1 ... \to_1 D_k$. Clearly, we can find some j with $D_j \in \inf(\sigma)$ and $D_{j+1} \notin \inf(\sigma)$, so we assume wlog $C \to_1 C'$

Since there are only finitely many choices for C', we can pick one where $i := \max(\{0\} \cup \{j \leq i : C_j = C'\})$ is maximal. (Note that C' appears only finitely often in σ, so the maximum exists.)

As C appears infinitely often in σ, fix some index $j > i$ with $C_j = C$ and $\inf(\sigma) \subseteq \{C_{i+1}, ..., C_j\}$. Consider the choice of configuration C_{j+1} and let S and f as above. In particular, we have $f(C') = i$. For any $D \in \inf(\sigma)$ our choice of j implies $f(D) > i$, so $C_{j+1} \neq D$. But if $C_{j+1} \notin \inf(\sigma)$, then $C_{j+1} \in S$ implies $C = C_j \to_1 C_{j+1}$, and we either have $C_{j+1} \neq C'$, and i is not maximal, or $C_{j+1} = C'$, contradicting the definition of i.

In either case we have a contradiction; therefore, σ is a run. □

Finally, we give a different characterisation of which inputs are accepted, using the notion of stable consensuses.

Definition 8. *Let C denote a b-consensus. We say that C is* stable*, if for all configurations C' with $C \to C'$ we have that C' is a b-consensus as well.*

Intuitively, a configuration is a stable consensus if all agents have the same opinion, and it is not possible for any agent to change their opinion.

Lemma 9. *Assume that $\mathcal{P}$ decides a predicate φ. Let $C_0 \in \mathbb{N}^I$ denote an initial configuration and $b \in \{0, 1\}$. Then C_0 reaches a stable b-consensus iff $\varphi(C_0) = b$.*

Proof. "⇒": Let C denote a stable b-consensus with $C_0 \to C$. Using Lemma 7 there is a run starting from C. By adding a prefix we obtain a run $\sigma = C_0 C_1 ...$ with $C_i = C$ for some i. Since we have $C \to C_j$ for every $j > i$, C_j is a b-consensus and $\inf(\sigma)$ contains only b-consensuses. By Definition 4, $\varphi(C_0) = b$.

"⇐": Let $\sigma = C_0 C_1 ...$ denote a run and pick some $C \in \inf(\sigma)$. Let $C' \in \mathbb{N}^Q$ be arbitrary, with $C \to C'$. As $\inf(\sigma)$ is closed under $\to$ we have $C' \in \inf(\sigma)$. Using Definition 4, we obtain that C' is a $\varphi(C_0)$-consensus, i.e. a b-consensus. Hence C is a stable b-consensus. □

4. State Complexity of Population Protocols

> They seem to be getting smaller, Charlie. Are you sure you wouldn't like some more?
>
> *Michael Collins, Apollo 11, 1969*

Given a predicate φ, what is the size of the smallest protocol deciding φ? This is the essential question of *space complexity* of population protocols. Here, the size of a population protocol is simply the number of states.

Minimising the number of states is of particular importance in the application of chemical computing, where the total number of agents is enormous (e.g. roughly 10^{23} molecules participating in a reaction), but the number of states is limited to perhaps 100-1000. Similar discrepancies, albeit not as stark, exist in other areas of natural computing.

In addition to its practical relevance, state complexity is also one of the fundamental measures in theoretical computer science (the other being time complexity). Analysing the ability of population protocols to succinctly express predicates gives us fundamental insights into the power of the model.

Definition 10. *Let φ denote a Presburger predicate. The* space complexity *of φ is the largest $k \in \mathbb{N}$, such that every population protocol deciding φ has at least k states.*

Note that space complexity is well-defined since every Presburger predicate can be decided by a population protocol.

As always, it is less interesting to consider just a single instance (i.e. one specific predicate), and determine the exact space complexity of that one predicate. Instead, we look at the asymptotic space complexity of families of predicates.

Of particular interest are *flock-of-birds* predicates, which are threshold predicates of the form $\varphi_k(x) \Leftrightarrow x \geq k$. They are introduced in Section 4.1, together with some simple protocols deciding them.

For these predicates, I present *lower* bounds on their space complexity in Section 4.2, i.e. impossibility results ruling out sufficiently small protocols. Matching these bounds, Section 4.3 describes a construction that is exponentially smaller than the simple constructions presented in the section preceding it.

Finally, in Section 4.4 I turn towards arbitrary Presburger predicates and give a general construction that constructs population protocols which are:

- *fast* — they stabilise within linear parallel time. Since this (almost) matches a known lower bound for population protocols, we refer to it as fast in this context.
- *succinct* — the number of states is polynomial relative to the length of the predicate decided by the protocol.

4.1. Flock-of-Birds Predicates

To analyse space complexity, we need some family of predicates to consider. A natural choice are *flock-of-birds* predicates, which are defined as follows.

Definition 11. *The family* $(\varphi_k)_{k\in\mathbb{N}}$ *with* $\varphi_k(x) \Leftrightarrow x \geq k$ *are the* flock-of-birds predicates.

Throughout the remainder of this chapter, we will always write φ_k to refer to the corresponding flock-of-birds predicate.

Note that the number of agents present in the computation does not change. Since φ_k has only one input, the value of that input is equal to the number of agents n — so intuitively the predicate φ_k can be restated as "Are there at least k agents?".

These predicates are, arguably, the simplest possible infinite family of Presburger predicates. As such, determining their space complexity is an important stepping stone towards the analysis of space complexity of arbitrary Presburger predicates. In fact, the results on flock-of-birds predicates presented in this thesis have already led to advances both for general predicates (see Section 4.4), as well as areas unrelated to space complexity (see Section 5.1).

Token counting. Let us begin with a population protocol $\mathcal{P}_{\text{tok}} := (Q, \delta, I, O)$ to decide φ_k, for some arbitrary $k \in \mathbb{N}$. This construction can be considered as a special case of the construction of [5], already proposed in 2004.

The idea of the protocol is that every agent holds some number of tokens, up to k. Hence, the states are $Q := \{0, ..., k\}$. Initially, each agent holds one token, so we set $I := \{1\}$ as initial states.

There are only two types of transitions. First, if two agents meet and the total number of tokens carried by them is below k, then one agent takes all tokens (and the other leaves empty-handed).

$$i, j \mapsto i + j, 0 \qquad \text{for } i + j < k \qquad \langle \mathsf{collect} \rangle$$

Conversely, if they have at least k tokens in total, they both leave with k tokens.

$$i, j \mapsto k, k \qquad \text{for } i + j \geq k \qquad \langle \mathsf{accept} \rangle$$

Finally, we set k as the only accepting state, i.e. $O(k) := 1$ and $O(q) := 0$ for $q \in \{0, ..., k-1\}$.

The protocol works in two phases. First, agents collect tokens until one agent has collected at least k. Once that happens, that agent "knows" that at least k agents exist and can accept. Since every interaction with an agent in state k results in both agents

being in state k, this will spread throughout the population until every agent is in state k and the population accepts.

On the other hand, if fewer than k agents are present, eventually all tokens are held by a single agent and the configuration becomes terminal, with all agents rejecting.

Lemma 12. *$\mathcal{P}_{\text{tok}}$ decides φ_k.*

Proof. For a configuration C we define

$$\text{val}(C) := \min\{k, \sum_{q \in Q} C(q) \cdot q\} \qquad \text{pot}(C) := \max[\![C]\!]$$

Note that executing transitions does not change the value of val and can never decrease the value of pot.

Let $\sigma = C_0 C_1 ...$ denote a run and let $C \in \inf(\sigma)$.

Case 1: $\text{val}(C) < k$**.** Clearly, $C(k) = 0$ by the definition of val, so C is a 0-consensus, and $k > \text{val}(C) = \text{val}(C_0)$, implying $\varphi_k(C_0) = 0$.

Case 2: $\text{val}(C) = k$**.** If in C one agent is in a state $q < k$, then another agent must be in a state $p > 0$ (as $\text{val}(C) = k$ could not hold otherwise). These two agents can interact, resulting in one agent moving to state $\min\{q + p, k\} > q$. In particular, if $\text{pot}(C) < k$, then we find a configuration C' with $\text{pot}(C') > \text{pot}(C)$ and $C \to C'$. However, $C' \not\to C$, a contradiction (Definition 3).

Therefore, C must have at least one agent in state k. Since interacting with that agent increases the number of agents in state k, and that number cannot decrease, we find that all agents are in state k. Hence C is a 1-consensus, and $k = \text{val}(C) = \text{val}(C_0)$ implies $\varphi_k(C_0) = 1$. □

Succinct protocols for flock-of-birds predicates. While $\mathcal{P}_{\text{tok}}$ is simple to describe, it uses $\Omega(k)$ states to decide φ_k. Going back to our motivation of chemical reactions, we can consider, say, the case $k = 10^{23}$, where $\Omega(k)$ requires 10^{23} states, clearly too many.

Definition 13. *Let $(\mathcal{P}_k)_{k \in \mathbb{N}}$ denote a family of population protocols, where $\mathcal{P}_k$ decides φ_k. We say that $\mathcal{P}_k$ is* succinct *if it has $\mathcal{O}(\text{polylog}\, k)$ states.*

With the above definition, we can formalise the search for a smaller protocol: Does a succinct population protocol for φ_k exist?

As it turns out, this is the case, and it is not difficult to see why. Essentially, we take the protocol from above and restrict the number of tokens held by each agent to be a power of two.

More precisely, let $k = 2^p$ be a power of two, with $p \in \mathbb{N}$. We will construct a succinct protocol $\mathcal{P}_{\text{tok2}} := (Q, \delta, I, O)$ for φ_k. (It is possible to generalise this construction to arbitrary k, see [15].)

As states we use the powers of two up to k, as well as 0, so $Q := \{0\} \cup \{2^0, 2^1, ..., 2^p\}$. Input and output are as for $\mathcal{P}_{\text{tok}}$: $I := \{1\}$, $O(k) := 1$ and $O(q) := 0$ for $q < k$.

For the transitions, we simply take all transitions of $\mathcal{P}_{\text{tok}}$ that are still valid:

$$i, i \mapsto 2i, 0 \qquad \text{for } 2i < k \qquad \langle \textsf{collect} \rangle$$
$$i, j \mapsto k, k \qquad \text{for } i + j \geq k \qquad \langle \textsf{accept} \rangle$$

Lemma 14. *$\mathcal{P}_{\text{tok2}}$ decides φ_k.*

Proof (sketch). Since $\mathcal{P}_{\text{tok2}}$ results from taking $\mathcal{P}_{\text{tok}}$ and deleting transitions, most of the arguments from the proof of Lemma 12 still apply. Intuitively, it remains to argue that $\mathcal{P}_{\text{tok2}}$ can always make progress towards reaching state k if at least k agents are present.

Consider any configuration C with at least k tokens in total. We have

$$\sum_{q \in Q \setminus \{0,k\}} q = \sum_{i=0}^{p-1} 2^i = 2^p - 1 < k$$

Therefore, if every state in $Q \setminus \{0, k\}$ had at most one agent, the total number of tokens is less than k. Since we assume the contrary, there must be at least one state $q \in Q$ with $0 < q < k$ and $C(q) \geq 2$. Hence two agents in state q can execute transition $\langle \mathsf{collect} \rangle$. □

The protocol $\mathcal{P}_{\text{tok2}}$ uses only $\mathcal{O}(\log k)$ states to decide φ_k. In particular, it is a succinct protocol, and e.g. for $k \approx 10^{23}$, it would use roughly 77 states. This provides our first nontrivial upper bound on the space complexity of flock-of-birds predicates.

Theorem 15 ([15])**.** *φ_k has space complexity $\mathcal{O}(\log k)$.*

4.2. Lower Bounds

As we have seen in the previous section, it is relatively easy to construct succinct protocols for flock-of-birds predicates. It is entirely unclear, however, how one could improve further upon this construction, and it seems difficult to do so, if not impossible.

In this section we will endeavour to prove this impossibility, and provide *lower bounds* on the state complexity. We start by discussing an existential lower bound proved in [15], which applies only to infinitely many φ_k, before moving to one of the main results in this thesis: a *general* lower bound on the state-complexity of flock-of-birds predicates.

Existential lower bound. The first lower bound relies on a simple counting argument: a population protocol $\mathcal{P} = (Q, \delta, I, O)$ with $m := |Q|$ states can be described using $\mathcal{O}(m^4)$ bits (the most expensive part being $\delta \subseteq Q^4$). So there is at least one number $k \leq 2^{\mathcal{O}(m^4)}$ such that φ_k is not decided by any protocol with m states. (There are not enough protocols to decide all such k.)

Theorem 16 ([15])**.** *There are infinitely many k such that φ_k has space complexity $\Omega(\log^{1/4} k)$.*

So in some sense the construction of the previous section is already optimal. This is not very satisfying, however, since it does not exclude the possibility of certain φ_k having much lower space complexity — and in practice, one often is not concerned with determining whether a specific threshold is met. Instead, it suffices to check a threshold that has roughly the right order of magnitude.

Therefore, we look for a *general* lower bound, which applies to all φ_k.

4.2.1. General lower bound

We prove the following:

Theorem 17 ([26, 29])**.** *φ_k has space complexity $\Omega(\log \log k)$*

The proof is technically involved and makes use of results from the theory of Petri nets and integer linear programming. In this section, I will sketch the main ideas of the proof, focusing on the parts as they relate to population protocols.

Notation. To ease presentation, I will simplify all polynomials of m to be just m. For example, instead of $m! \cdot m^{2m}$ I write 2^m, and instead of $2^{m^2 2^{m \log m}}$ I write 2^{2^m}.

Let us also recall some notation regarding multisets. In particular, for a multiset $C \in \mathbb{N}^Q$ and $k \in \mathbb{N}$ we write $C \geq k$ if $C(q) \geq k$ for each $q \in Q$, and kC denotes the multiset with $(kC)(q) = k \cdot C(q)$ for $q \in Q$.

Overview. Let $\mathcal{P} = (Q, \delta, \{q_{\text{init}}\}, O)$ be a population protocol deciding φ_k, where $k > 2^{2^m}$, with $m := |Q|$. For technical reasons, we assume wlog that all states of $\mathcal{P}$ are reachable. Our overall goal is to derive a contradiction by showing that $\mathcal{P}$ rejects some input $x \geq k$.

The proof proceeds as follows.

- We start with an initial configuration $C_0 := (k-1)q_{\text{init}}$, which is the largest inputs that should be rejected.
- We show that $C_0 \to C_{\text{many}} \to C_{\text{cons}}$, for some configuration $C_{\text{many}} \geq 2^m$ and a stable 0-consensus C_{cons}.
- We find a (nonempty) configuration D fulfilling two properties:
  (a) $C_{\text{many}} + |D| \cdot q_{\text{init}} \to C_{\text{many}} + D$, and
  (b) $C_{\text{cons}} + D$ is a stable 0-consensus.
- Overall, we have $(k - 1 + |D|)q_{\text{init}} \to C_{\text{cons}} + D$ (see Lemma 6). But this is a contradiction, since $C_{\text{cons}} + D$ is a stable 0-consensus, but the initial configuration has $k - 1 + |D| \geq k$ agents and must be accepted (see Lemma 9).

Many agents in every state. First, we consider the configuration C_{many}. Its main feature is that every state has at least 2^m agents. It is possible to reach such a configuration:

Lemma 18 ([29, Lemma 5.4])**.** *There is some configuration C_{many} with $C_{\text{many}} \geq 2^m$ and $C_0 \to C_{\text{many}}$.*

Proof. In the configuration C_0, some transition to a state $q_1 \notin [\![C_0]\!] = \{q_{\text{init}}\}$ must be enabled. We can execute that transition $|C_0|/4$ times, reaching a configuration C with $C(q) \geq |C_0|/4 = 2^{2^m - 2}$ for $q \in \{q_1, q_{\text{init}}\}$.

Again, some transition to a state $q_2 \notin [\![C]\!]$ must be enabled, and the above process can be repeated, until all states have agents. Since this requires up to m repetitions, and the number of agents is divided by 4 in each iteration, we end up with at least $2^{2^m - 2m} \geq 2^m$ agents in each state. □

Moreover, from C_{many} we can reach some stable 0-consensus. This is simply a consequence of the fact that $\mathcal{P}$ decides φ_k and C_{many} has fewer than k agents.

Lemma 19 ([29, Lemma 5.4]). *$C_{\text{many}} \to C_{\text{cons}}$, for some stable 0-consensus C_{cons}.*

Proof. Since $C_0 \to C_{\text{many}}$ by construction and $\varphi_k(C_0) = 0$, we can take some execution from C_0 to C_{many} and extend it to a run σ. As $\mathcal{P}$ decides φ_k, the (nonempty) set $\inf(\sigma)$ contains only 0-consensuses. Fix some $C_{\text{cons}} \in \inf(\sigma)$. Since σ is a run, the set $\inf(\sigma)$ is closed under $\to$, so C_{cons} is stable as well. □

Invisible agents. We have found some stable 0-consensus C_{cons}. We are going to argue a remarkable fact: there is some state $q \in Q$, such that the configuration $C_{\text{cons}} + q$ is still a stable 0-consensus. In fact, any state q with $C_{\text{cons}}(q) \geq 2^{2^m}$ is suitable, and we set S to be the set of such states. This insight comes from the theory of Petri nets and, in particular, Rackoff's theorem [50]. Essentially, it states that in a stable consensus, the protocol can "count" only up to 2^{2^m}.

Our goal is to "smuggle" one or more agents into S. We deliberately construct a sequence of transitions that will move agents from the initial state into S, while the remaining agents still reach C_{cons}. Once it has moved to $C_{\text{cons}} + D$, for some nonzero $D \in \mathbb{N}^S$, it is too late for the protocol — that configuration is a stable 0-consensus, so the protocol has rejected an input with at least k agents. This would contradict the assumption that the protocol decides φ_k.

Lemma 20 ([29, Lemma 3.2]). *Let $S := \{q \in Q : C_{\text{cons}}(q) \geq 2^{2^m}\}$, and $D \in \mathbb{N}^S$. Then $C_{\text{cons}} + D$ is a stable 0-consensus.*

Smuggling. Lemma 20 gives us a set S such that adding an agent to any state in that set keeps C_{cons} a 0-consensus. However, we still need to "smuggle" some agents into S, i.e. construct an execution that reaches $C_{\text{cons}} + D$. Intuitively, we need to prove that it is possible to execute transitions to move agents from the initial state q_{init} to *precisely* S — if we have one agent anywhere else, $C_{\text{cons}} + D$ need no longer be a stable 0-consensus. However, the only thing we know about S is that it is possible to put "many" agents into those states.

As it turns out, this is enough, and one can prove the following:

Lemma 21 ([29, lemmata 5.1, 5.8]). *There is some $D \in \mathbb{N}^S$ with $C_{\text{many}} + |D| \cdot q_{\text{init}} \to C_{\text{many}} + D$.*

Proof (sketch). The proof first considers a relaxed notion of reachability, where the number of agents is allowed to go negative temporarily. Under this assumption, reachability can be described by an integer linear program. We then use a theorem due to Pottier [49], showing that every solution to this integer linear program can be decomposed into "small" parts. Using the fact that $C_0 \to C_{\text{cons}}$ also yields a solution, we find a configuration $D \in \mathbb{N}^S$ and a solution that moves agents from the initial state into D — under the relaxed notion of reachability.

Hence, in the relaxed execution, the number of agents in some states might become negative. However, one can prove that it never drops below -2^m, and, using $C_{\text{many}} \geq 2^m$, we get $C_{\text{many}} + |D| \cdot q_{\text{init}} \to C_{\text{many}} + D$. □

This concludes the proof of Theorem 17.

4.3. Very Succinct Protocols

With the lower bound of the previous section, and the construction in the section before that, we have now determined that the space complexity of φ_k lies in $\Omega(\log\log k) \cap \mathcal{O}(\log k)$.

This raises a new question: Are there "very succinct" flock-of-birds protocols? I.e. can we find infinitely many k for which φ_k can be decided by a protocol with $o(\log k)$ states? There are several reasons why this seems unlikely.

- While there is a very succinct construction in population protocols with leaders, it relies on leaders in a crucial way, making it unlikely that the same construction would work in the basic model. (An explanation of leaders is given below.)
- There is a conditional lower bound, showing that very succinct, 1-aware protocols are impossible [15]. Essentially, 1-aware means that accepting runs can be characterised by the protocol reaching a certain state (see Definition 45). All existing constructions for flock-of-birds predicates, including the very succinct construction using leaders, are 1-aware.
- Most steps of the lower bound also apply to succinct flock-of-birds protocols. Only the use of Rackhoff's theorem does not. Since Rackhoff's theorem is a general result for Petri nets, there is hope that a more precise bound specific to population protocols can improve on it.
- Building on the previous point, one can improve the lower bound for silent population protocols, i.e. protocols which eventually reach a terminal configuration. In this case, Rackhoff's theorem is not needed, so we could show that very succinct, silent protocols are impossible. All existing constructions, for *any* predicate, are silent.

This led to the prevailing opinion at the time that very succinct flock-of-birds protocols were impossible. As the reader might suspect, however, it turns out that they do, in fact, exist — and since they must evade the conditional impossibility results mentioned above, they have certain interesting properties not seen in previous constructions.

In [25], I prove the following:

Theorem 22 ([25])**.** *For infinitely many k, the space complexity of φ_k is $\mathcal{O}(\log\log k)$, i.e. there is a population protocol $\mathcal{P}_{\text{tiny}}$ deciding φ_k with $\mathcal{O}(\log\log k)$ states.*

The proof of this theorem is technically involved.

- First, I introduce *population programs*, a small structured programming language with loops, procedures and branches.
- Within the model of population programs, I construct a very succinct program that can decide φ_k. This program builds upon an earlier construction by Lipton [44], but has to work under much weaker guarantees.

- I then show how to convert population programs into population protocols, via an intermediate model. This involves a leader election, trading one set of guarantees (agents start in a proper initial configuration) for another (the existence of a unique leader).

Over the next two sections, we will try to gain some intuition about the last two steps. In particular, in Section 4.3.1 we are going to look at population protocols with leaders, an extension of the population protocol model which is related to population programs. I then sketch how it is possible to perform a leader election, which imposes strong restrictions on the remainder of the protocol. Metaphorically speaking, the leader election leads to "unrest", meaning that the states of the non-leader agents become, in some sense, arbitrary.

Building on those restrictions, we then consider a simplified model in Section 4.3.2 and discuss the high-level ideas of construction, which strengthens a construction by Lipton to still work under the weak guarantees provided here.

4.3.1. Elections cause unrest

A common extension of population protocols adds a *leader* to the model — a unique agent in the initial configuration which can help synchronise the computation. It was already shown by [15] that the equivalent of Theorem 22 holds in the model with leaders. This result is not used in the proof of Theorem 22, but it is still instructive to look at the model with leaders and how it relates to the leaderless case.

Definition 23. *A* population protocol with leaders *is a tuple* $\mathcal{P} = (Q, \delta, I, O, q_{\mathrm{ldr}})$*, where* (Q, δ, I, O) *is a population protocol and* $q_{\mathrm{ldr}} \in Q \setminus I$ *is the* initial leader state*.*

The initial configurations *of* $\mathcal{P}$ *are precisely the configurations in* $q_{\mathrm{ldr}} + \mathbb{N}^I$ *(i.e. all configurations* C *with* $[\![C]\!] \subseteq I \cup \{q_{\mathrm{ldr}}\}$ *and* $C(q_{\mathrm{ldr}}) = 1$*). All other notions in Definitions 2 to 4 are defined analogously.*

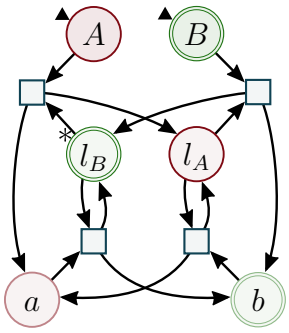


Figure 4.1.: $\mathcal{P}_{\mathrm{ldr}}$.

Let us consider an illustrative example for a population protocol with leaders $\mathcal{P}_{\mathrm{ldr}} := (Q, \delta, I, O, q_{\mathrm{ldr}})$ deciding the majority predicate $\varphi(A, B) \Leftrightarrow A \leq B$.

We use states $Q := \{A, B, a, b, l_A, l_B\}$, initial states $I := \{A, B\}$ and outputs $O(q) := 1$ for $q \in \{B, b, l_B\}$ and $O(q) := 0$ otherwise. The initial leader state is $q_{\mathrm{ldr}} := l_B$, and we have transitions

$$l_B, A \mapsto l_A, a \qquad \langle \mathsf{advise} \rangle$$
$$l_A, B \mapsto l_B, b$$
$$l_B, a \mapsto l_B, b \qquad \langle \mathsf{follow} \rangle$$
$$l_A, b \mapsto l_A, a$$

The protocol is drawn in Figure 4.1. As in the protocol $\mathcal{P}_{\mathrm{maj}}$ from Chapter 3, we have "active" agents in states A, B and "passive" agents in states a, b. Additionally, here we have a leader, either in state l_A or l_B. The state of the leader indicates the leader's

SPLIT(i):

goto s_2

s_1: $x_{i-1} := x_{i-1} - 1$

$x_i := x_i + 1$

$x_i := x_i + 1$

s_2: **if** $x_{i-1} \geq_{\text{nd}} 1$**: goto** s_1

MAIN:

$x_0 := x_0 + 1$

s_0: SPLIT(1)

$\vdots$

SPLIT(m)

goto s_0

Figure 4.3.: The counter machine $\mathcal{M}_{\text{pow}}$. (left) The macro SPLIT(i), $i \in \mathbb{N}$. (right) The instructions of $\mathcal{M}_{\text{pow}}$.

opinion on which side has the majority. Any active agent of the opposing side can change the leader's opinion using ⟨advise⟩, becoming passive in the process. So eventually the leader will have the correct opinion. After that point, all passive agents will follow the leader's opinion using transition ⟨follow⟩.

Counter machines. One of the main uses of leaders is the simulation of (a restricted kind of) *counter machines*. Grossly simplifying, these counter machines are equivalent to the model of population programs mentioned above.

For example, consider the counter machine $\mathcal{M}_{\text{ex}}$ with two registers x, y holding values in $\mathbb{N}$, as depicted in Figure 4.2. We always start our counter machines from an empty initial configuration, so we have $x = y = 0$ initially. The instructions of the counter machine can increment or decrement a register (instructions 1, 2 and 4, respectively), or execute a conditional jump (instructions 3 and 5). The test "$y \geq_{\text{nd}} 1$" is always false if $y = 0$, and it nondeterministically returns true or false if $y \geq 1$. Essentially, the counter machine can "detect" agents: if they are present it is possible (but not guaranteed) that they are found, but if they are absent, then they will certainly not be found.

1: $x := x + 1$
2: $y := y + 1$
3: **if** $x \geq_{\text{nd}} 1$**: goto** 1
4: $y := y - 1$
5: **if** $y \geq_{\text{nd}} 1$**: goto** 4
6: **goto** 6

Figure 4.2.: The counter machine $\mathcal{M}_{\text{ex}}$.

Due to this nondeterminism, the loop on lines 1-3 will eventually terminate, and the loop on lines 4-5 might terminate "prematurely", i.e. while $y > 0$. Overall, $\mathcal{M}_{\text{ex}}$ will end up in a configuration with $x = l_1$, $y = l_2$, for some values $l_1 \geq l_2$ chosen nondeterministically.

Please note that this is an instructive example and the behaviour of $\mathcal{M}_{\text{ex}}$ is not useful to us. Moreover, the counter machine model here has been simplified slightly for the benefit of presentation, and would not suffice for the proof of the full construction.

Counting with counter machines. Even with this limited set of primitives, these counter machines can decide large flock-of-birds predicates with a small number of instructions and registers. One such construction is shown in Figure 4.3, describing a counter machine $\mathcal{M}_{\text{pow}}$.

The machine $\mathcal{M}_{\text{pow}}$ uses registers $X = \{x_0, ..., x_m\}$, for some $m \in \mathbb{N}$. Initially, all registers have value 0. Then we run the counter machine, i.e. the instructions in MAIN. The first instruction sets x_0 to 1. We then execute the macro SPLIT(i) for $i \in \{1, ..., m\}$ some number of times in a loop. Whenever this happens, the machine tries to decrement

x_{i-1} and increase x_i by 2.

As a result, the initial value $x_0 = 1$ will eventually be doubled between each register, until we end up in the configuration with $x_0 = ... = x_{m-1} = 0$ and $x_m = 2^m$.

We shall now try to formalise this notion. Consider some counter machine with l instructions and registers X. We define a *configuration* to be a pair $D = (i, C) \in \{1, ..., l\} \times \mathbb{N}^X$, where i is the index of the instruction that is currently being executed, and C maps each register to its current value. We say that D is *terminal* if the i-th instruction is "**goto** i". The *size* of D is $|D| := |C|$.

Observation 24. *$\mathcal{M}_{\text{pow}}$ reaches a terminal configuration $C \in \mathbb{N}^X$ with $|C| = 2^m$.*

Let me remark that the construction of the counter machine is very similar to the protocol $\mathcal{P}_{\text{tok2}}$ in Section 4.1.

So in some sense $\mathcal{M}_{\text{pow}}$ "counts" up to 2^m, using $\mathcal{O}(m)$ instructions (and thus $\mathcal{O}(m)$ registers as well). Intuitively, it stands to reason that such a counter machine can be used to succinctly decide the predicate φ_{2^m} — in a moment, we will discuss this in more detail. First, let us turn towards the construction of Lipton, which can be seen as an improvement to $\mathcal{M}_{\text{pow}}$:

Theorem 25 ([44])**.** *There is a counter machine $\mathcal{M}_{\text{lip}}$ with $\mathcal{O}(m)$ instructions, such that $\mathcal{M}_{\text{lip}}$ reaches a terminal configuration $C \in \mathbb{N}^X$ with $|C| = 2^{2^m}$.*

Note also that this is the best possible result, according to Rackoff's theorem [50].

Simulating counter machines. Now, let us see how the counter machine $\mathcal{M}_{\text{ex}}$ could be implemented using a protocol with leaders. While only an example, the construction readily generalises to arbitrary counter machines.

Let $\mathcal{P}_{\text{cm}} := (Q, \delta, I, O, q_{\text{ldr}})$, with states $Q := \{x, y, \mathsf{R}\} \cup \{l_1, ..., l_6\}$ and input $I := \{\mathsf{R}\}$. The state R functions as a "reservoir" — when incrementing and decrementing registers, the agents are taken from and returned to R, respectively. Initially, all registers are 0, so R is the only initial state.

The leader stores the index of the current instruction. So we set $q_{\text{ldr}} := l_1$. Executing increment and decrement instructions is simple:

$$l_1, \mathsf{R} \mapsto l_2, x \qquad \text{"1: } x := x + 1\text{"} \qquad \langle \text{1-incr} \rangle$$
$$l_2, \mathsf{R} \mapsto l_3, y \qquad \text{"2: } y := y + 1\text{"} \qquad \langle \text{2-incr} \rangle$$
$$l_4, y \mapsto l_5, \mathsf{R} \qquad \text{"4: } y := y - 1\text{"} \qquad \langle \text{4-decr} \rangle$$

We assume that the "reservoir" R holds a sufficiently large number of agents. Hence we can increment and decrement registers by moving agents to and from the reservoir, respectively. If this assumption is not met, the protocol will get "stuck" at one of the increment instructions.

The conditional jumps are slightly more involved.

$$\begin{matrix} l_3, x \mapsto l_1, x \\ l_3, \mathsf{R} \mapsto l_4, \mathsf{R} \end{matrix} \qquad \text{"3: } \textbf{if } x \geq_{\text{nd}} 1\textbf{: goto } 1\text{"} \qquad \langle \text{3-cond} \rangle$$

$$\begin{matrix} l_5, y \mapsto l_4, y \\ l_5, \mathsf{R} \mapsto l_6, \mathsf{R} \end{matrix} \qquad \text{"5: } \textbf{if } y \geq_{\text{nd}} 1\textbf{: goto } 4\text{"} \qquad \langle \text{5-cond} \rangle$$

In $\langle$3-cond$\rangle$, the leader tries to meet an agent in state x. If it does, then the check returns 1 and we have to jump to instruction 1. However, if no agent is in state x then the first transition can never occur. To ensure that progress is made, the leader can also interact with an agent in R. Since we assume that R contains sufficiently many agents, the second transition is always enabled and will eventually be executed, if x is empty.

Clearly, it is also possible that x is nonempty, but the second transition is still executed. This is fine, since "$x \geq_{\text{nd}} 1$" is allowed to nondeterministically return 0 in that case.

Finally, the unconditional jump in instruction 6 remains. However, unconditional jumps do not require any transitions. Instead of jumping from instruction i to instruction j, we can identify l_i and l_j. Here, this results in identifying l_6 with itself, which results in the desired behaviour of an infinite loop once instruction 6 is reached.

Overall, the number of states is linear in the number of instructions used by the counter machine. Using the above method to simulate $\mathcal{M}_{\text{lip}}$, one can thus already prove the existence of very succinct protocols in the model with leaders.

Theorem 26 ([15])**.** *For infinitely many k there exists a population protocol with leaders $\mathcal{P}$ with $\mathcal{O}(\log \log k)$ states deciding φ_k.*

Proof (sketch). We use the construction sketched above to simulate the counter machine $\mathcal{M}_{\text{lip}}$ of Theorem 25. In that construction, R is the initial state, so the counter machine starts with a reservoir of n agents, where n is the size of the initial configuration of the population protocol.

Based on the guarantees provided by Theorem 25, there is a number $k \geq 2^{2^m}$ such that one of two things will happen:

- If $n \geq k$, the reservoir R is sufficiently large to accommodate the counter machine simulation and it will eventually reach a terminal configuration.
- Otherwise, at some point the counter machine simulation becomes stuck when incrementing a register and will not reach the terminal configuration.

To decide the predicate $\varphi_{2^{2^m}}$ it thus suffices to distinguish between the two cases. This can easily be done: as soon as the leader enters a terminal configuration, it moves itself, and every agent it meets, to some accepting state. All other states are rejecting. □

However, it should also be noted that [15] does not prove the theorem in this way.

Leader election. Seeing how one can simulate counter machines using leaders, and construct very succinct protocols in that way, the obvious next step would be to build a (leaderless) population protocol that first executes a leader election, and simulates the counter machine after the leader election has finished.

To make this more concrete, let us modify the protocol $\mathcal{P}_{\text{cm}} = (Q, \delta, I, O, q_{\text{ldr}})$ from above. We define $\mathcal{P}'_{\text{cm}} := (Q, \delta', I', O, l_1)$. As new initial state we choose $I' := \{l_1\}$ (instead of $I = \{\mathsf{R}\}$), and we set $\delta' := \delta \cup \langle$elect$\rangle$, with the new transition defined as follows:

$$l_i, l_j \mapsto l_1, \mathsf{R} \qquad \text{for } i, j \in \{1, ..., 6\} \qquad \langle\text{elect}\rangle$$

So in the initial configuration we now have a large number of agents occupying state l_1. But whenever two leaders meet, one moves to the reservoir R, and the other resets back to state l_1, restarting the counter machine. We can formalise this as follows:

Lemma 27. *Let $C_0 \in \mathbb{N}^{I'}$ denote a nonempty initial configuration of $\mathcal{P}'_{\mathrm{cm}}$. Then any run from C_0 reaches some $C \in \mathbb{N}^Q$ with $C(l_1) = 1$ and $C(l_2) = ... = C(l_6) = 0$.*

Proof. Recall that for a set $S \subseteq Q$ we write $C(S) := \sum_{q \in S} C(q)$. Let $L := \{l_1, ..., l_6\}$.

Let $\sigma = C_0 C_1 C_2 ...$ denote a run starting from C_0 and $C' \in \inf(\sigma)$. We first note that $C_i(L)$ can never increase, and can only decrease via $\langle \mathsf{elect} \rangle$, but that transition cannot decrease it to 0. Since $C_0(L) = |C_0| > 0$ and $C_0 \to C'$, we find that $C'(L) \geq 1$. If we had $C'(L) \geq 2$, then $\langle \mathsf{elect} \rangle$ would be enabled at C' and we could, irreversibly, reduce the number of agents in L, contradicting that $\inf(\sigma)$ is closed under $\to$.

So $C'(L) = 1$. If $C_0(L) = 1$, then the statement already holds (note that l_1 is the only initial state), so assume $C_0(L) \geq 2$. Hence there is some minimal i with $C_i(L) = 1$, and we have $C_{i-1}(L) = 2$. Therefore, $C_{i-1} \to_{\langle \mathsf{elect} \rangle} C_i$ and $C_i(L) = C_i(l_1) = 1$ while l_2, l_3, l_4 are empty in C_i, proving the lemma. □

To summarise, without much trouble we can build a leader election and will eventually end up with exactly one leader, and that leader will even be in its initial state. However, there is one important drawback: in the configuration C from Lemma 27 we no longer have the guarantee that $C(x) = C(y) = 0$, as we would have for the initial configuration of the protocol with leaders, $\mathcal{P}_{\mathrm{ldr}}$.

To emphasise: adding a leader election to our counter machine simulation means that the counter machine no longer starts from an initial register configuration (i.e. all registers set to 0), but instead starts from an *arbitrary* register configuration.

The problem is easy to see. Before the configuration with exactly one leader is reached, multiple leaders are already present and may start execution of the counter machine. In doing that, agents can be moved between the non-leader states x, y and R. When two leaders eventually meet, the state of the leader is reset, but the registers remain as they were.[1]

There are two obvious approaches for solving this problem, neither of which works:

- We could try delaying the start of the counter machine simulation until the leader election has finished. Unfortunately, this is impossible within a population protocol. Consider e.g. an execution $C_0 \to C$, where C is due to Lemma 27. In particular, the leader election has finished in C and we can reach some configuration C' where the counter machine simulation has started. But now $C_0 \to C'$, which implies $C_0 + l_1 \to C' + l_1$ (see Lemma 6).
- Instead, perhaps we do not simply reset the leader back to its initial state, but either "undo" the prior computation, or "merge" the states of the two simulated counter machines. This faces a similar issue as before, in that a leader cannot

[1] Admittedly, this does not show that an arbitrary configuration can be reached; and one could indeed slightly restrict the possible register configurations after the leader election concludes. However, I believe that these restrictions would be too weak to help.

track the values of the registers, having only finitely many states, and there is no way to determine conclusively whether a register is empty. Consider also that the two counter machines that have been running in parallel have not been running independently, but rather operate on the same set of registers.

4.3.2. Making Lipton's construction robust

Instead of restoring the guarantees on the initial configuration, we can strengthen Lipton's construction to deal with the weaker guarantees provided by the leader election. In this section, I will attempt to sketch the main ideas of the construction in a simplified model.

A new primitive. Consider the counter machine model from the previous section. It used the primitive "$x \geq_{\text{nd}} 1$", which nondeterministically returns true or false if $x \geq 1$, and always returns false otherwise.

We now assume that we have a counter machine with registers $X = \{x_0, ..., x_m\}$, and we augment the model with the primitives "$x_i \geq_{\text{lip}} 2^{2^i}$" and "$x_i >_{\text{lip}} 2^{2^i}$", for $i \in \{0, ..., m\}$. They function analogously to $x \geq_{\text{nd}} 1$, i.e. they nondeterministically return true or false if $x_i \geq 2^{2^i}$ and $x_i > 2^{2^i}$, respectively, and always return false otherwise.

It should be intuitive that it is possible to construct a very succinct population protocol for φ_k given this primitive. However, there is a catch: the checks "$x_i \geq_{\text{lip}} 2^{2^i}$" and "$x_i >_{\text{lip}} 2^{2^i}$" are only guaranteed to work correctly if the precondition $x_{i-1} = 2^{2^{i-1}}$ holds and if "$x_{i-1} >_{\text{lip}} 2^{2^{i-1}}$" works correctly, for $i \in \{1, ..., m\}$. The case $i = 0$ is guaranteed to always work. If the precondition is not met, the primitive might return an arbitrary value, and it also might not terminate within finite time.

Lipton's construction, essentially, shows how this primitive can be implemented in our counter machine model, and this leads to the machine $\mathcal{M}_{\text{lip}}$ of Theorem 25. In that context, one can ensure that the precondition always holds. However, since we are trying to design a counter machine that is able to run from *any* initial configuration, we cannot do the same.

I will not describe Lipton's construction in detail, so we shall assume that the primitive works as stated above (and later on, we will have to make further assumptions). In the remainder of this section, we will construct a counter machine $\mathcal{M}_{\text{chk}}$ that, starting from a configuration $(1, C)$, always terminates if there is an $i \in \{0, ..., m\}$ with $C(x_i) \neq 2^{2^i}$, and otherwise it is possible (but not guaranteed) that it does not terminate. This is an error checking routine — if a deviation from the expected configuration is detected, computation is terminated to indicate the error. We allow for false positives — even if C is as desired, $\mathcal{M}_{\text{chk}}$ may terminate.

Now, it is not clear how we would go from $\mathcal{M}_{\text{chk}}$ to a very succinct population protocol deciding φ_k, and this is another point I will not elucidate here. Suffice it to say that $\mathcal{M}_{\text{chk}}$ enables us to implement the $x_i \geq_{\text{lip}} 2^{2^i}$ primitive, and from there only a few technicalities remain.

The checking procedure. The overall structure of the error-checking routine $\mathcal{M}_{\text{chk}}$ is shown in Figure 4.4. To improve readability, it uses $\neg$ as a macro to simply invert the output of a condition; of course, this is straightforward to implement.

For the moment, let us assume that $\geq_{\text{lip}}$ and $>_{\text{lip}}$ work as intended, regardless of whether the precondition holds. Then, to check whether $x_i = 2^{2^i}$ holds, we essentially split the check into two parts:

- First, we check $x_i \geq_{\text{lip}} 2^{2^i}$ once and terminate if it returns false. If this returns true, we know that $x_i \geq 2^{2^i}$ holds, and conversely, if the latter holds, it is possible (though not guaranteed) that the former returns true.
- Second, we check $x_i > 2^{2^i}$ infinitely often, and terminate if it is true. Since we already know $x_i \geq 2^{2^i}$, we either have $x_i = 2^{2^i}$ and this is guaranteed to always return false. Otherwise, it will eventually return true.

s_0: **if** $\neg(x_0 \geq_{\text{lip}} 2^{2^0})$**: goto** t
$\vdots$
s_m: **if** $\neg(x_m \geq_{\text{lip}} 2^{2^m})$**: goto** t
r_0: **if** $x_0 >_{\text{lip}} 2^{2^0}$**: goto** t
$\vdots$
r_m: **if** $x_m >_{\text{lip}} 2^{2^m}$**: goto** t
goto r_0
t: **goto** t

Figure 4.4.: The error checking routine $\mathcal{M}_{\text{chk}}$.

Failure conditions. Of course, we have to show correctness without the simplifying assumption that $\geq_{\text{lip}}$ and $>_{\text{lip}}$ work as intended. But before dropping it entirely, we consider the case where $\geq_{\text{lip}}$ and $>_{\text{lip}}$ are guaranteed to return in finite time, and otherwise behave as described above.

Let i be minimal with $x_i \neq 2^{2^i}$. (Clearly, if no such i exists, it is possible that $\mathcal{M}_{\text{chk}}$ does not terminate.) In particular, this means that the primitives $x_i \geq_{\text{lip}} 2^{2^i}$ and $x_i >_{\text{lip}} 2^{2^i}$ work as intended. So if $x_i < 2^{2^i}$, line s_i will immediately move to t and terminate. Otherwise, $x_i > 2^{2^i}$. Since we assume that $\geq_{\text{lip}}$ and $>_{\text{lip}}$ return in finite time, line r_i is executed infinitely often, and must eventually move to t.

All that remains is ensuring that $\geq_{\text{lip}}$ and $>_{\text{lip}}$ always return in finite time. For this, the following observation is crucial: in the case $x_i < 2^{2^i}$, no care is needed, since line s_i will *always* move to t, and lines $s_0, ..., s_{i-1}$ only operate on $x_0, ..., x_{i-1}$, where the precondition holds.

So we are left with the case $x_i > 2^{2^i}$. Here we indeed have a problem: already line s_{i+1} might never terminate. We solve this by modifying the implementation of $\geq_{\text{lip}}$ and $>_{\text{lip}}$, such that checks on variables x_j with $j > i$ run the same check as line s_i infinitely often. (Naturally, we do not know i, so we do it for all $i \in \{1, ..., m\}$.) In other words, when executing e.g. $x_{i+1} \geq_{\text{lip}} 2^{2^{i+1}}$, we also check $x_i >_{\text{lip}} 2^{2^i}$ infinitely often, and if one of these checks returns true, we immediately terminate.

Summary. Overall, we get the following:

Theorem 22 ([25])**.** *For infinitely many k, the space complexity of φ_k is $\mathcal{O}(\log \log k)$, i.e. there is a population protocol $\mathcal{P}_{\text{tiny}}$ deciding φ_k with $\mathcal{O}(\log \log k)$ states.*

Proof (idea). Let $m \in \mathbb{N}$. We start with a counter machine $\mathcal{M}_{\text{vs}}$, using the ideas sketched in this section. Running $\mathcal{M}_{\text{vs}}$ on an arbitrary configuration C will result in one of two things: either $\mathcal{M}_{\text{vs}}$ terminates, or it runs infinitely long and in some way indicates whether $|C| \geq 2^{2^m}$ holds. In the first case, we nondeterministically move agents around and restart $\mathcal{M}_{\text{vs}}$. This ensures that eventually the second case occurs.

We then construct a population protocol $\mathcal{P}_{\text{tiny}}$ simulating $\mathcal{M}_{\text{vs}}$ with the above restarting behaviour, by using a counter machine simulation with an added leader election, similar to the description of Section 4.3.1. The leader election is guaranteed to eventually result in a single leader in its initial state (Lemma 27). After that point, the counter machine simulation decides the predicate. □

4.4. Arbitrary Predicates

Now that we have thoroughly discussed the space complexity of threshold predicates, let us turn towards arbitrary Presburger predicates. To do that, we will formally specify what a Presburger predicate is, and then extend the notion of succinct population protocols to predicates other than flock-of-birds predicates.

Presburger predicates. Presburger predicates, sometimes also called semilinear predicates, are precisely the properties expressible in *Presburger arithmetic*, the first-order theory of the natural numbers with addition. As is well-known, Presburger arithmetic admits a quantifier elimination procedure. Since the resulting quantifier-free fragment is a natural representation of Presburger predicates, which is well-suited to use in constructions of population protocols, this is the representation we will use.

Definition 28. *Let $X = \{x_1, ..., x_t\}$ and $\varphi : \mathbb{N}^X \to \{0, 1\}$. We say that φ is a* threshold predicate *if there exist $a_1, ..., a_t, k \in \mathbb{Z}$ with*

$$\varphi(x_1, ..., x_t) \Leftrightarrow \sum_{i=1}^{t} a_i x_i \geq k$$

We say that φ is a modulo predicate*, if there are $a_1, ..., a_t, k, m \in \mathbb{N}$ with $m > 0$ such that*

$$\varphi(x_1, ..., x_t) \Leftrightarrow \sum_{i=1}^{t} a_i x_i \equiv k \pmod{m}$$

Finally, a quantifier-free Presburger representation (QFPR) *is a boolean combination of threshold and modulo predicates, with coefficients encoded in binary. Any φ representable in this way is a* Presburger predicate*, and its* size $|\varphi|$ *(of φ) is the smallest length of such a representation.*

For example, the flock-of-birds predicate $\varphi_k(x) \Leftrightarrow x \geq k$ is a threshold predicate, and it has size $|\varphi_k| = \Theta(\log k)$.[2]

Using the size of a predicate, we can extend the definition of succinctness from above (Definition 13) to arbitrary Presburger predicates.

Definition 29. *Let $(\mathcal{P}_k)_{k\in\mathbb{N}}$ denote a family of population protocols and ψ_k a family of Presburger predicates, where $\mathcal{P}_k$ decides ψ_k. We say that $(\mathcal{P}_k)_{k\in\mathbb{N}}$ is* succinct *if it has $\mathcal{O}(\text{poly}|\psi_k|)$ states.*

[2]This is not entirely trivial — one must prove that the standard representation is the shortest.

Succinct protocols for Presburger predicates. Population protocols decide exactly the Presburger predicates, so it is possible to construct a population protocol for any such predicate [5]. But is it also possible to construct a *succinct* protocol? This question is resolved in the affirmative by [14]:

Theorem 30 ([14])**.** *There is a succinct family of population protocols deciding all Presburger predicates.*

This marks an important step towards practical constructions of population protocols. However, their construction has one glaring drawback: the protocols are slow, needing exponential (!) parallel time to stabilise.

We improve on their result and show that it is possible to construct succinct protocols that stabilise within linear parallel time [33] — matching a known lower bound for the majority predicate [1, 9].

Theorem 31 ([33])**.** *There is a succinct family of population protocols $(\mathcal{P}_k)_{k\in\mathbb{N}}$ deciding all Presburger predicates, such that each $\mathcal{P}_k$ stabilises in parallel time $\mathcal{O}(n)$.*

So far, we have only defined parallel time intuitively. A formal definition can be found in Section 4.4.4.

Major obstacles. Before we discuss how these constructions work, we will consider two major obstacles to producing succinct protocols. We are already familiar with the first: since the numbers are represented in binary, a coefficient k uses $\log k$ bits, and the protocol can only have $\mathcal{O}(\text{polylog}\, k)$ additional states for this coefficient. This is also the essential problem when constructing succinct flock-of-birds protocols, and it can be solved in the same way here as before, by having states represent different powers of two (see e.g. $\mathcal{P}_{\text{tok2}}$ in Section 4.1).

The second problem appears as a result of boolean combinations. The standard construction for a boolean combination in population protocols is the *parallel composition*, analogous to the product construction for finite automata.

In particular, given two protocols with m_1 and m_2 states, respectively, this construction will result in a protocol with $m_1 \cdot m_2$ states. Since a predicate φ may contain $\Omega(|\varphi|)$ boolean combinations, parallel composition results in protocols with exponentially many states. Clearly, this would not be succinct, so a new procedure for boolean combinations is needed.

Overview of the construction. The constructions of theorems 30 and 31 both follow the same overall structure.

- Represent the predicate φ as a boolean combination of threshold and modulo predicates $\varphi_1, ..., \varphi_s$. We refer to $\varphi_1, ..., \varphi_s$ as *subpredicates*.
- Construct a succinct (sub-)protocol for each subpredicate φ_i. It does not decide the subpredicate "fully", i.e. it does not try to reach a consensus. Instead, on input C_0 it will reach a terminal configuration C, and based on $[\![C]\!]$ one can determine the value of $\varphi_i(C_0)$. (Recall that $[\![C]\!] := \{q : C(q) > 0\}$ is the support of C.)

- Compose the subprotocols for the subpredicates by taking them "side by side", i.e. taking the union of their states and transitions. Then add new states and transitions that distribute the inputs into the subpredicates. A single agent in an input state will be split into multiple agents, such that it is represented in each subprotocol.
- Finally, determine the outputs of the subprotocols and combine them according to the boolean combination in φ.

These steps are implemented differently in the two constructions, and both are intricate. In this section, I will illustrate how to construct succinct subprotocols, which is a foundational idea present in both constructions. I will also demonstrate our version of the input distribution, which is the key idea allowing for linear time protocols. The presentation here is simplified in multiple ways:

- Instead of giving a general construction, I will focus on convenient special cases and one particular working example.
- An important technique present in both constructions is the use of *multiway transitions*, an extension of population protocols that allows more than two agents to interact at the same time. We will avoid these entirely here.
- As in the subprotocols, from an initial configuration C_0 the construction eventually reaches a terminal configuration C, and the output can be determined based on $[\![C]\!]$. In our construction, we show how to obtain a stable consensus from this kind of output condition. This is technically involved, so for us it will suffice to prove that $[\![C]\!]$ can be mapped to the correct output in some fashion.
- Instead of proving that the protocols stabilise in linear time, we only prove that they stabilise in quadratic time. (Though both are true.)
- We also adjust some minor technical details of the construction.

We take the predicate $\varphi(x, y) \Leftrightarrow (x + 2y \equiv 1 \pmod 5) \wedge (2x - y \geq 3)$ as running example, and write $\varphi_1(x, y) \Leftrightarrow x + 2y \equiv 1 \pmod 5$ and $\varphi_2(x, y) \Leftrightarrow 2x - y \geq 3$ for the two subpredicates.

The remainder of this section is structured as follows. In Sections 4.4.1 and 4.4.2 we will construct the subprotocols for the subpredicates φ_1 and φ_2, respectively. We combine those two in Section 4.4.3 into a protocol for φ, and finally prove its time complexity in Section 4.4.4.

4.4.1. Remainder Predicates

Let us start by sketching the construction for a general modulo predicate $\psi(x_1, ..., x_t) \Leftrightarrow \sum_i a_i x_i \equiv k \pmod m$. We fix $h := \max\{i : 2^i < m\}$ such that 2^h is the largest power of two below m. We make the following simplifying assumptions:

(A1) $0 \leq k < m$, a_i is a power of two with $0 < a_i < m$ for each i, and

(A2) $2^{h+1} - m = 2^a + 2^b$ for some $a, b \in \mathbb{N}$.

Assumption (A1) can be justified to not lose generality; however, (A2) cannot. We still assume it here, as it avoids the use of multiway transitions.

The protocol $\mathcal{P}_1 := (Q_1, \delta_1, I_1, O_1)$ uses similar ideas as $\mathcal{P}_{\text{tok2}}$ from Section 4.1. We use $Q_1 := \{0\} \cup \{2^0, 2^1, ..., 2^h\}$ as states, with $I_1 := \{a_1, ..., a_k\}$ (note that (A1) ensures $a_1, ..., a_k \in Q$). The idea is that every agent holds some value, and the total value of the agents is preserved during the computation.

$$q, q \mapsto 2q, 0 \qquad \text{for } q \in \{2^0, ..., 2^{h-1}\} \qquad \langle \textsf{double} \rangle$$

So if two agents meet, one takes the entire value. Of course, we need a special case if two agents in state 2^h meet:

$$2^h, 2^h \mapsto 2^a, 2^b \qquad \text{for } a, b \text{ as in (A2)} \qquad \langle \textsf{modulo} \rangle$$

Applying the above to $\varphi_1(x, y) \Leftrightarrow x + 2y \equiv 1 \pmod 5$ yields the subprotocol shown in Figure 4.5. (Recall that the dashed arrows all go to state 0.) Since the subprotocol does not stabilise to a consensus, we do not define an output. Instead, we can determine the output in the following manner, which depends on the support of the terminal configuration.

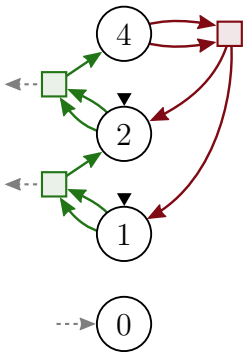


Figure 4.5.: The φ_1 subprotocol.

Lemma 32. *Let $C_0 \in \mathbb{N}^{I_1}$. Then every run from C_0 reaches a terminal configuration C_{term} with $\psi([\![C_{\text{term}}]\!]) = \psi(C_0)$.*

Proof. Let $\text{pot}(C) := \sum_q q \cdot C(q) - C(0)$, for any $C \in \mathbb{N}^{Q_1}$. We observe that executing any transition strictly reduces pot, and it is bounded below by $|C_0|$. So the protocol must reach some terminal configuration C_{term} in any run. Again, by inspecting the transitions we observe $\sum_q q \cdot C_{\text{term}}(q) \equiv \sum_q q \cdot C_0(q) \pmod m$. The statement then follows by noting $C_{\text{term}}(q) \leq 1$ for all $q \neq 0$, since otherwise C_{term} would not be terminal. □

This way of computing an output is not very useful by itself. But, importantly, the output now only depends on the *set* — not the multiset — of states in the configuration. As we show in the paper [33], it is possible to embed a component into the protocol that determines for each state whether it is present or not and can execute an arbitrary boolean circuit based on those results.

More concretely, we take some circuit, where the inputs correspond to the states Q, and which contains standard $\wedge, \vee, \neg$ gates, with one designated output gate. Given a boolean assignment $a : Q \to \{0, 1\}$ to the states (indicating whether those states are present, i.e. $a(q) \Leftrightarrow C_{\text{term}}(q) > 0$), the circuit will output $\psi([\![C_{\text{term}}]\!]) = \psi(C_0)$. For each gate of the circuit, we designate one agent to simulate that gate. The agent responsible for the output gate will thus eventually stabilise to the correct output, and broadcast this to all other agents. (It remains to show that a small boolean circuit can decide ψ for binary inputs, but that is easily done.)

4.4.2. Threshold Predicates

Moving on, we consider the case of threshold predicates, i.e. we fix some $\psi(x_1, ..., x_k) \Leftrightarrow \sum_{i=1}^{k} a_i x_i \geq t$. Similar to modulo predicates, we set $h := \min\{i : 2^i \geq 2m\}$, where $m := \max\{|a_1|, ...|a_k|\} \cup \{|t|\}$. (The precise choice for h is not obvious at this point, but will later become clear when we prove correctness.) We again assume that $a_1, ..., a_k$ are powers of two (but t need not be).

Let us define the protocol $\mathcal{P}_2 := (Q_2, \delta_2, I_2, O_2)$. The states are $Q_2 := \{-2^h, ..., -2^0\} \cup \{0\} \cup \{2^0, ..., 2^h\}$, with initial states $I_2 := \{a_1, ..., a_k\}$. We only have one kind of transition.

$$q, p \mapsto q + p, 0 \qquad \text{for } q, p, q + p \in Q_2 \qquad \langle \text{collect} \rangle$$

This is a general transition rule, stating "if two agents meet, one collects the total value, if that total can be represented within the state space". We can also look at it more concretely, and enumerate the possible transitions that may occur.

$$\begin{aligned} 2^i, -2^i &\mapsto 0, 0 && \text{for } i \in \{0, ..., h\} \\ 2^i, 2^i &\mapsto 2^{i+1}, 0 && \text{for } i \in \{0, ..., h-1\} \\ 2^{i+1}, -2^i &\mapsto 2^i, 0 && \text{for } i \in \{0, ..., h-1\} \end{aligned}$$

The result of applying this construction to the predicate $\varphi_2(x, y) \Leftrightarrow 2x - y \geq 3$ from above is depicted in Figure 4.6.

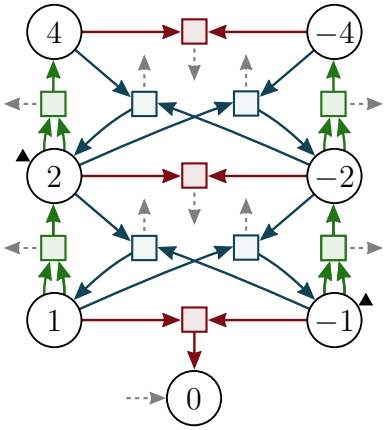


Figure 4.6.: $\mathcal{P}_2$, the subprotocol for φ_2.

We can prove the correctness in the same style as before, i.e. eventually the protocol will reach a terminal configuration, where the support of the configuration indicates the output of the predicate.

Lemma 33. *Let $C_0 \in \mathbb{N}^{I_2}$. Then every run from C_0 reaches a terminal configuration C_{term} with $\psi([\![C_{\text{term}}]\!]) = \psi(C_0)$.*

Proof. Clearly, every transition strictly increases the number of agents in state 0. (To be precise: some transitions instead leave the configuration unchanged, but those can be disregarded without issue.)

Hence eventually the protocol reaches a terminal configuration C_{term}. Observe that we can no longer argue $\sum_q q \cdot C_{\text{term}}(q) = \sum_{q \in [\![C_{\text{term}}]\!]} q$, as in the proof of Lemma 32, since it is possible that $C_{\text{term}}(q) \geq 2$ for some $q \neq 0$. In particular, this can happen for $q \in \{2^h, -2^h\}$ — and no other state, since every state p with $0 < |p| < 2^h$ has a transition $(p, p \mapsto 2p, 0) \in \delta_2$.

Let us assume that $C_{\text{term}}(2^h) \geq 2$. (The case $C_{\text{term}}(-2^h) \geq 2$ is analogous.) Then $C_{\text{term}}(-2^h) = C_{\text{term}}(-2^{h-1}) = 0$, as otherwise a transition would be enabled. We find

$$\sum_{q \in Q} q \cdot C_{\text{term}}(q) \overset{(1)}{\geq} \sum [\![C_{\text{term}}]\!] \overset{(2)}{\geq} 2^h - \sum_{i=0}^{h-2} 2^i \geq 2^h - 2^{h-1} = 2^{h-1} \overset{(3)}{\geq} k$$

where for step (1) we note that every state $q < 0$ has $C_{\text{term}}(q) \leq 1$; (2) follows from $C_{\text{term}}(2^h) \geq 1$ and $C_{\text{term}}(-2^h) = C_{\text{term}}(-2^{h-1}) = 0$; and (3) is due to our choice of h.

Overall, we have

$$\begin{aligned}
\psi(C_0) &\Leftrightarrow \sum_{i=1}^{k} a_i \cdot C_0(a_i) \geq k \\
&\Leftrightarrow \sum_{i=1}^{k} a_i \cdot C_{\text{term}}(a_i) \geq k && \text{(each transition preserves total value)} \\
&\Leftrightarrow \sum [\![C_{\text{term}}]\!] \geq k && \text{(both are true, due to } C_{\text{term}}(2^h) \geq 2\text{)} \\
&\Leftrightarrow \psi([\![C_{\text{term}}]\!])
\end{aligned}$$

□

4.4.3. Input Distribution

Now that we have the two subprotocols for the subpredicates $\varphi_1(x, y) \Leftrightarrow x + 2y \equiv 1 \pmod 5$ and $\varphi_2(x, y) \Leftrightarrow 2x - y \geq 3$, it is time to combine them towards a protocol for $\varphi \Leftrightarrow \varphi_1 \wedge \varphi_2$.

As a first step, we put the two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ "side-by-side", i.e. we consider the population protocol $(Q_1 \uplus Q_2, \delta_1 \uplus \delta_2, I, O)$ — inputs and outputs do not matter at this point, and we take the disjoint union of all states and transitions.

However, we do not want a fully disjoint union — the state 0 should be shared between all subprotocols and we will use it later. The idea is that each subprotocol "produces" agents in state 0 and that they can be consumed by any subprotocol, as well as the initialisation routine. Hence we will merge all 0 states of all subprotocols and rename the resulting state R, similar to the reservoir from Section 4.3.

So formally, we construct a population protocol $\mathcal{P}_{\text{fas}} := (Q, \delta, I, O)$, with $Q := f_1(Q_1) \cup f_2(Q_2) \cup Q'$, where Q' are states that we describe later, $f_i(q) := q_i$ for $i \in \{1, 2\}$, $q \in Q_i \setminus \{0\}$, and $f_i(0) := \mathsf{R}$. For example, f_1 renames the state 4 to 4_1, and 0 to R. We extend f_i to transitions in the obvious way and set $\delta := f_1(\delta_1) \cup f_2(\delta_2)$.

To get to a working protocol, we need two things:

- We need to distribute the input, i.e. have one common input state x that is distributed to the states $1_1, 2_1$ and one state y that is distributed to $2_1, -1_2$ (the states correspond to the coefficients of the variables in the subpredicates).
- The output of the subprotocols needs to be determined, combined, and turned into a consensus. As mentioned above, we will not do this here.

The running example has been carefully chosen so that the input distribution can be implemented without the use of multiway transitions. To simplify things further, we shall not give a general construction here (not even in a special case), and just focus on the example.

Input distribution. We start by adding states x, y to Q'. Based on the description above, the following transitions present themselves.

$$\begin{aligned} x, 0 &\mapsto 1_1, 2_2 \\ y, 0 &\mapsto 2_1, -1_2 \end{aligned} \qquad \langle \mathsf{distribute} \rangle$$

The idea is to distribute the input agents into the subprotocols using an additional agent in state R. While this decreases the number of agents in R, the agents in the subprotocols will interact with each other and generate more.

For example, we can have the following transition sequence.

$$2 \cdot x + 2 \cdot \mathsf{R} \to 2 \cdot 1_1 + 2 \cdot 2_2 \to 2_1 + 4_2 + 2 \cdot \mathsf{R}$$

However, there is one glaring problem: at the beginning of the execution, state R is empty and no input can be distributed. But without any input distribution, the subprotocol cannot generate any agents in R.

To resolve this issue, we allow the input distribution to "kickstart" itself. We define $Q' := \{x, y, X, Y\}$. Intuitively, one agent in X represents two agents in x. In the following transition, two input agents are combined in this way:

$$\begin{aligned} x, x &\mapsto X, \mathsf{R} \\ y, y &\mapsto Y, \mathsf{R} \end{aligned} \qquad \langle \mathsf{combine} \rangle$$

This will generate enough R agents to start the computation. Of course, we then also need to distribute the agents in X and Y.

$$\begin{aligned} X, 0 &\mapsto 2_1, 4_2 \\ Y, 0 &\mapsto 4_1, -2_2 \end{aligned} \qquad \langle \mathsf{distribute} \rangle$$

Finally, as input states we define $I := \{x, y\}$. The resulting protocol is sketched in Figure 4.7.

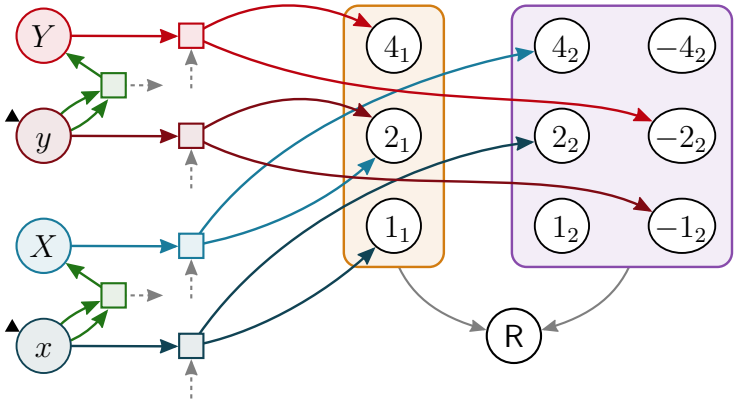


Figure 4.7.: Input distribution for $\varphi(x, y) \Leftrightarrow (x + 2y \equiv 1 \pmod 5) \wedge (2x - y \geq 3)$.

Small inputs. One final caveat remains. Consider e.g. the input $x = y = 1$. In this case states X, Y cannot be reached, and no agents in R can be generated. So we again have the problem that the computation cannot start at all.

However, this problem occurs only for "small" inputs. More precisely, there is some constant μ such that every input with at least μ agents can distribute its entire input.

This issue is also encountered by [14]. They solve it by constructing a protocol specifically for small inputs and combining it with the other construction. Since the time complexity of a protocol depends only on the speed on large inputs, we can simply re-use their small-input construction unchanged.

4.4.4. Time Complexity

We now turn towards the final task: proving that the above protocols are fast, i.e. they stabilise in linear time. To do this, we must first define the concept of time. Afterwards, we show that the protocol in the example stabilises in quadratic time, illustrating the techniques used in the time complexity analysis in the paper.

Definition of stabilisation time. To define the concept of parallel time, we introduce a particular random process, modelling how the agents interact. For this to make sense, it is best if a pair of agents can execute at most one transition — otherwise, we would also have to define how the agents choose between the possible transitions.

Definition 34. *A population protocol $\mathcal{P} = (Q, \delta, I, O)$ is* deterministic *if δ is a partial function, i.e. for $(q, p) \in Q^2$ there exists at most one pair $(q', p') \in Q^2$ with $(q, p \mapsto q', p') \in \delta$. In this case we write $\delta(q, p) := (q', p')$; and if no such pair exists, we set $\delta(q, p) := (q, p)$.*

The idea is that we execute one step of the protocol by picking two distinct agents uniformly at random. Those two agents execute a transition, if one exists, otherwise nothing happens.

Definition 35. *Let $\mathcal{P} = (Q, \delta, I, O)$ be a deterministic population protocol. Fix some configuration $C \in \mathbb{N}^Q$. Let q, p denote Q-valued random variables, where q is obtained from sampling the distribution $C/|C|$, and p from sampling $(C - q)/(|C| - 1)$. Then we define* $\mathrm{step}(C)$ *as* $\mathrm{step}(C) := C - q - p + \delta(q, p)$.

The random run (starting in $C_0 \in \mathbb{N}^Q$) *is the homogeneous Markov chain $\sigma = C_0 C_1 ...$ with initial state C_0 and transition probabilities corresponding to* step.

In other words, $\mathrm{step}(C)$ is a $\mathbb{N}^Q$-valued random variable with distribution

$$\mathbb{P}(\mathrm{step}(C) = C') = \sum_{(q,p) \in S(C,C')} \frac{C(q) \cdot (C - q)(p)}{|C|(|C| - 1)}, \quad \text{where}$$

$$S(C, C') := \{(q, p) \in Q^2 : q + p \leq C \wedge C' = C - q - p + \delta(q, p)\}$$

for $C' \in \mathbb{N}^Q$. An example illustrating the computation of $\mathrm{step}(C)$ is given in Figure 4.8.

So step is a random variable assigning a probability to the next configuration. By repeatedly moving according to this process, we get a (homogeneous) Markov chain, from which we can sample executions of the protocol.

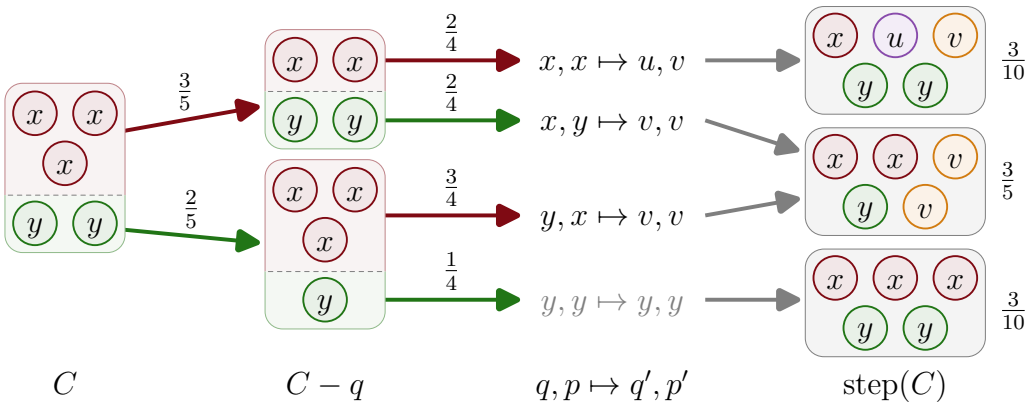


Figure 4.8.: Example for step(C), with $\delta = \{(x, x \mapsto u, v), (x, y \mapsto v, v), (y, x \mapsto v, v)\}$ and configuration $C = 3 \cdot x + 2 \cdot y$.

Definition 36. *Assume that $\mathcal{P}$ decides φ. Let $\mathcal{C} \subseteq \mathbb{N}^Q$, $C_0 \in \mathbb{N}^I$ and let $\sigma = C_0 C_1 \ldots$ denote a random run. Let $T := \min\{t : C_t \in \mathcal{C}\}$. We say that $\mathcal{P}$ reaches $\mathcal{C}$ in* (parallel) time $f(n)$*, if for all C_0 with $|C_0| = n$ the expectation of T is at most $n \cdot f(n)$. We say that $\mathcal{P}$* stabilises in (parallel) time $f(n)$ *if it reaches a stable consensus in time $f(n)$.*

At first glance, it might be unclear how the above definition is connected to Definition 4, i.e. the definition of how a protocol decides a predicate. Essentially, any trace of the random run is an infinite execution by the definition of step, and with probability 1 it is a run. Moreover, every run of $\mathcal{P}$ must eventually reach a stable $\varphi(C_0)$-consensus (see Lemma 9). In particular, after reaching a stable consensus, the output can no longer change, and we could stop the execution of the protocol at that point.

Note that the above definition allows for $n \cdot f(n)$ steps to occur in time $f(n)$ (instead of, say, $f(n)$ steps). This is the standard definition, since one assumes that n interactions occur "in parallel".

Proving quadratic stabilisation time. As mentioned above, our construction stabilises in linear parallel time. However, we will not prove this here and instead content ourselves with quadratic parallel time. Still, both proofs are based on the use of linear potential functions, and so I hope to also give some insight into the challenges faced by the linear-time proof.

The concept of potential functions was introduced in Section 3.1.2, and we have used them in many correctness proofs since. For the following time-complexity analysis, we use potential functions with two additional properties:

- They are a *linear* function of the configuration, and
- they are *strictly decreasing*, so every transition must strictly reduce the potential.

Definition 37. *Let (Q, δ, I, O) denote a population protocol. A function $\Phi : \mathbb{N}^Q \to \mathbb{N}$ is a* strong potential*, if*

(1) Φ is linear*, i.e. there is a $w : Q \to \mathbb{N}$ and $\Phi(C) = \sum_{q \in Q} w(q)C(q)$ for $C \in \mathbb{N}^Q$, and*

(2) $\Phi(q + p) > \Phi(q' + p')$ for all $(q, p \mapsto q', p') \in \delta$.

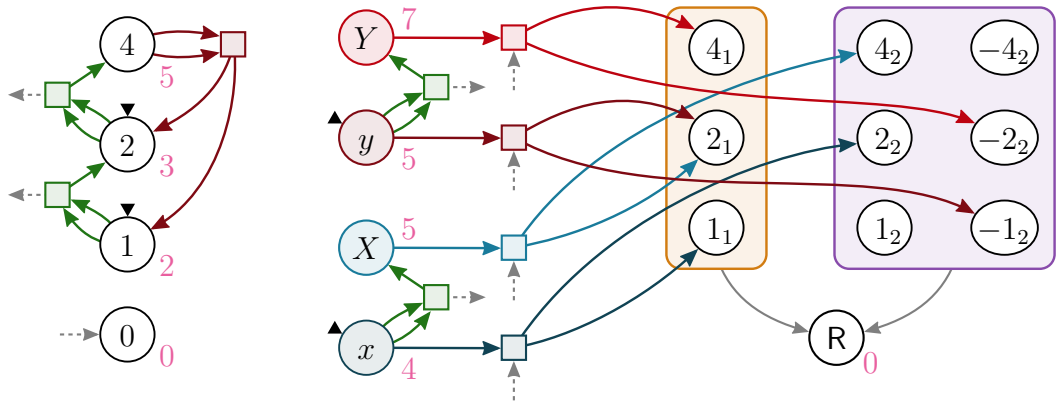


Figure 4.9.: Strong potentials for (a) $\mathcal{P}_1$ and (b) $\mathcal{P}_{\text{fas}}$

As an example, consider $\mathcal{P}_1 = (Q_1, \delta_1, I_1, O_1)$. We can define a strong potential Φ by choosing the weights $w(q) := 2q + 1$ for $q \in Q_1 \setminus \{0\}$ and $w(0) := 0$. This is illustrated in Figure 4.9(a). Executing any transition strictly reduces the potential of a configuration: ⟨double⟩ preserves total value but increases the number of agents in state 0, and ⟨modulo⟩ reduces total value.

Constructing a strong potential for $\mathcal{P}_2$ and $\mathcal{P}_{\text{fas}}$ is similarly straightforward. The latter is drawn in Figure 4.9(b).

Observation 38. *There are strong potentials for $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_{\text{fas}}$.*

Clearly, the existence of a strong potential proves that any run reaches some terminal configuration (which must be a stable consensus, if the protocol decides some predicate). But we can make an even stronger statement, namely that a terminal configuration is reached within quadratic parallel time. Essentially, we use that every non-terminal configuration enables some transition. This transition occurs after linear expected time and strictly decreases the (at most linear) potential.

Lemma 39 ([33, Proposition 24])**.** *Let $\mathcal{P} = (Q, \delta, I, O)$ denote a deterministic population protocol with a strong potential Φ. Then, from any initial configuration, $\mathcal{P}$ reaches a terminal configuration within $\mathcal{O}(n^2)$ parallel time.*

Proof. Let $C_0 \in \mathbb{N}^Q$ and $n := |C_0|$. We prove that C_0 reaches a terminal configuration within $\Phi(C_0) \cdot n$ parallel time, for any configuration C_0. This implies the overall statement due to $\Phi(C_0) \in \mathcal{O}(n)$.

Clearly, the statement holds for all terminal C_0. For the non-terminal configurations, we show it by induction on $\Phi(C_0)$. Assume that C_0 is not terminal and that we have already shown the statement for all C_0' with $\Phi(C_0') < \Phi(C_0)$.

Let $\sigma = C_0 C_1 \ldots$ be a random run, and T minimal such that $C_T \neq C_0$. In particular, we have $\Phi(C_0) > \Phi(C_T)$, since Φ is a strong potential. Moreover, T is geometrically distributed with success probability p, where p is the probability of $C_0 \neq C_1$.

As C_0 is not terminal, there is some $t = (q, p \mapsto q', p') \in \delta$ and $C \in \mathbb{N}^Q$ with $C_0 \to_t C$ and $C_0 \neq C$. We thus get $C_0 \geq q + p$, implying that $\mathrm{step}(C_0) = C$ with probability at least $1/n(n-1)$. Hence $p \geq 1/n(n-1)$, and the expectation of T is at most $n(n-1)$.

By induction hypothesis, C_T reaches a terminal configuration in parallel time $\Phi(C_T) \cdot n$, and thus C_0 does so in time

$$\Phi(C_T) \cdot n + \frac{n(n-1)}{n} \leq (\Phi(C_0) - 1) \cdot n + n - 1 \leq \Phi(C_0) \cdot n$$

□

Towards linear time. As we have shown above, the mere existence of a strong potential proves quadratic stabilisation time. Intuitively, the reason for this is that the proof only makes use of the fact that *some* agents can interact and lower the potential. But it is possible that there is *precisely one* pair of agents that can do so, and the entire population has to wait until those two meet.

In other words, the existence of a strong potential does not exclude the existence of "bottlenecks", long periods of time where the protocol is waiting for specific interactions to occur.

In the paper [33], we introduce the concept of *rapidly decreasing* strong potentials. This additional property guarantees that "many" agents in a configuration can interact to decrease potential.

Let us make this slightly more precise and consider a configuration C and a terminal configuration C_{term} with $C \to C_{\mathrm{term}}$. Then $\mu(C) := \Phi(C) - \Phi(C_{\mathrm{term}})$ quantifies the amount of work that still needs to be done, i.e. it is the reduction in potential until a terminal configuration is reached.

For Φ to be rapidly decreasing in C we then essentially require that there are states q, p with $C(q), C(p) \in \Omega(\mu(C))$.

This guarantees that the probability of decreasing the potential grows quadratically with $\mu(C)$. Very roughly, this means that we need parallel time $n/\mu(C)^2$ for the potential to decrease once, and the total parallel time is at most $\sum_{i=1}^{\infty} n/i^2 \in \mathcal{O}(n)$.

5. Fault-Tolerant Population Protocols

> A student of history wouldn't look at the failures of the past and conclude that nothing could be done about them. They would instead learn lessons from those failures and correct for them.
>
> *Amaryllis Penndraig, Worth the Candle*

We imagine population protocols as modelling systems with a large number of participants, e.g. millions, billions, or even 10^{23}. Such systems will inevitably encounter individual failures simply due to their size. Additionally, we want to model natural systems, such as chemical reactions, where contamination cannot be excluded.

As such, I consider the question of how the presence of errors affects the power of the population protocol model. I focus on two types of errors:

- **Addition of agents.** In a population protocol (Q, δ, I, O) we start with some initial configuration $C_0 \in \mathbb{N}^I$, and thereafter the number of agents is fixed. Our first type of error has an adversary add agents to the computation, in arbitrary states. It executes *contamination transitions*, which move from a configuration $C \in \mathbb{N}^Q$ to a configuration $C + q$, with $q \in Q$.
- **Removal of agents.** Conversely, one can imagine agents disappearing from the computation. Again, the adversary selects which agents to remove and at which point, executing *snipe transitions*, moving from a configuration $C \in \mathbb{N}^Q$ to a configuration $C - q$, with $q \in [\![C]\!]$.

In both cases, giving the adversary unlimited access to these errors makes the problem trivial. Hence we will define common-sense restrictions, which — we believe — allow for a realistic level of errors to occur, but exclude degenerate cases.

We start in Section 5.1 with contamination transitions, where we shall find a — perhaps unexpected — connection to state complexity. In particular, we are going to show that it is still possible to construct protocols for flock-of-birds predicates.

In Section 5.2, we then consider the other kind of error, snipe transitions. We again look at flock-of-birds predicates, and show that they can still be decided. Finally, we extend this construction to arbitrary threshold or modulo predicates (see Definition 28).

Before we do either, let us briefly make the above notions formally precise.

Definition 40. *Let (Q, δ, I, O) denote a population protocol and $C, C' \in \mathbb{N}^Q$. For any $q \in Q$ we write $C \xrightarrow{+} C + q$ and refer to this as* contamination transition*, and for any $q \in [\![C]\!]$ we write $C \xrightarrow{-} C - q$ and call this* snipe transition*. For $\rightsquigarrow \in \{\xrightarrow{+}, \xrightarrow{-}\}$ and $k \in \mathbb{N}$ we say $C \rightsquigarrow^k C'$ if there exist D, D' with $C \rightsquigarrow^{k-1} D \rightsquigarrow D' \to C'$. For $k = 0$ we instead set $\rightsquigarrow^0 := \to$.*

In particular, we write e.g. $C \xrightarrow{+}{}^k C'$ if there is some sequence of transitions, normal or contamination, from C to C', and that sequence contains precisely k contamination transitions.

5.1. Addition of agents

Imagine a chemical reaction. Two carefully chosen substances x and y are poured into a container and then begin to react in a sophisticated way, forming new substances in the process. Does the initial configuration of this reaction contain only agents in states x and y? Clearly not! Trace amounts of essentially any molecule will be present and can possibly contaminate the reaction.[1]

To model this, we allow an adversary to add agents to the computation at arbitrary points in time. However, it is easy to see that this is equivalent to adding all of the agents at the beginning of the computation.

Lemma 41. *$C \xrightarrow{+}{}^k C'$ iff $C + D \to C'$ for some $D \in \mathbb{N}^Q$ with $|D| = k$.*

Proof. We prove by induction on k. The base $k = 0$ is trivial. In the induction step, assume $C \xrightarrow{+}{}^k C'$. By definition, there are $D, D' \in \mathbb{N}^Q$ with $C \xrightarrow{+}{}^{k-1} D \xrightarrow{+} D' \to C'$. By induction hypothesis, we find some $D'' \in \mathbb{N}^Q$ with $|D''| = k - 1$ and $C + D'' \to D$.

Using the definition of $\xrightarrow{+}$, we get $D' = D + q$ for some $q \in Q$. By Lemma 6 we have

$$C + (D'' + q) = (C + D'') + q \to D + q = D' \to C'$$

□

In the remainder of this section, we will first consider how to define fault-tolerance in the presence of contamination transitions, i.e. what it means for a protocol to be correct. Afterwards, we will see that this is possible to achieve, at least in a special case.

Contamination fault-tolerance. Let $\mathcal{P} = (Q, \delta, I, O)$ denote a population protocol. We would like to execute $\mathcal{P}$ with the addition of contamination transitions, and still have the protocol arrive at a "correct" consensus. But it is not entirely clear what the correct consensus should be. If we consider e.g. the majority predicate $\varphi(x, y) \Leftrightarrow x \leq y$ and add some agent in a state $q \in Q \setminus \{x, y\}$ — does that agent count towards x, or towards y?

In this thesis, we are going to dodge this question by focusing only on flock-of-birds predicates (see Definition 11), which have only one input.

So we fix some predicate $\varphi_k(x) \Leftrightarrow x \geq k$ and set $I := \{x\}$. Consider some initial configuration $C_0 \in \mathbb{N}^I$.

[1] cf. https://en.wikipedia.org/wiki/Disappearing_polymorph

The strongest possible version of fault-tolerance would require any configuration C' with $C_0 \xrightarrow{+}{}^{s} C'$ for some $s \in \mathbb{N}$ to stabilise to $\varphi_k(|C'|)$. Using Lemma 41, we find a $C \in \mathbb{N}^Q$ such that this is equivalent to $C_0 + C$ stabilising to $\varphi_k(|C|)$. This matches the notion of *self-stabilisation* from the literature.

Definition 42. *$\mathcal{P}$ is* self-stabilising *if for any $C_0 \in \mathbb{N}^I$ and $C \in \mathbb{N}^Q$ any run from $C_0 + C$ stabilises to $\varphi_k(|C_0 + C|)$.*

Of course, the C_0 in the above definition is superfluous, since C can be an arbitrary configuration. Self-stabilisation is known to be a strong requirement; too strong, in fact.

Lemma 43. *There is no self-stabilising protocol $\mathcal{P}$.*

Proof. Let C denote some configuration with $|C| = k$. Since $\mathcal{P}$ is self-stabilising, any run from C stabilises to 1, and C reaches some stable 1-consensus C' by Lemma 9.

There exist $q \in Q$, $D \in \mathbb{N}^Q$ with $C' = D + q$.[2] Since $|D| < k$ and $\mathcal{P}$ is self-stabilising, we can reach a stable 0-consensus D' from D, via the same argument as before. By Lemma 6, $C' = D + q \to D' + q$, but $D' + q$ is not a 1-consensus, contradicting the fact that C' is a stable 1-consensus. □

Intuitively, self-stabilisation is such a strong requirement because it places no limits on the amount of contamination that may occur. However, one would imagine that in a practical reaction the amount of contamination is "small" relative to the intended reactants. Our formalisation of this allowance is quite modest: we still allow for arbitrary amounts of contamination C, we merely place a lower bound on the size of C_0.

In other words, instead of requiring for any $C_0 \in \mathbb{N}^I$ any C' with $C_0 \xrightarrow{+}{}^{s} C'$ stabilises to $\varphi_k(|C'|)$, we only require this for $|C_0| \geq |Q|$, where Q are the states of the protocol.

Definition 44. *$\mathcal{P}$ is* almost self-stabilising*, if for $C_0 \in \mathbb{N}^I$ and $C \in \mathbb{N}^Q$ with $|C_0| \geq |Q|$ any run from $C_0 + C$ stabilises to $\varphi_k(|C_0| + |C|)$.*

Admittedly, this notion seems artificial. However, if the protocol is succinct, it is very strong: the contamination C can be arbitrarily large, much larger than C_0 even, and it can be added at arbitrary points in the computation, and in arbitrary states. The only requirement is a minimum number of agents in the initial state, but since $|Q| \in \mathcal{O}(\log k)$ for a succinct protocol, the number of agents needed is very modest compared to the property that we are trying to decide.

Almost self-stabilisation is difficult. Recall the simple protocol $\mathcal{P}_{\text{tok}}$, the first example deciding a flock-of-birds predicate. In particular, let us instantiate the protocol specifically for the predicate $\varphi_{64}(x) \Leftrightarrow x \geq 64$ and consider a run starting with states $1, 1, 64$. A sample execution is shown in Figure 5.1.

[2] One can get into technicalities if $k \leq 2$, but we shall simply ignore those.

Even though only 3 agents are present, much fewer than the threshold of 64, the single agent in state 64 will cause all other agents to move to that state as well, making the protocol accept. The same thing happens in the succinct version of the protocol $\mathcal{P}_{\text{tok2}}$ as well as to many other constructions.

The reason for this is a common property shared by these constructions, which [15] refers to as *1-awareness*.

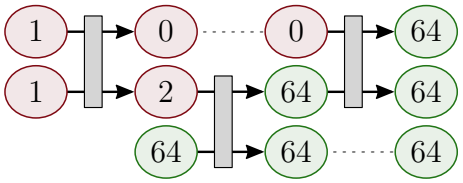


Figure 5.1.: A run of $\mathcal{P}_{\text{tok}}$ with contamination.

Definition 45. *Let $\mathcal{P} = (Q, \delta, I, O)$. $\mathcal{P}$ is* 1-aware*, if there is a state $q_1 \in Q$ such that any run $\sigma = C_0 C_1 \ldots$ stabilises to 1 iff $C_i(q_1) > 0$ for some i.*

Essentially, this property states that there is some state q_1 such that agents in it "know" that the computation will accept. So any accepting run must have an agent in q_1 at some point, and any rejecting run must avoid q_1 altogether. To the best of my knowledge, prior to this thesis all known constructions for flock-of-birds predicates were 1-aware.

Of course, a succinct 1-aware protocol can never be almost self-stabilising, since the adversary can put an agent into q_1 and make arbitrarily small configurations accept.

Almost self-stabilisation is possible. As it turns out, we need not look far to find an almost self-stabilising population protocol. In fact, the protocol $\mathcal{P}_{\text{tiny}}$ of Theorem 22 in Section 4.3 is already self-stabilising!

Recall that in Section 4.3 we construct a "very succinct" protocol, i.e. a protocol $\mathcal{P}_{\text{tiny}}$ deciding φ_k with only $\mathcal{O}(\log \log k)$ states. This construction was not originally intended to also provide fault-tolerance guarantees, but they arise naturally from the requirements placed on very succinct protocols. This again relates to the notion of 1-awareness: a 1-aware protocol cannot be very succinct [15], so our construction, by necessity, is not.

Theorem 46. *$\mathcal{P}_{\text{tiny}}$ is almost self-stabilising.*

Proof (sketch). While we have not described the construction of $\mathcal{P}_{\text{tiny}}$ in detail, the argument for it being almost self-stabilising only relies on the implementation of the leader election. So we instead consider the protocol $\mathcal{P}'_{\text{cm}}$ (see Lemma 27), for which the argument is analogous.

Inspecting the proof of Lemma 27, we find that it also works for an initial configuration $C_0 + C$, where $C \in \mathbb{N}^Q$, as long as $|C_0| \geq 1$.

Since we want to show that $\mathcal{P}_{\text{tiny}}$ is almost self-stabilising, we have $|C_0| \geq |Q| \geq 1$. □

5.2. Removal of agents

Let us now turn towards snipe transitions, i.e. an adversary that is able to remove agents from the computation. For contamination transitions we showed that multiple such transitions in a larger execution can be replaced by a single "contamination event" at the beginning. For snipe transitions we will be able to get a similar result, the difference being that the single "snipe event" happens at the end of the execution.

Lemma 47. *$C \xrightarrow{-}{}^{k} C'$ iff $C \to C' + D$ for some $D \in \mathbb{N}^Q$ with $|D| = k$.*

The proof is analogous to the proof of Lemma 41.

So far, there appears to be a symmetry between the contamination and snipe transitions. However, as we will see shortly, the models are quite different. Similar to before, we start by developing a notion of *robustness*, i.e. fault-tolerance under snipe transitions, that is strong but also achievable.

In Section 5.2.2 we then consider flock-of-birds predicates and show that the constructions we have already seen are not robust. We proceed to give a simple construction that does work, though it is not succinct. To simplify proofs of robustness, we develop a sufficient condition in Section 5.2.3, which can be shown in almost mechanical fashion and is a novel contribution of this thesis. Finally, in Section 5.2.2 we present a construction for a much larger class of predicates, although we fall short of being able to decide all Presburger predicates.

Robustness. Fix some population protocol $\mathcal{P} = (Q, \delta, I, O)$ deciding a predicate φ. Our goal is to define a notion of *robustness*, i.e. say that $\mathcal{P}$ should still give the correct output even with an adversary who can execute snipe transitions. For this, we first have to decide on what the output of the protocol *should* be.

The obvious choice would be to require that when started in initial configuration $C_0 \in \mathbb{N}^I$ the protocol still stabilises to $\varphi(C_0)$, regardless of snipe transitions. But this clearly cannot work, as the adversary would be able to snipe agents before any transitions have taken place, and the protocol will never know that these agents existed. The following lemma formalises this intuition, and makes an even stronger statement.

Lemma 48. *Let $s \in \mathbb{N}$, $C_0, C \in \mathbb{N}^I$ with $|C| \leq s$ and $C \leq C_0$. Then $C_0 \xrightarrow{-}{}^{s} C'$ for some stable $\varphi(C_0 - C)$-consensus C'.*

In other words, we have some initial configuration C_0 and an adversary that executes *exactly* s snipes. Then the adversary can make the protocol output any value that would be correct for an initial configuration that results from removing *at most* s agents from C_0. (Of course, the same holds for an adversary that is allowed at most s snipes.)

Proof (of Lemma 48). We have $C_0 \xrightarrow{-}{}^{|C|} C_0 - C$. Since $C_0 - C \in \mathbb{N}^I$ is still an initial configuration, there is some stable $\varphi(C_0 - C)$-consensus D reachable from C_0 (see Lemma 9). Let $C' \in \mathbb{N}^Q$ denote some configuration that results by removing $s - |C|$ agents from D, i.e. $C' := D - D'$ for some $D' \leq D$ with $|D| = s - |C|$. Overall, we have $C_0 \xrightarrow{-}{}^{s} C'$, so it remains to show that C' is a stable $\varphi(C_0 - C)$-consensus.

Assume the contrary. Then there is some $C'' \in \mathbb{N}^Q$ with $C' \to C''$ which is not a $\varphi(C_0 - C)$-consensus. But this would yield $D = C' + D' \to C'' + D'$ (Lemma 6) since $C'' + D'$ is not a $\varphi(C_0 - C)$-consensus either, this contradicts the choice of D. □

Consider for a moment some fixed initial configuration $C_0 \in \mathbb{N}^I$ and an adversary that is allowed precisely s snipe transitions. This induces a region of initial configurations that are "close" to C_0, in particular the set $S := \{C_0 - C : C \in \mathbb{N}^I, |C| \leq s\}$. The above lemma states that it is always possible to make $\mathcal{P}$ stabilise to any value in $\varphi(S)$. Our

definition of robustness will then result immediately from this: we simply require that the protocol *only* stabilises to values in $\varphi(S)$.

Definition 49. *For a run σ which stabilises to $b \in \{0,1\}$ we set* $\mathrm{out}(\sigma) := b$*, and otherwise set* $\mathrm{out}(\sigma) := \bot$*. For $C \in \mathbb{N}^Q$ we write* $\mathrm{out}(C) := \{\mathrm{out}(\sigma) : \sigma \text{ is a run from } C\}$.

We say that $\mathcal{P}$ robustly decides *φ if for any $s \in \mathbb{N}$, $C_0 \in \mathbb{N}^I$ and $C' \in \mathbb{N}^Q$ with $C_0 \xrightarrow{\leq s} C'$ we have* $\mathrm{out}(C') \subseteq \{\varphi(C_0 - C) : C \leq C_0, |C| \leq s\}$.

As an example, consider the majority predicate $\varphi(x,y) \Leftrightarrow x \leq y$ with input $x := 100$, $y := 120$. Then a robust protocol must output 1 for any number $s \leq 20$ of snipes, since one has to remove at least 21 agents to change the value of the predicate.

Let me remark that in the paper [45], we give a slightly different definition of robustness — the difference is that the above definition requires that the protocol stabilises to *some* output in all cases. So the definition here is slightly stronger. However, it does not matter for the construction considered in the following sections, and I believe that one can even generically convert a protocol that is robust based on the paper's definition to a robust protocol with the above definition.

5.2.1. Constructing robust protocols is difficult

Many protocols in the literature work by centralising information. The most obvious example is leader election, which we find e.g. in the almost-stabilising construction discussed in Section 5.1. In that construction, the leader stores the instruction that is currently executed by the counter machine, and without the leader, the counter machine simulation cannot continue.

In other protocols there is no explicit leader, but the protocol still centralises information. An example of this is $\mathcal{P}_{\mathrm{tok}}$, which we discussed in Section 4.1. In that protocol, each agent starts with one token. Whenever two agents meet, one of them takes both their tokens, and the other leaves empty. Hence the tokens will accumulate in a small number of agents.

This centralisation of information limits the applicability of protocols, since computations in chemical or biological contexts are inherently unreliable. A single leader can easily disappear during the computation.

In [35], the authors allow for a constant number of failures l. However, the protocol is allowed to depend on l, so in their construction they can simply elect $l+1$ leaders, leading to the same lack of meaningful redundancy as before. (Unless one chooses l to be quite large, but then the protocol would have an impractical number of states.)

Conversely, robust protocols have to tolerate an unbounded number of failures. Intuitively, this forces the protocols to keep information decentralised.

We can see how this plays out for $\mathcal{P}_{\mathrm{tok}}$, which is not robust. Fix $k := 3$ and the input $x := 4$. A sample execution with a single snipe transition is shown in Figure 5.2. Initially, all tokens are collected by one agent, then that agent is removed. As a result, the remaining agents are in states $0, 0, 1$; a stable 0-consensus.

But robustness requires that the protocol output either $\varphi(4)$ or $\varphi(3)$, since we have one snipe transition here. As $\varphi(4) = \varphi(3) = 1$, this is violated.

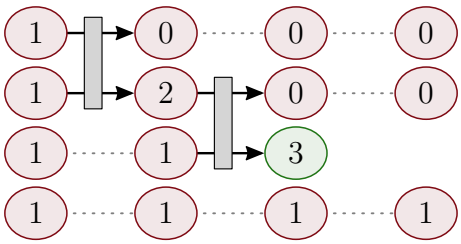


Figure 5.2.: A run of $\mathcal{P}_{\text{tok}}$ with one snipe transition.

As we will see, this requirement that information remains decentralised at all times is quite strong, and protocols must be carefully designed to achieve robustness.

5.2.2. Flock-of-birds Predicates

We consider flock-of-birds predicates $\varphi_k(x) \Leftrightarrow x \geq k$. As just discussed, $\mathcal{P}_{\text{tok}}$ is not robust.

None of the other constructions we have seen so far are robust either. In particular, $\mathcal{P}_{\text{tok2}}$ also has agents collect tokens and fails in the same way, and the very succinct construction $\mathcal{P}_{\text{tiny}}$ performs a leader election. The adversary can simply remove one leader and the computation never starts, regardless of the input.

A robust flock-of-birds protocol. There is a different construction for flock-of-birds protocols, which does not suffer the same issues. We now construct a population protocol $\mathcal{P}_{\text{lvl}} := (Q, \delta, I, O)$ to decide the predicate φ_k. As states we choose $Q := \{1, ..., k\}$, and the initial state is $I := \{1\}$. However, instead of imagining that an agent holds a certain number of tokens, we think of an agent in state i as occupying level i, in a tower with k levels.

Whenever two agents on the same level meet, one of them moves to the next level.

$$i, i \mapsto i+1, i \qquad \text{for } i \in \{1, ..., k-1\} \qquad \langle \textsf{elevate} \rangle$$

This ensures that one agent remains at every level. If one agent reaches level k, they convince the others to accept.

$$k, i \mapsto k, k \qquad \text{for } i \in \{1, ..., k-1\} \qquad \langle \textsf{accept} \rangle$$

Correspondingly, the output is defined as $O(i) := 0$ for $i \in \{1, ..., k-1\}$ and $O(k) := 1$. This construction is sketched in Figure 5.3.

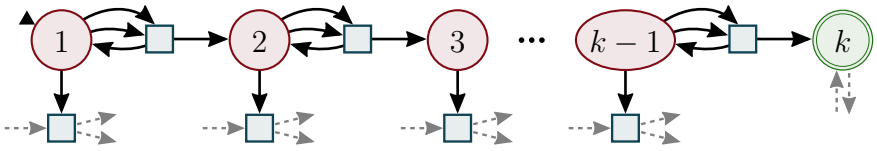


Figure 5.3.: The $\mathcal{P}_{\text{lvl}}$ construction, deciding the predicate φ_k.

Lemma 50. *$\mathcal{P}_{\text{lvl}}$ robustly decides φ_k.*

We will defer the proof to the following section.

This protocol $\mathcal{P}_{\text{lvl}}$ is a nice variation on the simple $\mathcal{P}_{\text{tok}}$ construction. But it is not succinct — it uses $\Omega(k)$ states, a stark contrast to the very succinct protocol we saw for the case of almost self-stabilisation. Are there protocols that are both succinct and robust? We do not know.

5.2.3. Proving Robustness

Proving robustness by directly using the definition above, perhaps in conjunction with Lemma 47 is quite possible. However, in this section we are going to develop a sufficient condition that is easier to check. In particular, the condition can be checked *locally*, in some sense, by considering the possible transitions one by one.

Let $\mathcal{P} = (Q, \delta, I, O)$ denote a population protocol. We will now give the intuition for the property, which we will refer to as *locally robust*. Roughly speaking, in the definition of robustness we have $C_0 \xrightarrow{\text{-}}{}^{s} C'$, for some C_0, C', s. To prove robustness, we have to find a configuration C, such that $C_0 - C$ and C' stabilise to the same output.

Note that, using Lemma 47, $C_0 \xrightarrow{\text{-}}{}^{s} C'$ is equivalent to $C_0 \to C' + D$, for some $D \in \mathbb{N}^Q$ with $|D| \leq s$. Our idea is that we can prove the above *inductively*, by considering the execution $C_0 \to C' + D$ one transition at a time. In particular, we essentially execute one transition in reverse, and try to adjust C', D accordingly. If we can reach the beginning of the execution, i.e. $C_0 = C' + D$, we have found our C by simply taking $C := D$.

Now, the crucial insight is that the only interesting transitions are those where one of the interacting two agents ends up in C', and the other in D. Intuitively, a protocol is locally robust if it is possible to adjust C', D for any such transition.

Definition 51. *We say that $\mathcal{P}$ is* locally robust*, if for any $C' \in \mathbb{N}^Q$, $b \in \text{out}(C')$, $t = (q_1, q_2 \mapsto q_1', q_2') \in \delta$, and $q' := q_1' + q_2' \cap C' \in \{q_1', q_2'\}$ there is $q \in \{q_1, q_2, q_1 + q_2\}$ such that $b \in \text{out}(C' - q' + q)$.*

Recall that for multisets $C, D \in \mathbb{N}^Q$ we write $C \cap D$ for the multiset with $(C \cap D)(q) := \min\{C(q), D(q)\}$ for $q \in Q$. So the notation "$q' := q_1' + q_2' \cap C' \in \{q_1', q_2'\}$" means "$C'$ contains exactly one of q_1', q_2', namely q'".

As mentioned above, the notion of local robustness is a novel (albeit minor) contribution of this thesis and does not appear in our paper [45]. We now prove that this property is sufficient to prove robustness.

Lemma 52. *If $\mathcal{P}$ is locally robust and decides φ, then $\mathcal{P}$ robustly decides φ.*

Proof. To prove robustness, let $s \in \mathbb{N}$, $C_0 \in \mathbb{N}^I$, $C' \in \mathbb{N}^Q$ with $C_0 \xrightarrow{\text{-}}{}^{s} C'$, and let $b \in \text{out}(C')$. We need to show that there is a $C \leq C_0$ with $|C| \leq s$ such that $\varphi(C_0 - C) = b$. Also, from Lemma 47 we get a D with $C_0 \to C' + D$. We choose a D which minimises the length of the execution $C_0 \to C' + D$.

We proceed via induction on the length of this execution from C_0 to $C' + D$. In the base case, the length is 0, i.e. $C_0 = C' + D$, so we can choose $C := D$ (using that $\mathcal{P}$ decides φ).

For the induction step, let $t = (q_1, q_2 \mapsto q_1', q_2') \in \delta$ be the last transition of the execution, so there is some $C'' \in \mathbb{N}^Q$ with $C'' \to_t C' + D$. Due to the induction hypothesis it suffices to show that there is some D' with $|D'| \leq s$ and $b \in \mathrm{out}(C'' - D')$.

Case 1: $q_1' + q_2' \leq D$**.** We choose $D' := D - q_1' - q_2' + q_1 + q_2$, which yields $C'' - D' = C'$.

Case 2: $q_1' + q_2' \leq C'$**.** Here we can set $D' := D$, yielding $C'' - D' \to_t C'$, and thus $\mathrm{out}(C'' - D') \subseteq \mathrm{out}(C')$.

Case 3: $q' := q_1' + q_2' \cap C' \in \{q_1', q_2'\}$**.** From local robustness we get $q \in \{q_1, q_2, q_1 + q_2\}$ where some run from $C' - q' + q$ stabilises to the same value as σ'. We then define $D' := D - p' + p$, where $p := q_1 + q_2 - q$ and $p' := q_1' + q_2' - q$, giving $b \in \mathrm{out}(C' - q' + q)$. □

The protocol $\mathcal{P}_{\mathrm{lvl}}$ is robust. Now we return to the protocol $\mathcal{P}_{\mathrm{lvl}}$ of the previous section and prove its robustness. We start with an important helper lemma that characterises the output of every configuration. I remark that this lemma is, by itself, entirely unconnected to robustness. It merely makes a statement on how certain configurations stabilise.

Lemma 53. *Let $C \in \mathbb{N}^Q$ be a configuration of $\mathcal{P}_{\mathrm{lvl}}$. Then $\mathrm{out}(C) = \{1\}$ if there is some $i \in \{1, ..., k\}$ with $\sum_{j=i}^{k} C(j) > k - i$, and $\mathrm{out}(C) = \{0\}$ otherwise.*

Proof. Let σ be a run from C and $C' \in \inf(\sigma)$. Any transition moves agents to higher states, so only finitely many transitions can be executed on any run, and thus C' is terminal. Hence we may proceed by induction on the length of the shortest execution from C to some terminal configuration.

In the base case, $C = C'$. If $C(k) > 0$, then all agents in C must be in state k, since otherwise $\langle \mathsf{accept} \rangle$ would be enabled. We thus have $\mathrm{out}(C) = 1$, and $\sum_{j=i}^{k} C(j) > k - i$ holds for e.g. $i := k$. Otherwise $C(k) = 0$ so $\mathrm{out}(C) = 0$ (noting that k is the only state with output 1). Let $i \in \{1, ..., k\}$ be arbitrary. If $\sum_{j=i}^{k} C(j) > k - i$, then there are more than $k - i$ agents among states $i, ..., k - 1$. By the pigeonhole principle, one state $j \in \{i, ..., k - 1\}$ has $C(j) > 1$, enabling $\langle \mathsf{elevate} \rangle$ and contradicting the fact that C is terminal.

For the induction step, let $C' \in \mathbb{N}^Q$ with $C \to_t C'$ for some $t \in \delta$. It suffices to show

$$\exists i : \sum_{j=i}^{k} C(j) > k - i \;\Leftrightarrow\; \exists i : \sum_{j=i}^{k} C'(j) > k - i$$

If $t \in \langle \mathsf{accept} \rangle$, both sides must be true (with $i := k$), so assume $t = (q, q \mapsto q, q + 1) \in \langle \mathsf{elevate} \rangle$. Clearly, "⇒" holds. For "⇐", fix i to make the right-hand side true. If $i \neq q + 1$, then $\sum_{j=i}^{k} C(j) = \sum_{j=i}^{k} C'(j)$, and we are done. Finally, in the case $i = q + 1$ we have

$$\sum_{j=q}^{k} C(j) \overset{(1)}{=} \sum_{j=q}^{k} C'(j) \overset{(2)}{\geq} 1 + \sum_{j=i}^{k} C'(j) \overset{(3)}{>} 1 + k - i = k - q$$

where (1) is due to $C \to_t C'$, (2) uses that $C'(q) \geq 1$ after the transition, and (3) is implied by the choice of i. □

At this point we have all the ingredients necessary to prove robustness, and the proof is rather simple. Essentially, we use the characterisation provided by Lemma 53 to check for all possible transitions that local robustness holds.

Lemma 50. *$\mathcal{P}_{\text{lvl}}$ robustly decides φ_k.*

Proof. From Lemma 53 we immediately get that $\mathcal{P}_{\text{lvl}}$ decides φ_k, so by Lemma 52 it suffices to show local robustness.

Let $C' \in \mathbb{N}^Q$, $b \in \text{out}(C')$, $t = (q_1, q_2 \mapsto q_1', q_2') \in \delta$, and $q' := q_1' + q_2' \cap C' \in \{q_1', q_2'\}$. We need to find $q \in \{q_1, q_2, q_1 + q_2\}$ such that $b \in \text{out}(C' - q' + q)$.

If $t \in \langle \mathsf{accept} \rangle$, we can choose $q := q'$ and are done. So assume $t = (p, p \mapsto p, p+1) \in \langle \mathsf{elevate} \rangle$. If $q' = p$ we can again choose $q := q'$, so $q' = p+1$. We can now use Lemma 53 to see that in the case $b = 0$ the choice $q := p$ works, since $\sum_{j=i}^{k}(C' - q' + q)(j) \leq \sum_{j=i}^{k} C'(j)$ for all i.

Finally, the case $b = 1$ remains. Here we use Lemma 53 to fix an i with $\sum_{j=i}^{k} C'(j) > k - i$ and set $q := p + p$. We then get $\sum_{j=i-1}^{k}(C' - q' + q)(j) = \sum_{j=i}^{k} C'(j) + 1 > k - i + 1$, implying $\text{out}(C' - q' + q) = 1$. □

5.2.4. Threshold and Modulo Predicates

Let us now turn our attention towards predicates other than flock-of-birds predicates. As defined in Section 4.4, there are two major subclasses of Presburger predicates, namely the threshold and the modulo predicates. (One obtains the class of Presburger predicates by taking an arbitrary boolean combination of these two subclasses.)

In our paper [45], we show that robustness can be achieved for *all* threshold and modulo predicates, significantly generalising the construction we have seen above.

Theorem 54 ([45])**.** *Let φ denote either a threshold or a modulo predicate. Then there is a population protocol robustly deciding φ.*

Here, I will attempt to highlight the main ideas of the threshold construction. As the modulo construction already appears in the literature [35], albeit in a different context, it will not be covered here.

Let $\varphi(x_1, ..., x_t) \Leftrightarrow \sum_{i=1}^{t} a_i x_i \geq k$ denote a threshold predicate. On a high level, we identify two subclasses of threshold predicates:

- **Upwards-closed threshold predicates.** In this case, we require all coefficients to be positive, i.e. $a_i > 0$ for $i \in \{1, ..., t\}$. This class can be seen as a generalisation of flock-of-birds predicates. In particular, they are *upwards-closed*: if $\varphi(C) = 1$, then $\varphi(C') = 1$ for all $C' \geq C$.
- **Homogeneous threshold predicates.** Here, we have $k = 0$. This is a generalisation of the majority predicate $\varphi(x, y) \Leftrightarrow x \leq y \Leftrightarrow x - y \leq 0$.

We give one construction for each of the above. As we show in the paper, it is then possible to combine the two constructions into a single one for general thresholds.

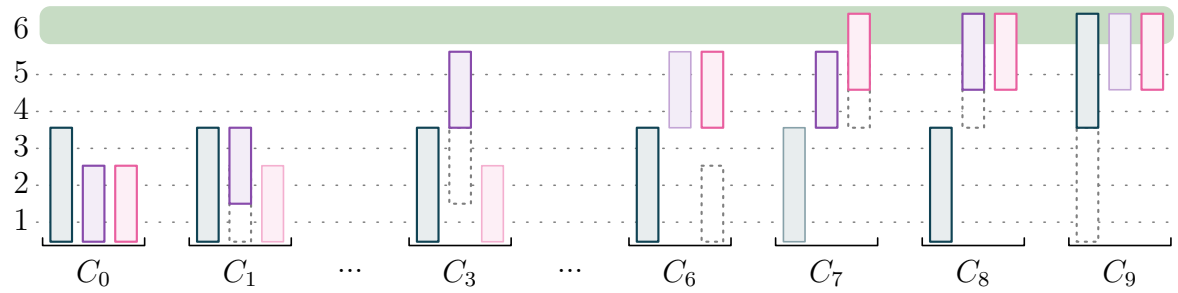


Figure 5.4.: An execution of $\mathcal{P}_{\text{lvl2}}$ for the predicate $\varphi(x, y) \Leftrightarrow 3x + 2y \geq 6$, with input $x = 1, y = 2$, i.e. $C_0 = \{1, 2, 3\} + 2 \cdot \{1, 2\}$.

Upward-closed threshold predicates. We generalise the protocol $\mathcal{P}_{\text{lvl}}$ from above. Again, we imagine a tower with k levels. Instead of having each agent occupy a single level of the tower, however, we have them occupying an *interval* of levels. For an agent starting in state x_i, this interval will have size a_i.

Let us assume that φ is as defined above, and $a_i > 0$ for $i \in \{1, ..., t\}$. Wlog we also assume $a_i \leq k$. We define a population protocol $\mathcal{P}_{\text{lvl2}} := (Q, \delta, I, O)$. The states are $Q := \{\{i, ..., j\} : 1 \leq i \leq j \leq k\}$. (Note that we denote the intervals simply as sets of integers.) The input states are $I := \{\{1, ..., a_i\} : i \in \{1, ..., t\}\}$, meaning that at the beginning, an agent in input a_i occupies levels $1, ..., a_i$.

As before, we have two kinds of transitions. When two agents sharing a level meet, one of them moves upwards by one level, i.e. we shift the entire interval by 1. There is one important detail: we must take care which agent to move, so that we avoid introducing "gaps" in the tower.

$$q, p \mapsto q + 1, p \qquad \text{for } q, p \in Q \text{ with } k \notin p \cup q, q \cap p \neq \emptyset, \min(q) \geq \min(p) \quad \langle \textsf{elevate} \rangle$$

We write $q + 1$ with the obvious meaning, i.e. $q + 1 := \{i + 1 : i \in q\}$. As soon as one agent occupies level k, the protocol accepts.

$$q, p \mapsto q, p + k - \max(p) \qquad \text{for } q \in Q, k \in q \qquad \langle \textsf{accept} \rangle$$

Consequently, we choose the output function $O(q) := 1$ for $q \in Q$ with $k \in q$, and $O(q) := 0$ otherwise. A sample run of this construction is shown in Figure 5.4.

Lemma 55 ([45, Theorem 4.1])**.** *$\mathcal{P}_{\text{lvl2}}$ robustly decides φ.*

The proof is omitted here. The lemma can be proven analogously to Lemma 50, by generalising the property of Lemma 53. The key idea is the same: as long as the total size of the intervals is at least the height of the tower, but the top has not yet been reached, there must be two intervals that are not disjoint, enabling a transition.

Homogeneous threshold predicates. As we mentioned above, homogeneous threshold predicates can be seen as generalisations of the majority predicate. Conveniently, $\mathcal{P}_{\text{maj}}$ — the majority protocol described in Section 3.1 — is already robust, as is its straightforward generalisation to arbitrary coefficients.

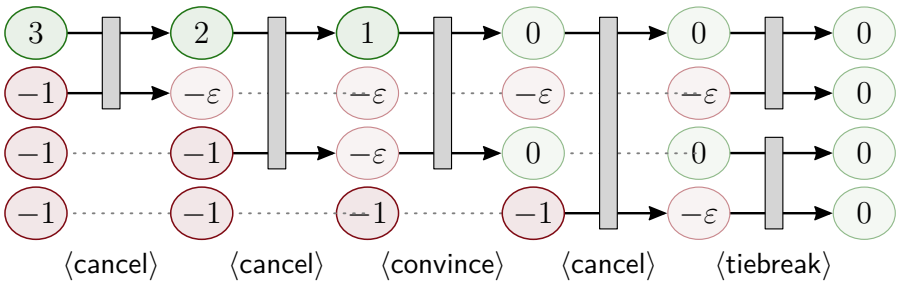


Figure 5.5.: An execution of $\mathcal{P}_{\text{hom}}$ for the predicate $\varphi(x, y) \Leftrightarrow 3x - y \geq 0$, with input $x = 1, y = 3$.

Let φ be as above, with $k = 0$. Wlog we assume $a_1 \leq a_2 \leq ... \leq a_t$. We construct the protocol $\mathcal{P}_{\text{hom}} := (Q, \delta, I, O)$. As states we choose $Q := \{a_1, a_1 + 1, ..., a_t\} \cup \{-\varepsilon\}$. If we recall the states of $\mathcal{P}_{\text{maj}}$ (which are $\{A, B, a, b\}$), deciding the predicate $A \leq B$, this might appear confusing at first glance. However, $\mathcal{P}_{\text{maj}}$ will still result from the following construction as a special case, with the following mapping:

$$A \mapsto -1, \quad B \mapsto 1, \quad a \mapsto -\varepsilon, \quad b \mapsto 0$$

The protocol will follow the same basic principle as $\mathcal{P}_{\text{maj}}$. The opinion of an agent is given by its sign, so we define $O(q) := 1$ for $q \in Q$ if $q \geq 0$, and $O(q) := 0$ otherwise. In particular, $O(-\varepsilon) := 0$.

Agents that have a nonzero value, i.e. agents in states $Q \setminus \{0, -\varepsilon\}$, are "active" and try to cancel their values if they have opposing signs.

$$i, j \mapsto i + j, -\varepsilon \qquad \text{for } i, j \in Q \setminus \{0, -\varepsilon\},\ i > 0 > j \qquad \langle\text{cancel}\rangle$$

I remark that the $-\varepsilon$ is only for consistency with $\mathcal{P}_{\text{maj}}$ — it could be changed to 0.

States $\{0, -\varepsilon\}$ are "passive" — their opinion is changed whenever they meet an active agent with a different opinion.

$$\begin{aligned} i, -\varepsilon &\mapsto i, 0 \qquad && \text{for } i \in \{1, ..., a_t\} \\ i, 0 &\mapsto i, -\varepsilon \qquad && \text{for } i \in \{a_1, ..., -1\} \end{aligned} \qquad \langle\text{convince}\rangle$$

Finally, to resolve ties we allow agents in state 0 to convince agents in state $-\varepsilon$.

$$0, -\varepsilon \mapsto 0, 0 \qquad \langle\text{tiebreak}\rangle$$

For the following proofs, we refer to agents in states $\{0, \varepsilon\}$ as *passive*, and all other agents as *active*.

Lemma 56. *Let* $C \in \mathbb{N}^Q$ *and* $\text{val}(C) := \sum_{q \in Q \cap \mathbb{Z}} q \cdot C(q)$. *Then* $\text{out}(C) = \{\beta\}$, *with*

(a) $\beta = 1$, *if* $\text{val}(C) > 0$,

(b) $\beta = 1$, *if* $\text{val}(C) = 0$ *and* $[\![C']\!] \neq \{-\varepsilon\}$,

(c) $\beta = 0$, *if* $\text{val}(C) = 0$ *and* $[\![C']\!] = \{-\varepsilon\}$, *and*

(d) $\beta = 0$, *if* $\mathrm{val}(C) < 0$.

Proof. Let $\mathcal{C}_a, \mathcal{C}_b, \mathcal{C}_c, \mathcal{C}_d \subseteq \mathbb{N}^Q$ denote the configurations fulfilling conditions (a), (b), (c), and (d), respectively. Let $i \in \{a, b, c, d\}$ with $C \in \mathcal{C}_i$. First, we argue that $\mathcal{C}_i$ is closed under $\to$.

For $i \in \{a, d\}$ this follows from the fact that val is an invariant, i.e. that $\mathrm{val}(D) = \mathrm{val}(D')$ for every $D \to D'$. For $i = b$ we simply observe that any transition produces at least one agent in a state other than $-\varepsilon$; and if $i = c$, then C' is already terminal.

Fix some $C' \in \inf(\sigma)$. From the above, we now have $C' \in \mathcal{C}_i$. We observe that the number of passive agents does not decrease when executing any transition, and strictly increases when executing $\langle \mathsf{cancel} \rangle$. So $\langle \mathsf{cancel} \rangle$ cannot be enabled at C' (otherwise $\inf(\sigma)$ would not be closed under $\to$), and thus all active agents in C' must have the same sign.

If $i \in \{a, d\}$, that sign must match $\mathrm{val}(C')$. So we can find one active agent in a state q with $O(q) = \beta$. Let $p \in \{0, -\varepsilon\}$ with $O(p) = O(q)$. By executing $\langle \mathsf{convince} \rangle$ some number of times with an agent in q, we can move to a configuration C'' where all passive agents are in p. Since $\langle \mathsf{tiebreak} \rangle$ is not enabled at C'', we have that C'' is terminal and thus $C' = C''$ by closure of $\inf(\sigma)$. Hence C' is a terminal β-consensus.

If $i = b$, then there are no active agents in C' (due to $\mathrm{val}(C') = 0$ and $\langle \mathsf{cancel} \rangle$ not being enabled). So we have $C'(0) > 0$, and we can execute $\langle \mathsf{tiebreak} \rangle$ sufficiently often to reach a terminal configuration C'' with all agents in 0. Again we get $C' = C''$, and C' is a terminal 1-consensus.

Finally, for $i = c$ the configuration C' is already a terminal 0-consensus, since all agents are in state $-\varepsilon$. □

Lemma 57. $\mathcal{P}_{\mathrm{hom}}$ *robustly decides* φ.

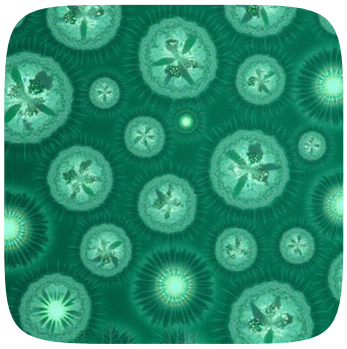
Proof. From Lemma 56 we have that $\mathcal{P}$ decides φ, using that $\varphi(C_0) = 1$ holds iff $\mathrm{val}(C_0) > 0$, for any $C_0 \in \mathbb{N}^I$. Using Lemma 52 it suffices to show that $\mathcal{P}$ is locally robust.

Let $C' \in \mathbb{N}^Q$, $b \in \mathrm{out}(C')$, $t = (q_1, q_2 \mapsto q_1', q_2') \in \delta$, and $q' := q_1' + q_2' \cap C' \in \{q_1', q_2'\}$. We need to find $q \in \{q_1, q_2, q_1 + q_2\}$ such that $b \in \mathrm{out}(C' - q' + q)$.

If $t \in \langle \mathsf{tiebreak} \rangle$ we choose $q := q'$. If $t \in \langle \mathsf{cancel} \rangle$, we have $\mathrm{val}(q_1) > \mathrm{val}(q') > \mathrm{val}(q_2)$. Choosing $q := q_1$ if $b = 1$ and $q := q_2$ otherwise, we get $\mathrm{out}(C' - q' + q) = b$ by Lemma 56(a,d).

So let $t \in \langle \mathsf{convince} \rangle$. If $q' = q_1'$, we can choose $q := q'$ and are done, so assume $q' \in \{0, -\varepsilon\}$. If $\mathrm{val}(C') \neq 0$, we set $q := q' \in \{0, \varepsilon\}$, and get $\mathrm{out}(C' - q + q) = b$ by Lemma 56(a,d). Otherwise, we either have $q' = 0$, implying $b = 1$ (Lemma 56(b)) and choose $q := q_1$, or $q' = -\varepsilon$ and choose depending on b. If $b = 1$, we set $q := q_2 = 0$, and if $b = 0$ we set $q := q_1 < 0$. □

6. Other Models



> Well, this isn't just about modeling, though.
>
> *Serena van der Woodsen,*
> *Gossip Girl*

So far, this thesis has been focused entirely on the population protocol model, as well as restrictions to it. In this section we will finally turn towards extensions and variants of the model. Admittedly, one of the extensions is simply population protocols with a non-constant number of states. In the other two, however, we will encounter new kinds of properties:

- **Communication structure.** So far, all pairs of agents were able to interact with each other, modelling the case of e.g. a "well-stirred" chemical reaction. Now we consider agents that are located on a graph, where only neighbouring agents can interact with each other.
- **Broadcast transitions.** Instead of just two agents interacting, in a broadcast transition a single agent sends a signal that is then received by all other agents, which react in some way.
- **Neighbourhood transitions.** The agents now act fully independently (instead of in pairs). They observe the states of the adjacent agents and react to those states in some manner.
- **Fairness.** In population protocols, the scheduler is assumed to be stochastic (or behave as though it were). Here, we are also going to look at weaker requirements.

In particular, I will present results on three different models. First, we have population protocols with non-constant state space in Section 6.2. Second, we have *broadcast consensus protocols*, which are essentially population protocols augmented with broadcast transitions. These are discussed in Section 6.3. Third, in Section 6.4, I consider distributed graph machines, a family of models where agents are located on a graph and interact via neighbourhood transitions.

To avoid the overhead of defining three different formal models, Section 6.1 starts by defining a general model which then contains the three models as special cases.

6.1. Transition Systems

For the general model, we start with an arbitrary set of *configurations*, and consider a *transition relation* on those configurations. This is simply a transition system. Since we are usually interested in the expressive power of the model, we also define input and output.

Definition 58. *An* (input-output) transition system (TS) *is a tuple* $\mathcal{P} = (\mathcal{C}, \to_1, I, \sim_{\text{inp}}, O)$*, where*

- $\mathcal{C}$ *is a set of* configurations*,*
- $\to_1 \subseteq \mathcal{C} \times \mathcal{C}$ *is a* transition relation*,*
- I *is a finite set of* inputs*,*
- $\sim_{\text{inp}} \subseteq \mathbb{N}^I \times \mathcal{C}$ *is an* input relation*, and*
- $O : \mathcal{C} \to \{0, 1, \bot\}$ *is an* output function*.*

Finally, we require that the connected components of $\to_1$ *are finite.*

I remark that the transition relation $\to_1$ is the "one-step" relation, and we will define $\to$ as its reflexive-transitive closure. The requirement that the connected components of $\to_1$ are finite is not strictly necessary, but holds in all models we consider and simplifies matters. For example, in a population protocol we have that the size of the population does not change, ensuring that any configuration can reach only finitely many configurations.

The output function maps each configuration to an output, or to $\bot$ if there is no output. (In a population protocol, the latter would correspond to configurations that are not a consensus.)

Let $\mathcal{P} = (\mathcal{C}, \to_1, I, \sim_{\text{inp}}, O)$ denote a TS. We now generalise the definitions we had for population protocols.

Definition 59. *Let* $C, C' \in \mathcal{C}$*. We write* $C \to C'$ *if* $C = C'$ *or there is a sequence* $C_0, ..., C_k \in \mathcal{C}$ *with* $C = C_0, C' = C_k$ *and* $C_{i-1} \to_1 C_i$ *for* $i \in \{1, ..., k\}$*. We say that* C *is* initial*, if* $D \sim_{\text{inp}} C$ *for some* $D \in \mathbb{N}^I$*,* reachable *if* $C_0 \to C$ *for some initial* $C_0 \in \mathcal{C}$*, and* terminal *if* $C \to C'$ *implies* $C = C'$*.*

Next are the definitions of runs and of the predicate decided by a TS. These are analogous to the definitions of population protocols.

Definition 60. *Let* $\sigma = C_0 C_1 ...$ *denote an infinite sequence of configurations. We set* $\inf(\sigma) := \bigcap_i \{C_i, C_{i+1}, ...\}$*. We say that* σ *is a* run*, if* $\inf(\sigma)$ *is closed under* $\to$*, and that it* stabilises to b*, for* $b \in \{0, 1\}$*, if* $O(\inf(\sigma)) = \{b\}$*. For* $D \in \mathbb{N}^I$ *we say that* σ *has* input D *if* $D \sim_{\text{inp}} C_0$*.*

Let $\varphi : \mathbb{N}^I \to \{0, 1\}$ *denote a predicate. The protocol* $\mathcal{P}$ decides φ*, if for* $D \in \mathbb{N}^I$ *any run with input* D *stabilises to* $\varphi(D)$*.*

TSs are very general — we are not going to use them directly. Instead, we will define models of protocols that then induce a TS, and use the notions defined for the induced GDP. In particular, if we define a protocol $\mathcal{P}$ and say that it induces some TS $\mathcal{P}'$, then we implicitly extend all definitions of $\mathcal{P}'$ to $\mathcal{P}$ as well. For example, we could then say "$\mathcal{P}$ decides a predicate φ" to mean that $\mathcal{P}'$ decides φ.

6.2. Non-constant State Space

Population protocols can decide the majority predicate $\varphi(x, y) \Leftrightarrow x \leq y$, but not within polylogarithmic parallel time [1, 2]. Indeed, they need almost linear parallel time, and it is known that the same holds for "most" Presburger predicates [9]. Since we usually consider population protocols to operate in a setting with a large number of agents, linear parallel time is too slow for many applications.

Loosening the restriction that the states do not depend on the size of the input is one way to address this issue. In this popular extension of the population protocol model, it is possible to construct protocols that stabilise in logarithmic parallel time, using $\mathcal{O}(\log n)$ states, where n is the number of agents [36].

Let $S : \mathbb{N} \to \mathbb{N}$ denote the space bound, i.e. for an input with n agents we may use up to $S(n)$ states. The idea is that we know an upper bound $N \geq n$ on the number of agents, and can choose a protocol based on $S(N)$. There are multiple variants on how exactly this choice happens:

- **Weakly-uniform population protocols (WUPPs).** These are also referred to as "non-uniform" in the literature. This is the strongest model, where the protocol can depend on $S(N)$ in arbitrary ways, i.e. we have a family of protocols $\mathcal{P}_1, \mathcal{P}_2, \ldots$. Contrary to what the name "non-uniform" might suggest, the family of protocols can usually still be generated by a single Turing machine.
- **Strongly-uniform population protocols (SUPPs).** Here we are given a single, infinitely large protocol with states $Q := \mathbb{N}$. When run on an input with n agents, it will only use states $1, ..., S(n)$. This is usually referred to as "uniform" in the literature.
- **Bounded-uniform population protocols (BUPPs).** This variant is similar to SUPPs, in that we have one infinitely large protocol $\mathcal{P}_\infty$. Instead of requiring $\mathcal{P}_\infty$ to restrict itself to $S(n)$ states at runtime, we require that removing all but the first $S(N)$ states from $\mathcal{P}_\infty$ results in a protocol that works for all inputs with $n \leq N$ agents.

I remark that BUPPs have not (yet) appeared in the literature. However, I argue that they are a more natural model than SUPPs and better suited to explore the expressive power of population protocols with $S \in o(\log n)$ states. In SUPPs, a protocol must be able to restrict the amount of states used by itself. However, in practice space bounds are usually imposed externally, e.g. by the amount of memory available in a device or the number of compounds that can be synthesised. Moreover, as discussed in Section 6.2.2, the SUPP requirement simply excludes all protocols with $\omega(1) \cap o(\log n)$ states, while

being equivalent to BUPP outside that interval. Hence, I believe that the notion of uniformity is best expressed by BUPPs.

In the remainder of this section, we first define the three models above in Section 6.2.1. In Section 6.2.2, I compare the two notions of uniformity in the SUPP and BUPP models. Then Section 6.2.3 discusses the expressive power for $S \in \Omega(\log n)$ and presents our results in this area, and finally in Section 6.2.4 we make a conjecture about the case $S \in o(\log n)$ and relate it to our results from Section 4.3.

6.2.1. Definition

For all of the following definitions we assume $Q \subseteq \mathbb{N}$, i.e. the name of every state is simply an integer. We also implicitly assume that all space bounds $S(n)$ are "well-behaved" in some manner. In particular, $S(n)$ should be space-constructible and monotonic.

WUPPs. We start by defining WUPPs, from which we will then obtain SUPPs and BUPPs as special cases.

Definition 61. *A* weakly-uniform population protocol (WUPP) *is a tuple* (Q, δ, I, O), *where*

- $Q = \mathbb{N}$, $I \subseteq Q$ *is finite,* $O : Q \to \{0, 1\}$,
- $\delta = (\delta_i)_{i \in \mathbb{N}}$ *with* $\delta_i \subseteq Q^2 \times Q^2$ *for* $i \in \mathbb{N}$, *and*
- δ *can be computed by a Turing machine.*

It induces the TS $(\mathbb{N}^Q, \to_1, I, \sim_{\text{inp}}, O')$, *where*

- *for* $C, C' \in \mathbb{N}^Q$ *we have* $C \to_1 C'$ *if* $C \to_t C'$ *for some* $t \in \delta_{|C|}$,
- $\sim_{\text{inp}} := \{(C, C) : C \in \mathbb{N}^I\}$ *is the identity, and*
- $O'(C) := b$ *if* $O([\![C]\!]) = \{b\}$ *for* $b \in \{0, 1\}$ *and* $O'(C) := \perp$ *otherwise.*

So a WUPP has an infinite sequence of transition functions, where δ_n is used for inputs of size n. This is ensured by the definition of $\to_1$.

Next, we define what it means for a WUPP to only use a certain number of states.

Definition 62. *Let* $\mathcal{P} = (Q, \delta, I, O)$ *denote a WUPP and* $S : \mathbb{N} \to \mathbb{N}$. *We set* $[\![\delta_i]\!] := \{q, p, q', p' : (q, p \mapsto q', p') \in \delta_i\}$. *We say that* $\mathcal{P}$ *is* $S(n)$-bounded *if* $\delta_i = \delta_j$ *for all* $i, j \in \mathbb{N}$ *with* $S(i) = S(j)$ *and* $[\![\delta_i]\!] \subseteq \{0, ..., S(i)\}$ *for all* $i \in \mathbb{N}$.

Intuitively, δ_n is the transition function that is used for populations with n agents. The requirement $[\![\delta_i]\!] \subseteq \{0, ..., S(i)\}$ ensures that δ_n can only use $S(n)$ states. If this were the only restriction, the protocol would be able to use different transition functions depending on the precise value of n, i.e. the transitions get a precise estimate of n as "advice". This is undesirable; instead, we want the transitions to only have the value of $S(n)$; we ensure this by requiring $\delta_i = \delta_j$ for $S(i) = S(j)$.

Still, as defined above the transitions δ_i only have to work for n with $S(n) = i$, and not for smaller populations. To ensure that the protocol fulfils this as well, we require *monotonicity*:

Definition 63. *Let $\mathcal{P} = (Q, \delta, I, O)$ denote a WUPP deciding a predicate φ. We say that $\mathcal{P}$ is* monotonic*, if for every $i \in \mathbb{N}$ the WUPP (Q, δ', I, O) with $\delta'_j := \delta_{\max\{i,j\}}$ also decides φ.*

All results in this section apply equally to monotonic and non-monotonic protocols.

BUPPs. In a BUPP, we do not have a family of transition functions $\delta_0, \delta_1, ...$ — instead, there is a single transition function δ, which induces δ_i by deleting all transitions that involve states outside of the state bound. Consequently, the definition of a BUPP already involves a space bound.

Definition 64. *Let $S : \mathbb{N} \to \mathbb{N}$. An $S(n)$-bounded uniform population protocol (BUPP) is a WUPP (Q, δ, I, O), if there is a $\delta_\infty \subseteq Q \times Q$ such that $\delta_i = \{(q, p \mapsto q', p') \in \delta_\infty : q, p, q', p' \leq S(i)\}$ for all $i \in \mathbb{N}$.*

I remark that $\delta_\infty = \bigcup_{i \in \mathbb{N}} \delta_i$.

SUPPs. For strong uniformity, we take a BUPP and further restrict it by requiring that it is impossible to reach states outside of the space bound. More precisely, if we have an initial configuration C with $|C| = n$ agents, and we execute transitions according to the (unrestricted) transition function δ_∞, then we cannot reach any state $q > S(n)$.

Definition 65. *Let $S : \mathbb{N} \to \mathbb{N}$. A $S(n)$-bounded strongly uniform population protocol (SUPP) is an $S(n)$-bounded BUPP (Q, δ, I, O), if in the WUPP $(Q, (\delta_\infty)_{i \in \mathbb{N}}, I, O)$ any reachable configuration C fulfils $[\![C]\!] \subseteq \{0, ..., S(|C|)\}$.*

Note that SUPPs are always monotonic.

6.2.2. The right notion of uniformity

In the previous section, we have defined two different notions of (non-weak) uniformity: BUPPs and SUPPs. In both cases, the family of size-bounded population protocols is induced by a single infinitely-large protocol. The difference is, essentially, that a SUPP must not exceed the state bound even when more states are available, while a BUPP is limited to use only states within the state bound. In this section, I will compare these two models.

In Turing machines, the latter case would be analogous to having markers on the tape denote the amount of space the machine is allowed to use. There, such an assumption can easily be made for the kinds of space bounds that are commonly considered, since the Turing machine is able to, by itself, compute how much space it can use and place markers accordingly.

For population protocols I believe that the same principle can be applied to space bounds $S(n)$ with $S \in \Omega(\log n)$. In particular, I believe that it is possible to give a generic conversion from BUPP to SUPP. One can construct a counter that stores that precise number of agents n in binary using $\mathcal{O}(\log n)$ states, and use it to calculate the number of bits that may be used by each agent to store its state. Each agent stores this number, incurring a $\mathcal{O}(\log S(n))$ overhead, and only executes transitions where the resulting states do not exceed the allowed number of bits.

Conjecture 66. *Any BUPP with $S(n)$ states with $S \in \Omega(\log n)$ can be converted to a SUPP with $\mathcal{O}(S(n) \log S(n))$ states, deciding the same predicate.*

So apart from a $\log n$ factor, the two models coincide for space bounds $S \in \Omega(\log n)$. If $S \in o(\log n)$, however, it is known that WUPPs decide only Presburger predicates, i.e. the same as constant-state population protocols [22]. Essentially, this bound generalises the observation that in a population protocol every reachable state can be reached from an input configuration with roughly $2^{|Q|}$ agents (see Lemma 18). So either finitely many states are reachable, or $S \in \Omega(\log n)$.

Conversely, the BUPP model mostly coincides with SUPPs for $S \in \Omega(\log n)$, and, as we will discuss in Section 6.2.4, exhibits interesting behaviour in the $o(\log n)$ region. In other words, there simply are no SUPPs that are $o(\log n)$-space bounded, but not $\mathcal{O}(1)$-space bounded.

Hence we have to ask whether the requirement imposed by the SUPP model is a reasonable one. I believe not — space restrictions are usually imposed externally by the environment e.g. by the available memory in a chip or the need to synthesise compounds with particular properties. This is precisely the BUPP model.

Therefore, I believe BUPPs to be the "right" notion of uniformity.

6.2.3. Expressive power with $\Omega(\log n)$ states

To discuss the expressive power of population protocols with $\omega(1)$-states, we usually use the classes $\mathsf{SNSPACE}(g(n))$ [46], which are defined as the set of predicates $\varphi : \mathbb{N}^k \to \{0, 1\}$ which can be decided by a $\mathcal{O}(g(n))$-space bounded nondeterministic Turing machine (NTM), with input in unary. (So an input $(x_1, ..., x_k) \in \mathbb{N}^k$ would be represented as the word $1^{x_1}\#...\#1^{x_k}$, with length $\mathcal{O}(x_1 + ... + x_k)$.)

So $\mathsf{SNSPACE}(g(n))$ is similar to $\mathsf{NSPACE}(g(n))$, but only considers predicates. Confusingly, in the literature the class $\mathsf{SNSPACE}(g(n))$ is often referred to as "predicates in $\mathsf{NSPACE}(g(n))$", or even just $\mathsf{NSPACE}(g(n))$. The same holds for the class $\mathsf{SNL} := \mathsf{SNSPACE}(\log n)$ and NL.

Additionally, we use PA for the set of Presburger predicates.

Expressive power. Consider the class of $S(n)$-bounded population protocols, either WUPP, BUPP or SUPP. If $S \in \Omega(n)$, then exactly the predicates in $\mathsf{SNSPACE}(n \log S(n))$ can be decided [22].

However, population protocols with $\Omega(n)$ states are considered impractical. Most constructions in the literature have $\mathcal{O}(\operatorname{polylog} n)$ states instead.

In our paper [30], we investigate precisely this region, and prove that the precise space complexity for population protocols with $\Omega(\log n) \cap o(n^{1-\varepsilon})$ states, for any $\varepsilon > 0$, is $\mathsf{SNSPACE}(\log n \cdot S(n))$. Combined with the aforementioned bound, we get the following.

Theorem 67 ([22, 30])**.** *Let $S \in \Omega(\log n)$ denote a space bound, and $\varepsilon > 0$. Then $S(n)$-bounded SUPPs, BUPPs, or WUPPs, have expressive power*

(a) $\mathsf{SNSPACE}(\log(n) \cdot S(n))$*, for $S \in \mathcal{O}(n^{1-\varepsilon})$, and*

(b) $\mathsf{SNSPACE}(n \cdot \log(S(n)))$*, for $S \in \Omega(n)$.*

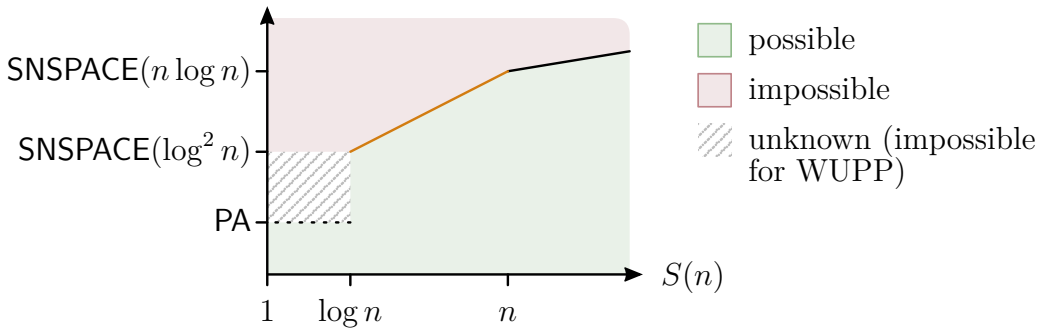


Figure 6.1.: Expressive power of population protocols.

This is illustrated in Figure 6.1, where the x-axis denotes the space bound $S(n)$, and the y-axis denotes expressive power — either in terms of SNSPACE, or using PA for the set of Presburger predicates. In the region $S \in \omega(1) \cap o(\log n)$, expressive power is known only for the SUPP model (since, as discussed above, there are no SUPP protocols in this region), indicated by the dashed line.

Part (b) of the above theorem is due to [22], part (a) is our contribution. For this, we prove both upper and lower bounds on the expressive power.

Upper bound. Essentially, we simulate a population protocol inside a space bounded NTM, representing the number of agents in a state via a binary counter. This leads immediately to the $\log(n) \cdot S(n)$ bound, since there are $S(n)$ states, each needing $\log(n)$ bits. The difficult part is detecting whether the population protocol has reached a stable consensus, which makes use of the Immerman–Szelepcsényi theorem.

Lower bound. The lower bound is proven via construction, and is more involved. On a high level, our result uses the assumption $S \in \Omega(\log n)$ to create a binary counter that will eventually store the number of agents n. This is done by counting agents one by one. While the protocol cannot determine whether that process has finished, it can restart subsequent computations whenever a new agent is found. So eventually the counter will be correct. Once the value of the counter stabilises, the protocol can use it to perform zero-tests, i.e. detecting whether a state is empty.

Using zero-checks, the protocol then simulates a counter machine with $S(n)$ counters. Each of these counters is represented by a single state, and the value of the counter corresponds to the number of agents in the state. By distributing the agents uniformly across the counters, each counter can count up to $n/S(n)$.[1]

6.2.4. Expressive power with $o(\log n)$ states

As shown in Figure 6.1, for space bounds $S \in \omega(1) \cap o(\log n)$ the expressive power is still open. We know it to lie between PA (i.e. the Presburger predicates) and SNL := SNSPACE($\log n$), since that is the power of protocols with $\Theta(1)$ and $\Theta(\log n)$ states,

[1] As a technical aside, this is not quite enough. But we can instead use $kS(n)$ counters, for some constant $k \in \mathbb{N}$, which count up to $n/kS(n)$.

respectively. However, this is a large gap: Presburger predicates contain only simple arithmetic, and cannot decide, for example, whether a number is prime, square, or a power of two. It also cannot multiply by non-constant terms, so e.g. the predicate $\varphi(x, y, z) \Leftrightarrow xy = z$ is not in PA. Conversely, all of these predicates are in SNL.

So in some sense, we know that a "phase transition" occurs in population protocols somewhere in the $S \in \omega(1) \cap o(\log n)$ region, where they go from only Presburger arithmetic to arbitrary space-bounded computations. Where exactly does that phase transition occur? We do not know — but I suspect that the answer is related to our results on the space complexity of flock-of-bird predicates, discussed in Sections 4.2 and 4.3.

A pattern that emerges from multiple results is the importance of the ability of a protocol to perform a deterministic zero-test, i.e. check whether some specific state is empty. This has been considered in the literature [47], showing that adding zero-tests to population protocols allows them to decide any predicate in SNL. We also see their importance in the construction in Section 4.3, which is built around constructing virtual registers that do allow for such a check, in the construction of Section 6.2.4, in the lower bound of the previous section, as well as the constructions we will discuss in Sections 6.3 and 6.4.

In particular, in the lower bound of the previous section, this zero-test was implemented using a simple binary counter with $\mathcal{O}(\log n)$ states. I conjecture that it is possible to instead use the very succinct flock-of-birds construction of Section 4.3 to implement the zero-test. Conversely, I speculate that the lower bound of Section 4.2 can be extended to show that protocols with $o(\log \log n)$ states decide only PA, though this seems less clear.

Conjecture 68. *Let $S \in \mathcal{O}(\log n)$ denote a space bound. Then $S(n)$-bounded BUPPs or WUPPs have expressive power*

(a) PA*, for $S \in o(\log \log n)$, and*

(b) $\mathsf{SNSPACE}(\log(n) \cdot S(n))$*, for $S \in \Omega(\log \log n)$.*

6.3. Broadcast Consensus Protocols

For the next model we return to having a constant state space, and instead allow for *broadcast transitions*. Given a state space Q, a broadcast transition has the form $q \mapsto q', f$, where $q, q' \in Q$ and $f : Q \to Q$. So one agent in state q initiates the transition, moving to state q'. Intuitively, this agent broadcasts the signal f — all other agents react to that signal and change states accordingly. More concretely, another agent in state p would move to state $f(p)$.

Definition 69. *A* broadcast consensus protocol (BCP) *is a tuple (Q, δ, I, O), where*

- *Q is a finite set of* states*,*
- *$\delta \subseteq Q \times Q \times Q^Q$ is a set of* transitions*,*
- *$I \subseteq Q$ are* input states*, and*

- $O : Q \to \{0,1\}$ *is an* output mapping.

It induces the TS $(\mathbb{N}^Q, \to_1, I, \sim_{\text{inp}}, O')$*, where* $\sim_{\text{inp}} := \{(C, C) : C \in \mathbb{N}^I\}$*,* $O'(C) := b$ *if* $O([\![C]\!]) = \{b\}$ *for* $b \in \{0,1\}$ *and* $O'(C) := \bot$ *otherwise, and*

$$C \to_1 f(C - q) + q' \qquad \text{if } (q \mapsto q', f) \in \delta,\ C \in \mathbb{N}^Q, C(q) > 0,$$

Recall that the notation $f(C)$, for a multiset $C \in \mathbb{N}^Q$ and $f : Q \to Q$, denotes the multiset that results from C if every agent in state q moves to state $f(q)$. So $(f(C))(q') = \sum_{q \in f^{-1}(q')} C(q)$. Note also that the definition of $\sim_{\text{inp}}$ and O' are precisely identical to WUPPs (see Definition 61).

The BCP model was first proposed in [16], as an extension of population protocols with broadcasts. They also allow for so-called rendezvous transitions, which are the transitions of the population protocol model. However, as we show in our paper [34], one can simulate rendezvous transitions using only broadcast transitions, so I have not included them in the definition above.

The expressive power of the model is settled in [16] to be precisely SNL. Our paper continues this investigation and analyses the time complexity of the model. To keep this section brief, I will not give a formal definition of time complexity here. (It would be analogous to the definitions in Section 4.4.4.) We consider a scheduler that repeatedly picks agents uniformly at random (recall Definition 58); when picked, an agent initiates a broadcast. The resulting sequence of configurations is almost surely a run, so the protocol will eventually enter a stable consensus. The time complexity then is the expected number of interactions until this occurs.

In a population protocol, we measure parallel time, i.e. we assume that $\Theta(n)$ interactions are executed in one time step (n being the number of agents). This assumption relies on the locality of the interactions, each involving only two agents, meaning that executing linearly many of them in parallel is both feasible and expected in the settings we are considering. However, in BCPs a single transition involves all agents, since every agent potentially changes state according to the broadcast. Hence we will not use parallel time here, and instead refer to the number of interactions directly.

Our paper has two main results:

- BCPs decide all Presburger predicates within $\mathcal{O}(n \log n)$ expected interactions, which is optimal.
- Within a polynomial number of interactions, BCPs decide precisely SZPL, i.e. the predicates that can be decided by a randomised log-space Turing machine, when input is encoded in unary.

The second result is proven by simulating the execution of the Turing machine, going over multiple intermediate models. As such, the construction is quite technical and will not be presented here. Instead, the following section describes the first result, on Presburger predicates.

6.3.1. Presburger predicates

Let us start by considering the majority predicate, $\varphi(x, y) \Leftrightarrow x - y \leq 0$. We construct the protocol $\mathcal{P} = (Q, \delta, I, O)$, where $Q := \{x, y, \mathsf{R}\} \times \{0, 1\}$. Intuitively, each agent stores two things: up to one token corresponding to the initial state, and the value of a global counter. We will ensure that the second value is always the same in all agents.

As initial states we thus choose $I := \{x, y\} \times \{0\}$, and the output is defined as $O((q, c)) := 1$ for $(q, c) \in Q$ with $c \leq 0$, and $O((q, c)) := 0$ otherwise.

There is only one kind of transition, which consumes the token held by an agent and updates the global counter accordingly.

$$\begin{aligned}(x, 0) &\mapsto (\mathsf{R}, 1), \{(q, 0) \mapsto (q, 1) : (q, 0) \in Q\} \\ (y, 1) &\mapsto (\mathsf{R}, 0), \{(q, 1) \mapsto (q, 0) : (q, 1) \in Q\}\end{aligned} \qquad \langle \mathsf{count} \rangle$$

It is easy to see that this protocol is correct.

Lemma 70. *$\mathcal{P}$ decides φ.*

Proof. Let σ denote a run from some initial configuration C_0. As the number of agents in states $\{\mathsf{R}\} \times \{0, 1\}$ is nonincreasing, and decreases strictly with each transition, it suffices to prove that any configuration C reachable from C_0 is a $\varphi(C_0)$-consensus or can execute a transition.

First, we observe the invariant that for any reachable configuration C and states $(q_1, c_1), (q_2, c_2) \in [\![C]\!]$ we have $c_1 = c_2$. So we can define $\mathrm{ctr}(C) := c_1$ as the value of that global counter.

Next, we set $\mathrm{val}(C) := [C]((x, \cdot)) - [C]((x, \cdot)) + \mathrm{ctr}(C)$, and observe that this value is not changed by any transition. In particular, for any C reachable from C_0 we get $\mathrm{val}(C) = \mathrm{val}(C_0)$. Further, we have $\varphi([C_0]) \Leftrightarrow \mathrm{val}(C) \leq 0$.

If in C there is an agent in a state $(x, \mathrm{ctr}(C))$, and $\mathrm{ctr}(C) = 0$, then $\langle \mathsf{count} \rangle$ is enabled. Conversely, the same holds for state $(y, \mathrm{ctr}(C))$ and $\mathrm{ctr}(C) = 1$. So we are left with two cases.

Case 1: $[C]((x, \cdot)) > 0 \wedge \mathrm{ctr}(C) = 1$**.** Using $\mathrm{ctr}(C) = 1$, we can conclude that C is a 0-consensus. Since $0 < \mathrm{val}(C) = \mathrm{val}(C_0)$, we also have $\varphi([C_0]) = 0$.

Case 2: $[C]((y, \cdot)) > 0 \wedge \mathrm{ctr}(C) = 0$**.** This case is analogous to case 1. □

We can generalise this construction to arbitrary Presburger predicates. We proceed in the usual fashion: first, we define constructions for both threshold and modulo predicates, then we give a construction for arbitrary boolean combinations. The boolean construction we use in the paper [34] is the standard product construction (first used by [5]), and the constructions for threshold and modulo are straightforward variants of each other. Therefore, in this thesis I will only give the construction for modulo predicates.

Modulo predicates. Let $\varphi(x_1, ..., x_k) \Leftrightarrow a_1x_1 + ... + a_kx_k \equiv t \pmod{m}$ denote a modulo predicate. We proceed similarly to the example from above and construct the BCP $\mathcal{P} = (Q, \delta, I, O)$ with $Q := (\{x_1, ..., x_k\} \cup \{\mathsf{R}\}) \times \{0, ..., m - 1\}$, $I := \{x_1, ..., x_k\} \times \{0\}$, and for $(q, c) \in Q$ we set $O((q, c)) := 1$ if $c = t$, and $O((q, c)) := 0$ else.

As transitions we have the following.

$$(x_i, c) \mapsto (\mathsf{R}, c'), \{(q, c) \mapsto (q, c') : (q, c) \in Q\} \quad \text{for } i \in \{1, ..., k\},\ c' \equiv c + a_i \pmod{m} \qquad \langle \mathsf{count} \rangle$$

The principle is exactly the same as before: each agent holds up to one token indicating its input, and can consume that token in a transition to update the global counter.

Lemma 71. *$\mathcal{P}$ decides φ.*

The proof is omitted here, since it is analogous to the proof of Lemma 70. In fact, for modulo predicates correctness is easier to prove, since eventually all tokens will be consumed.

Finally, let us briefly consider time complexity. The protocol needs to execute exactly n $\langle \mathsf{count} \rangle$ transitions to stabilise, since every agent initiates precisely one transition. Moreover, at any point in time all agents holding a token can initiate a transition. The time bound then results from a coupon-collector analysis: In the beginning, n agents can initiate a $\langle \mathsf{count} \rangle$ transition. After the first transition, $n - 1$ agents can do so. This continues until stabilisation.

So the expected number of transitions until the first $\langle \mathsf{count} \rangle$ is n/n, the second takes $n/(n-1)$, and so on, leading to

$$\sum_{i=1}^{n} \frac{n}{i} = n \sum_{i=1}^{n} \frac{1}{i} \in \Theta(n \log n)$$

Conversely, $\Omega(n \log n)$ transitions are needed for every agent to initiate one transition (by the same analysis), and for some predicates, the output depends on the value of every input agent (e.g. $\varphi(x, y) \Leftrightarrow x \equiv 0 \pmod 2$). So the protocol achieves an optimal running time in expectation.

6.4. Distributed Graph Machines

So far, we have only seen variants of the population protocol model. In this section, we consider models that deviate in two fundamental aspects:

- The computation no longer operates on the clique (which allows all pairs of agents to communicate), but on an arbitrary connected graph.
- Instead of executing the rendezvous transitions of population protocols, agents receive information about the states of their neighbourhood, and move to a new state based on that information. We refer to these as *neighbourhood transitions*.

This model was introduced in [40]. The authors attempt to unify many different models that have appeared in distributed computing into one common framework, to enable principled comparisons between those models. To this end, their model is not fully specified, but leaves open four different parameters, giving rise to 24 possible model variants. Those parameters are as follows.

- **Detection.** Transitions are executed based on the states of neighbouring agents. For each state, the agent learns how many neighbours are in that state. However, the agent does not learn the exact number — there is a counting bound $\beta \in \mathbb{N}$, and the agent gets $\min\{x, \beta\}$, where x is the true count. The detection parameter $x \in \{\mathtt{d}, \mathtt{D}\}$ determines β: if $x = \mathtt{d}$, $\beta = 1$, and otherwise the protocol can fix any $\beta \in \mathbb{N}$ (independent of the input).
- **Acceptance.** The protocol always accepts by consensus. The parameter $y \in \{\mathtt{a}, \mathtt{A}\}$ controls whether agents are allowed to change their opinions. If $y = \mathtt{a}$, the protocol must be *halting*, meaning that an agent ceases further computation once it assumes an opinion. It must not change state afterwards. Otherwise no restrictions are applied, as for population protocols.
- **Selection.** Agents can either execute their transitions *exclusively*, i.e. only one agent executes a transition at a time (and they do so atomically), *synchronously*, meaning that all agents move at the same time, or *liberal*, which means that some subset of agents moves.
- **Fairness.** The final parameter $z \in \{\mathtt{f}, \mathtt{F}\}$ controls the scheduler. If $z = \mathtt{f}$ the scheduler is only *weakly fair*. Under weak fairness each agent must execute infinitely many transitions, but there are no further guarantees. With $z = \mathtt{F}$ we have *strong fairness*, and every possible finite sequence of transitions must occur infinitely often.

The six possible combinations of the acceptance and selection parameters are known to collapse into two equivalence classes [40]. In particular, it is sufficient to consider the two combinations (1) synchronous selection and weak fairness, and (2) exclusive selection and strong fairness.

Models are thus identified by a three-letter word $xyz \in \{\mathtt{d}, \mathtt{D}\} \times \{\mathtt{a}, \mathtt{A}\} \times \{\mathtt{f}, \mathtt{F}\}$. Since the selection parameter can be inferred from the fairness condition, it is omitted. Uppercase letters indicate fewer restrictions, so `DAF` is the most powerful model, while `daf` is the least powerful.

In our paper [31], we determine the expressive power of each of the eight classes. We also investigate the expressive power along an additional axis. Inspired by biological systems, we consider the assumption that the degree of the graph is bounded, i.e. there is a constant k such that the protocol need only be correct on graphs with maximum degree at most k (and the protocol may depend on the value of k).

Section 6.4.1 will introduce the model formally and present two examples. I then present our results on expressive power in Section 6.4.2, and highlight some of the main technical ideas in Section 6.4.3.

6.4.1. Formal definition

Let us now give a formal definition.

Definition 72. *A* distributed graph machine (DGM) *is a tuple* $\mathcal{P} = (Q, \beta, \delta, I, O, \mathcal{S})$, *where*

- Q *is a finite set of* states,
- $\beta \in \mathbb{N}$ *is a* counting bound,
- $\delta \subseteq Q \times \{0, ..., \beta\}^Q \times Q$ *is a set of* transitions,
- $I \subseteq Q$ *are* input states,
- $O : Q \to \{0, 1, \bot\}$, *is an* output function, *and*
- $\mathcal{S} \in \{\mathsf{one}, \mathsf{all}\}$ *is a* selection rule.

Before defining the TS induced by a DGM, let me first give the underlying intuition and define the necessary notation. We want to execute DGMs on graphs, so a configuration is going to be an undirected, Q-labelled graph $G = (V, E, \lambda)$, i.e. V, E are nodes and edges, respectively, and $\lambda : V \to Q$ are the labels. We write $[G]$ for the multiset of labels, i.e. $[G](q) := |\lambda^{-1}(q)|$ for $q \in Q$.

A single node $v \in V$ executes a transition depending on the states of its neighbours. In particular, it determines $N := [\lambda \restriction \Gamma(v)] \cap B$, where $\restriction$ denotes function restriction and $B(q) := \beta$, for $q \in Q$. In other words, $N(q) = \min\{s, \beta\}$, where s is the number of neighbours of v that are in state q. Then, if there is a transition $(q, N \mapsto q') \in \delta$, v moves to state q'. We write this concisely as follows:

Definition 73. *Let $G = (V, E, \lambda)$, $v \in V$, $\lambda' : V \to Q$. We write $\lambda \vdash_v \lambda'$ if there is $(q, N \mapsto q') \in \delta$ with $\lambda(v) = q$, $N = [\lambda \restriction \Gamma(v)] \cap B$, and $\lambda'(v) = q'$, or if no such transition exists and $\lambda'(v) = q$. We further write $(V, E, \lambda) \vdash_S (V, E, \lambda')$ if $\lambda \vdash_v \lambda'$ for $v \in S$ and $\lambda(v) = \lambda'(v)$ for $v \notin S$.*

Finally, we have $\mathcal{S}$. If $\mathcal{S} = \mathsf{one}$, the DGM makes a transition by selecting precisely one agent and having that agent execute a transition. Conversely, if $\mathcal{S} = \mathsf{all}$, a transition of the DGM involves all agents moving simultaneously.

Definition 74. *The DGM $\mathcal{P} = (Q, \beta, \delta, I, O, \mathcal{S})$ induces the TS $(\mathcal{G}, \to_1, I, \sim_{\text{inp}}, O')$, where $\mathcal{G}$ is the set of all connected, Q-labelled graphs[2], $[C] \sim_{\text{inp}} C$ for $C \in \mathcal{G}$, $O'(C) := b$ for $b \in \{0, 1\}$ if $O(\llbracket [C] \rrbracket) = \{b\}$, and $O'(C) := \bot$ otherwise. Let $C = (V, E, \lambda), C' \in \mathcal{G}$. If $\mathcal{S} = \mathsf{one}$, we have $C \to_1 C'$ if $C \vdash_{\{v\}} C'$ for some $v \in V$, and if $\mathcal{S} = \mathsf{all}$, we have $C \to_1 C'$ if $C \vdash_V C'$.*

Note that our definition of DGMs does not allow for different fairness conditions — it will always use strong fairness (cf. Definition 60). This is not an issue, as we only need weak fairness in combination with synchronous selection, and for synchronous selection there is no difference between strong and weak fairness.

Finally, we define the 8 classes based on the three parameters described above.

Definition 75. *Let $xyz \in \{\mathtt{d}, \mathtt{D}\} \times \{\mathtt{a}, \mathtt{A}\} \times \{\mathtt{f}, \mathtt{F}\}$. We say that $\mathcal{P}$ is a xyz-protocol, if*

- $x = \mathtt{d}$ *implies* $\beta = 1$,
- $y = \mathtt{a}$ *implies* $O(q) = \bot$ *for all* $(q, N \mapsto q') \in \delta$, *and*
- $z = \mathtt{f}$ *implies* $\mathcal{S} = \mathsf{all}$.

We write xyz for the set of predicates that can be decided by a xyz-protocol.

[2]There are some corner cases when graphs have fewer than 3 nodes, which we ignore here.

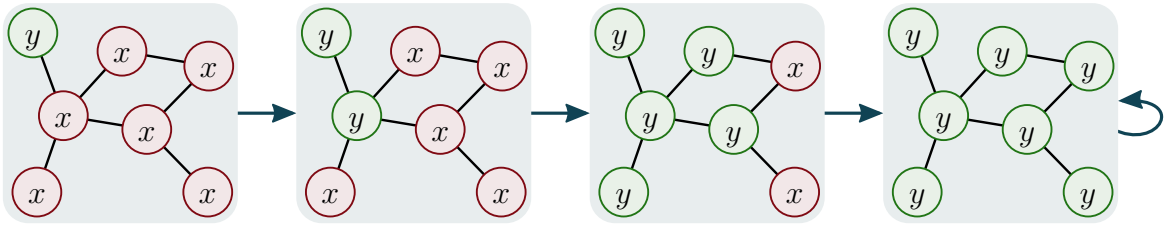


Figure 6.2.: A run of the first example $\mathcal{P}_1$.

First example. Let us start with a simple example. We construct a DAF-protocol $\mathcal{P}_1 = (Q, \beta, \delta, I, O, \mathsf{all})$, with states $\{x, y\}$, counting bound $\beta = 1$, inputs $I = \{x, y\}$ and output $O(x) := 0$, $O(y) := 1$. The only transition is

$$x, N \mapsto y \qquad \text{for } N(y) > 0 \qquad \langle \mathsf{infect} \rangle$$

So an agent in state x can move to y if it has a neighbour in state y. This is an "epidemic" — if there is at least one agent in state y, this spreads through adjacent agents until eventually all agents are in state y. Hence the protocol decides the predicate $\varphi_1(x, y) \Leftrightarrow y > 0$.

Clearly, this is a DAF-protocol, since DAF does not impose any further restrictions. But this is also a dAf-protocol: the counting bound is 1, and we are using all as selection. It is not, however, a daf-protocol, as state x has $O(x) \neq \bot$, but there is a transition initiated by x.

Can we adjust the protocol to be daf? A first try would be to change $O(x)$ to $\bot$. Unfortunately, the protocol would no longer be correct, since an input consisting only of agents in x would no longer be rejected. In fact, it is not possible to give a daf-protocol for φ_1 — as [40] proves, daf-protocols can only decide the two trivial predicates, true and false.

Second example. The second example intends to illustrate the difference between strong and weak fairness. We construct a DAF-protocol $\mathcal{P}_2 = (Q, \beta, \delta, I, O, \mathsf{one})$, with states $\{x, y, T, x', y', y^*\}$. The initial states are $I := \{x, y\}$, the output is $O(T) := 1$, and $O(q) := 0$ for $q \neq T$. The counting bound is $\beta = 2$.

Our goal is to decide the predicate $\varphi_2(x, y) \Leftrightarrow y \geq 2$ — this turns out to be significantly more difficult to decide than φ_1.

State T indicates that we have found two agents in y and so the protocol should accept. We start with a transition that does exactly that.

$$q, N \mapsto T \qquad \text{for } (q + N)(y) \geq 2 \vee N(T) \geq 1 \qquad \langle \mathsf{accept} \rangle$$

This transition alone is not sufficient, as the two agents in y might be far apart in the graph. Therefore, we design transitions that let states y "move around". To this end, we define $f : \mathbb{N}^Q \to Q \cup \{\bot\}$ to yield the unique agent in x', y' or y^*, if it exists, and $\bot$ otherwise. Formally, $f(N) := q$ for $q \in \{x', y', y^*\}$ if $N(q) = N(x') + N(y') + N(y^*) = 1$

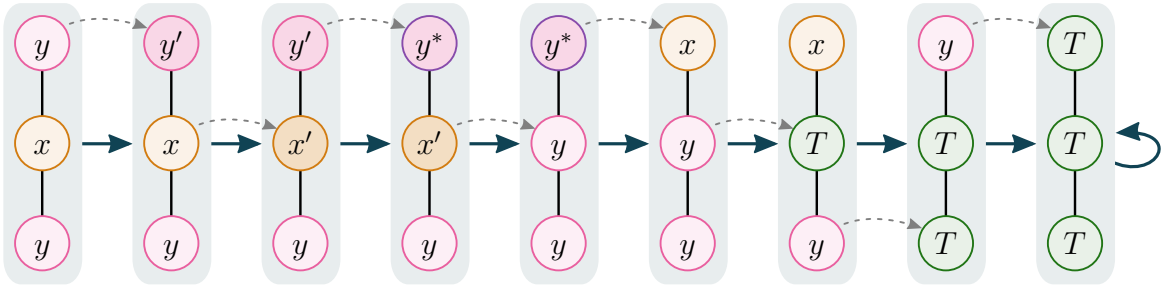


Figure 6.3.: An accepting run of $\mathcal{P}_2$, with exclusive selection. Dashed arrows indicate which agents execute transitions.

and $f(N) := \bot$ otherwise.

$$
\begin{aligned}
y, N &\mapsto y' && \text{for } f(N) = \bot \\
x, N &\mapsto x' && \text{for } f(N) = y' \\
y', N &\mapsto y^* && \text{for } f(N) = x' \\
x', N &\mapsto y && \text{for } f(N) = y^* \\
y^*, N &\mapsto x && \text{for } f(N) = \bot \\
q', N &\mapsto q && \text{for } f(N) = \bot, q \in \{x, y\}
\end{aligned}
\qquad \langle \mathsf{swap} \rangle
$$

These transitions intend to take two agents in states y and x, respectively, and "swap" their states. This is done by a sequence of actions: (1) y moves to y', indicating that it intends to start a swap; (2) x sees y' and moves to x', signalling its intention to participate in the swap; (3) y' moves to y^*, confirming the swap; (4) x' moves to y, y^* moves to x, completing the swap.

Importantly, these steps ensure that there is only one possible candidate for the swap. For example, if multiple x indicate interest in participating in the swap, step (3) will not proceed, and instead y' moves back to y via the last transition. This then eventually causes the x' to revert to x, cancelling the transaction.

Eventually, this protocol causes two agents in state y to move next to each other. Hence $\mathcal{P}_2$ decides φ_2.

However, this relies on strong fairness and exclusive selection. If we instead used synchronous selection (i.e. $\mathcal{S} = \mathsf{all}$), it is possible that no swap is ever completed. For example, consider a triangle with two nodes in state y and one node in state x, as depicted in Figure 6.4. The first transition would move both y nodes to y'. In the second

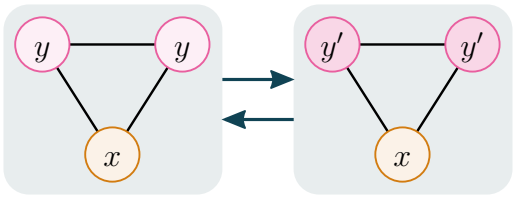


Figure 6.4.: A run of $\mathcal{P}_2$ with synchronous selection.

transition, x does not change state (since it sees two neighbours in state y') and the two nodes in y' move back to y. This repeats infinitely often.

Indeed, the results of our paper [31], discussed in the next section, show that φ_2 requires strong fairness — it can only be decided by DAF- and dAF-protocols.

6.4.2. Expressive power of DGMs

We are interested in the expressive power of the DGM model, so we want to characterise which predicates can be decided by DGMs. While not directly investigated by [40], its analysis does have implications for expressive power. As already mentioned, it shows that the selections can be assumed to be exclusive, and we do so here. Further, it is shown that daf and daF have the same expressive power, so we disregard the latter and seven classes remain. (More precisely, those two classes can only decide the two trivial predicates, true and false.)

They also prove inclusions between the classes, which are illustrated in Figure 6.5. Using this as the starting point of our analysis, we are able to show the following [31]

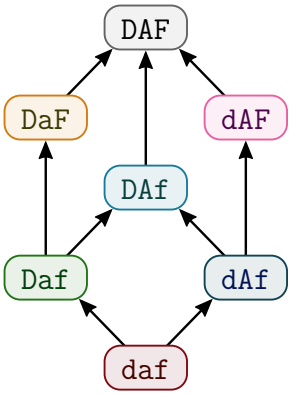


Figure 6.5.: The seven models.

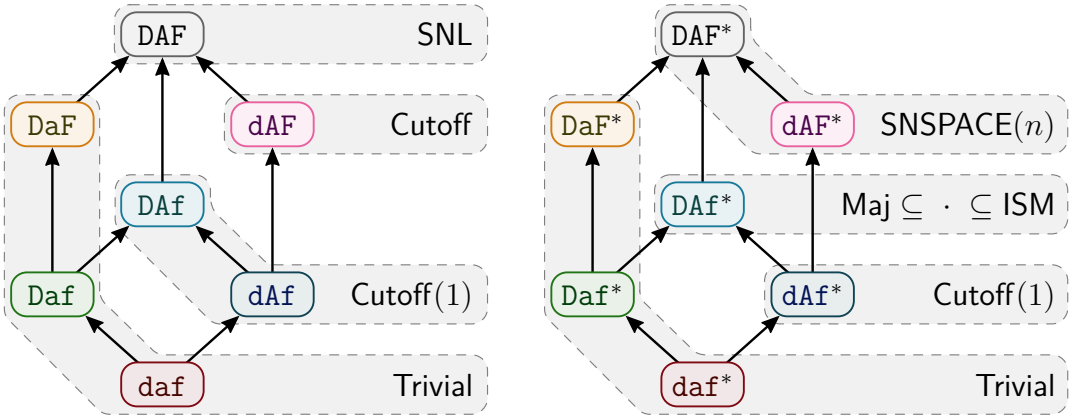
- All classes with halting acceptance, i.e. daf, Daf, DaF, can only decide the two trivial predicates.
- Classes missing one of detection and strong fairness can only count up to a finite bound. More precisely, the classes dAf and DAf decide predicates φ where $\varphi(C)$ depends only on $[\![C]\!]$, for $C \in \mathbb{N}^I$. And for any $\varphi \in$ dAF there is a $B \in \mathbb{N}^I$ with $\varphi(C) = \varphi(C \cap B)$ for all C.
- The strongest model, DAF, decides exactly SNL.

With these results, we characterise the power of all classes precisely, as shown in Figure 6.6(a), using classes of predicates defined as follows:

Definition 76. *Let I denote a finite set of inputs.*

- $\varphi \in \mathsf{Trivial}$ *if there is a* $b \in \{0,1\}$ *with* $\varphi(C) = b$ *for all* $C \in \mathbb{N}^I$.
- $\varphi \in \mathsf{Cutoff}(k)$ *for* $k \in \mathbb{N}$ *if* $\varphi(C) = \varphi(C \cap B)$ *for all* $C \in \mathbb{N}^I$, *with* $B \in \mathbb{N}^I$ *defined as* $B(x) := k$ *for* $x \in I$.
- $\mathsf{Cutoff} := \bigcup_{k=1}^{\infty} \mathsf{Cutoff}(k)$.
- Maj *contains all homogeneous threshold predicates, i.e. predicates* φ *of the form* $\varphi(x_1, ..., x_k) \Leftrightarrow \sum_{i=1}^{k} a_i x_i \geq 0$ *for* $a_1, ..., a_k \in \mathbb{Z}$ *(cf. Section 5.2.4),*
- $\varphi \in \mathsf{ISM}$ *if for all* $C \in \mathbb{N}^I$ *and* $k \in \mathbb{N}$ *we have* $\varphi(C) = \varphi(kC)$.

Bounded-degree graphs. In applications where agents occupy a permanent position in space, it makes sense to consider only communication graphs that have bounded degree. This makes the model stronger, as the adversary must now choose from a limited set of graphs. Formally, we define this as follows.



(a) (b)

Figure 6.6.: Expressive power of DGMs on (a) arbitrary and (b) bounded-degree graphs.

Definition 77. *Let $k \in \mathbb{N}$. A* k-bounded DGM *is defined analogously to a DGM, except that in the induced TS $\mathcal{G}$ is limited to graphs with maximum degree at most k.*

Let $xyz \in \{\mathtt{d},\mathtt{D}\} \times \{\mathtt{a},\mathtt{A}\} \times \{\mathtt{f},\mathtt{F}\}$. We write $xyz(k)$ for the set of predicates decided by k-bounded DGMs that are xyz-protocols, and set $xyz^* := \bigcup_{k=1}^{\infty} xyz(k)$.

The results from [40] still hold in the bounded-degree setting, so $\mathtt{daf}^* = \mathtt{daF}^* = \mathsf{Trivial}$ and we have the same implications, as depicted in Figure 6.5.

In our paper [31] we almost obtain a full characterisation:

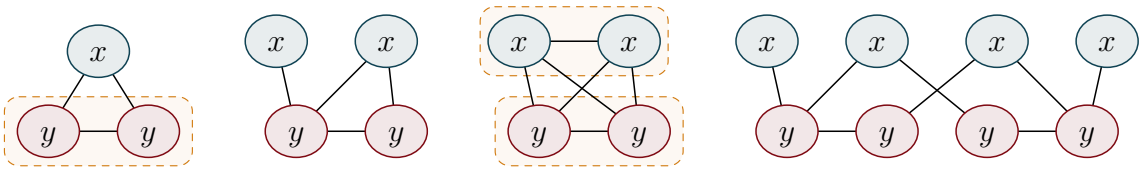
- With halting acceptance, the bounded-degree assumption does not change the expressive power, and we again get $\mathtt{daf}^* = \mathtt{Daf}^* = \mathtt{DaF}^* = \mathsf{Trivial}$.
- We also have $\mathtt{dAf} = \mathtt{dAf}^* = \mathsf{Cutoff}(1)$, same as before.
- The class $\mathtt{dAF}$ has the most dramatic uplift: while $\mathtt{dAF} = \mathsf{Cutoff}$, we have $\mathtt{dAF}^* = \mathsf{SNSPACE}(n)$, i.e. $\mathtt{dAF}^*$ contains all predicates decidable by a linearly space-bounded nondeterministic Turing machine. This is the maximum, and so $\mathtt{DAF}^* = \mathtt{dAF}^*$ as well.
- Finally, the class $\mathtt{DAf}^*$ is interesting. We are unable to provide a full characterisation, but we show $\mathtt{DAf}^* \subseteq \mathsf{ISM}$. This severely limits the predicates that can be decided within the class. In terms of lower bounds, we show that the majority predicate is included in $\mathtt{DAf}^*$, as well as all other homogeneous threshold predicates.

These results are drawn in Figure 6.6(b).

6.4.3. Selected results

In this section we highlight some of the main ideas necessary to prove the results above.

Limitations of synchronous selection. Consider Figure 6.7(a). In the graph G, the two nodes in state y are *symmetric*, i.e. there exists an automorphism mapping one to



(a) G (b) H (c) H' (d) $H^{(2)}$

Figure 6.7.: Graph symmetries. (a) A graph G with one pair of symmetric nodes. (b) A graph H without symmetries. (c) A clique with the same labels as H. (d) H connected to a copy of itself by swapping the ends of one edge.

the other. Under synchronous selection, it is easy to see that those two nodes will always be in the same state.

Since the protocol has to work on an arbitrary graph, we can deliberately choose graphs that exhibit such symmetries. For example, take the graph H in Figure 6.7(b). While it has no symmetries, we can instead choose the clique H' with the same labels, and the protocol must stabilise also on H'. This observation would already be enough to prove $\texttt{dAf} = \mathsf{Cutoff}(1)$.

We can also identify symmetries between two different graphs. For example, we obtain the graph $H^{(2)}$ by taking two copies of H, and then adjusting one edge to "cross-over" and bridge the two copies. In the resulting graph, every node is symmetric to the corresponding node in the other copy of H. Additionally, the states in the neighbourhood of a node are identical to its neighbourhood in the original graph H. So if we take two runs, one of H and one of $H^{(2)}$, then any node of H will traverse exactly the same sequence of states as each of the corresponding two nodes in H.

In particular, this means that both runs stabilise to the same value, so a protocol must either accept both $H, H^{(2)}$, or neither of them. Generalising this argument, we find that any model with synchronous selection can only decide predicates in ISM.

Limitations of halting acceptance. The problem lies in the requirement that a node v has to commit to an output in some finite time t. Consider again the graph $H^{(2)}$, but now with exclusive selection. Let σ denote some finite execution, which we generate by selecting first an arbitrary node, and then the corresponding node in the other copy of H. After these two transitions, the two nodes will be in the same state — so we can create an arbitrarily long execution with the same symmetry as before.

Crucially, we cannot continue this pattern for an infinite number of repetitions to obtain a run, since it contradicts the fairness criterion of Definition 60. But a finite execution shall suffice, since we can induce a node to commit to the wrong output in finite time — with halting acceptance, this is fatal.

Roughly, let t denote the number of steps after which the protocol terminates when run on H. We then generalise the construction of $H^{(2)}$ to obtain a graph $H^{(2t+4)}$, for some $k \in \mathbb{N}$. The nodes of $H^{(2t+4)}$ will still be symmetric to H and traverse the same states in an execution, but the graph now has diameter at least $t + 2$. So we can find some nodes u, v with that distance, and modify the state of v. After t steps, this modification cannot result in a change to the state of u, since information propagates at a maximum

speed of one edge per step. So u will still halt with the same output.

This line of argument can be used to derive a contradiction from any protocol with halting acceptance that decides a non-trivial predicate, showing $\mathtt{Daf} = \mathtt{DaF} = \mathsf{Trivial}$.

More powerful transitions. Many of our constructions rely on simulation results, where we show that certain DGM models can emulate more powerful kinds of transitions. We use three different kinds of transitions, which have all appeared in some form in previous sections of this thesis.

- **Rendezvous transitions.** These transitions, employed by population protocols, allow two agents to perform an atomic, pairwise interaction. The fact that the transition is executed by a pair of agents atomically is particularly useful. For example, the second example protocol of Section 6.4.1 uses the six-part transition $\langle\mathsf{swap}\rangle$ to implement the rendezvous transition $x, y \mapsto y, x$. Our construction to simulate rendezvous transitions is a straightforward generalisation of that example, but it only works in DAF-protocols.
- **Weak broadcasts.** Similar to the broadcast transitions introduced in Section 6.3, an agent broadcasts a signal to which other agents react. However, weak broadcasts provide weaker guarantees. In particular, multiple weak broadcasts may be executed at the same time. If that happens, we only guarantee that every agent receives *some* broadcast. We give a construction to simulate weak broadcasts in all DGM models.
- **Weak zero-tests.** We have discussed the importance of zero-tests in Section 6.2.4, and they also appear in the literature [47]. We model a zero-test as a kind of inverse broadcast: all other agents remain in their states, but the initiator transitions based on the set of states occupied by the others. Similar to weak broadcasts, in our weak zero-tests it is possible that multiple tests occur at the same time, in which case the agents are partitioned and each initiator moves based on the states of one of the partitions. We implement weak zero-tests only in the bounded-degree case and only for models with detection, but we believe that it is possible to generalise the construction somewhat.

Flock-of-bird predicates. Using weak broadcasts, let me sketch the construction that we use to prove $\mathtt{dAF} = \mathsf{Cutoff}$. To simplify, we consider a flock-of-bird-style predicate $\varphi(x, y) \Leftrightarrow x \geq k$. Note that a protocol for φ is already sufficient: using different k and renaming inputs, every predicate in Cutoff can be written as a boolean combination of φ-like predicates.

We use states $Q := \{0, 1, ..., k\}$, with inputs $\{0, 1\}$ (identifying 0 with y and 1 with x) and output $O(q) := 1$ for $q = k$ and $O(q) := 0$ otherwise. The transitions are as follows.

$$i \mapsto i, \{i \mapsto i+1\} \cup \{j \mapsto j : j \neq i\} \qquad \text{for } i \in \{1, ..., k-1\} \qquad \langle\mathsf{elevate}\rangle$$

Syntactically, the transition matches the broadcast transitions of Section 6.3. The idea of this protocol is the same as for $\mathcal{P}_{\text{lvl}}$ of Section 5.2.2 — if we have two agents in state i, one of them moves to state $i + 1$. As long as k nonzero agents are present, state k will

eventually be reached, and then the following transition causes the protocol to accept.

$$k \mapsto k, \{j \mapsto k : j \in Q\} \qquad \langle \mathsf{accept} \rangle$$

Since we have not defined weak broadcasts formally, we also cannot formally prove the correctness of this protocol. Intuitively, correctness relies on the fact that executing the broadcasts partially causes no harm. Due to strong fairness, eventually a correct broadcast will be executed, and the protocol will make progress.

In Section 5.2.2, we used essentially the same protocol to decide flock-of-bird predicates with robust population protocols, i.e. protocols that still work under an adversary that removes agents. Is there a connection between weak broadcasts and robustness? We do not know — but if so, this could open an exciting avenue for designing robust population protocols, by using the weak broadcast model.

Simulating population protocols. Using our construction for rendezvous transitions, it is straightforward to simulate population protocols (on graphs). It is known that population protocols on bounded-degree graphs decide exactly $\mathsf{SNSPACE}(n)$ [19], and $\mathtt{DAF}' = \mathtt{dAF}'$ was shown in [40]. This yields our $\mathtt{DAF}' = \mathtt{dAF}' = \mathsf{SNSPACE}(n)$ result.

Simulating BCPs. We have discussed BCPs in Section 6.3. While the results in this thesis focus on their running time, prior literature shows that BCPs decide exactly the predicates in SNL [16]. To simulate BCPs we cannot use weak broadcasts directly, but we can combine them with rendezvous transitions. More precisely, we use rendezvous transitions to implement a leader election, resulting in a unique leader token. This token is allowed to move around, and broadcasts are restricted to agents holding the leader token. In this manner, we ensure that only one weak broadcast is executed at a time, restoring the guarantees of the BCP model.

Majority without symmetry breaking. As mentioned above, all models with synchronous selection, including $\mathtt{DAf}^*$, can only decide predicates in ISM. Except for $\mathtt{Daf}^*$, we can combine this restriction with others, to show that only predicates in $\mathsf{Cutoff}(1)$ are possible. This notably excludes the majority predicate $\varphi(x, y) \Leftrightarrow x \leq y$.

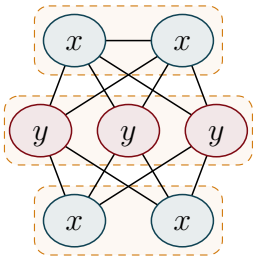


Figure 6.8.: Graph with symmetries.

For $\mathtt{DAf}^*$, however, we are unable to prove further restrictions, so we only know $\mathtt{DAf}^* \subseteq \mathsf{ISM}$ — still, this is a severe limitation. In particular, it relies on the inability of the protocol to perform "symmetry breaking". Consider e.g. the graph in Figure 6.8. As discussed above, the nodes in each of the highlighted groups are symmetric, and the synchronous selection will ensure that they always stay in the same state.

However, the majority predicate is included in ISM, and we prove that it is also included in $\mathtt{DAf}^*$. While the construction is the most technically involved one of this paper, its high-level ideas can be explained succinctly. The protocol has two "layers":

- On the bottom layer, we run a kind of diffusion process. Each agent holds a number of tokens, which are either x or y tokens. Agents with "many" tokens distribute some to those of their neighbours that have "few" tokens. The two types of tokens cancel out, so an agent will never hold both types at the same time. We prove that this process can be implemented with finitely many states and always terminates.

- On the top layer, we use weak zero-tests to wait until the diffusion has terminated and then send a weak broadcast for each agent to double the number of tokens held by the agent. This doubles the size of the gap between x and y. We start the diffusion again, and repeat this process. After enough repetitions, only one type of token remains and we have determined the majority.

This is only a rough sketch — for example, the full construction also has to deal with the case of a tie, which causes the process to run infinitely long. Additionally, we have to simulate weak zero-test and weak broadcasts, and we have to deal with the intermediate states used for the simulation. (In particular, when the weak broadcast is executed it is possible that a zero-test is currently in progress.)

6.4.4. Outlook

Of the 14 classes we set out to analyse, we have found exact characterisations for 13, and given upper and lower bounds for the $\mathtt{DAf}^*$, the remaining one. There, a gap remains: we know that it can decide majority, as well as all other homogeneous threshold predicates, and that it is limited to predicates that are invariant under scalar multiplication. This excludes many, perhaps even all, other Presburger predicates. In particular, non-homogeneous threshold predicates and all modulo predicates.

However, $\mathtt{DAf}^*$ may also include non-Presburger predicates that happen to be invariant under scalar multiplication. One example would be the divisibility predicate $\varphi(x, y) \Leftrightarrow (\exists k : x = ky)$. Can $\mathtt{DAf}^*$ break the Presburger barrier? I do not know.

As we have shown, of the other classes only $\mathtt{DAF} = \mathsf{SNL}$ and $\mathtt{DAF}^* = \mathtt{dAF}^* = \mathsf{SNSPACE}(n)$ achieve significant computational power. None of the others can decide even the majority predicate. While this result may be interpreted to mean that those classes are uninteresting, too limited to perform interesting computations, I would caution that our setting features a strong adversary.

In ongoing work, we investigate a different setting, namely the *emptiness problem*: Given a DGM $\mathcal{P}$, does there exist a labelled graph G on which $\mathcal{P}$ accepts?

We show that this problem is undecidable for all classes except for and *, even given the promise that $\mathcal{P}$ either accepts or rejects any G.

So even the "weaker" classes are able to execute complex computations, as long as the communication structure is suitable. I thus believe that it would also be interesting to investigate different restrictions similar to the bounded-degree restriction.

7. Conclusions

> Look at that date. It's over two years ago.
>
> *Linus Torvalds, 2015*

Weak models of distributed computing, such as population protocols or distributed graph machines, are a rich field of study. To close out this thesis, I would like to again highlight a few connections between the results described above, as well as the most exciting directions for further research.

We have shown that population protocols can decide flock-of-birds predicates very succinctly. It would be interesting to see whether the same can be done for other families of predicates.

There is a connection to the area of fault-tolerant population protocols. The very succinct construction is unusual in that it is not 1-aware, making it almost self-stabilising. So far, the property of almost-self-stabilisation has not been investigated apart from this observation, but I believe that almost-self-stabilisation is both a strong guarantee and possible to achieve, at least in some cases. Many questions arise: What is the expressive power of almost self-stabilising protocols? Can this property be achieved by less complex (and not necessarily succinct) constructions?

For robustness, our results also leave much space to cover still. Is it possible to decide flock-of-birds predicates succinctly with robust protocols? What is the expressive power of robust population protocols? There is an intriguing connection to distributed graph machines — in both cases, the same idea was used to construct flock-of-birds protocols. Of course, this might very well be superficial, but I speculate that weak broadcasts could be helpful for achieving robustness.

In this thesis, I have discussed two notions of robustness, which correspond to adding and removing agents, respectively. Is it possible to construct protocols that are robust against both? In my opinion, this requires a careful definition of which kinds of additions and removals are allowed, since doing this naively quickly makes the problem impossible.

Returning to the very succinct construction for flock-of-birds protocols, I believe that it will be useful in determining the expressive power of non-constant state space population protocols in the "phase transition" region of $\omega(1) \cap o(\log n)$ states. Looking further, I hope that the construction can be used in the construction of fast population protocols with $o(\log n)$ states. While there are known impossibility results in that case, they are not unconditional — it is possible that the very succinct construction evades these bounds as well.

Finally, the model of distributed graph machines is an interesting sandbox to explore how different distributed computing models relate to each other. While our results on expressive power suggest that many of these models are quite weak, this depends a lot on how the graphs are chosen. In ongoing work, we consider the emptiness problem where the graph is part of the input — i.e. deciding whether there exists a labelled graph on which the protocol accepts. We believe that this problem is undecidable for all but one class. I take this to indicate that even the weaker classes exhibit interesting behaviours, in the right circumstances.

8. Publication Summary

8.1. Lower bounds on the state complexity of population protocols

It is known that population protocols decide all Presburger predicates; however, little is known about their *space complexity*. We define the space complexity of a predicate φ as the smallest number of states of any protocol deciding φ. In particular, while some constructions exist and provide upper bounds for each Presburger predicate, no general lower bounds are known.

We consider the family of flock-of-birds predicates, which are the predicates of the form $\varphi_k(x) \Leftrightarrow x \geq k$, for $k \in \mathbb{N}$. We prove two lower bounds, one for the basic model, and one for population protocols extended with leaders.

In the basic model, we show that the state complexity is $\Omega(\log \log k)$, for all k, leaving only an exponential gap to the best known upper bound $\mathcal{O}(\log k)$. This is the main technical contribution of this paper.

If leaders are allowed, we prove a lower bound of $\Omega(\text{ack}^{-1}(k))$ for the state complexity, where ack is some Ackermann-type function. The best known upper bound is $\mathcal{O}(\log \log k)$ in this case, leaving a large gap.

Own contributions. I contributed the first version of the lower bound for the leaderless case, proving an $\Omega(\log \log \log k)$ bound. This version already included many of the technical innovations of the published proof. In particular, it used the technique of first moving to a configuration with many agents in every state, followed by subsequent exploitation of the fact that the behaviour of the population protocol can then be captured by an integer linear program (in some sense). It also included the use of Rackhoff's theorem to argue that adding agents to particular states in a stable consensus does not create instability. The major difference to the final version lay in the proof that the integer linear program has a suitable solution — this part was much simplified by Jérôme Leroux, also improving the overall bound.

8.2. Breaking Through the $\Omega(n)$-Space Barrier: Population Protocols Decide Double-Exponential Thresholds

I consider the *space complexity* of population protocols, i.e. the space complexity of a Presburger predicate φ is the smallest number of states of any protocol deciding φ.

It is possible to represent any such predicate φ in quantifier-free Presburger arithmetic — a natural representation commonly used in the context of population protocols. Encoding coefficients in binary, we define its *size* $|\varphi|$ as the length of the smallest such representation.

For example, the flock-of-birds predicates $\varphi_k(x) \Leftrightarrow x \geq k$ would be represented by "$x \geq k$", i.e. $|\varphi_k| \in \mathcal{O}(\log k)$.

Prior research has shown that the space complexity of φ_k lies in $\Omega(\log|\varphi_k|) \cap \mathcal{O}(|\varphi_k|)$, and that the space complexity of any predicate φ is in $\mathcal{O}(|\varphi_k|)$. From a simple counting argument, one can also see that the space complexity of "most" predicates φ is roughly $\Omega(|\varphi|)$.

Is it possible to break this barrier? More concretely, can we find *any* infinite family of predicates such that every φ in the family has space complexity $o(|\varphi|)$?

I answer this question affirmatively and show that this can already be done for flock-of-birds predicates. In particular, I show that the space complexity of φ_k is $\Theta(\log\log k)$ for infinitely many k.

My construction evades previously known conditional impossibility results, by being the first construction for flock-of-birds predicates that is not 1-aware. Partly due to this, it also exhibits interesting fault-tolerance properties, making it almost self-stabilising.

To give the construction, I develop *population programs*, a small structured programming language, which allows for the construction of small population protocols. Using this model, I show that it is possible to extend a construction due to Lipton, to make it work in a setting where the initial values of the registers are not known. I then give a conversion procedure from population programs to population protocols.

Own contributions. As the sole author, all contributions in this paper are my own.

8.3. Fast and Succinct Population Protocols for Presburger Arithmetic

Already in 2004, when population protocols were introduced, it was shown that they can decide all Presburger predicates φ, by giving a construction to this effect. However, this construction, while simple and fast, produces protocols with an exponential number of states relative to the size of the predicate $|\varphi|$. As usual, we represent predicates via quantifier-free Presburger arithmetic, with coefficients encoded in binary.

For example, the protocol for $\varphi_k(x) \Leftrightarrow x \geq k$ would have $\Omega(k)$ states, while the protocols for $\varphi_{k_1,\dots,k_l}(x) \Leftrightarrow x = k_1 \vee \dots \vee x = k_l$ would have $\Omega(k_1 \cdot \dots \cdot k_l)$ states.

This was improved upon in 2020, showing that for any Presburger predicate φ there is a protocol with $\mathcal{O}(\text{poly}|\varphi|)$ states. However, their construction is slow, needing exponential time to stabilise.

This suggests the existence of a time-space tradeoff, where protocols can either be large and fast, or succinct and slow. However, we prove that this is not the case, by giving a construction which achieves the same size as the 2020 construction, while matching (and even slightly improving) the time of the 2004 construction. Essentially, the resulting protocols are both succinct and optimally fast (for a general construction).

Additionally, as corollary we also obtain a construction with only linear size, i.e. $\mathcal{O}(|\varphi|)$ states for a predicate φ, which is only guaranteed to work for inputs with at least $\mathcal{O}(|\varphi|)$ agents. Considering that the coefficients in φ are of size $2^{\mathcal{O}(|\varphi|)}$, this is a modest assumption.

Own contributions. I suggested the method of ensuring that the entire input is distributed, which relies on allowing higher powers of two in the subprotocols, and thus compute a more efficient representation of the input. I also proposed the restriction of k-way transitions to having at most two distinct input states, which allows the transitions to be executed quickly, and the method of handling output by including a circuit into the model. Additionally, I developed the technique of using rapidly-decreasing potential functions to analyse time complexity. Finally, I also contributed to aspects of the presentation and overall writing, in particular in the time complexity analysis.

8.4. The Black Ninjas and the Sniper: On Robust Population Protocols

We investigate how population protocols can deal with removal of agents. In our fault model, there is an adversary (a "sniper") which is allowed to remove agents from the configuration at any point throughout the computation. However, we restrict the number of removed agents to be at most $k - 1$, where k is the *input tolerance* — the smallest amount of agents one has to remove from the initial configuration to change the output of the predicate.

For example, for the majority predicate $\varphi(x, y) \Leftrightarrow x \leq y$ and input $x = 5$, $y = 7$ we would get $k = 3$.

We refer to protocols that always return the correct output even under this adversary as *robust*. In the paper, we show that the standard construction for flock-of-birds predicates is not robust, and describe an alternative construction that is. The standard construction for majority is already robust — we prove this, and then combine a generalised version of this construction with our construction for flock-of-birds predicates to construct a robust protocol for an arbitrary threshold predicate. Additionally, we give a robust construction for any modulo predicate.

Own contributions. The idea for generalising the tower protocol to arbitrary upwards-closed threshold predicates by considering agents as intervals was suggested by me, as was combining this protocol with the generalised majority protocol by having the positive coefficients first climb the tower. I also contributed to aspects of the presentation and overall writing.

8.5. Brief Announcement: The Expressive Power of Uniform Population Protocols with Logarithmic Space

A popular extension of population protocols allows the state space to depend on the number of agents n. Usually, constructions use $\mathcal{O}(\operatorname{polylog} n)$ states. However, the expressive power of the non-constant state space model was only known for the case of having either $\Omega(n)$ states, or for a restricted class of protocols and $\mathcal{O}(\log n)$ states. This fails to capture most protocols proposed in the literature.

We fill this gap by characterising the expressive power of protocols with $f(n)$ states for $f \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$, for any $\varepsilon > 0$. Let $\mathsf{SNSPACE}(S(n))$ denote the number predicates that can be decided by a nondeterministic $S(n)$-space bounded Turing machine, with input encoded in unary. Then the expressive power is precisely $\mathsf{SNSPACE}(\log(n) f(n))$.

The upper bound relies on a straightforward simulation with a Turing machine, invoking the Immerman–Szelepcsényi theorem to determine whether the protocol has stabilised. For the lower bound we construct a protocol that determines the precise value of n by storing it in a binary counter (using $\log n$ states), and then use this value to implement a zero-test and simulate a counter machine.

Own contributions. The construction hinges on the ability to represent the state of the counter machine within the population protocol. I suggested using population splitting to have $\mathcal{O}(f(n))$ counters each with $\mathcal{O}(n/f(n))$ agents, which is necessary to achieve the desired bound. (The standard method of equipping each agent with one bit per counter needs 2^k states to store k counters, each of which is n-bounded. This is less efficient and would not suffice here.) I have also contributed to the presentation of the construction, by developing notation and simplifying certain aspects.

8.6. Running Time Analysis of Broadcast Consensus Protocols

Many different extensions of the population protocol model have been considered in the literature. Here, we equip agents with *broadcasts*, a kind of transition where one agent sends a signal that is received by all other agents. We refer to this model as *broadcast consensus protocols (BCPs)*.

This model has already been considered previously, and it was shown that its expressive power is precisely SNL, i.e. the number predicates that can be decided by a log-space nondeterministic Turing machine, when input is encoded as unary.

We consider the time complexity of the BCP model. We show that BCPs can decide all Presburger predicates within $\mathcal{O}(n \log n)$ interactions, and prove that this is optimal. The main idea is the use of a global counter, which is shared across all agents and updated via the use of broadcasts.

Our main contribution is a characterisation of polynomial-time BCPs: we show that they correspond exactly to SZPL, meaning the number predicates decidable by randomised Turing machine with logarithmic space and polynomial time. Again, inputs are encoded in unary.

To prove this result, we give a construction that simulates a counter machine equipped with multiplication and division by a constant. Such a counter machine can efficiently simulate a Turing machine. The multiplications and divisions are implemented using broadcasts in combination with ordinary (rendezvous) transitions. The former updates all values stored in the agents across the population at once, while the latter then distributes those values among the agents.

It is possible that the next step of the computation is started before the distribution has finished. However, in this case the broadcast will cause one agent to enter an error state, eventually restarting the computation. As this happens only with negligible probability, the expected running time is not affected.

Own contributions. I developed all of the theoretical results. I also contributed to the presentation and overall writing, particularly in the technical parts.

8.7. Decision Power of Weak Asynchronous Models of Distributed Computing

We consider weak models of distributed computing, where a large number of computationally limited agents interact. Many such models have been proposed in the literature, and these models are often merely slight variations on each other. In 2018, Esparza and Reiter proposed a classification of many of these models. They defined a framework for a distributed computing model, which has four different parameters. Via different combinations of these parameters, 24 models can be obtained.

These models operate on graphs, which provide both communication structure and input. Each node in the graph is one agent, occupying one of finitely many states. Initially, the graph is labelled, and the labels determine the initial states of the agents. The computation proceeds via *neighbourhood transitions*, where an agent receives information about the states present in adjacent agents, and moves to a new state accordingly.

In their analysis, a protocol decides a language of labelled graphs. In this framework, they prove that the 24 classes collapse into 7 different ones.

We build on their analysis by characterising the expressive power of the 7 classes. To do this, we consider the graph as adversarially chosen, so the input is now just a multiset of labels. We give precise characterisations of all 7 classes.

Many e.g. biological systems have agents with limited mobility and a fixed location in space. In such a setting, it makes sense to only consider bounded-degree graphs as inputs. We also analyse expressive power under this restriction and give full characterisations for 6 of the 7 classes, while providing upper and lower bounds for the remaining one.

Own contributions. I proposed the notion of extending the model with different types of transitions, which became an integral part of many constructions in the paper. This includes the formalisation of extensions using reorderings, which provide a sound framework in which we can give precise proofs. I contributed to many individual results, most notably the proof that `dAF` decides all predicates in Cutoff. Finally, I also co-developed the most technically involved result together with Roland Guttenberg, giving a construction for all homogeneous threshold protocols in `DAf` for the degree-bounded case, and contributed the final presentation of this result, using a diffusion process with layered absence detection and weak broadcast transitions.

Bibliography

[1] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2560–2579. SIAM, 2017. doi:10.1137/1.9781611974782.169.

[2] Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2221–2239. SIAM, 2018. doi:10.1137/1.9781611975031.144.

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

[4] Dan Alistarh, Rati Gelashvili, and Joel Rybicki. Fast graphical population protocols. In Quentin Bramas, Vincent Gramoli, and Alessia Milani, editors, *25th International Conference on Principles of Distributed Systems, OPODIS 2021, December 13-15, 2021, Strasbourg, France*, volume 217 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.OPODIS.2021.14, doi:10.4230/LIPICS.OPODIS.2021.14.

[5] 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.

[6] 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.

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

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

[9] Amanda Belleville, David Doty, and David Soloveichik. Hardness of computing and approximating predicates and functions with leaderless population protocols. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors,

44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland, volume 80 of *LIPIcs*, pages 141:1–141:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. URL: https://doi.org/10.4230/LIPIcs.ICALP.2017.141, doi:10.4230/LIPICS.ICALP.2017.141.

[10] Stav Ben-Nun, Tsvi Kopelowitz, Matan Kraus, and Ely Porat. An $O(\log^{3/2} n)$ parallel time population protocol for majority with $O(\log n)$ states. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 191–199. ACM, 2020. doi:10.1145/3382734.3405747.

[11] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. A population protocol for exact majority with o(log5/3 n) stabilization time and theta(log n) states. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.10.

[12] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. Time-space trade-offs in population protocols for the majority problem. *Distributed Comput.*, 34(2):91–111, 2021. doi:10.1007/s00446-020-00385-0.

[13] Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. Population protocols for leader election and exact majority with o(logˆ2 n) states and o(logˆ2 n) convergence time. *CoRR*, abs/1705.01146, 2017. URL: http://arxiv.org/abs/1705.01146, arXiv:1705.01146.

[14] 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.

[15] 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.

[16] Michael Blondin, Javier Esparza, and Stefan Jaax. Expressive power of broadcast consensus protocols. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2019.31, doi:10.4230/LIPICS.CONCUR.2019.31.

[17] Michael Blondin and François Ladouceur. Population protocols with unordered data. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International*

Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany, volume 261 of *LIPIcs*, pages 115:1–115:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.ICALP.2023.115, doi:10.4230/LIPICS.ICALP.2023.115.

[18] Olivier Bournez, Johanne Cohen, and Mikaël Rabie. Homonym population protocols. *Theory Comput. Syst.*, 62(5):1318–1346, 2018. URL: https://doi.org/10.1007/s00224-017-9833-2, doi:10.1007/S00224-017-9833-2.

[19] Olivier Bournez and Jonas Lefèvre. Population protocols on graphs: A hierarchy. In Giancarlo Mauri, Alberto Dennunzio, Luca Manzoni, and Antonio E. Porreca, editors, *Unconventional Computation and Natural Computation - 12th International Conference, UCNC 2013, Milan, Italy, July 1-5, 2013. Proceedings*, volume 7956 of *Lecture Notes in Computer Science*, pages 31–42. Springer, 2013. doi:10.1007/978-3-642-39074-6_5.

[20] Janna Burman, Ho-Lin Chen, Hsueh-Ping Chen, David Doty, Thomas Nowak, Eric E. Severson, and Chuan Xu. Time-optimal self-stabilizing leader election in population protocols. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 33–44. ACM, 2021. doi:10.1145/3465084.3467898.

[21] Shukai Cai, Taisuke Izumi, and Koichi Wada. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theory Comput. Syst.*, 50(3):433–445, 2012. doi:10.1007/s00224-011-9313-z.

[22] Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. Passively mobile communicating machines that use restricted space. In Augustin Chaintreau and Dariusz R. Kowalski, editors, *FOMC'11, The Seventh ACM SIGACT/SIGMOBILE International Workshop on Foundations of Mobile Computing (part of FCRC 2011), San Jose, CA, USA, June 9, 2011, Proceedings*, pages 6–15. ACM, 2011. doi:10.1145/1998476.1998480.

[23] Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, volume 6343 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2010. doi:10.1007/978-3-642-15763-9_15.

[24] Eszter Couillard, Philipp Czerner, Javier Esparza, and Rupak Majumdar. Making sf ip=sf PSPACE practical: Efficient interactive protocols for BDD algorithms. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III*, volume 13966 of *Lecture Notes in Computer Science*, pages 437–458. Springer, 2023. doi:10.1007/978-3-031-37709-9_21.

[25] Philipp Czerner. Breaking through the Ω(n)-space barrier: Population protocols decide double-exponential thresholds. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain*, volume 319 of *LIPIcs*, pages 17:1–17:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.DISC.2024.17, doi:10.4230/LIPICS.DISC.2024.17.

[26] Philipp Czerner and Javier Esparza. Lower bounds on the state complexity of population protocols. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 45–54. ACM, 2021. doi:10.1145/3465084.3467912.

[27] Philipp Czerner, Javier Esparza, and Valentin Krasotin. A resolution-based interactive proof system for UNSAT. In Naoki Kobayashi and James Worrell, editors, *Foundations of Software Science and Computation Structures - 27th International Conference, FoSSaCS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part II*, volume 14575 of *Lecture Notes in Computer Science*, pages 116–136. Springer, 2024. doi:10.1007/978-3-031-57231-9_6.

[28] Philipp Czerner, Javier Esparza, Valentin Krasotin, and Christoph Welzel-Mohr. Computing inductive invariants of regular abstraction frameworks. In Rupak Majumdar and Alexandra Silva, editors, *35th International Conference on Concurrency Theory, CONCUR 2024, September 9-13, 2024, Calgary, Canada*, volume 311 of *LIPIcs*, pages 19:1–19:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2024.19, doi:10.4230/LIPICS.CONCUR.2024.19.

[29] Philipp Czerner, Javier Esparza, and Jérôme Leroux. Lower bounds on the state complexity of population protocols. *Distributed Comput.*, 36(3):209–218, 2023. URL: https://doi.org/10.1007/s00446-023-00450-4, doi:10.1007/S00446-023-00450-4.

[30] Philipp Czerner, Vincent Fischer, and Roland Guttenberg. Brief announcement: The expressive power of uniform population protocols with logarithmic space. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain*, volume 319 of *LIPIcs*, pages 44:1–44:7. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.DISC.2024.44, doi:10.4230/LIPICS.DISC.2024.44.

[31] Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. Decision power of weak asynchronous models of distributed computing. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 115–125. ACM, 2021. doi:10.1145/3465084.3467918.

[32] Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. Fast and succinct population protocols for presburger arithmetic. In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022, March 28-30, 2022, Virtual Conference*, volume 221 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAND.2022.11.

[33] Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. Fast and succinct population protocols for presburger arithmetic. *J. Comput. Syst. Sci.*, 140:103481, 2024. URL: https://doi.org/10.1016/j.jcss.2023.103481, doi:10.1016/J.JCSS.2023.103481.

[34] Philipp Czerner and Stefan Jaax. Running time analysis of broadcast consensus protocols. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 164–183. Springer, 2021. doi:10.1007/978-3-030-71995-1_9.

[35] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When birds die: Making population protocols fault-tolerant. In Phillip B. Gibbons, Tarek F. Abdelzaher, James Aspnes, and Ramesh R. Rao, editors, *Distributed Computing in Sensor Systems, Second IEEE International Conference, DCOSS 2006, San Francisco, CA, USA, June 18-20, 2006, Proceedings*, volume 4026 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2006. doi:10.1007/11776178_4.

[36] David Doty, Mahsa Eftekhari, Leszek Gasieniec, Eric E. Severson, Przemyslaw Uznanski, and Grzegorz Stachowiak. A time and space optimal stable population protocol solving exact majority. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1044–1055. IEEE, 2021. doi:10.1109/FOCS52979.2021.00104.

[37] Robert Elsässer and Tomasz Radzik. Recent results in population protocols for exact majority and leader election. *Bull. EATCS*, 126, 2018.

[38] Yuval Emek and Roger Wattenhofer. Stone age distributed computing. In *PODC*, pages 137–146. ACM, 2013.

[39] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, 54(2):191–215, 2017. URL: https://doi.org/10.1007/s00236-016-0272-3, doi:10.1007/S00236-016-0272-3.

[40] Javier Esparza and Fabian Reiter. A classification of weak asynchronous models of distributed computing. In Igor Konnov and Laura Kovács, editors, *31st International*

Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference), volume 171 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2020.10, doi:10.4230/LIPICS.CONCUR.2020.10.

[41] Leszek Gasieniec and Grzegorz Stachowiak. Fast space optimal leader election in population protocols. In *SODA*, pages 2653–2667. SIAM, 2018.

[42] Rachid Guerraoui and Eric Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 484–495. Springer, 2009. doi:10.1007/978-3-642-02930-1_40.

[43] Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. In Darek Kowalski and Alessandro Panconesi, editors, *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 185–194. ACM, 2012. doi:10.1145/2332432.2332466.

[44] Richard J. Lipton. The reachability problem requires exponential space. Technical report, Yale University, Dept. of CS, 1976. URL: http://www.cs.yale.edu/publications/techreports/tr63.pdf.

[45] Benno Lossin, Philipp Czerner, Javier Esparza, Roland Guttenberg, and Tobias Prehn. *The Black Ninjas and the Sniper: On Robust Population Protocols*, pages 206–233. Springer Nature Switzerland, Cham, 2025. Reproduced with permission from Springer Nature. doi:10.1007/978-3-031-75778-5_10.

[46] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Mediated population protocols. *Theor. Comput. Sci.*, 412(22):2434–2450, 2011. URL: https://doi.org/10.1016/j.tcs.2011.02.003, doi:10.1016/J.TCS.2011.02.003.

[47] Othon Michail and Paul G. Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *J. Parallel Distributed Comput.*, 81-82:1–10, 2015. URL: https://doi.org/10.1016/j.jpdc.2015.02.005, doi:10.1016/J.JPDC.2015.02.005.

[48] Saket Navlakha and Ziv Bar-Joseph. Distributed information processing in biological and computational systems. *Commun. ACM*, 58(1):94–102, 2015. doi:10.1145/2678280.

[49] Loic Pottier. Minimal solutions of linear Diophantine systems: Bounds and algorithms. In *RTA*, volume 488 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 1991.

[50] Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

[51] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992. doi:10.1145/146585.146609.

[52] Steffen van Bergerem, Roland Guttenberg, Sandra Kiefer, Corto Mascle, Nicolas Waldburger, and Chana Weil-Kennedy. Verification of population protocols with unordered data. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 156:1–156:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.ICALP.2024.156, doi:10.4230/LIPICS.ICALP.2024.156.

A. Lower bounds on the state complexity of population protocols

Title	Lower bounds on the state complexity of population protocols
Authors	Philipp Czerner, Javier Esparza, Jérôme Leroux
Venue	Distributed Computing, 2023
Publisher	Springer
DOI	10.1007/s00446-023-00450-4

Distributed Computing (2023) 36:209–218
https://doi.org/10.1007/s00446-023-00450-4

Lower bounds on the state complexity of population protocols

Philipp Czerner[1] · **Javier Esparza**[1] · **Jérôme Leroux**[2]

Received: 29 November 2021 / Accepted: 25 April 2023 / Published online: 15 June 2023

Abstract
Population protocols are a model of computation in which an arbitrary number of indistinguishable finite-state agents interact in pairs. The goal of the agents is to decide by stable consensus whether their initial global configuration satisfies a given property, specified as a predicate on the set of configurations. The state complexity of a predicate is the number of states of a smallest protocol that computes it. Previous work by Blondin et al. has shown that the counting predicates $x \geq \eta$ have state complexity $\mathcal{O}(\log \eta)$ for leaderless protocols and $\mathcal{O}(\log \log \eta)$ for protocols with leaders. We obtain the first non-trivial lower bounds: the state complexity of $x \geq \eta$ is $\Omega(\log \log \eta)$ for leaderless protocols, and the inverse of a non-elementary function for protocols with leaders.

Keywords Population protocols · State complexity

1 Introduction

Population protocols are a model of computation in which an arbitrary number of indistinguishable finite-state agents interact in pairs to decide if their initial global configuration satisfies a given property. Population protocols were introduced in [5, 6] to study the theoretical properties networks of mobile sensors with very limited computational resources, but they are also very strongly related to chemical reaction networks, a discrete model of chemistry in which agents are molecules that change their states due to collisions.

Population protocols decide a property by *stable consensus*. Each state of an agent is assigned a binary output (yes/no). In a correct protocol, all agents eventually reach the set of states whose output is the correct answer to the question "did our initial configuration satisfy the property?", and stay in it forever. An example of a property decidable by population protocols is majority: initially agents are in one of two initial states, say A and B, and the property to be decided is whether the number of agents in A is larger than the number of agents in B. In a seminal paper, Angluin et al. showed that population protocols can decide exactly the properties expressible in Presburger arithmetic, the first-order theory of addition [8].

Assume that at each step a pair of agents is selected uniformly at random and allowed to interact. The *parallel runtime* is defined as the expected number of interactions until a stable consensus is reached (i.e. until the property is decided), divided by the number of agents. Even though the parallel runtime is computed using a discrete model, under reasonable and commonly accepted assumptions the result coincides with the runtime of a continuous-time stochastic model. Many papers have investigated the parallel runtime of population protocols, and several landmark results have been obtained. In [6] it was shown that every Presburger property can be decided in $\mathcal{O}(n \log n)$ parallel time, where n is the number of agents, and [7] showed that population protocols with a fixed number of leaders can compute all Presburger predicates in polylogarithmic parallel time. (Loosely speaking, leaders are auxiliary agents that do not form part of the population of "normal" agents, but can interact with them to help them decide the property.) More recent results have studied protocols for majority and leader election in which the number of states grows with the number of agents, and shown that polylogarithmic time is achievable by protocols without leaders, even for very slow growth functions, see e.g. [2–4, 16, 19].

✉ Philipp Czerner
czerner@in.tum.de

Javier Esparza
esparza@in.tum.de

Jérôme Leroux
jerome.leroux@labri.fr

[1] Department of Computer Science, CIT School, Technical University of Munich, Munich, Germany

[2] LaBRI, University of Bordeaux, CNRS, Bordeaux, France

However, many protocols have a high number of states. For example, a quick estimate shows that the fast protocol for majority implicitly described in [7] has tens of thousands of states. This is an obstacle to implementations of protocols in chemistry, where the number of states corresponds to the number of chemical species participating in the reactions. The number of states is also important because it plays the role of memory in sequential computational models; indeed, the total memory available to a protocol is the logarithm of the number of states multiplied by the number of agents. Despite these facts, the *state complexity* of a Presburger property, defined as the minimal number of states of any protocol deciding the property, has received comparatively little attention[1]. In [11, 12] Blondin et al. have shown that every predicate representable by a boolean combination of threshold and modulo constraints (every Presburger formula can be put into this form), with numbers encoded in binary, can be decided by a protocol with polynomially many states in the length of the formula. In particular, it is not difficult to see that every property of the form $x \geq \eta$, stating that the number of agents is at least η, can be decided by a leaderless protocol with $\mathcal{O}(\log \eta)$ states. A theorem of [11] also proves the existence of an infinite family of thresholds η such that $x \geq \eta$ can be decided by a protocol (with leaders) having $\mathcal{O}(\log \log \eta)$ states. However, to the best of our knowledge there exist no *lower* bounds on the state complexity, i.e. bounds showing that a protocol for $x \geq \eta$ needs $\Omega(f(\eta))$ states for some function f. This question, which was left open in [12], is notoriously hard due to its relation to fundamental questions in the theory of Vector Addition Systems.

In this paper we first show that every protocol, with or without leaders, needs a number of states that, roughly speaking, grows like the inverse Ackermann function, and then prove our main result: every leaderless protocol for $x \geq \eta$ needs $\Omega(\log \log \eta)$ states. The proof of the first bound relies on results on the maximal length of controlled antichains of $\mathbb{N}^d$, a topic in combinatorics with a long tradition in the study of Vector Addition Systems and other models, see e.g. [1, 9, 18, 23, 26]. The double logarithmic bound follows from Pottier's small basis theorem, a useful result of the theory of Diophantine equations [24].

The paper is organised as follows. Section 2 introduces population protocols, the state complexity function, and its inverse, the busy beaver function, which assigns to a number of states n the largest η such that a protocol with n states decides $x \geq \eta$. Instead of lower bounds on state complexity, we present upper bounds on the busy beaver function for convenience. Section 3 presents some results on the mathematical structure of stable sets of configurations that are used throughout the paper. Section 4 shows an Ackermannian upper bound on the busy beaver function, valid for protocols with or without leaders, and explains why this very large bound might be optimal. Section 5 gives a triple exponential upper bound on the busy beaver function for leaderless protocols.

[1] Notice that the time-space trade-off results of [2–4, 16, 19] refer to a more general model in which the number of states of a protocol grows with the number n of agents; in other words, a property is decided by a *family* of protocols, one for each value of n. Trade-off results bound the growth rate needed to compute a predicate within a given time. We study the minimal number of states of a *single* protocol that decides the property for *all* n.

2 Population protocols and state complexity

2.1 Mathematical preliminaries

For sets A, B we write A^B to denote the set of functions $f : B \to A$. If B is finite we call the elements of $\mathbb{N}^B$ *multisets* over B. We sometimes write multisets using set-like notation, e.g. $\{\!\{a, b, b\}\!\}$ and $\{\!\{a, 2 \cdot b\}\!\}$ denote the multiset m such that $m(a) = 1, m(b) = 2$ and $m(c) = 0$ for every $c \in B \setminus \{a, b\}$. Given a multiset $m \in \mathbb{N}^B$ and $B' \subseteq B$, we define $m(B') := \sum_{b \in B'} m(b)$. The *size* of m is $|m| := m(B)$; in other words, the total number of elements of m. The *support* of m is the set $[\![m]\!] = \{b \in B \mid mb) > 0\}$. Abusing language we identify an element $b \in B$ with the one-element multiset containing it, i.e. with the multiset $m \in \mathbb{N}^B$ given by $m(b) = 1$ and $m(b') = 0$ for $b' \neq b$.

We call the elements of $\mathbb{Z}^B$ *vectors* over B of dimension $|B|$. Observe that every multiset is also a vector. Arithmetic operations on vectors in $\mathbb{Z}^B$ are defined as usual, extending the vectors with zeroes if necessary. For example, if $B' \subseteq B$, $u \in \mathbb{Z}^B$, and $v \in \mathbb{Z}^{B'}$, then $u + v \in \mathbb{Z}^B$ is defined by $(u + v)(b) = u(b) + v(b)$, where $v(b) = 0$ for every $b \in B \setminus B'$. For $u, v \in \mathbb{Z}^B$ we write $u \leq v$ if $u_i \leq v_i$ for all $i \in B$, and $u \lneq v$ if $u \leq v$ and $u \neq v$. Given a vector $v \in \mathbb{Z}^k$, we define $\|v\|_1 = \sum_{i=1}^k |v_i|$ and $\|v\|_\infty = \max_{i=1}^k |v_i|$.

2.2 Population protocols

We recall the population protocol model of [6], with explicit mention of leader agents. A *population protocol* is a tuple $\mathcal{P} = (Q, T, L, X, I, O)$ where

- Q is a finite set of *states*;
- $T \subseteq Q_2 \times Q_2$ is a set of *transitions*, where Q_2 denotes the set of multisets over q of size 2;
- $L \in \mathbb{N}^Q$ is the *leader multiset*;
- X is a finite set of *input variables*;
- $I : X \to Q$ is the *input mapping*; and
- $O : Q \to \{0, 1\}$ is the *output mapping*.

We write $p, q \mapsto p', q'$ to denote that the pair $(\{\!| p, q |\!\}, \{\!| p', q' |\!\})$ is a transition. We assume that for every multiset $\{\!| p, q |\!\}$ there is at least one transition of the form $p, q \mapsto p', q'$.

Inputs and configurations An *input* to $\mathcal{P}$ is a multiset $m \in \mathbb{N}^X$ such that $|m| \geq 2$. 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* for input m is defined as

$$IC(m) := L + \sum_{x \in X} m(x) \cdot I(x).$$

When $\mathcal{P}$ has a unique input x, i.e. $X = \{x\}$, we abuse language and write $IC(i)$ instead of $IC(i \cdot x)$ to denote the initial configuration for input $i \in \mathbb{N}$.

The *output* $O(C)$ of a configuration C is b if $C(q) \geq 1$ implies $O(q) = b$ for all $q \in Q$, and undefined otherwise. So a population has output b if all agents have output b.

Executions A transition $t = p, q \mapsto p', q'$ is *enabled* at a configuration C if $C \geq p + q$, and *disabled* otherwise. As $|C| \geq 2$ by the definition of configuration, every configuration enables at least one transition. If t is enabled at C, then it can be *fired* leading to configuration $C' := C - p - q + p' + q'$, which we denote $C \xrightarrow{t} C'$. We write $C \to C'$ if $C \xrightarrow{t} C'$ for some $t \in T$. Given a sequence $\sigma = t_1 t_2 \dots t_n$ of transitions, we write $C \xrightarrow{\sigma} C'$ if there exist configurations $C_1, C_2, \dots, C_n$ such that $C \xrightarrow{t_1} C_1 \xrightarrow{t_2} C_2 \cdots C_n \xrightarrow{t_n} C'$, and $C \xrightarrow{*} C'$ if $C \xrightarrow{\sigma} C'$ for some sequence $\sigma \in T^*$. For every set of transitions $T' \subseteq T$, we write $C \xrightarrow{T'} C'$ if $C \xrightarrow{t} C'$ for some $t \in T'$; we write $C \xrightarrow{T'*} C'$, and say that C' is *reachable* from C, if $C \xrightarrow{\sigma} C'$ for some sequence $\sigma \in T'^*$. Given a set $\mathcal{C}$ of configurations, $C \xrightarrow{*} \mathcal{C}$ denotes that $C \xrightarrow{*} C'$ for some $C' \in \mathcal{C}$.

An *execution* is a sequence of configurations $\sigma = C_0 C_1 \dots$ such that $C_i \to C_{i+1}$ for every $i \in \mathbb{N}$. The *output* $O(\sigma)$ of σ is b if there exist $i \in \mathbb{N}$ such that $O(C_i) = O(C_{i+1}) = \dots = b$, otherwise $O(\sigma)$ is undefined.

Executions have the *monotonicity property*: If $C_0 C_1 C_2 \dots$ is an execution, then for every configuration D the sequence $(C_0 + C)(C_1 + C)(C_2 + C) \dots$ is an execution too. We often say that a statement holds "by monotonicity", meaning that it is a consequence of the monotonicity property.

Computations An execution $\sigma = C_0 C_1 \dots$ is *fair* if for every configuration C the following holds: if C is reachable from C_i for infinitely many $i \in \mathbb{N}$, then $C_j = C$ for infinitely many $j \in \mathbb{N}$. In other words, fairness ensures that an execution cannot avoid a reachable configuration forever. We say that a population protocol *computes* a predicate $\varphi \colon \mathbb{N}^X \to \{0, 1\}$ (or *decides* the property represented by the predicate) if for every $v \in \mathbb{N}^X$ every fair execution σ starting from $IC(v)$ satisfies $O(\sigma) = \varphi(v)$. Two protocols are *equivalent* if they compute the same predicate. It is known that population protocols compute precisely the Presburger-definable predicates [8].

Example 2.1 Let $\mathcal{P}_k = (Q, T, 0, \{x\}, I, O)$ be the protocol where $Q := \{0, 1, 2, 3, ..., 2^k\}$, $I(x) := 1$, $O(a) = 1$ iff $a = 2^k$, and the set T of transitions contains $a, b \mapsto 0, a + b$ if $a + b < 2^k$, and $a, b \mapsto 2^k, 2^k$ if $a + b \geq 2^k$ for every $a, b \in Q$. It is readily seen that $\mathcal{P}_k$ computes $x \geq 2^k$ with $2^k + 1$ states. 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^k. Once an agent reaches 2^k, all agents eventually get converted to 2^k.

Now, consider the protocol $\mathcal{P}'_k = (Q', T', 0, \{x\}, I', O')$, where $Q' := \{0, 2^0, 2^1, ..., 2^k\}$, $I'(x) := 2^0$, $O'(a) = 1$ iff $a = 2^k$, and T' contains $2^i, 2^i \mapsto 0, 2^{i+1}$ for each $0 \leq i < k$, and $a, 2^k \mapsto 2^k, 2^k$ for each $a \in Q'$. It is easy to see that $\mathcal{P}'_k$ also computes $x \geq 2^k$, but more succinctly; while $\mathcal{P}_k$ has $2^k + 1$ states, $\mathcal{P}'_k$ has only $k + 1$ states.

Leaderless protocols A protocol $\mathcal{P} = (Q, T, L, X, I, O)$ has a multiset L of leaders. If $L = 0$, then the protocol is *leaderless*. Protocols with leaders and leaderless protocols compute the same predicates [8]. For $L = 0$ we have

$$IC(\lambda v + \lambda' v') = \lambda IC(v) + \lambda' IC(v')$$

for all inputs $v, v' \in \mathbb{N}^X$ and $\lambda, \lambda' \in \mathbb{N}$. In other words, any linear combination of initial configurations with natural coefficients is also an initial configuration.

2.3 State complexity of population protocols

Informally, the state complexity of a predicate is the minimal number of states of the protocols that compute it. We would like to define the state complexity function as the function that assigns to a number ℓ the maximum state complexity of the predicates of size at most ℓ. However, defining the size of a predicate requires to fix a representation. Population protocols compute exactly the predicates expressible in Presburger arithmetic [8], and so there are at least three natural representations: formulas of Presburger arithmetic, existential formulas of Presburger arithmetic, and semilinear sets [20]. Since the translations between these representations involve superexponential blow-ups, we focus on threshold predicates of the form $x \geq \eta$, for which the size of the predicate is the size of η, independently of the representation. We choose to encode numbers in unary, and so we define $STATE(\eta)$ as the number of states of the smallest protocol computing $x \geq \eta$.

The inverse of $STATE(\eta)$ is the function that assigns to a number n the largest η such that a protocol with n states computes $x \geq \eta$. Recall that the busy beaver function assigns to a number n the largest η such that a Turing machine with n states started on a blank tape writes η consecutive ones on the

tape and terminates. Due to this analogy, we call the inverse of the state complexity function the *busy beaver function*, and call protocols computing predicates of the form $x \geq \eta$ busy beaver protocols, or just *busy beavers*.

Definition 1 The *busy beaver function* $BB\colon \mathbb{N} \to \mathbb{N}$ is defined as follows: $BB(n)$ is the largest $\eta \in \mathbb{N}$ such that the predicate $x \geq \eta$ is computed by some leaderless protocol with at most n states. The function $BB_L(n)$ is defined analogously, but for general protocols, possibly with leaders.

In [12] Blondin et al. give lower bounds on the busy beaver function:

Theorem 2.2 ([12]) *For every number of states n: $BB(n) \in \Omega(2^n)$ and $BB_L(n) \in \Omega(2^{2^n})$.*

However, to the best of our knowledge no upper bounds have been given.

3 Mathematical structure of stable sets

We define the *stable configurations* of a protocol:

Definition 2 Let $b \in \{0, 1\}$. A configuration C óf a protocol is *b-stable* if $O(C') = b$ for every configuration C' reachable from C. The set of b-stable configurations is denoted SC_b, and we let $SC = SC_0 \cup SC_1$.

It follows easily from the definitions that a population protocol *computes* a predicate $\varphi\colon \mathbb{N}^X \to \{0, 1\}$ iff for every input v and configuration C with $IC(v) \xrightarrow{*} C$ the condition $C \xrightarrow{*} SC_{\varphi(v)}$ holds.

Moreover, given a protocol computing φ, $\varphi(v) = b$ for an input v iff $IC(v) \xrightarrow{*} SC_b$.

A set $\mathcal{C}$ of configurations is *downward closed* if $C \in \mathcal{C}$ and $C' \leq C$ implies $C' \in \mathcal{C}$. The sets SC_0, SC_1, and SC are downward closed:

Lemma 3.1 *Let $\mathcal{P}$ be a protocol with n states. For every $b \in \{0, 1\}$ the set SC_b is downward closed.*

Proof Assume $C \in SC_b$ and $C' \leq C$. We prove $C' \in SC_b$ by contradiction, so assume that $C' \xrightarrow{*} C''$ for some C'' such that $O(C'') \neq b$. By monotonicity, $C = C' + (C - C') \xrightarrow{*} C'' + (C - C')$, and since $O(C'') \neq b$ we have $O(C'' + (C - C')) \neq b$. So $C' \in SC_b$. □

Given a downward closed set $\mathcal{C}$, a pair (B, S), where B is a configuration and $S \subseteq Q$, is a *basis element* of $\mathcal{C}$ if $B + \mathbb{N}^S \subseteq \mathcal{C}$. A *base* of $\mathcal{C}$ is a finite set $\mathcal{B}$ of *basis elements* such that $\mathcal{C} = \bigcup_{(B,S)\in\mathcal{B}}(B + \mathbb{N}^S)$. We define the *norm* of a basis element (B, S) as $\|(B, S)\|_\infty := \|B\|_\infty$, and the norm of a basis as the maximal norm of its elements.

It is well-known that every downward-closed set of configurations has a base. We prove a stronger result: the sets SC_0, SC_1, and SC have bases of small norm.

Lemma 3.2 *Let $\mathcal{P}$ be a protocol with n states. Every $\mathcal{C} \in \{SC_0, SC_1, SC\}$ has a basis of norm at most $2^{2(2n+1)!+1}$ with at most $\vartheta(n) := 2^{(2n+2)!}$ elements.*

Proof For the bound on the norm, let $\beta := 2^{2(2n+1)!}$ and fix a b-stable configuration C. Let $S := \{q \in Q \mid C(q) > 2\beta\}$, and define $B \leq C$ as follows: $B(i) := C(i)$ for $i \notin S$ and $B(i) := 2\beta$ for $i \in S$. Since $B \leq C$ and C is b-stable, so is B. We show that (B, S) is a basis element of SC_b, which proves the result for SC_0 and SC_1. Assume the contrary. Then some configuration $C' \in B + \mathbb{N}^S$ is not b-stable. So $C' \xrightarrow{*} C''$ for some C'' satisfying $C''(q) \geq 1$ for some state $q \in Q$ with $O(q) \neq b$; we say that C'' covers q.

By Rackoff's Theorem [25], C'' can be chosen so that $C' \xrightarrow{\sigma} C''$ for a sequence σ of length $2^{2^{\mathcal{O}(n)}}$; a more precise bound is $|\sigma| \leq \beta$ (see Theorem 3.12.11 in [17]). Since a transition moves at most two agents out of a given state, σ moves at most 2β agents out of a state. So, by the definition of B, the sequence σ is also executable from B, and also leads to a configuration that covers q. But this contradicts that B is b-stable. This concludes the proof for SC_0 and SC_1. For SC, just observe that the union of the bases of SC_0 and SC_1 is a basis of SC.

To prove the bound on the number of elements of the bases, observe that the number of pairs (B, S) such that B has norm at most k and $S \subseteq Q$ is at most $(k + 2)^n$. Indeed, for each state q there are at most $k + 2$ possibilities: $q \in S$, or $q \notin S$ and $0 \leq B(q) \leq k$. So $\vartheta \leq (2^{2(2n+1)!+1} + 2)^n \leq 2^{(2n+2)!}$. □

From now on we use the following terminology:

Definition 3 We call $\beta := 2^{2(2n+1)!+1}$ the *small basis constant* for the protocol $\mathcal{P}$. A *small basis* of SC_b or SC is a basis of norm at most β (guaranteed to exist by Lemma 3.2). Its elements are called *small basis elements*.

4 A general upper bound on the busy beaver function

We obtain a bound on the busy-beaver function $BB_L(n)$.

Fix a protocol $\mathcal{P}_n = (Q, T, L, \{x\}, I, O)$ with n states computing a predicate $x \geq \eta$. Observe that the unique input state is x, and so $IC(a) = a \cdot x + L$ for every input a.

Observe that for every input i we have $IC(i) \xrightarrow{*} C_i$ for some configuration $C_i \in SC$, and so $C_i \in B_i + \mathbb{N}^{S_i}$ for some basis element (B_i, S_i) of SC. If $i < \eta$, then $C_i \in SC_0$, and if $i \geq \eta$ then $C_i \in SC_1$. Lemma 4.1 below uses this observation to provide a sufficient condition for an input a to lie above η. The rest of the section shows that for a protocol with n states some number $a < f(n)$ satisfies the condition, where $f(n)$ is a function from the Fast Growing Hierarchy [18]. While the function $f(n)$ grows very fast, it is a recursive function. So, contrary to Turing machines, the busy-beaver function for

population protocols does not grow faster than any recursive function.

Lemma 4.1 *If there exist $a, b \in \mathbb{N}$, a basis element (B, S) of SC, and configurations $D_a, D_b \in \mathbb{N}^S$ satisfying*

1. $IC(a) \xrightarrow{*} B + D_a$, *and*
2. $b \cdot x \xrightarrow{*} D_b$,

then $\eta \leq a$.

Proof We first claim that $IC(a + \lambda b) \xrightarrow{*} B + D_a + \lambda D_b$ holds for every $\lambda \geq 0$. Observe that

$$IC(a + \lambda b) = (a + \lambda b) \cdot x + L = IC(a) + \lambda b \cdot x.$$

We have:

$$\begin{aligned} IC(a) + \lambda b \cdot x &\xrightarrow{*} B + D_a + \lambda b \cdot x \quad \text{by (1)} \\ &\xrightarrow{*} B + D_a + \lambda D_b \quad \text{by (2)} \end{aligned}$$

and the claim is proved.

Assume now that $\eta > a$, i.e. $\mathcal{P}_n$ rejects a. Since $D_a \in \mathbb{N}^S$, we have $B + D_a \in SC$, and so $B + D_a \in SC_0$ because $\mathcal{P}_n$ rejects a. Since $D_b \in \mathbb{N}^S$, we have $B + D_a + \lambda D_b \in B + \mathbb{N}^S$, and so $B + D_a + \lambda D_b \in SC_0$ for every $\lambda \geq 0$. So $\mathcal{P}_n$ rejects $a + \lambda b$ for every $\lambda \geq 0$, contradicting that $\mathcal{P}$ computes $x \geq \eta$. □

We now start our search for a number a satisfying the conditions of Lemma 4.1. First we identify a sequence of configurations $C_2, C_3, C_4 \ldots$ of SC satisfying conditions close to 1. and 2. in Lemma 4.1.

Lemma 4.2 *There exists a sequence $C_2, C_3, C_4 \ldots$ of configurations of SC satisfying:*

1. $IC(i) \xrightarrow{*} C_i$ *for every* $i \geq 2$, *and*
2. $C_i + j \cdot x \xrightarrow{*} C_{i+j}$ *for every* $j \geq 0$.

Proof Since $\mathcal{P}_n$ computes $x \geq \eta$, for every $i \geq 2$ every fair run of $\mathcal{P}$ starting at $IC(i)$ eventually reaches SC_0 or SC_1, depending on whether $i < \eta$ or $i \geq \eta$, and stays there forever. We define $C_2, C_3, C_4, \ldots$ as follows. First, we let C_2 be any configuration of SC reachable from $IC(2)$. Then, for every $i \geq 2$, assume that C_i has already been defined and satisfies $IC(i) \xrightarrow{*} C_i$. Observe that $IC(i + 1) = IC(i) + x$. Since $IC(i) \xrightarrow{*} C_i$, we also have $IC(i + 1) = IC(i) + x \xrightarrow{*} C_i + x$. This execution can be extended to a fair run, which eventually reaches SC. We let C_{i+1} be any configuration of SC reachable from $C_i + x$.

Let us show that $C_2, C_3, C_4 \ldots$ satisfies1.and 2.. Property 1. holds for C_2 by definition, and for $i \geq 2$ because $IC(i + 1) = IC(i) + x \xrightarrow{*} C_i + I(x) \xrightarrow{*} C_{i+1}$. For property 2., by monotonicity and the definition of C_i we have for every $2 \leq i \leq k$:

$$\begin{aligned} C_i + j \cdot x &\xrightarrow{*} C_{i+1} + (j - 1) \cdot x \\ &\xrightarrow{*} \cdots \xrightarrow{*} C_{i+j-1} + x \xrightarrow{*} C_{i+j} \end{aligned}$$

□

We can now easily prove the existence of a number a satisfying the conditions of Lemma 4.1. We start by recalling Dickson's Lemma:

Lemma 4.3 (Dickson's lemma) *For every infinite sequence $v_1, v_2, \ldots$ of vectors of the same dimension there is an infinite sequence $i_1 < i_2 < \ldots$ of indices such that $v_{i_1} \leq v_{i_2} \leq \ldots$.*

By Dickson's lemma, the sequence $C_2, C_3, C_4 \ldots$ of configurations of SC constructed in Lemma 4.2 contains an ordered subsequence $C_{i_1} \leq C_{i_2} \leq C_{i_3} \cdots$. Since SC has a finite basis, by the pigeonhole principle there exist numbers $k < \ell$ and a basis element (B, S) such that $C_{i_k}, C_{i_\ell} \in B + \mathbb{N}^S$. Since $C_k \leq C_\ell$, we have $C_k - C_\ell \in \mathbb{N}^S$, and so we can take:

$$a := k; b := \ell - k; D_a := C_k - B; D_b := C_\ell - C_k.$$

However, the proof of Dickson's lemma is non-constructive, and gives no bound on the size of a. To solve this problem we observe that, in the terminology of [18], the sequence $C_2\, C_3 \cdots$ is *linearly controlled*: there is a linear control function $f : \mathbb{N} \to \mathbb{N}$ satisfying $|C_i| \leq f(i)$. Indeed, since $InCi \xrightarrow{*} C_i$, we have $|C_i| = |IC(i)| = |L| + i$, and so we can take $f(n) = |L| + n$. This allows us to use a result on linearly controlled sequences from [18]. Say a finite sequence $v_0, v_1, \cdots, v_s$ of vectors of the same dimension is *good* if there are two indices $0 \leq i_1 < i_2 \leq s$ such that $v_{i_1} \leq v_{i_2}$. The maximal length of good linearly controlled sequences has been studied in [9, 18, 23]. In particular, this lemma follows easily from results of [18]:

Lemma 4.4 [18] *For every $\delta \in \mathbb{N}$ and for every elementary function $g : \mathbb{N} \to \mathbb{N}$, there exists a function $F_{\delta,g} : \mathbb{N} \to \mathbb{N}$ at level $\mathcal{F}_\omega$ of the Fast Growing Hierarchy satisfying the following property: For every infinite sequence $v_0, v_1, v_2 \ldots$ of vectors of $\mathbb{N}^n$ satisfying $|v_i| \leq i + \delta$, there exist $i_0 < i_1 < \ldots < i_{g(n)} \leq F_{\delta,g}(n)$ such that $v_{i_0} \leq v_{i_1} \leq \cdots \leq v_{g(n)}$.*

We do not need the exact definition of the Fast Growing Hierarchy (see [18]); for our purposes it suffices to know that the level $\mathcal{F}_\omega$ contains functions that, crudely speaking, grow like the Ackermann function. From this lemma we obtain:

Theorem 4.5 *Let $\mathcal{P}_n$ be a population protocol with n states and ℓ leaders computing a predicate $x \geq \eta$ for some $\eta \geq 2$. Then $\eta < F_{\ell,\vartheta}(n)$, where $\vartheta(n)$ is the function of Lemma 3.2.*

Proof By Lemma 4.4 there exist $\vartheta(n)+1$ indices $i_0 < i_1 < \ldots < i_{\vartheta(n)} \leq F_{\ell,\vartheta}(n)$ such that $C_{i_0} \leq C_{i_1} \leq \cdots \leq C_{i_{\vartheta(n)}}$. By the definition of ϑ and the pigeonhole principle, there are indices $k < \ell$ and a basis element (B, S) of SC_0 such that $C_{i_k}, C_{i_\ell} \in B + \mathbb{N}^S$ and $C_{i_k} \leq C_{i_\ell}$. By Lemma 4.1, $\eta \leq i_k \leq F_{\ell,\vartheta}(n)$. □

4.1 Is the bound optimal?

The function $F_{\ell,\vartheta}(n)$ grows so fast that one can doubt that the bound is even remotely close to optimal. However, recent results show that this would be less strange than it seems. If a protocol $\mathcal{P}$ computes a predicate $x \geq \eta$, then η is the smallest number such that $IC(\eta) \xrightarrow{*} SC_1$. Therefore, letting $\mathbf{BBP}(n)$ denote the busy beaver protocols with at most n states, and letting $SC_1^{\mathcal{P}}$ and $IC^{\mathcal{P}}$ denote the set SC_1 and the initial mapping of the protocol $\mathcal{P}$, we obtain:

$$BB_L(n) = \max_{\mathcal{P}\in\mathbf{BBP}(n)} \min\{i \in \mathbb{N} \mid \exists C \in SC_1^{\mathcal{P}} : IC^{\mathcal{P}}(i) \xrightarrow{*} C\}$$

Consider now a deceptively similar function. Let All_1 be the set of configurations C such that $O(C) = 1$, i.e. all agents are in states with ouput 1. Further, let $\mathbf{PP}(n)$ denote the set of *all* protocols with alphabet $X = \{x\}$, possibly with leaders, and n states. Notice that we include also the protocols that do not compute any predicate. Define

$$f(n) = \max_{\mathcal{P}\in\mathbf{PP}(n)} \min\{i \in \mathbb{N} \mid \exists C \in All_1^{\mathcal{P}} : IC^{\mathcal{P}}(i) \xrightarrow{*} C\}$$

Using recent results in Petri nets and Vector Addition Systems [14, 15, 21, 22] it is easy to prove that $f(n)$ grows faster than any primitive recursive function.[2] However, a recent result [10] by Balasubramanian et al. shows $f(n) \in 2^{\mathcal{O}(n)}$ for leaderless protocols.

These results suggest that a non-elementary bound on $BB_L(n)$ might well be optimal. However, in the rest of the paper we prove that this can only hold for population protocols with leaders. We show $BB(n) \in 2^{2^{\mathcal{O}(n)}}$, i.e. leaderless busy beavers with n states can only compute predicates $x \geq \eta$ for numbers η at most double exponential in n.

[2] The paper [21] considers protocols with one leader, and studies the problem of moving from a configuration with the leader in a state q_{in} and all other agents in another state r_{in}, to a configuration with the leader in a state q_f and all other agents in state r_f. Combined with [14, 15, 22], this shows that the smallest number of agents for which this is possible grows faster than any elementary function in the number of states of the protocol.

5 An upper bound for leaderless protocols

Fix a *leaderless* protocol $\mathcal{P}_n = (Q, T, \emptyset, \{x\}, I, O)$ with $|Q| = n$ states computing a predicate $x \geq \eta$. Observe that the unique input state is x, and so $IC(a) = a \cdot x$ for every input a. We prove that $\eta \leq 2^{(2n+2)!} \in 2^{2^{\mathcal{O}(n)}}$. We first introduce some well-known notions from the theory of Petri nets and Vector Addition Systems.

5.1 Potentially realisable multisets of transitions

The *displacement* of a transition $t = p, q \mapsto p', q'$ is the vector $\Delta_t \in \{-2, -1, 0, 1, 2\}^Q$ given by $\Delta_t := p' + q' - p - q$. Intuitively, $\Delta_t(q)$ is the change in the number of agents populating q caused by the execution of t. For example, if $Q = \{p, q, r\}$ and $t = p, q \mapsto p, r$ we have $\Delta_t(p) = 0$, $\Delta_t(q) = -1$, and $\Delta_t(r) = 1$. The *displacement* of a multiset $\pi \in \mathbb{N}^T$ is defined as $\Delta_\pi := \sum_{t\in T} \pi(t) \cdot \Delta_t$. We use the following notation:

$C \xRightarrow{\pi} C'$ denotes that $C' = C + \Delta_\pi$.

Intuitively, $C \xRightarrow{\pi} C'$ states that if C enables some sequence $t_1\, t_2 \ldots t_k \in T^*$ such that $\wr t_1, \ldots, t_k \wr = \pi$, then the execution of σ leads to C'. However, such a sequence may not exist. We call the multiset $\wr t_1, \ldots, t_k \wr$ the *Parikh mapping* of $t_1\, t_2 \ldots t_k$.

Say a configuration C is *j-saturated* if $C(q) \geq j$ for every $q \in Q$, i.e. if it populates all states with at least j agents. We have the following relations between $\xrightarrow{\sigma}$ and $\xRightarrow{\pi}$:

Lemma 5.1 (i) *If $C \xrightarrow{\sigma} C'$ then $C \xRightarrow{\pi} C'$, where π is the Parikh mapping of σ*

(ii) *If $C \xRightarrow{\pi} C'$ and C is $2|\pi|$-saturated, then $C \xrightarrow{\sigma} C'$ for any σ with Parikh mapping π.*

Proof (i): Easy induction on $|\sigma|$.

(ii): By induction on $|\pi|$. The basis case $\pi = \emptyset$ is trivial. Otherwise, let σ be any sequence with Parikh mapping π. Since this sequence is non empty, it can be decomposed as $t\sigma'$ where t is a transition and σ' is a sequence with Parikh mapping π' defined by $\pi'(t) = \pi(t) - 1$ and $\pi'(t') = \pi(t')$ for every $t' \neq t$. As C is $2|\pi|$-saturated, we have $C \xrightarrow{t} C''$ for some configuration C''. Further, C'' is $2|\pi'|$-saturated and $C'' \xRightarrow{\pi'} C'$. By induction hypothesis $C'' \xrightarrow{\sigma'} C'$. It follows that $C \xrightarrow{\sigma} C'$, and we are done. □

We introduce the set of potentially realisable multisets of transitions of a protocol:

Definition 4 A multiset π of transitions is *potentially realisable* if there are $i \in \mathbb{N}$ and $C \in \mathbb{N}^Q$ such that $IC(i) \xRightarrow{\pi} C$.

The reason for the name is as follows. If π is not potentially realisable, then by Lemma 5.1(i) no sequence $\sigma \in T^*$ with

Parikh mapping π can be executed from any initial configuration, and so π cannot be "realised". In other words, potential realisability is a necessary but not sufficient condition for the existence of an execution that "realises" π.

5.2 Structure of the proof

We can now give a high-level view of the proof of the bound $\eta \leq 2^{(2n+2)!}$. The starting point is a version of Lemma 4.1 in which, crucially, the condition $IC(b) \xrightarrow{*} D_b$ is replaced by the weaker $IC(b) \overset{*}{\Rightarrow} D_b$.

Lemma 5.2 *If there exist $a, b \in \mathbb{N}$, a basis element (B, S) of SC, configurations $D_a, D_b \in \mathbb{N}^S$, and a configuration D satisfying*

(i) $IC(a) \xrightarrow{*} D \xrightarrow{*} B + D_a$, *and*
(ii) $IC(b) \overset{\pi}{\Rightarrow} D_b$ *for some $\pi \in \mathbb{N}^T$ such that D is $2|\pi|$-saturated,*

then $\eta \leq a$.

Proof We first claim that

$$IC(a + \lambda b) \xrightarrow{*} B + D_a + \lambda D_b$$

holds for every $\lambda \geq 0$. To prove this, observe first that $IC(b) \overset{\pi}{\Rightarrow} D_b$ implies $D + IC(b) \overset{\pi}{\Rightarrow} D + D_b$. Since D is $2|\pi|$-saturated so is $D + IC(b)$, and, by Lemma 5.1(ii), we have $D + IC(b) \xrightarrow{\sigma} D + D_b$ where σ is any sequence with Parikh mapping π. With an induction on λ, we immediately derive

$$D + \lambda IC(b) \xrightarrow{\sigma^\lambda} D + \lambda D_b. \qquad (*)$$

Since $\mathcal{P}_n$ is leaderless, $IC(a + \lambda b) = IC(a) + \lambda IC(b)$ holds, and:

$$\begin{aligned} IC(a) + \lambda IC(b) &\xrightarrow{*} D + \lambda IC(b) && \text{by}(i) \\ &\xrightarrow{*} D + \lambda D_b && \text{by}(*) \\ &\xrightarrow{*} B + D_a + \lambda D_b && \text{by}(i). \end{aligned}$$

This proves the claim.

Assume now that $\eta > a$, i.e. $\mathcal{P}_n$ rejects a. Since $D_a, D_b \in \mathbb{N}^S$, we have $B + D_a + \lambda D_b \in SC$, and so $B + D_a + \lambda D_b \in SC_0$ for every $\lambda \geq 0$. So, by the claim, $\mathcal{P}_n$ rejects $a + \lambda b$ for every $\lambda \geq 0$, contradicting that it computes $x \geq \eta$. □

In the next sections we show that $a := 2^{(2n+2)!}$ satisfies the conditions of Lemma 5.2. We proceed in three steps:

(a) Section 5.3 proves that for every $j \in \mathbb{N}$ and input $a \geq j3^n$ the initial configuration $IC(a)$ can reach a j-saturated configuration D. We remark that this result is only true for leaderless protocols.
(b) Let $S \subseteq Q$. Section 5.4 proves that $IC(a) \overset{*}{\Rightarrow} B + D_a$, for $a \in \mathbb{N}$, $D_a \in \mathbb{N}^S$ and a configuration B, implies $IC(b) \overset{*}{\Rightarrow} D_b$ for some $b \in \mathbb{N}$, $D_b \in \mathbb{N}^S$, provided that a is "large" relative to $|B|$.
(c) Section 5.5 puts everything together, and gives the final bound.

5.3 Reaching *j*-saturated configurations

Recall our assumption that for every state $q \in Q$ there exists an input i_q such that $IC(i_q) \xrightarrow{*} C_q$ for some configuration C_q such that $C_q(q) > 0$. By monotonicity, we have $IC(i) \xrightarrow{*} C$ for the input $i := \sum_{q \in q} i_q$ and the 1-saturated configuration $C := \sum_{q \in Q} C_q$. We show that we can choose $i < 3^n$ and that $IC(i) \xrightarrow{\sigma} C$ for some σ such that $|\sigma| \leq 3^n$. It follows that for every $j \in \mathbb{N}$ the input $j3^n$ can reach a j-saturated configuration by executing j times σ.

Lemma 5.3 *Let C be a configuration satisfying $x \in [\![C]\!] \subset Q$. There exists a transition $p, q \mapsto p', q'$ such that $\{p, q\} \subseteq [\![C]\!]$ and $\{p', q'\} \not\subseteq [\![C]\!]$.*

Proof Since $[\![C]\!]$ is strictly included in Q, there exist $i \in \mathbb{N}$, a word $\sigma \in T^*$, a configuration C' such that $IC(i) \xrightarrow{\sigma} C'$ and $[\![C']\!] \not\subseteq [\![C]\!]$. Assume w.l.o.g. that σ has minimal length. If $\sigma = \epsilon$ then $[\![C']\!] \subseteq \{x\}$ contradicting $[\![C']\!] \not\subseteq [\![C]\!]$. So $\sigma = \sigma' t$ for some transition $t = p, q \mapsto p', q'$, and $IC(i) \xrightarrow{\sigma'} C'' \xrightarrow{t} C'$ for some configuration C''. From $C'' \xrightarrow{t} C'$ we derive $\{p, q\} \subseteq [\![C'']\!]$ and $[\![C']\!] \subseteq [\![C'']\!] \cup \{p', q'\}$. By minimality of σ and $|\sigma'| < |\sigma|$, we deduce that $[\![C'']\!] \subseteq [\![C]\!]$. In particular $\{p, q\} \subseteq [\![C]\!]$, and $[\![C']\!] \subseteq [\![C]\!] \cup \{p', q'\}$. As $[\![C']\!] \not\subseteq [\![C]\!]$, we deduce that $\{p', q'\} \not\subseteq [\![C]\!]$. □

Lemma 5.4 *There exists a word $\sigma \in T^*$, a 1-saturated configuration C and a sequence σ of length at most 3^n such that $IC(3^n) \xrightarrow{\sigma} C$.*

Proof We build a sequence $\sigma_0, \sigma_1 \ldots$ of words in T^* and a non decreasing sequence $C_0, C_1 \ldots$ of configurations such that for every $j \geq 0$ we have (with the convention $C_{-1} = 0$)

- $IC(3^j) \xrightarrow{\sigma_j} C_j$;
- $[\![C_{j-1}]\!]$ is saturated, or $[\![C_{j-1}]\!] \subset [\![C_j]\!]$, and
- $|\sigma_j| = (3^j - 1)/2$.

Since $\mathcal{P}_n$ has n states, some C_j with $0 \leq j \leq n$ is saturated.

The sequence is built inductively. Choose $C_0 = IC(1)$ and $\sigma_0 = \epsilon$. Assume that $\sigma_0, \ldots, \sigma_k$ has been built. If C_k is saturated, then choose $C_{k+1} := 3C_k$ and $\sigma_{k+1} := \sigma_k^3$. Assume C_k is not saturated. Since $x \in [\![C_1]\!]$ we have $x \in [\![C_k]\!] \subset Q$. By Lemma 5.3 there exists a transition $t = p, q \mapsto p', q'$ such that $p, q \in [\![C_k]\!]$ and $\{p', q'\} \not\subseteq [\![C_k]\!]$. Since $IC(3^k) \xrightarrow{\sigma_k} C_k$ and $\mathcal{P}_n$ is leaderless, we have $IC(3^{k+1}) \xrightarrow{\sigma_k^3} 3C_k$. Since $p, q \in [\![C_k]\!]$, transition t is enabled at $2C_k$. We choose

$C_{k+1} := C_k + C'_k$ where C'_k is the configuration satisfying $2C_k \xrightarrow{t} C'_k$. Now, just set $\sigma_{k+1} := \sigma_k^3 t$. □

5.4 Reaching ϵ-concentrated stable configurations

This section contains the core of the proof. We want to find some $b \in \mathbb{N}$ and a configuration $D_b \in \mathbb{N}^S$ with $IC(b) \xRightarrow{*} D_b$, where $S \subseteq Q$ is given by a basis element (B, S). We start by noting that any sequence $IC(a) \xrightarrow{*} B + D_a$, again with $a \in \mathbb{N}$ and $D_a \in \mathbb{N}^S$, already fulfils this condition *approximately* if a is large and (B, S) is a small basis element, i.e. B is "small" relative to D_a. The next lemma formalises this notion.

Definition 5 Let $0 < \epsilon \leq 1$ and $S \subseteq Q$. A configuration C is *ϵ-concentrated in S* if $C(S) \geq (1-\epsilon)|C|$ or, equivalently, $C(Q \setminus S) \leq \epsilon |C|$.

Lemma 5.5 *Let $k \geq 1$, let $a := kn\beta$, where β is the constant of Definition 3, and let D denote a configuration with $IC(a) \xrightarrow{*} D$. There exists a small basis element (B, S) of SC, and a $D_a \in \mathbb{N}^S$ s.t.*

1. *$IC(a) \xrightarrow{*} D \xrightarrow{*} B + D_a$, and*
2. *$B + D_a$ is $\frac{1}{k}$-concentrated in S.*

Proof As D is reachable from an input configuration, it can reach a configuration $B + D_a$, where $D_a \in \mathbb{N}^S$ and (B, S) is a small basis element of SC. The latter implies $\|B\|_\infty \leq \beta$, so we have $|B| \leq n\beta = a/k = |B + D_a|/k$, and $B + D_a$ is $\frac{1}{k}$-concentrated in S. □

In the remainder of this section we use another small basis theorem, Pottier's small basis theorem for Diophantine equations, to extend the above result: not only can we reach ϵ-concentrated configurations for arbitrarily small $\epsilon > 0$, we can also *potentially* reach a 0-concentrated configuration, i.e. a configuration in which *all* agents populate S.

For this, we observe that the potentially realisable multisets π (see Definition 4) are precisely the solutions of the system of $|Q| - 1$ Diophantine equations over the variables $\{\pi(t)\}_{t \in T}$ given by

$$\sum_{t \in T} \pi(t)\Delta_t(q) \geq 0 \qquad \text{for q} \in Q \setminus \{x\}.$$

Let $A \cdot y \geq 0$ be a homogeneous system of linear Diophantine inequalities, i.e. a solution is a vector m over the natural numbers such that $A \cdot m \geq 0$. A set $\mathcal{B}$ of solutions is a basis if every solution is the sum of a multiset of solutions of $\mathcal{B}$. Formally, $\mathcal{B}$ is a *basis* if for every solution m, there exists a multiset $M \in \mathbb{N}^{\mathcal{B}}$ such that $m = \sum_{b \in \mathcal{B}} M(b) \cdot b$. It is easy to see that every system has a finite basis. Pottier's theorem shows that it has a small basis:

Theorem 5.6 ([24]) *Let $A \cdot y \geq 0$ be a system of e linear Diophantine equations on v variables. There exists a basis $\mathcal{B} \subseteq \mathbb{N}^v$ of solutions such that for every $m \in \mathcal{B}$:*

$$\|m\|_1 \leq \left(1 + \max_{i=1}^{e} \sum_{j=1}^{v} |a_{ij}|\right)^e.$$

Since the potentially realisable multisets are the solutions of a system of Diophantine equations, by applying Pottier's theorem we obtain:

Corollary 5.7 *Let $\xi := 2(2|T| + 1)^{|Q|}$. There exists a basis of potentially realisable multisets such that every element π of the basis satisfies $|\pi| \leq \xi/2$. Moreover, $IC(i) \xRightarrow{\pi} C$ for some $i \leq \xi$ and some C such that $C(Q) \leq \xi$ and $C(x) = 0$.*

Proof The potentially realisable multisets are precisely the solutions of a system of $|Q| - 1$ Diophantine equations on $|T|$ variables. The matrix A of the system satisfies $|a_{ij}| = |\Delta_j(q_i)|$, and so $|a_{ij}| \leq 2$ for every $1 \leq i < |Q|$ and every $1 \leq j \leq |T|$. The result follows from Theorem 5.6.

For $C(x) = 0$, note that $IC(i) \xRightarrow{\pi} C$ implies $IC(i - C(x)) \xRightarrow{\pi} C - x \cdot C(x)$. □

Recall that Definition 3 introduced the small basis constant β. Analogously, we now introduce the Pottier constant:

Definition 6 We call $\xi := 2(2|T|+1)^{|Q|}$ the *Pottier constant* for the protocol $\mathcal{P}$.

Remark 1 For a deterministic population protocol (i.e. a protocol such that for every pair of states there is at most one transition) we could instead use $\xi = 2(|Q| + 2)^{|Q|}$.

The following lemma is at the core of our result. Intuitively, it states that if some potentially realisable multiset leads to a $(1/\xi)$-concentrated configuration in S, then some *small* potentially realisable multiset leads to a 0-concentrated configuration in S.

Lemma 5.8 *Let $IC(i) \xRightarrow{*} C$ and $S \subseteq Q$. If C is $(1/\xi)$-concentrated in S, then $IC(j) \xRightarrow{\theta} C'$ for some number j, potentially realisable multiset $\theta \in \mathbb{N}^T$, and configuration C' such that $|\theta| \leq \xi/2$, $j \leq \xi$ and C' is 0-concentrated on S, i.e. $C' \in \mathbb{N}^S$.*

Proof If $x \notin S$ then we take θ as the empty multiset and are done. Otherwise let $l := C(x)$ and set $i^* := i - l$ and $C^* := C - l \cdot x$. Clearly, $C^*(S) < i^*/\xi$ follows from $C(S) < i/\xi$. It therefore suffices to prove the lemma in the case of $C^* = C$ and $i^* = i$, i.e. $l = 0$.

Since $IC(i) \xRightarrow{*} C$, we have $IC(i) \xRightarrow{\pi} C$ for some potentially realisable multiset π. By Corollary 5.7 there exist potentially realisable multisets $\pi_1, \ldots, \pi_k \in \mathbb{N}^T$ s.t. $\pi = \pi_1 + \cdots + \pi_k$, numbers $i_1, \ldots, i_k \in \mathbb{N}$ and configurations $C_1, \ldots, C_k$ (not necessarily distinct) such that

- $i = \sum_{j=1}^{k} i_j$, $\pi = \sum_{j=1}^{k} \pi_j$, and $C = \sum_{j=1}^{k} C_j$.
- $IC(i_j) \stackrel{\pi_j}{\Rightarrow} C_j$.
- $i_j \leq \xi$ and $C_j(Q) \leq \xi$ for $j \in \{1, .., k\}$.

Let $J = \{j \in \{1, \ldots, k\} \mid i_j > 0\}$. We have

$$i = \sum_{j=1}^{k} i_j = \sum_{j \in J} i_j \leq \xi \cdot |J|.$$

Since $i > \xi \cdot C(S)$ by hypothesis, we deduce that $|J| > C(S)$. Assume that $C_j(S) > 0$ for every $j \in J$. Then $C(S) = \sum_{j \in J} C_j(S) \geq |J|$, a contradiction. So $C_j(S) = 0$ for some $j \in J$, and we take $\theta := \pi_j$. □

5.5 The upper bound

We have now obtained all the results needed for our final theorem.

Theorem 5.9 *Let $\mathcal{P}_n$ be a leaderless population protocol with n states, and let β and ξ be the constants of Definition 3 and Definition 6. If $\mathcal{P}_n$ computes a predicate $x \geq \eta$, then*

$$\eta \leq \xi n \beta 3^n \leq 2^{(2n+2)!}.$$

Proof Assume $\mathcal{P}_n$ computes $x \geq \eta$. We first prove that $\eta \leq a$ holds for $a := \xi n \beta 3^n$. It suffices to show that a satisfies the conditions of Lemma 5.2 for a suitable choice of b, (B, S), D_a, D_b, D, and π.

Lemma 5.4 shows that a 1-saturated configuration C with $IC(3^n) \to C$ exists. Choose $D := \xi n \beta \cdot C$. Observe that D is $(\xi n \beta)$-saturated and reachable from $IC(a)$.

Now choose (B, S) and D_a as the configurations and basis element of Lemma 5.5, where $k := \xi 3^n$ and D is chosen as above. With this choice of k Lemma 5.5 yields:

1. $IC(a) \stackrel{*}{\to} D \stackrel{*}{\to} B + D_a$; and
2. $B + D_a$ is $(1/\xi)$-concentrated in S.

(Regarding 2., note that $(1/\xi 3^n)$-concentrated implies $(1/\xi)$-concentrated.) Property 1. coincides with condition (i) of Lemma 5.2. Finally, we choose b, $D_b \in \mathbb{N}^S$, and π so that condition (ii) of Lemma 5.2 holds as well. In particular, b, D_b, and π must satisfy:

$$IC(b) \stackrel{\pi}{\Rightarrow} D_b \text{for some} \pi \text{s.t.} D \text{is} 2|\pi|\text{-saturated.}$$

By 1. we have $IC(a) \stackrel{*}{\Rightarrow} B + D_a$. Applying Lemma 5.8 with $i := a$ and $C := B + D_a$, we conclude that $IC(j) \stackrel{\theta}{\Rightarrow} C'$ for some $0 < j \leq \xi$, some multiset θ such that $|\theta| \leq \xi/2$, and some configuration C' 0-concentrated in S, i.e. satisfying $C'(Q \setminus S) = 0$.

Using $b := j$, $\pi := \theta$, and $D_b := C'$, we have $IC(b) \stackrel{\pi}{\Rightarrow} D_b$, which fulfils condition (ii) of Lemma 5.2 as well. Further, since D is $(n\beta\xi)$-saturated and $n\beta\xi \geq 2(\xi/2) \geq 2|\pi|$, the configuration D is $2|\pi|$-saturated. Applying the lemma we get $\eta \leq a$, concluding the first part of our proof.

We now show that $\xi n \beta 3^n \leq 2^{(2n+2)!}$. Since $\xi = 2(2|T| + 1)^{|Q|}$ and $\beta = 2^{2(2n+1)!+1}$, we deduce (for $n \geq 2$):

$$\begin{aligned}
\eta &\leq 2\,(2|T| + 1)^n\, n\, 2^{2(2n+1)!+1}\, 3^n \\
&\leq 4\,(2n^4 + 1)^n\, n\, 2^{2(2n+1)!}\, 2^{2n} \\
&\leq 4 \cdot 3^n n^{4n} n 2^{2(2n+1)!}\, 2^{2n} \\
&\leq 4 \cdot 3^n n^{4n+1} 2^{2(2n+1)!}\, 2^{2n}
\end{aligned}$$

and so

$$\begin{aligned}
2^{\eta} &\leq 2 + 2n + (4n + 1)\log_2 n + 2(2n + 1)! + 2n \\
&\leq 2(2n + 1)! + (2n + 1)(2 + 2\log_2 n) \\
&\leq (2n + 2)!
\end{aligned}$$

Therefore $\eta \leq 2^{(2n+2)!}$. □

6 Conclusion

We have obtained the first non-trivial lower bounds on the state complexity of population protocols computing counting predicates of the form $x \geq \eta$, a fundamental question about the model. The obvious open problems are to close the gap between the $\Omega(\log \log \eta)$ lower bound and the $\mathcal{O}(\log \eta)$ upper bound for the leaderless case, and the even larger gap between $\mathcal{O}(\log \log \eta)$ and (roughly speaking), the $\Omega(\alpha(\eta))$ lower bound for protocols with leaders, where $\alpha(\eta)$ is the inverse of the Ackermann function.

Acknowledgements This work was supported by an ERC Advanced Grant (787367: PaVeS), by the Research Training Network of the Deutsche Forschungsgemeinschaft (DFG) (378803395: ConVeY), and by the grant ANR-17-CE40-0028 of the French National Research Agency ANR (project BRAVAS).A previous version of this paper appeared in the proceedings of PODC 2021 [13].

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Abriola, S., Figueira, S., Senno, G.: Linearizing well quasi-orders and bounding the length of bad sequences. Theor. Comput. Sci. **603**, 3–22 (2015)
2. Alistarh, D., Aspnes, J., Eisenstat, D., Gelashvili, R., Rivest, R.L.: Time-space trade-offs in population protocols. In: SODA, pp. 2560–2579. SIAM, Philadelphia (2017)
3. Alistarh, D., Aspnes, J., Gelashvili, R.: Space-optimal majority in population protocols. In: SODA, pp. 2221–2239. SIAM, Philadelphia (2018)
4. Alistarh, D., Gelashvili, R.: Recent algorithmic advances in population protocols. SIGACT News **49**(3), 63–73 (2018)
5. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: PODC, pp 290–299. ACM (2004)
6. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distrib. Comput. **18**(4), 235–253 (2006)
7. Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. Distrib. Comput. **21**(3), 183–199 (2008)
8. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. **20**(4), 279–304 (2007)
9. Balasubramanian, A.R.: Complexity of controlled bad sequences over finite sets of $\mathbb{N}^d$. In: LICS, pp. 130–140. ACM (2020)
10. Balasubramanian, A.R., Esparza, J., Raskin, M.A.: Finding cut-offs in leaderless rendez-vous protocols is easy. In: FoSSaCS. Lecture Notes in Computer Science, vol. 12650. Springer, New York (2021)
11. Blondin, M., Esparza, J., Genest, B., Helfrich, M., Jaax, S.: Succinct population protocols for Presburger arithmetic. In: STACS, vol. 154, pp. 40:1–40:15. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
12. Blondin, M., Esparza, J., Jaax, S.: Large flocks of small birds: On the minimal size of population protocols. In *STACS*, vol. 96, pp. 16:1–16:14, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018)
13. Czerner, P., Esparza, J.: Lower bounds on the state complexity of population protocols. In: PODC, pp. 45–54. ACM (2021)
14. Czerwinski, W., Lasota, S., Lazic, R., Leroux, J., Mazowiecki, F.: The reachability problem for Petri nets is not elementary. J. ACM **68**(1), 7:1-7:28 (2021)
15. Czerwinski, W., Orlikowski, L.: Reachability in vector addition systems is Ackermann-complete. CoRR, arxiv:2104.13866 (2021)
16. Elsässer, R., Radzik, T.: Recent results in population protocols for exact majority and leader election. Bull. EATCS **126** (2018)
17. Esparza, J.: Petri nets lecture notes (2019). https://archive.model.in.tum.de/um/courses/petri/SS2019/PNSkript.pdf
18. Figueira, D., Figueira, S., Schmitz, S., Schnoebelen, P.: Ackermannian and primitive-recursive bounds with Dickson's lemma. In: LICS, pp. 269–278. IEEE Computer Society (2011)
19. Gąsieniec, L., Stachowiak, G.: Enhanced phase clocks, population protocols, and fast space optimal leader election. J. ACM **68**(1), 1–12 (2020)
20. Haase, C.: A survival guide to Presburger arithmetic. ACM SIGLOG News **5**(3), 67–82 (2018)
21. Horn, F., Sangnier, A.: Deciding the existence of cut-off in parameterized rendez-vous networks. In: CONCUR, vol. 171, pp. 46:1–46:16. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
22. Leroux, J.: The reachability problem for Petri nets is not primitive recursive. CoRR arxiv:2104.12695 (2021)
23. McAloon, K.: Petri nets and large finite sets. Theor. Comput. Sci. **32**, 173–183 (1984)
24. Pottier, L.: Minimal solutions of linear diophantine systems: bounds and algorithms. In: RTA. Lecture Notes in Computer Science, vol. 488, pp. 162–173. New York, Springer (1991)
25. Rackoff, C.: The covering and boundedness problems for vector addition systems. Theor. Comput. Sci. **6**, 223–231 (1978)
26. Schmitz, S.: Complexity hierarchies beyond elementary. ACM Trans. Comput. Theory **8**(1), 3;1-3:36 (2016)

B. Breaking Through the $\Omega(n)$-Space Barrier: Population Protocols Decide Double-Exponential Thresholds

Title	Breaking Through the $\Omega(n)$-Space Barrier: Population Protocols Decide Double-Exponential Thresholds
Authors	Philipp Czerner
Venue	DISC, 2024
Publisher	Dagstuhl
DOI	10.4230/LIPIcs.DISC.2024.17

Breaking Through the $\Omega(n)$-Space Barrier: Population Protocols Decide Double-Exponential Thresholds

Philipp Czerner
Department of Informatics, TU München, Germany

Abstract

Population protocols are a model of distributed computation in which finite-state agents interact randomly in pairs. A protocol decides for any initial configuration whether it satisfies a fixed property, specified as a predicate on the set of configurations. A family of protocols deciding predicates φ_n is *succinct* if it uses $\mathcal{O}(|\varphi_n|)$ states, where φ_n is encoded as quantifier-free Presburger formula with coefficients in binary. (All predicates decidable by population protocols can be encoded in this manner.) While it is known that succinct protocols exist for all predicates, it is open whether protocols with $o(|\varphi_n|)$ states exist for *any* family of predicates φ_n. We answer this affirmatively, by constructing protocols with $\mathcal{O}(\log|\varphi_n|)$ states for some family of threshold predicates $\varphi_n(x) \Leftrightarrow x \geq k_n$, with $k_1, k_2, \ldots \in \mathbb{N}$. (In other words, protocols with $\mathcal{O}(n)$ states that decide $x \geq k$ for a $k \geq 2^{2^n}$.) This matches a known lower bound. Moreover, our construction for threshold predicates is the first that is not 1-aware, and it is almost self-stabilising.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Distributed computing, population protocols, state complexity

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.17

Related Version *Full Version*: `https://arxiv.org/abs/2204.02115` [18]

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

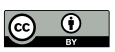
1 Introduction

Population protocols are a distributed model of computation where a large number of indistinguishable finite-state agents interact randomly in pairs. The goal of the computation is to decide whether an initial configuration satisfies a given property. The model was introduced in 2004 by Angluin et al. [4, 5] to model mobile sensor networks with limited computational capabilities (see e.g. [28, 22]). It is also closely related to the model of chemical reaction networks, in which agents, representing discrete molecules, interact stochastically [17].

A protocol is a finite set of transition rules according to which agents interact, but it can be executed on an infinite family of initial configurations. Agents decide collectively whether the initial configuration fulfils some (global) property by *stable consensus*; each agent holds an opinion about the output and may freely change it, but eventually all agents agree.

An example of a property decidable by population protocols is *majority*: initially all agents are in one of two states, x and y, and they try to decide whether x has at least as many agents as y. This property may be expressed by the predicate $\varphi(x, y) \Leftrightarrow x \geq y$.

In a seminal paper, Angluin et al. [7] proved that the predicates that can be decided by population protocols correspond precisely to the properties expressible in Presburger arithmetic, the first-order theory of addition.



To execute a population protocol, the scheduler picks two agents uniformly at random and executes a pairwise transition on these agents. These two agents interact and may change states. The number of agents does not change during the computation. It will be denoted m throughout this paper.

Population protocols are often extended with a *leader* – an auxiliary agent not part of the input, which can assist the computation. It is known that this does not increase the expressive power of the model, i.e. it can still decide precisely the predicates expressible in Presburger arithmetic. However, it is known that leaders enable an exponential speed-up [6, 1] in terms of the time that is needed to come to a consensus.

Space complexity. Many constructions in the literature need a large number of states. We estimate, for example, that the protocols of [6] need tens of thousands of states. This is a major obstacle to implementing these protocols in chemical reactions, as every state corresponds to a chemical compound.

This motivates the study of *space complexity*, the minimal number of states necessary for a population protocol to decide a given predicate. Predicates are usually encoded as quantifier-free Presburger formulae with coefficients in binary. For example, the predicates $\varphi_n(x) \Leftrightarrow x \geq 2^n$ have length $|\varphi_n| \in \Theta(n)$. Formally we define $\mathrm{space}(\varphi)$ as the smallest number of states of any protocol deciding φ, and $\mathrm{space}_L(\varphi)$ as the analogous function for protocols with a leader. Clearly, $\mathrm{space}(\varphi)_L \leq \mathrm{space}(\varphi)$.

The original construction in [4] showed $\mathrm{space}(\varphi) \in \mathcal{O}(2^{|\varphi|})$ – impractically large. For the family of *threshold predicates* $\tau_n(x) \Leftrightarrow x \geq n$ Blondin, Esparza and Jaax [14] prove $\mathrm{space}(\tau_n) \in \mathcal{O}(|\tau_n|)$, i.e. they have polynomial space complexity. For several years it was open whether similarly succinct protocols exist for every predicate. This was answered positively in [13], showing $\mathrm{space}(\varphi) \in \mathcal{O}(\mathrm{poly}(|\varphi|))$ for all φ.

Is it possible to do much better? For most predicates it is not; based on a simple counting argument one can show that for every family φ_n with $|\varphi_n| \in \mathcal{O}(n)$ there is an infinite subfamily $(\varphi'_n)_n \subseteq (\varphi_n)_n$ with $\mathrm{space}_L(\varphi'_n) \in \Omega(|\varphi_n|^{1/4-\varepsilon})$, for any $\varepsilon > 0$ [14].

This covers threshold predicates and many other natural families of protocols (e.g. $\varphi_n(x) \Leftrightarrow x \equiv 0 \pmod{n}$ or $\varphi_n(x, y) \Leftrightarrow x \geq ny$). But it is not an impenetrable barrier, even for the case of threshold protocols: it does not rule out constructions that work for *infinitely many* (but not all) thresholds and use only, say, logarithmically many states. Indeed, if leaders are allowed this is known to be possible: [14] shows $\mathrm{space}_L(\tau'_n) \in \mathcal{O}(\log|\tau'_n|)$ for some subfamily τ'_n of threshold predicates.

Recently, *general* lower bounds have been obtained, showing $\mathrm{space}(\tau_n) \in \Omega(\log^{1-\varepsilon}|\tau_n|)$ for all $\varepsilon > 0$ [19, 20]. The same bound (up to $\varepsilon = 1/2$) holds even if the model is extended with leaders [24].

For leaderless population protocols, these results leave an exponential gap. In this paper we settle that question and show that, contrary to prevailing opinion, $\mathrm{space}(\tau'_n) \in \mathcal{O}(\log|\tau'_n|)$ for some subfamily τ'_n of threshold predicates. In other words, we construct the first family of *leaderless* population protocols that decide double-exponential thresholds and break through the polynomial barrier.

Robustness. Since population protocols model computations where large numbers of agents interact, it is desirable that protocols deal robustly with noise. In a chemical reaction, for example, there can be trace amounts of unwanted molecules. So the initial configuration of the protocol would have the form $C_I + C_N$, where C_I is the "intended" initial configuration, containing only agents in the designated initial states, and C_N is a "noise" configuration, which can contain agents in arbitrary states.

Table 1 Prior results on the state complexity of threshold predicates $\varphi(x) \Leftrightarrow x \geq k$, for $k \in \mathbb{N}$. Upper bounds need only hold for infinitely many k. We elide exponentially dominated factors from lower bounds.

year	result	type	ordinary	with leaders
2018	Blondin, Esparza, Jaax [14]	construction	$\mathcal{O}(\lvert\varphi\rvert)$	$\mathcal{O}(\log\lvert\varphi\rvert)$
2021	Czerner, Esparza [19]	impossibility	$\Omega(\log\log\lvert\varphi\rvert)$	$\Omega(\text{ack}^{-1}\lvert\varphi\rvert)$
2021	Czerner, Esparza, Leroux [20]	impossibility	$\Omega(\log\lvert\varphi\rvert)$	
2022	Leroux [24]	impossibility		$\Omega(\log\lvert\varphi\rvert)$
2024	this paper	construction	$\mathcal{O}(\log\lvert\varphi\rvert)$	

For threshold predicates, specifically, we want to decide whether $|C_I| + |C_N|$ exceeds some threshold $k \in \mathbb{N}$, under some reasonable restrictions to C_I, C_N. However, all known threshold protocols fail even for the case $|C_N| = 1$. Is it possible to do better?

If C_N can be chosen arbitrarily, then the protocol has to work correctly for *all* input configurations. This property is known as *self-stabilisation*, and it has also been investigated in the context of population protocols [8, 16, 15]. However, it can only be achieved in extensions of the model (e.g. on specific communication graphs, or with a non-constant number of states). This is easy to see in the case of threshold predicates: if any configuration is stably accepting, then any smaller configuration is stably accepting as well. In particular, there is a stably accepting configuration with $k - 1$ agents.

While full self-stabilisation is impossible, in this paper we show that one can come remarkably close. We prove that our construction is *almost self-stabilising*, meaning that it computes the correct output for all C_I, C_N with $|C_I| \geq n$, where n is the number of states of the protocol. We do not constraint C_N at all. Since $n \in \mathcal{O}(\log\log k)$ in our protocol, this means that one can take an arbitrary configuration C_N one wishes to count, add a tiny amount of agents to the initial state, and the protocol will compute the correct output.

Related work. We consider the space complexity of families of protocols, each of which decides a different predicate. In another line of research, one considers a family of protocols for the *same* predicate, where each protocol is specialised for a fixed population size m.

In the original model of population protocols (which is also the model of this paper), the set of states is fixed, and the same protocol can be used for an arbitrary number of agents. Relaxing this requirement has opened up a fruitful avenue of research; here, the number of states depends on m (e.g. the protocol has $\mathcal{O}(\log m)$ states, or even $\mathcal{O}(\log\log m)$ states). In this model, faster protocols can be achieved [3, 26, 27].

It has also led to space-efficient, fast protocols, which stabilise within $\mathcal{O}(\text{polylog}\, m)$ parallel time, using a state-space that grows only slowly with the number of agents, e.g. $\mathcal{O}(\text{polylog}\, m)$ states [1, 12, 2, 10, 9, 11, 21]. These protocols have focused on the majority predicate. Moreover, lower bounds and results on time-space tradeoffs have been developed in this model [1, 2].

2 Main result

We construct population protocols (without leaders) for an infinite family of threshold predicates $\varphi_n(x) \Leftrightarrow x \geq k_n$, with $k_1, \ldots \in \mathbb{N}$, proving an $\mathcal{O}(\log|\varphi_n|)$ upper bound on their state complexity. This closes the final gap in the state complexity of threshold predicates.

As in prior work, our result is not a construction for arbitrary thresholds k, only for an infinite family of them. It is, therefore, easier to formally state by fixing the number of states n and specifying the largest threshold k that can be decided by a protocol with n states.

▶ **Theorem 1.** *For every $n \in \mathbb{N}$ there is a population protocol with $\mathcal{O}(n)$ states deciding the predicate $\varphi(x) \Leftrightarrow x \geq k$ for some $k \geq 2^{2^n}$.*

Proof. This will follow from theorems 3 and 5. ◀

The result is surprising, as prevailing opinion was that the existing constructions are optimal. This was based on the following:

- It is intuitive that population protocols with leaders have an advantage. In particular, one can draw a parallel to time complexity, where an exponential gap is proven: for some predicates protocols with leaders have $\mathcal{O}(\operatorname{polylog} m)$ parallel time, while all leaderless protocols have $\Omega(m)$ parallel time.
- The $\mathcal{O}(\log \log k)$-state construction from [14] crucially depends on having leaders.
- The technique to show the $\Omega(\log \log k)$ lower bound could, for the most part, also be used for a $\Omega(\log k)$ bound. Only the use of Rackoff's theorem, a general result for Petri nets, does not extend.
- There is a conditional impossibility result, showing that $\Omega(\log k)$ states are necessary for leaderless 1-aware protocols. [14] (Essentially, protocols where some agent knows at some point that the threshold has been exceeded.) All prior constructions are 1-aware.

Regarding the last point, our protocol evades the mentioned conditional impossibility result by being the first construction that is not 1-aware. Intuitively, our protocol only accepts provisionally and continues to check that no invariant has been violated. Based on this, we also obtain the following robustness guarantee:

▶ **Theorem 2.** *The protocols of Theorem 1 are almost self-stabilising.*

Overview. We build on the technique of Lipton [25], which describes a double-exponential counting routine in vector addition systems. Implementing this technique requires the use of procedure calls; our first contribution are *population programs*, a model in which population protocols can be constructed by writing structured programs, in Section 4. Every such program can be converted into an equivalent population protocol.

However, population programs provide weaker guarantees than the model of parallel programs used in [25]. Both models access registers with values in $\mathbb{N}$. In a parallel program these are initialised to 0, while in a population program *all* registers start with arbitrary values. This limitation is essential for our conversion into population protocols.

A straightforward implementation is, therefore, impossible. Instead, we have to adapt the technique to work with arbitrary initial configurations. Our second contribution, and the main technical difficulty of this result, is extending the original technique with error-checking routines to work in our model. We use a detect-restart loop, which determines whether the initial configuration is "bad" and, if so, restarts with a new initial configuration. The stochastic behaviour of population protocols ensures that a "good" initial configuration is reached eventually. Standard techniques could be used to avoid restarts with high probability and achieve an optimal running time, but this is beyond the scope of this paper.

A high level overview of both the original technique as well as our error-checking strategy is given in Section 5. We then give a detailed description of our construction in Section 6.

To get population protocols, we need to convert from population programs. We split this into two parts. First, we use standard techniques to lower population programs to *population machines*, an assembly-like programming language. In a second step we simulate arbitrary population machines by population protocols. This conversion is described in Section 7.

Finally, we introduce the notion of being almost self-stabilising in Section 8, and prove that our construction has this property.

To start out, Section 3 introduces the necessary mathematical notation and formally defines population protocols as well as the notion of stable computation.

3 Preliminaries

Multisets. We assume $0 \in \mathbb{N}$. For a finite set Q we write $\mathbb{N}^Q$ to denote the set of multisets containing elements in Q. For such a multiset $C \in \mathbb{N}^Q$, we write $C(S) := \sum_{q \in S} C(q)$ to denote the total number of elements in some $S \subseteq Q$, and set $|C| := C(Q)$. Given two multisets $C, C' \in \mathbb{N}^Q$ we write $C \leq C'$ if $C(q) \leq C'(q)$ for all $q \in Q$, and we write $C + C'$ and $C - C'$ for the componentwise sum and difference (the latter only if $C \geq C'$). Abusing notation slightly, we use an element $q \in Q$ to represent the multiset C containing exactly q, i.e. $C(q) = 1$ and $C(r) = 0$ for $r \neq q$.

Stable computation. We are going to give a general definition of stable computation not limited to population protocols, so that we can later reuse it for population programs and population machines. Let $\mathcal{C}$ denote a set of configurations and $\to$ a left-total binary relation on $\mathcal{C}$ (i.e. for every $C \in \mathcal{C}$ there is a $C' \in \mathcal{C}$ with $C \to C'$). Further, we assume some notion of output, i.e. some configurations have an output $b \in \{\mathsf{true}, \mathsf{false}\}$ (but not necessarily all).

A sequence $\tau = (C_i)_{i \in \mathbb{N}}$ with $C_i \in \mathcal{C}$ is a *run* if $C_i \to C_{i+1}$ for all $i \in \mathbb{N}$. We say that τ *stabilises to* b, for $b \in \{\mathsf{true}, \mathsf{false}\}$, if there is an i s.t. C_j has output b for every $j \geq i$. A run τ is *fair* if $\cap_{i \geq 0}\{C_i, C_{i+1}, ...\}$ is closed under $\to$, i.e. every configuration that *can* be reached infinitely often *is*.

Population protocols. A *population protocol* is a tuple $PP = (Q, \delta, I, O)$, where

- Q is a finite set of *states*,
- $\delta \subseteq Q^4$ is a set of *transitions*,
- $I \subseteq Q$ is a set of *input states*, and
- $O \subseteq Q$ is a set of *accepting states*.

We write transitions as $(q, r \mapsto q', r')$, for $q, r, q', r' \in Q$. A *configuration* of PP is a multiset $C \in \mathbb{N}^Q$ with $|C| > 0$. A configuration C is *initial* if $C(q) = 0$ for $q \notin I$ (one might also say $C \in \mathbb{N}^I$ instead). It has output true if $C(q) = 0$ for $q \notin O$, and output false if $C(q) = 0$ for $q \in O$. For two configurations C, C' we write $C \to C'$ if $C = C'$ or if there is a transition $(q, r \mapsto q', r') \in \delta$ s.t. $C \geq q + r$ and $C' = C - q - r + q' + r'$.

Let $\varphi : \mathbb{N}^I \to \{\mathsf{true}, \mathsf{false}\}$ denote a predicate. We say that PP *decides* φ, if every fair run starting at an initial configuration $C \in \mathbb{N}^I$ stabilises to $\varphi(C)$, where fair run and stabilisation are defined as above.

4 Population Programs

We introduce population programs, which allows us to specify population protocols using structured programs. An example is shown in Figure 1.

Formally, a *population program* is a tuple $\mathcal{P} = (Q, \mathrm{Proc})$, where Q is a finite set of *registers* and Proc is a list of *procedures*. Each procedure has a name and consists of (possibly nested) while-loops, if-statements and instructions. These are described in detail below.

```
1: procedure Main
2:   OF := false
3:   while ¬Test(4) do
4:     Clean
5:   OF := true
6:   while ¬Test(7) do
7:     Clean
8:   OF := false
9:   while true do
10:    Clean
```

```
1: procedure Test(i)
2:   for j = 1, ..., i do
3:     if detect x > 0 then
4:       x ↦ y
5:     else
6:       return false
7:   return true
```

```
1: procedure Clean
2:   if detect z > 0 then
3:     restart
4:   swap x, y
5:   while detect y > 0 do
6:     y ↦ x
```

Figure 1 A population program for $\varphi(x) \Leftrightarrow 4 \leq x < 7$ using registers x, y, z. Main is run initially and decides the predicate, Test(i) tries to move i units from x to y and reports whether it succeeded, and Clean checks whether z is empty and moves some number of units from y to x. If Clean detects an agent in z, it restarts the computation. As every run calls Clean infinitely often, this serves to reject initial configurations where z is nonzero; eventually the protocol will be restarted with $z = 0$. This is an illustrative example and some simplifications are possible. E.g. the instruction (**swap** x, y) in Clean is superfluous; additionally, instead of checking $z > 0$ one could omit that register entirely.

Primitives. Each register $x \in Q$ can take values in $\mathbb{N}$. Only three operations on these registers are supported.

- The move instruction ($x \mapsto y$), for $x, y \in Q$, decreases the value of x by one, and increases the value of y by one. We also say that it moves one unit from x to y. If x is empty, i.e. its value is zero, the programs hangs and makes no further progress
- The nondeterministic nonzero-check (**detect** $x > 0$), for $x \in Q$, nondeterministically returns either false or whether $x > 0$. In other words, if it does return true, it certifies that x is nonzero. If it returns false, however, no information has been gained. We consider only fair runs, so if x is nonzero the check cannot return false infinitely often.
- A swap (**swap** x, y) exchanges the values of the two registers x, y. This primitive is not necessary, but it simplifies the implementation.

Loops and branches. Population programs use while-loops and if-statements, which function as one would expect.

We also use for-loops. These, however, are just a macro and expand into multiple copies of their body. For example, in the program in Figure 1 the for-loop in Test expands into i copies of the contained if-statement.

Procedures. Our model has procedure calls, but no recursion. Procedures have no arguments, but we may have parameterised copies of a procedure. The program in Figure 1, for example, has four procedures: Main, Clean, Test(4), and Test(7).

Procedure calls must be acyclic. It is thus not possible for a procedure to call itself, and the size of the call stack remains bounded. We remark that one could inline every procedure call. The main reason to make use of procedures at all is succinctness: if our program contains too many instructions, the resulting population protocol has too many states.

Procedures may return a single boolean value, and procedure calls can be used as expressions in conditions of while- or if-statements.

Output flag. There is an output flag OF, which can be modified only via the instructions $OF :=$ true and $OF :=$ false. (These are special instructions; it is not possible to assign values to registers.) The output flag determines the output of the computation.

Initialisation and restarts. The only guarantee on the initial configuration is that execution starts at Main. In particular, all registers may have arbitrary values.

There is one final kind of instruction: **restart**. As the name suggests, it restarts the computation. It does so by nondeterministically picking any initial configuration s.t. the sum of all registers does not change.

Size. The *size* of $\mathcal{P}$ is defined as $|Q| + L + S$, where L is the number of instructions and S is the *swap-size*. The latter is defined as the number of pairs $(x, y) \in Q^2$ for which it is syntactically possible for x to swap with y via any sequence of swaps. [1] For example, in Figure 1 the swap-size is two: $(x, y), (y, x)$ can be swapped, but e.g. (x, z) cannot. If we add a (**swap** y, z) instruction at any point, then (x, z) can be swapped (transitively), and the swap-size would be 6.

Configurations and Computation. A *configuration* of $\mathcal{P}$ is a tuple $D = (C, OF, \sigma)$, where $C \in \mathbb{N}^Q$ is the *register configuration*, $OF \in \{\mathsf{true}, \mathsf{false}\}$ is the value of the output flag, and $\sigma \in (\mathrm{Proc} \times \mathbb{N})^*$ is the call stack, storing names and currently executed instructions of called procedures. (E.g. $\sigma = ((\mathsf{Main}, 3), (\mathsf{Test(4)}, 1))$ when Test is first called in Figure 1.) A configuration is *initial* if $\sigma = ((\mathsf{Main}, 1))$ and it has *output* OF. For two configurations D, D' we write $D \to D'$ if D can move to D' after executing one instruction.

Using the general notion of stable computation defined in Section 3, we say that $\mathcal{P}$ *decides* a predicate $\varphi(x)$, for $k \in \mathbb{N}$, if every run started at an initial configuration (C, OF, σ) stabilises to $\varphi(|C|)$. Note that this definition limits population programs to decide only unary predicates.

Notation. When analysing population programs it often suffices to consider only the register configuration Let $C, C' \in \mathbb{N}^Q$, $b \in \{\mathsf{false}, \mathsf{true}\}$ and let $f \in \mathrm{Proc}$ denote a procedure. We consider the possible outcomes when executing f in a configuration with registers C. Note that the program is nondeterministic, so multiple outcomes are possible. If f may return b with register configuration C', we write $C, f \to C', b$. For procedures not returning a value, we use $C, f \to C'$ instead. If f may initiate a restart, we write $C, f \to \mathbf{restart}$. If f may hang or not terminate, we write $C, f \to \bot$. Finally, we define $\mathrm{post}(C, f) := \{S : C, f \to S\}$.

5 High-level Overview

We give an intuitive explanation of our construction. This section has two parts. As mentioned, we use the technique of Lipton [25] to count to 2^{2^n} using $4n$ registers. We will give a brief explanation of the original technique in Section 5.1. Readers might also find the restatement of Liptons proof in [23] instructive – the Petri net programs introduced therein are closer to our approach, and more similar to models used in the recent Petri net literature.

A straightforward application of the above technique only works if some guarantees are provided for the initial configuration (e.g. that the $4n$ registers used are empty, while an additional register holds all input agents). No such guarantees are given in our model. Instead, we have to deal with adversarial initialisation, i.e. the notion that registers hold arbitrary values in the initial configuration. Section 5.2 describes the problems that arise, as well as our strategies for dealing with them.

[1] Unfortunately, without restrictions we would convert swaps to population protocols with a quadratic blow-up in states, so we introduce this technical notion to quantify the overhead.

5.1 Double-exponential counting

The biggest limitation of population programs is their inability to detect absence of agents. This is reflected in the (**detect** $x > 0$) primitive; it may return true and thereby certify that x is nonzero, but it may always return false, regardless of whether $x = 0$ actually holds. In particular, it is impossible to implement a zero-check.

However, Lipton observes that if we have two registers $x, \overline{x}$ and ensure that the invariant $x + \overline{x} = k$ holds, for some fixed $k \in \mathbb{N}$, then $x = 0$ is equivalent to $\overline{x} \geq k$. Crucially, it is possible to certify the latter property; if we have a procedure for checking $\overline{x} \geq k$, we can run both checks ($x > 0$ and $\overline{x} \geq k$) in a loop until one of them succeeds. Therefore, we may treat x as k-bounded register with deterministic zero-checks.

This seems to present a chicken-and-egg problem: to implement this register we require a procedure for $\overline{x} \geq k$, but checking such a threshold is already the overall goal of the program. Lipton solves this by implementing a bootstrapping sequence. For small k, e.g. $k = 2$, one can easily implement the required $\overline{x} \geq k$ check. We use that as subroutine for *two* k-bounded registers, x and y. Using the deterministic zero-checks, x and y can together simulate a single k^2-bounded register with deterministic zero-check; this then leads to a procedure for checking $\overline{z} \geq k^2$ (for some other register $\overline{z}$).

Lipton iterates this construction n times. We have n levels of registers, with four registers $x_i, y_i, \overline{x}_i, \overline{y}_i$ on each level $i \in \{1, ..., n\}$. For each level we have a constant $N_i \in \mathbb{N}$ and ensure that $x_i + \overline{x}_i = y_i + \overline{y}_i = N_i$ holds. These constants grow by repeated squaring, so e.g. $N_1 = 2$ and $N_{i+1} = N_i^2$. Clearly, $N_n = 2^{2^n}$. (Our actual construction uses slightly different N_i.)

We have not yet broached the topic of initialising these registers s.t. the necessary invariants hold. For our purposes, having a separate initialisation step is superfluous. Instead, we check whether the invariants hold in the initial configuration and restart (nondeterministically choosing a new initial configuration) if they do not.

5.2 Error detection

Our model provides only weak guarantees. In particular, we must deal with adversarial initialisation, meaning that the initial configuration can assign arbitrary values to any register. This is not limited to a designated set of initial registers; all registers used in the computation are affected.

Let us first discuss how the above construction behaves if its invariants are violated. As above, let $x, \overline{x}$ denote registers for which we want to keep the invariant $x + \overline{x} = k$, for some $k \in \mathbb{N}$. If instead $x + \overline{x} > k$, the "zero-check" described above is still guaranteed to terminate, as either $x > 0$ or $\overline{x} \geq k$ must hold. However, it might falsely return $x = 0$ when it is not. The procedure we use above, to combine two k-bounded counter to simulate a k^2-bounded counter, exhibits erratic behaviour under these circumstances. When we try to use it to count to k^2 we might instead only count to some lower value $k' < k^2$, even $k' \in \mathcal{O}(k)$.

If the invariant is violated in the other direction, i.e. $x + \overline{x} < k$ holds, we can never detect $x = 0$ and will instead run into an infinite loop.

The latter case is more problematic, as detecting it would require detecting absence. For the former, we can ensure that we check $x + \overline{x} \geq k + 1$ infinitely often; if $x + \overline{x} > k$, this check will eventually return true and we can initiate a restart. For the $x + \overline{x} > k$ case the crucial insight is that we cannot *detect* it, but we can *exclude* it: we issue a single check $x + \overline{x} \geq k$ in the beginning. If it fails, we restart immediately.

A simplified model. In the full construction, we have many levels of registers that rely on each other. Instead, we first consider a simplified model here to explain the main ideas.

In our simplified model there is only a single register x_i per level $i \in \{1, ..., n\}$ as well as one "level $n+1$" register R. For $i \in \{1, ..., n\}$ we are given subroutines $\textsc{Check}(x_i \geq N_i)$ and $\textsc{Check}(x_i > N_i)$ which we use to check thresholds; however, they are only guaranteed to work if $x_1 = N_1$, $x_2 = N_2$, ..., $x_{i-1} = N_{i-1}$ hold.

Our goal is to decide the threshold predicate $m \geq \sum_i N_i$, where $m := \sum_i x_i + \mathsf{R}$ is the sum of all registers. For each possible value of m we pick one initial configuration C_m and design our procedure s.t.

- every initial configuration different from C_m will cause a restart, and
- if started on C_m it is *possible* that the procedure enters a state where it cannot restart.

The structure of C_m is simple: we pick the largest i s.t. we can set $x_j := N_j$ for $j \leq i$ and put the remaining units into x_{i+1} (or R, if $i = n$). The procedure works as follows:

1. We nondeterministically guess $i \in \{0, ..., n\}$.
2. We run $\textsc{Check}(x_j \geq N_j)$ for all $j \in \{1, ..., i\}$. If one of these checks fails, we restart.
3. According to $i = n$ we set the output flag to true or false.
4. To verify that we are in C_m, we check the following infinitely often. For $j \in \{1, ..., i\}$ we run $\textsc{Check}(x_j > N_j)$ and restart if it succeeds. If $i < n$ we also restart if $\textsc{Check}(x_{i+1} \geq N_{i+1})$ or one of $x_{i+2}, ..., x_n, \mathsf{R}$ is nonempty.

Clearly, when started in C_m and i is guessed correctly, it is possible for step 2 to succeed, and it is impossible for step 4 to restart. If i is too large, step 2 cannot work, and if i is too small step 4 will detect $x_{i+1} \geq N_{i+1}$. So the procedure will restart until the right i is guessed and step 4 is reached.

Consider an initial configuration $C \neq C_m$, $|C| = m$. There are two cases: either there is a k with $C(x_k) < C_m(x_k)$, or some k has $C(x_k) > C_m(x_k)$. Pick a minimal such k.

In the former case, step 2 can only pass if $i < k$, but then one of $x_{i+2}, ..., x_n, \mathsf{R}$ is nonempty and step 4 will eventually restart.

The latter case is more problematic. Step 2 can pass regardless of i (for $i > k$ the precondition of $\textsc{Check}$ is not met). In step 4, either $i < k$ and then $x_{i+1} \geq N_{i+1}$ or one of $x_{i+2}, ..., x_n, \mathsf{R}$ is nonempty, or $i \geq k$ and one of the checks $\textsc{Check}(x_j > N_j)$ will eventually restart, for $j = k$.

This would be what we are looking for, but note that we implicitly made assumptions about the behaviour of $\textsc{Check}$ when called without its precondition being met. We need two things: all calls to $\textsc{Check}$ terminate and they do not change the values of any register. The second is the simpler one to deal with: later, we will have multiple registers per level and our procedures only need to move agents between registers of the same level. This keeps the sum of registers of one level constant, this weaker property suffices for correctness.

Ensuring that all calls terminate is more difficult. It runs into the problem discussed above, where a zero-check might not terminate if the invariant of its register is violated. In this simplified model it corresponds to the case $x_i < N_i$.

However, we note that $\textsc{Check}(x_i \geq N_i)$ and $\textsc{Check}(x_i > N_i)$ are only called if $(x_1, ..., x_{i-1}) \geq_{\text{lex}} (N_1, ..., N_{i-1})$, where $\geq_{\text{lex}}$ denotes lexicographical ordering. So if the precondition is violated, there must be a $j < i$ with $(x_1, ..., x_{j-1}) = (N_1, ..., N_{j-1})$ and $x_j > N_j$. This can be detected within the execution of $\textsc{Check}$ by calling itself recursively. In this manner, we can implement $\textsc{Check}$ in a way that avoids infinite loops as long as the weaker precondition $(x_1, ..., x_{i-1}) \geq_{\text{lex}} (N_1, ..., N_{i-1})$ holds.

Our actual construction follows the above closely; of course, instead of a single register per level we have four, making the necessary invariants more complicated. Additional issues arise when implementing Check, as registers cannot be detected erroneous while in use. Certain subroutines must hence take care to ensure termination, even when the registers they use are not working properly.

6 A Succinct Population Program

In this section, we construct a population program $\mathcal{P} = (Q, \mathrm{Proc})$ to prove the following:

▶ **Theorem 3.** *Let $n \in \mathbb{N}$. There exists a population program deciding $\varphi(x) \Leftrightarrow x \geq k$ with size $\mathcal{O}(n)$, for some $k \geq 2^{2^{n-1}}$.*

Full proofs and formal definitions of this section can be found in the full version of the paper [18].

We use registers $Q := Q_1 \cup ... \cup Q_n \cup \{\mathsf{R}\}$, where $Q_i := \{x_i, y_i, \overline{x}_i, \overline{y}_i\}$ are *level i* registers and R is a *level $n+1$* register. For convenience, we identify $\overline{\overline{x}}$ with x for any register x.

Types of Configurations. As explained in the previous section, x and $\overline{x}$ are supposed to sum to a constant N_i, for a level i register $x \in \{x_i, y_i\}$, which we define via $N_1 := 1$ and $N_{i+1} := (N_i + 1)^2$. If this invariant holds, we can use $x, \overline{x}$ to simulate a N_i-bounded register, which has value x.

We cannot guarantee that this invariant always holds, so our program must deal with configurations that deviate from this. For this purpose, we classify configurations based on which registers fulfil the invariant, and based on the type of deviation.

A configuration $C \in \mathbb{N}^Q$ is *i-proper*, if the invariant holds on levels $1, ..., i$, and their simulated registers have value 0. This is a precondition for most routines. Sometimes we relax the latter requirement on the level i registers; C is *weakly i-proper* if it is $(i-1)$-proper and the invariant holds on level i.

If C is $(i-1)$-proper and not i-proper, then there are essentially two possibilities. Either $C \leq C'$ for some i-proper C' and we call C *i-low*, or $C(x) \geq C'$ for a weakly i-proper C' and we call C *i-high*. Note that it is possible that C is neither i-low nor i-high – these configurations are easy to exclude and play only a minor role. We can mostly ensure that i-low configurations do not occur, but procedures must provide guarantees when run on i-high configurations.

Finally, we say that C is *i-empty* if all registers on levels $i, ..., n+1$ are empty.

	x_1	$\overline{x}_1$	y_1	$\overline{y}_1$	...	x_{i-1}	$\overline{x}_{i-1}$	y_{i-1}	$\overline{y}_{i-1}$	x_i	$\overline{x}_i$	y_i	$\overline{y}_i$	...
i-proper	0	N_1	0	N_1	...	0	N_{i-1}	0	N_{i-1}	0	N_i	0	N_i	...
weakly i-proper	0	N_1	0	N_1	...	0	N_{i-1}	0	N_{i-1}	3	$N_i - 3$	$N_i - 7$	7	...
i-low	0	N_1	0	N_1	...	0	N_{i-1}	0	N_{i-1}	0	$N_i - 3$	0	N_i	...
i-high	0	N_1	0	N_1	...	0	N_{i-1}	0	N_{i-1}	3	N_i	7	$N_i - 5$	...
i-empty	2	4	8	3	...	5	3	0	7	0	0	0	0	...

Figure 2 Example configurations exhibiting the different types.

Summary. We use the following procedures.

- Main. Computation starts by executing this procedure, and Main ultimately decides the predicate $\varphi(x) \Leftrightarrow x \geq 2\sum_{i=1}^{n} N_i$.

Algorithm AssertEmpty.

Parameter: $i \in \{1, ..., n+1\}$
Effect: If i-empty, do nothing, else it may restart

```
1: procedure AssertEmpty.(i) [i ≤ n]
2:     AssertEmpty(i + 1)
3:     for x ∈ Q_i do
4:         if detect x > 0 then
5:             restart
6: procedure AssertEmpty.(i) [i = n+1]
7:     if detect R > 0 then
8:         restart
```

Algorithm AssertProper.

Parameter: $i \in \{1, ..., n\}$
Effect: If i-proper or i-low, do nothing, else it may restart.

```
1: procedure AssertProper.(i)
2:     AssertProper(i − 1)
3:     for x ∈ {x_i, y_i} do
4:         if detect x > 0 then
5:             restart
6:         Large(x̄)
7:         if detect x > 0 then
8:             restart
```

Algorithm Zero Check whether a register is equal to 0.

Parameter: $x \in \{x_i, \overline{x}_i, y_i, \overline{y}_i\}$
Output: whether $x = 0$

```
1: procedure Zero(x)
2:     while true do
3:         AssertProper(i − 1)
4:         if detect x > 0 then
5:             return false
6:         if Large(x̄) then
7:             return true
```

Algorithm IncrPair Decrement a two-digit, base $\beta := N_i + 1$ register.

Parameter: $x \in \{x_i, \overline{x}_i\}$, $y \in \{y_i, \overline{y}_i\}$
Effect: $\beta x + y \pmod{\beta^2}$ decreases by 1

```
1: procedure IncrPair(x, y)
2:     if Zero(ȳ) then
3:         swap y, ȳ
4:         if Zero(x̄) then
5:             swap x, x̄
6:         else x̄ ↦ x
7:     else ȳ ↦ y
```

- AssertEmpty.. Check whether a configuration is i-empty and initiate a restart if not.
- AssertProper.. Check whether a configuration is i-proper or i-low, initiate a restart if not.
- Large. Nondeterministically check whether a register $x \in Q_i$ is at least N_i.
- Zero. Perform a deterministic zero-check on a register $x \in Q_i$.
- IncrPair. As described in Section 5.1, we use two level i registers (which are N_i bounded) to simulate an N_{i+1}-bounded register. This procedure implements the increment operation for the simulated register.

Procedures AssertEmpty., AssertProper.. The procedure AssertEmpty. is supposed to determine whether a configuration is i-empty, which can easily be done by checking whether the relevant registers are nonempty.

Similarly, AssertProper. is used to ensure that the current configuration is not i-high. If it is, it may initiate a restart. We remark that calls to AssertProper.(0) have no effect and can simply be omitted.

Procedure Zero. This procedure implements a deterministic zero-check, as long as the register configuration is weakly i-proper. To ensure termination, AssertProper. is called within the loop.

Algorithm Large Nondeterministically check whether a register is maximal.

Parameter: $x \in \{x_i, \overline{x}_i, y_i, \overline{y}_i\}, x \neq y$
Output: if $x \geq N_i$ return true and swap units of $x - N_i$ and $\overline{x}$; or return false

```
1:  procedure Large(x)   [for i = 1]
2:      if detect x > 0 then
3:          x ↦ x̄
4:          swap x, x̄
5:          return true
6:      else
7:          return false
8:  procedure Large(x)   [for i > 1]
9:      if ¬Zero(x_{i−1}) ∨ ¬Zero(y_{i−1}) then
10:         restart
11:     while true do
12:         CheckProper(i − 2)
13:         if detect x > 0 then
14:             x ↦ x̄
15:             IncrPair(x_{i−1}, y_{i−1})
16:             if Zero(x_{i−1}) ∧ Zero(y_{i−1}) then
17:                 swap x, x̄
18:                 return true
19:         else
20:             if Zero(x_{i−1}) ∧ Zero(y_{i−1}) then
21:                 return false
22:             if detect x̄ > 0 then
23:                 x̄ ↦ x
24:                 IncrPair(x̄_{i−1}, ȳ_{i−1})
```

Procedure IncrPair**.** This is a helper procedure to increment the "virtual", N_{i+1}-bounded counter simulated by x and y. It works by first incrementing the second digit, i.e. y. If an overflow occurs, x is incremented as well. It is also be used to decrement the counter, by running it on $\overline{x}$ and $\overline{y}$.

As we show later, IncrPair is "reversible" under only the weak assumption that the configuration $C \in \mathbb{N}^Q$ is i-high. More precisely, $C, \mathsf{IncrPair}(x, y) \to C'$ implies $C', \mathsf{IncrPair}(\overline{x}, \overline{y}) \to C$. Using this, we can show that Large, which calls IncrPair in a loop, terminates.

Procedure Large**.** This is the last of the subroutines, and the most involved one. The goal is to determine whether $x \geq N_i$, by using the registers of level $i - 1$ to simulate a "virtual" N_i-bounded register. To ensure termination, we use a "random" walk, which nondeterministically moves either up or down. More concretely, at each step either x is found nonempty, one unit is moved to $\overline{x}$ and the virtual register is incremented, or conversely $\overline{x}$ is nonempty, one unit moved to x, and the virtual register decremented. If the virtual register reaches 0 from above, Large had no effect and returns false. Once the virtual register overflows, a total of N_i units have been moved. These are put back into x by swapping x and $\overline{x}$ and true is returned.

As mentioned above, IncrPair is reversible even under weak assumptions. This ensures that the random walk terminates, as it can always retrace its prior steps to go back to its starting point.

Procedure Main**.** Finally, we put things together to arrive at the complete program. The implementation is very close to the steps described in Section 5.2 in the simplified model, but instead of guessing an i we iterate through the possibilities.

As mentioned before, Main considers a small set of initial configurations "good" and may stabilise. The following lemma formalises this.

▶ **Lemma 4.** Main*, run on register configuration $C \in \mathbb{N}^Q$, can only restart or stabilise, and*
(a) *it may stabilise to* false *if C is j-low and $(j+1)$-empty, for some $j \in \{1, ..., n\}$,*
(b) *it may stabilise to* true *if C is n-proper, and*
(c) *it always restarts otherwise.*

Algorithm Main Decide whether there are at least $2\sum_i N_i$ agents.

```
1: procedure Main
2:     OF := false
3:     for i = 1, ..., n do
4:         while ¬Large(x̄_i) ∨ ¬Large(ȳ_i) do
5:             AssertProper(i)
6:             AssertEmpty(i + 1)
7:     OF := true
8:     while true do
9:         AssertProper(n)
```

7 Converting Population Programs into Protocols

In the previous section we constructed succinct population programs for the threshold predicate. We now justify our model and prove that we can convert population programs into population protocols, keeping the number of states low. We do this in two steps; first we introduce population machines, which are a low-level representation of population programs, then we convert these into population protocols. This results in the following theorem:

▶ **Theorem 5.** *If a population program deciding φ with size n exists, then there is a population protocol deciding $\varphi'(x) \Leftrightarrow \varphi(x-i) \land x \geq i$ with $\mathcal{O}(n)$ states, for an $i \in \mathcal{O}(n)$.*

Population machines are introduced in Section 7.1, they serve to provide a simplified model. Converting population programs into machines is straightforward and uses standard techniques, similar to how one would convert a structured program to use only goto-statements. We will describe this in Section 7.2. The conversion to population protocols is finally described in Section 7.3. Here, we only highlight the key ideas of the conversion. The details can be found in the full version of the paper [18].

7.1 Formal Model

▶ **Definition 6.** *A* population machine *is a tuple $\mathcal{A} = (Q, F, \mathcal{F}, \mathcal{I})$, where Q is a finite set of* registers*, F a finite set of* pointers*, $\mathcal{F} = (\mathcal{F}_i)_{i \in F}$ a list of* pointer domains*, each of which is a nonempty finite set, and $\mathcal{I} = (\mathcal{I}_1, ..., \mathcal{I}_L)$ is a sequence of* instructions*, with $L \in \mathbb{N}$. Additionally, $OF, CF, IP \in F$, $\mathcal{F}_{OF} = \mathcal{F}_{CF} = \{\mathsf{false}, \mathsf{true}\}$ and $\mathcal{F}_{IP} = \{1, .., L\}$. For $x \in Q \cup \{\square\}$ we also require $V_x \in F$, and $x \in \mathcal{F}_{V_x} \subseteq Q$. The* size *of $\mathcal{A}$ is $|Q| + |F| + \sum_{X \in F} |\mathcal{F}_X| + |\mathcal{I}|$.*

Let $x, y \in Q$, $x \neq y$, $X, Y \in F$, $i \in \{1, ..., L\}$ and $f : \mathcal{F}_Y \to \mathcal{F}_X$. There are three types of instructions: $\mathcal{I}_i = (x \mapsto y)$, $\mathcal{I}_i = (\textbf{detect } x > 0)$, or $\mathcal{I}_i = (X := f(Y))$.

A population machine has a number of registers, as usual, and a number of pointers. While each register can take any value in $\mathbb{N}$, a pointer is associated with a finite set of values it may assume. There are three special pointers: the output flag OF, which we have already seen in population programs and is used to indicate the result of the computation, the condition flag CF used to implement branches, and the instruction pointer IP, storing the index of the next instruction to execute. To implement swap instructions we use a register map; the pointer V_x, for a register $x \in Q$, stores the register x is actually referring to. ($V_\square$ is a temporary pointer for swapping.) The model allows for arbitrary additional pointers, we will use a one per procedure to store the return address.

There are only three kinds of instructions: $(x \mapsto y)$ and $(\textbf{detect } x > 0)$ are present in population programs as well and have the same meaning here. (With the slight caveat that x and y are first transformed according to the register map. The instructions do not

operate on the actual registers x, y, but on the registers pointed to by V_x and V_y.) The third, $(X := f(Y))$ is a general-purpose instruction for pointers. It can change IP and will be used to implement control flow constructs.

A precise definition of the semantics can be found in the full version of the paper [18].

7.2 From Population Programs to Machines

Population machines do not have high-level constructs such as loops or procedures, but these can be implemented as macros using standard techniques. We show only an example here, a detailed description of the conversion can be found in the full version of the paper [18].

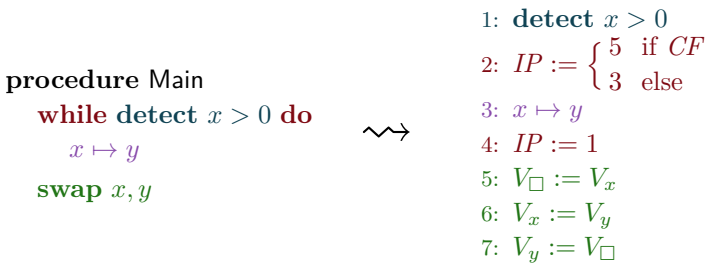


Figure 3 Conversion to a population machine.

Control-flow, i.e. **if**, **while** and procedure calls are implemented via direct assignment to IP, the instruction pointer, as in lines 2 and 4 above. The statements (**detect** $x > 0$) and $(x \mapsto y)$ are translated one-to-one, but note that in the population machine their operands are first translated via the register map. For example, (**detect** $x > 0$) in line 1 checks whether the register pointed to by V_x is nonzero. Correspondingly, **swap** statements result in direct modifications to the register map: lines 5-7 swap the pointers V_x and V_y (and leave the registers they point to unchanged).

7.3 Conversion to Population Protocols

In this section, we only present a simplified version of our construction. In particular, we make use of multiway transitions to have more than two agents interact at a time. Our actual construction, described in the full version of the paper [18], avoids them and the associated overhead.

Let $\mathcal{A} = (Q, F, \mathcal{F}, \mathcal{I})$ denote a population machine. To convert this into a population protocol, we use two types of agents: *register agents* to store the values of the registers, and *pointer agents* to store the pointers. For a register we have many identical agents, and the value of the register corresponds to the total number of those agents. They use states Q. For each pointer we use a unique agent, storing the value of the pointer in its state; they use states $\{X^v : X \in F, v \in \mathcal{F}_X\}$.

Let $X_1, ..., X_{|F|}$ denote some enumeration of F with $X_{|F|} = IP$, and let v_i denote the initial value of X_i. We use X_1 as initial state of the protocol. To goal is to have a unique agent for each pointer, so we implement a simple leader election. We use $*$ as wildcard.

$$X_i^*, X_i^* \mapsto X_i^{v_i}, X_{i+1}^{v_{i+1}} \qquad IP^*, IP^* \mapsto X_1^{v_1}, x$$

with $i \in \{1, ..., |F| - 1\}$. If two agents store the value of a single pointer, they eventually meet and one of them is moved to another state. When this happens, the computation is

restarted – but note that the values of the registers are not reset. Eventually, the protocol will thus reach a configuration with exactly one agent in $X_i^{v_i}$, for each i, and the remaining agents in Q.

Starting from this configuration, the instructions can be executed. We illustrate the mapping from instructions to transitions in the following example:

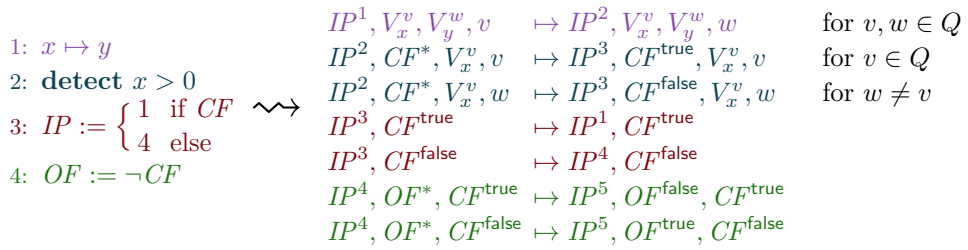


Figure 4 Converting instructions into transitions.

For example, in line 1 we want to move one agent from x to y and set the instruction pointer to 2 (from 1). Recall that the registers map to states of the population protocol via the register map, stored in pointers V_x, where $x \in Q$ is a register. We thus have the following agents initiating the transition:

- IP^1; the agents storing the instruction pointer currently stores the value 1,
- V_x^v; the register $x \in Q$ is currently mapped to state $v \in Q$,
- v; an agent in state v, i.e. representing one unit in register x,
- V_y^w; register y is mapped to state w.

The transition then moves v to state w, and increments the instruction pointer.

The above protocol does not come to a consensus. For this to happen, we use a standard output broadcast: we add a single bit to all states. In this bit an agent stores its current opinion. When any agent meets the pointer agent of the output flag OF, the former will assume the opinion of the latter. Eventually, the value of the output flag has stabilised and will propagate throughout the entire population, at which point a consensus has formed.

8 Robustness of Threshold Protocols

A major motivation behind the construction of succinct protocols for threshold predicates is the application to chemical reactions. In this, as in other environments, computations must be able to deal with errors. Prior research has considered *self-stabilising* protocols [8, 16, 15]. Such a protocol must converge to a desired output regardless of the input configuration. However, it is easy to see that no population protocol for e.g. a threshold predicate can be self-stabilising (and prior research has thus focused on investigating extensions of the population protocol model).

In our definition of population programs, the program cannot rely on any guarantees about its input configuration, so they are self-stabilising by definition. However, when we convert to population protocols, we retain only a slightly weaker property, defined as follows:

▶ **Definition 7.** *Let $PP = (Q, \delta, I, O)$ denote a population protocol deciding φ with $|I| = 1$. We say that PP is* almost self-stabilising*, if every fair run starting at a configuration $C \in \mathbb{N}^Q$ with $C(I) \geq |Q|$ stabilises to $\varphi(|C|)$.*

So the initial configuration can be almost arbitrary, but it must contain a small number of agents in the initial state. In many contexts, this is a mild restriction. In a chemical reaction, for example, the number of agents (i.e. the number of molecules) is many orders of magnitude larger than the number of states (i.e. the number of species of molecules).

In particular, this is also much stronger than any prior construction. All known protocols for threshold predicates are 1-aware [14], and can thus be made to accept by placing a single agent in an accepting state.

▶ **Theorem 2.** *The protocols of Theorem 1 are almost self-stabilising.*

9 Conclusions

We have shown an $\mathcal{O}(\log \log n)$ upper bound on the state complexity of threshold predicates for leaderless population protocols, closing the last remaining gap. Our result is based on a new model, population programs, which enable the specification of leaderless population protocols using structured programs.

As defined, our model of population programs can only decide unary predicates and it seems impossible to decide even quite simple remainder predicates (e.g. "is the total number of agents even"). Is this a fundamental limitation, or simply a shortcoming of our specific choices? We tend towards the latter, and hope that other very succinct constructions for leaderless population protocols can make use of a similar approach.

Our construction is almost self-stabilising, which shows that it is possible to construct protocols that are quite robust against *addition* of agents in arbitrary states. A natural next step would be to investigate the *removal* of agents: can a protocol provide guarantees in the case that a small number of agents disappear during the computation?

Threshold predicates can be considered the most important family for the study of space complexity, as they are the simplest way of encoding a number into the protocol. The precise space complexity of other classes of predicates, however, is still mostly open. The existing results generalise somewhat; the construction presented in this paper, for example, can also be used to decide $\varphi(x) \Leftrightarrow x = k$ for $k \geq 2^{2^n}$ with $\mathcal{O}(n)$ states. As mentioned, there also exist succinct constructions for arbitrary predicates, but – to the extent of our knowledge – it is still open whether, for example, $\varphi(x) \Leftrightarrow x = 0 \pmod{k}$ can be decided for $k \geq 2^{2^n}$, both with and without leaders.

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2560–2579. SIAM, 2017. doi:10.1137/1.9781611974782.169.

2 Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2221–2239. SIAM, 2018. doi:10.1137/1.9781611975031.144.

3 Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. Fast and exact majority in population protocols. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 47–56. ACM, 2015. doi:10.1145/2767386.2767429.

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

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. doi:10.1007/S00446-005-0138-3.

6 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Comput.*, 21(3):183–199, 2008. doi:10.1007/S00446-008-0067-Z.

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

8 Dana Angluin, James Aspnes, Michael J. Fischer, and Hong Jiang. Self-stabilizing population protocols. In James H. Anderson, Giuseppe Prencipe, and Roger Wattenhofer, editors, *Principles of Distributed Systems, 9th International Conference, OPODIS 2005, Pisa, Italy, December 12-14, 2005, Revised Selected Papers*, volume 3974 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2005. doi:10.1007/11795490_10.

9 Stav Ben-Nun, Tsvi Kopelowitz, Matan Kraus, and Ely Porat. An $O(\log^{3/2} n)$ parallel time population protocol for majority with $O(\log n)$ states. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 191–199. ACM, 2020. doi:10.1145/3382734.3405747.

10 Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. A population protocol for exact majority with o(log5/3 n) stabilization time and theta(log n) states. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.10.

11 Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. Time-space trade-offs in population protocols for the majority problem. *Distributed Comput.*, 34(2):91–111, 2021. doi:10.1007/s00446-020-00385-0.

12 Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. Population protocols for leader election and exact majority with o(logˆ2 n) states and o(logˆ2 n) convergence time. *CoRR*, abs/1705.01146, 2017. arXiv:1705.01146.

13 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. doi:10.4230/LIPICS.STACS.2020.40.

14 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. doi:10.4230/LIPICS.STACS.2018.16.

15 Janna Burman, Ho-Lin Chen, Hsueh-Ping Chen, David Doty, Thomas Nowak, Eric E. Severson, and Chuan Xu. Time-optimal self-stabilizing leader election in population protocols. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 33–44. ACM, 2021. doi:10.1145/3465084.3467898.

16 Shukai Cai, Taisuke Izumi, and Koichi Wada. How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model. *Theory Comput. Syst.*, 50(3):433–445, 2012. doi:10.1007/s00224-011-9313-z.

17 Ho-Lin Chen, Rachel Cummings, David Doty, and David Soloveichik. Speed faults in computation by chemical reaction networks. *Distributed Comput.*, 30(5):373–390, 2017. doi:10.1007/s00446-015-0255-6.

18 Philipp Czerner. Breaking through the $\omega(n)$-space barrier: Population protocols decide double-exponential thresholds, 2024. arXiv:2204.02115.

19 Philipp Czerner and Javier Esparza. Lower bounds on the state complexity of population protocols. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 45–54. ACM, 2021. doi:10.1145/3465084.3467912.

20 Philipp Czerner, Javier Esparza, and Jérôme Leroux. Lower bounds on the state complexity of population protocols. *CoRR*, 2021. doi:10.48550/arXiv.2102.11619.

21 David Doty, Mahsa Eftekhari, Leszek Gasieniec, Eric E. Severson, Przemyslaw Uznanski, and Grzegorz Stachowiak. A time and space optimal stable population protocol solving exact majority. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1044–1055. IEEE, 2021. doi:10.1109/FOCS52979.2021.00104.

22 Moez Draief and Milan Vojnovic. Convergence speed of binary interval consensus. *SIAM J. Control. Optim.*, 50(3):1087–1109, 2012. doi:10.1137/110823018.

23 Javier Esparza. Decidability and complexity of petri net problems - an introduction. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer, 1996. doi:10.1007/3-540-65306-6_20.

24 Jérôme Leroux. State complexity of protocols with leaders. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 257–264. ACM, 2022. doi:10.1145/3519270.3538421.

25 Richard J. Lipton. The reachability problem requires exponential space. Technical report, Yale University, Dept. of CS, 1976. URL: http://www.cs.yale.edu/publications/techreports/tr63.pdf.

26 Yves Mocquard, Emmanuelle Anceaume, James Aspnes, Yann Busnel, and Bruno Sericola. Counting with population protocols. In D. R. Avresky and Yann Busnel, editors, *14th IEEE International Symposium on Network Computing and Applications, NCA 2015, Cambridge, MA, USA, September 28-30, 2015*, pages 35–42. IEEE Computer Society, 2015. doi:10.1109/NCA.2015.35.

27 Yves Mocquard, Emmanuelle Anceaume, and Bruno Sericola. Optimal proportion computation with population protocols. In Alessandro Pellegrini, Aris Gkoulalas-Divanis, Pierangelo di Sanzo, and Dimiter R. Avresky, editors, *15th IEEE International Symposium on Network Computing and Applications, NCA 2016, Cambridge, Boston, MA, USA, October 31 - November 2, 2016*, pages 216–223. IEEE Computer Society, 2016. doi:10.1109/NCA.2016.7778621.

28 Etienne Perron, Dinkar Vasudevan, and Milan Vojnovic. Using three states for binary consensus on complete graphs. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*, pages 2527–2535. IEEE, 2009. doi:10.1109/INFCOM.2009.5062181.

C. Fast and Succinct Population Protocols for Presburger Arithmetic

Title	Fast and Succinct Population Protocols for Presburger Arithmetic
Authors	Philipp Czerner, Roland Guttenberg, Martin Helfrich, Javier Esparza
Venue	Journal of Computer and System Sciences, 2024
Publisher	Elsevier
DOI	10.1016/j.jcss.2023.103481

Journal of Computer and System Sciences 140 (2024) 103481

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

journal homepage: www.elsevier.com/locate/jcss

Fast and succinct population protocols for Presburger arithmetic ☆

Philipp Czerner *, Roland Guttenberg *, Martin Helfrich, Javier Esparza

Department of Computer Science, Technical University of Munich, Germany

ARTICLE INFO

Article history:
Received 10 September 2022
Received in revised form 11 September 2023
Accepted 18 September 2023
Available online 29 September 2023

Keywords:
Population protocols
Fast
Succinct
Population computers

ABSTRACT

In their 2006 seminal paper in Distributed Computing, Angluin et al. present a construction that, given any Presburger predicate, outputs a leaderless population protocol that decides the predicate. The protocol for a predicate of size m 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 is exponential in m. Blondin et al. presented at 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 generalization of population protocols, and show that our computers for Presburger predicates can be translated into fast and succinct population protocols.

1. Introduction

Population protocols 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 [5]. 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.

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 $\vec{a} \cdot \vec{x} \equiv_\theta c$, where $\vec{a}$ is an integer vector, c and θ are integers, and $\equiv_\theta$ denotes congruence modulo θ [21]. 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.

☆ **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).

* Corresponding authors.
E-mail addresses: czerner@in.tum.de (P. Czerner), guttenbe@in.tum.de (R. Guttenberg), helfrich@in.tum.de (M. Helfrich), esparza@in.tum.de (J. Esparza).
URLs: https://nicze.de/philipp (P. Czerner), https://rolandguttenberg.de (R. Guttenberg), https://martinhelfrich.de (M. Helfrich), https://www7.in.tum.de/~esparza (J. Esparza).

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

https://doi.org/10.1016/j.jcss.2023.103481
0022-0000/

Two fundamental parameters of a protocol are the expected number of interactions until a stable consensus is reached (that is, a consensus that cannot be broken by any possible continuation of the execution that leads to it), and the number of states of each agent. The expected number of interactions divided by the number of agents, also called the parallel stabilisation 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 many natural computing applications, where a state corresponds to a chemical species, it is difficult to implement protocols with many states (see [15] for a particularly simple example and, in general, the literature on programming chemical reaction networks, a model very close to population protocols [24]).

Given a formula φ of QFPA, let m be the number of bits needed to write φ with coefficients in binary, and let n be the number of agents participating in the protocol. The construction of [5] yields a protocol with $\mathcal{O}(m \cdot n^2 \log n)$ 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 $\Omega(2^m)$. This is well beyond the only known lower bound, showing that for every construction there exists an infinite subset of predicates φ for which the construction produces protocols with $\Omega(m^{1/4})$ states [12]. So the construction of [5] produces *fast* but *very large* protocols. The same happens with the constructions of [6], where protocols simulate register machines and the state of an agent stores (among other information) one bit for every register, and with those of [22], which embed the other two.

In [12,11] Blondin et al. exhibit a construction that produces *succinct* protocols, that is, protocols with $\mathcal{O}(\text{poly}(m))$ 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 [12,11] 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 of size m a protocol with $\mathcal{O}(\text{poly}(m))$ states and stabilizing after $\mathcal{O}(m^7 \cdot n^2)$ expected interactions. So our construction achieves optimal parallel stabilisation time in n, and, at the same time, yields protocols that are as succinct as the construction of [11]. Moreover, for inputs of size $\Omega(m)$ (a very mild constraint when agents are molecules), the protocols have $\mathcal{O}(m)$ states.

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 have *k-way interactions* between more than two agents (but these are limited to involve at most two types of 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*. The construction proceeds in three steps. First, we exhibit succinct population computers for all Presburger predicates in which every run is finite, that is, every run reaches a configuration at which no transition can occur. We call these computers *bounded*. In a second step we prove a very general conversion theorem: *Any* bounded computer of size m can be translated into a population protocol with $\mathcal{O}(m^2)$ states and stabilizing in $2^{\mathcal{O}(m^2 \log m)} \cdot n^3$ expected interactions.

An important ingredient of the proof is a novel simulation of interactions between an arbitrary number of agents (common in chemical reaction networks) by binary interactions. In previous work this simulation required to introduce "reverse" interactions that "undo" the effect of others, which led to a large slowdown [11]. We limit ourselves to interactions between two types of agents (but an arbitrary number), and provide a new simulation that avoids the use of "reverse" interactions.

Finally, we exploit that our computers for Presburger predicates are not only bounded, but satisfy an additional property, called *rapidness*, to improve the stabilisation time bound to $\mathcal{O}(m^7 \cdot n^2)$ interactions.

Related work. Our results are for the canonical population protocol model of [4,5], where (a) agents have a constant number of states, independent of the size of the population; (b) there are no leader agents; (c) time complexity is measured in terms of the expected number of interactions until stabilisation; and (d) protocols decide the predicate with probability 1 for all inputs. As mentioned above, in this model every protocol for the majority predicate needs $\Omega(n^2)$ expected interactions [1], i.e. $\Omega(n)$ parallel stabilisation time, and so our construction is optimal. However, there is a vast body of work concerning variants of the model in which one or more of (a)-(d) are relaxed in order to find faster protocols for specific tasks. We briefly discuss some of this work.

If condition (a) is relaxed, then protocols running in $\mathcal{O}(\text{polylog}(n))$ instead of $\Omega(n)$ parallel stabilisation time have been proposed for specific tasks like majority or leader election. In the first such protocol, presented in [3], the number of states still grew very rapidly in the number of agents. Much subsequent work led to protocols where the number of states grows much more slowly; for example, in 2018 two surveys were published devoted only to this question [2,18]. An asymptotically optimal protocol for majority with $\Theta(\log n)$ states and expected $\Theta(n \log n)$ interactions was given in [17], and an optimal

[2] See the related work section for other results when these assumptions are given up.

[3] Other output conventions for population protocols have been considered, see e.g. [14].

protocol for leader election with $\Theta(\log\log n)$ states per agent and expected $\Theta(n \log n)$ interactions was presented in [10]. However, the properties of the model in which the number of states can grow in the size of the population are very different from the canonical one. In particular, the decision power of the model may go beyond Presburger arithmetic, depending on the rate at which the number of states is allowed to grow with the size of the population, on whether the agents "know" an upper bound on the size of the population or not, etc. (For example, if agents know an upper bound on the size of the population and have enough memory to implement a counter that can count up to that number, then they can elect a leader, and then let the leader simulate a broadcast population protocol with only one broadcasting agent [13]. To simulate a broadcast, the leader counts the number of agents it interacts with until the bound is reached. These protocols can compute all predicates lying in the complexity class NL, which properly contains Presburger arithmetic.) To the best of our knowledge, the expressive power of many variants is not even known, and so the question of generic constructions yielding a protocol from a specification of the predicate cannot even be formulated. Our results might be used to produce succinct protocols for Presburger predicates in some of these variants, but this question is beyond the scope of this paper.

Angluin et al. consider in [6] a model that relaxes (b) by allowing one leader agent, (c) by measuring time in terms of the number of interactions until *convergence* (loosely speaking, an execution converges after t interactions if all configurations reached during the execution after t steps exhibit the same consensus, observe that at that point t it might still be theoretically possible to reach non-consensus configurations), and (d) by allowing a small probability of error. They exhibit a construction that, given any Presburger predicate, produces a protocol running in $\mathcal{O}(\text{polylog}(n))$ parallel convergence time. Further, they show that zero probability of error can be achieved by suitably combining a fast protocol with small probability of error and a slow but exact *backup protocol* (a technique later used in other works, like [9]). Our work provides the first *succinct* backup protocol. In future work we plan to investigate if there also exist succinct protocols running in $\mathcal{O}(\text{polylog}(n))$ parallel convergence time, with small probability of error.

Kosowski and Uznański improve the construction of [6] by showing that $\mathcal{O}(\text{polylog}(n))$ parallel convergence time can also be achieved without a leader, i.e. by relaxing only (c) and (d) [22]. Further, using the exact protocols of [6], they provide protocols that run in $\mathcal{O}(n^\epsilon)$ parallel convergence time for arbitrary ϵ (i.e. only (c) is relaxed). Again, an interesting question for future work is whether these constructions can be made succinct, and, again, our results can be seen as a first step that exhibits a succinct backup protocol.

Organization of the paper. We give preliminary definitions in Section 2 and introduce population computers in Section 3. Section 4 describes why previous constructions were either not succinct or slow. Section 5 gives an overview of the rest of the paper and summarises our main results. Section 6 describes bounded population computers for every Presburger predicate. Section 7 shows that every bounded computer can be converted into a succinct population protocol. Section 8 shows that the protocols obtained for the bounded computers of Section 6 are not only succinct but also fast.

2. Preliminaries

Multisets. Let E be a finite set. A multiset over E is a mapping $E \to \mathbb{N}$, 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 $u \geq v$ (i.e. $u(e) \geq v(e)$ for all $e \in E$).

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 $v \to_t v'$. We call $v \to_t v'$ a *step*.

Let $\delta \subseteq \mathbb{N}^E \times \mathbb{N}^E$ denote a finite set of transitions. The following definitions depend on δ. In the paper it will always be clear from context which δ is meant, hence we leave this dependence implicit. The multiset v is *terminal* if it does not enable any transition $t \in \delta$. An *execution* is a finite or infinite sequence $v_0, v_1, \ldots$ of multisets such that $v \to_{t_1} v_1 \to_{t_2} \cdots$ for some sequence $t_1, t_2, \ldots \in \delta$ of transitions. A multiset v' is *reachable* from v if there is an execution $v_0, v_1, \ldots, v_k$ 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 $v_0, v_1, \ldots$ 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 $j \geq 0$.

Presburger arithmetic and QFPA. Presburger arithmetic is the first-order theory of addition [21]. A formula $\varphi(x_1, \ldots, x_v)$ of Presburger arithmetic with $x_1, \ldots, x_v$ as free variables induces a predicate $P_\varphi : \mathbb{N}^v \to \{0, 1\}$ defined by: $P_\varphi(n_1, \ldots, n_v) = 1$ iff $\varphi(n_1, \ldots, n_v)$ is true. A predicate $\mathbb{N}^v \to \{0, 1\}$ is *definable in Presburger arithmetic* or just a *Presburger predicate* if it is induced by some formula of Presburger arithmetic. Presburger predicates are known to be the same as the *semilinear* predicates [21]. Using the quantifier elimination procedure of Presburger arithmetic, one can show that every Presburger predicate is equivalent to a boolean combination of threshold and remainder predicates, defined as follows: A predicate $\varphi : \mathbb{N}^v \to \{0, 1\}$ is a *threshold predicate* if $\varphi(x_1, \ldots, x_v) = \left(\sum_{i=1}^v a_i x_i \geq c\right)$, where $a_1, \ldots, a_v, c \in \mathbb{Z}$, and a *remainder predicate* if $\varphi(x_1, \ldots, x_v) = \left(\sum_{i=1}^v a_i x_i \equiv_\theta c\right)$, where $a_1, \ldots, a_v \in \mathbb{Z}$, $\theta \geq 1$, $c \in \{0, \ldots, \theta - 1\}$, and $a \equiv_\theta b$ denotes that a is congruent to b modulo θ [21].

We call the set of boolean combinations of threshold and remainder predicates *quantifier-free Presburger arithmetic*, or QFPA. We define the *size* of a Presburger predicate as the number of bits of a shortest formula of QFPA representing it, with coefficients written in binary. As mentioned in the introduction, Angluin et al. showed that population protocols can decide exactly the Presburger predicates, or, by the above, the predicates definable in QFPA [7].

3. Population computers

Population computers are a generalization of population protocols. They allow us to give very concise descriptions of 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 finite set of multiset rewriting transitions $r \mapsto s$ over Q such that $|r| = |s| \geq 2$ and $|\mathrm{supp}(r)| \leq 2$. Further, we require that δ is a partial function, i.e. if $(r \mapsto s_1), (r \mapsto s_2) \in \delta$ then $s_1 = s_2$. The *arity* of a transition $r \mapsto s$ is the size of the multiset r (or s). A transition is *binary* if it has arity two. A population computer is *binary* if all its transitions are binary.
- $I \subseteq Q$ is a set of *input states*. An *input* is a configuration C such that $\mathrm{supp}(C) \subseteq I$.
- $O : 2^Q \to \{0, 1, \bot\}$ is an *output function*. The *output* of a configuration C is $O(\mathrm{supp}(C))$. An output function O is a *consensus output function* 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 $O(Q') = \bot$ otherwise, for all $Q' \subseteq Q$.
- $H \in \mathbb{N}^{Q \setminus I}$ is a multiset of *helper agents* or just *helpers*. A *helper configuration* is a configuration C such that $\mathrm{supp}(C) \subseteq \mathrm{supp}(H)$ and $C \geq H$.

In particular, note that δ is restricted in that a transition can only involve two types of agents. For example, $\{\!|p, p, q|\!\} \mapsto \{\!|p, q, o|\!\}$ (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.

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 to execute a run 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 $C_0 C_1 \ldots$ *stabilises to* b if there exists an $i \geq 0$ such that $O(\mathrm{supp}(C_i)) = b$ and C_i only reaches configurations C' with $O(\mathrm{supp}(C')) = b$. 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 $\varphi : \mathbb{N}^I \to \{0, 1\}$ if every input C_I has output $\varphi(C_I)$. Observe that, crucially, the protocol has to work for *any* helper configuration C_H.

Terminating and bounded computers. A population computer is *bounded* if no run starting at any initial configuration C is infinite, and *terminating* if no fair run starting at C is infinite, i.e. every fair run ends at a terminal configuration.[4]

Example 1. Consider a population computer with states $\{q, p\}$, input state q, and transitions $t = \{\!|2 \cdot q|\!\} \mapsto \{\!|2 \cdot q|\!\}$ and $u = \{\!|2 \cdot q|\!\} \mapsto \{\!|2 \cdot p|\!\}$. The computer is not bounded, because, for example, there is an infinite run from the initial configuration $C = \{\!|2 \cdot q|\!\}$, namely $C\,C\,C\,C\ldots$. However, it is terminating because every fair run eventually reaches a terminal configuration of the form $\{\!|n \cdot p|\!\}$ or $\{\!|q, n \cdot p|\!\}$ for some $n \geq 0$.

Graphical notation. We visualise population computers as Petri nets (see e.g. the left part of Fig. 1 in page 10). Places (circles) and transitions (squares) represent respectively states and transitions. The number of agents currently occupying a state is written within the place representing the state. For example, the computer on the left of Fig. 1 has states 0,1,2,4,8,16 and $\{\!|4 \cdot 16|\!\} \mapsto \{\!|2 \cdot 0, 1, 8|\!\}$ is one of its transitions. Currently, there are 12 agents in state 0 and no agents elsewhere.

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 $\mathrm{size}(O)$ gates. For every transition t let $|t|$ be the arity of t. The *size* of $\mathcal{P}$ is $\mathrm{size}(\mathcal{P}) := |Q| + |H| + \mathrm{size}(O) + \sum_{t \in \delta} |t|$. If $\mathcal{P}$ is binary, then (as for population protocols) we do not count the arities and define the *adjusted size* $\mathrm{size}_2(\mathcal{P}) := |Q| + |H| + \mathrm{size}(O)$. For a binary computer $\mathcal{P}$ we have $\sum_{t \in \delta} |t| = 2|\delta| \leq 2|Q|^2$, and so in particular $\mathrm{size}(\mathcal{P}) \leq 2\,\mathrm{size}_2(\mathcal{P})^2$. Observe that both the arity of a transition $r \mapsto s$ and the size of the helper multiset H are defined as the number of elements of the multisets r and H, respectively, and not as the number of bits of these numbers. In other words, we consider their size in unary. This makes our result about the existence of succinct population computers stronger.

Population protocols and speed of a protocol. 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].

[4] This is the classical notion of termination under fairness in concurrent systems [19]. It differs from recent notions of termination in the literature on population protocols, e.g. the one of [16].

The speed of a binary population computer without helpers, and so in particular of a population protocol, is defined as follows. We assume a probabilistic execution model for a population protocol $\mathcal{P}$ in which at a configuration C two agents are picked uniformly at random and execute a transition, if possible, moving to a configuration C' (by assumption two agents enable at most one transition). This is called an *interaction*. Repeated occurrences of interactions, starting from an input C_0, produce an *execution* $C_0\, C_1\, C_2 \ldots$. An execution *stabilises at time* t if every configuration C reachable from C_t satisfies $O(\mathrm{supp}(C)) = O(\mathrm{supp}(C_t))$, and *converges* after t interactions if $C_{t'}$ satisfies $O(\mathrm{supp}(C'_t)) = O(\mathrm{supp}(C_t))$ for every $t' \geq t$. Let $\mathcal{P}$ be a protocol that decides a given predicate φ. Given an input C_I, we say that $\mathcal{P}$ decides $\varphi(C_I)$ *within* k *interactions* if the expected value of the earliest stabilisation time of the executions starting at C_I is at most k. (The earliest stabilisation time is the random variable that assigns to each execution the smallest time at which it stabilises.) Let $T : \mathbb{N} \to \mathbb{N}$. We say that $\mathcal{P}$ decides φ *within* T *interactions* if it decides $\varphi(C_I)$ within $T(n)$ interactions for every $n \geq 0$ and for every input C_I of size n. See e.g. [6] for more details. Notice that in this paper we study the stabilisation time, and not the convergence time [22].

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.
- They use a multiset H of auxiliary helper agents, but the addition of more helpers must 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, only the multiset H. Since, by definition, the initial configurations contain *at least* H helpers but possibly more, the agents only know a lower bound.
- They have a more flexible output condition. A population protocol accepts or rejects by moving all agents to accepting or rejecting states, respectively. In contrast, population computers look at the states that are present in the current configuration, and then choose an output based on that set.

Fast and succinct population protocols. As announced in the introduction, the goal of this paper is to show that every Presburger predicate has a *fast* and *succinct* population protocol. We formalise these notions.

Definition 2. Let $\mathcal{F}$ be a function that assigns to every predicate $\varphi \in \mathit{QFPA}$ a population protocol $\mathcal{F}(\varphi)$ deciding φ.

- $\mathcal{F}$ produces *succinct* protocols if there exists a constant $a \geq 0$ such that for every $\varphi \in \mathit{QFPA}$ the protocol $\mathcal{F}(\varphi)$ has $\mathcal{O}(|\varphi|^a)$ states.
- $\mathcal{F}$ produces *fixed-parameter fast* protocols if there exists a function $f : \mathbb{N} \to \mathbb{N}$ and a constant $b \geq 0$ such that for every $\varphi \in \mathit{QFPA}$ the protocol $\mathcal{F}(\varphi)$ decides φ within $\mathcal{O}(f(|\varphi|) \cdot n^b)$ interactions.
- $\mathcal{F}$ produces *fast* protocols if there exist constants $a, b \geq 0$ such that for every $\varphi \in \mathit{QFPA}$ the protocol $\mathcal{F}(\varphi)$ decides φ within $\mathcal{O}(|\varphi|^a \cdot n^b)$ interactions.

We call an effectively computable function $\mathcal{F}$ a *construction* or a *procedure*. The formalization of "every Presburger predicate has a fast and succinct protocol" is "there exists a construction that produces fast and succinct protocols". The next section explains why none of the constructions in the literature produces fast and succinct protocols. The rest of the paper describes a new construction that produces fast and succinct protocols.

4. Previous constructions: Angluin et al. and Blondin et al.

We show by means of some examples that the construction by Angluin et al. [5] does not produce succinct protocols, and the construction of Blondin et al. [11] does not produce fast protocols, not even fixed-parameter fast.

Example 3. 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 4. We give a protocol for $\varphi = (x - y \geq 2^d)$ with a polynomial number of states, very similar to the protocol of [11]. The protocol is defined in two steps. First, we remove states and transitions from the protocol of Example 3, retaining only the states (ℓ, b, u) such that u is a power of 2, and some of the transitions involving these states:

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

This 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 [11] this is solved in a second step that adds the following "reverse" transitions:

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

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, or $i + 1$ positive agents. One can see that the probability of $i + 1$ positive agents is $\Omega(1/n)$, the probability 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 $\Omega(n^{2^d-1})$. Let use see why. First, let us analyse the probabilities of $i + 1$ and $i - 1$ positive agents:

- $i \to i + 1$. This happens whenever a non-zero agent with wealth different from 1 or -1 meets a zero agent. Since i is the number of positive agents, the configuration C has $n - i > n - 2^d$ zero agents. Further, since the total wealth is 2^d and there are less than 2^d non-zero agents, at least one agent has wealth bigger than 1. So the probability is at least $p^+ := 2(n - 2^d)/n(n - 1)$, and so $\Omega(1/n)$.
- $i \to i - 1$. This can only happen if two non-zero agents meet. Since there are less than 2^d non-zero agents, $p^- := 2^d(2^d - 1)/n(n - 1)$ is an upper bound, and so the probability is $\mathcal{O}(1/n^2)$ for fixed d.

So we obtain a random walk with states $\{2^d, 2^{d-1}, \dots, 1\}$, initial state 2^d, target 1, and probabilities $p_1 := p^+ \in \mathcal{O}(1)$ of moving towards 2^d, and probability $p_2 := 1 - p^+ \in \Theta(1)$ of moving towards 1. The expected time to state 1 underapproximates the expected stabilisation time of the protocol, because in the walk one cannot stay in a state and must instead move towards 1. Standard results on the Gambler's Ruin problem yield $E[T_{2^d}] \in \Omega(n^{2^d-2})$ [25].

Recall that predicates of quantifier-free Presburger arithmetic are boolean combinations of threshold and remainder predicates. Therefore, the size of a predicate depends on the number d of bits of the largest coefficient, and the number s of predicates of the boolean combination. Examples 3 and 4 show that in the constructions of [5] and [11] the number of states, respectively the expected number of iterations, grows exponentially in d. The next example shows that the same holds for the parameter s.

Example 5. 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. (The states of $\mathcal{P}$ are all possible pairs of states (q, r), where q and r are states of $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively.) It follows that the number of states of a protocol for $\varphi := \varphi_1 \wedge \cdots \wedge \varphi_s$ grows exponentially in s, and so in $|\varphi|$.

Blondin et al. give an alternative construction with polynomially many states [11, Section 5.3]. However, the protocol 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 [11].) Starting with an even number $n \ge 2$ of agents in q_1, eventually all agents move to q_2 and stay there. We show that the expected number of interactions is $\Omega(2^{n/10})$.

Let $(C_0, C_1, \dots)$ be the stochastic process induced by the toy protocol, where C_i indicates the configuration after i interactions. Since at every step agents are chosen independently and uniformly at random, the process is a Markov chain. We can identify the state space of the chain with the set $\{0, 1, 2, \dots, n\}$ via the mapping $C_t \mapsto C_t(q_1)$. At state i, three transitions can happen, leading to states $i + 1$, i and $i - 2$. The probabilities of moving to $i + 1$ and $i - 2$ are $\frac{i(n-i)}{n(n-1)}$ and $\frac{i(i-1)}{n(n-1)}$, respectively. The goal is to reach state 0 from state n.

In order to obtain a lower bound on the number of steps, let us reduce the states to $\{0, \dots, \lfloor n/5 \rfloor\}$, replacing the transition $\lfloor n/5 \rfloor \mapsto \lfloor n/5 \rfloor + 1$ by a self-loop at $\lfloor n/5 \rfloor$, and starting at state $\lfloor n/5 \rfloor$ instead of n. This only reduces the number of steps to the goal. In this new chain, the quotient of the probabilities of moving to $i + 1$ and $i - 2$ is $\frac{i(i-1)}{i(n-i)} \le \frac{i}{n-i} \le \frac{1}{4}$ for all states i such that $i + 1$ and $i - 2$ exist. It is easy to see that we can simplify the chain further, without increasing the number of steps to the goal, into a chain with probability 4/5 and 1/5 of moving to from i to $i + 1$ and to $i - 2$, respectively. The expected number of steps to the goal in this chain is the same as for a random walk with states $\{0, \dots, \lfloor n/10 \rfloor\}$, biased by a factor of 2 in the "wrong" direction. (Indeed, the fact that in the chain we move from i to $i - 2$, while in the random walk we move from i to $i - 1$, is compensated by the probability in the chain being lower by a factor of 4). This biased random walk needs $\Omega(2^{n/10})$ steps in expectation until it reaches 0 from $\lfloor n/10 \rfloor$ [25].

5. Constructing fast and succinct protocols: overview

Given a predicate $\varphi \in \mathit{QFPA}$, in the rest of the paper we show how to construct a fast and succinct protocol deciding φ. We give an overview of the procedure, which first constructs a population computer for a different predicate, called double(φ), and then transforms it into a protocol for φ. We start by defining double(φ).

Definition 6. Let $\varphi \in \mathit{QFPA}$ be a predicate over variables $x_1, ..., x_v$. The predicate double(φ) $\in \mathit{QFPA}$ over variables $x_1, ..., x_v, x'_1, ..., x'_v$ is defined as follows: For every $i \in \{1, ..., v\}$, replace every occurrence of x_i in φ by $x_i + 2x'_i$.

For example, if $\varphi = (x - y \geq 0)$ then double(φ) $= (x + 2x' - y - 2y' \geq 0)$. Observe that size double(φ) $\in \mathcal{O}$(size φ). The procedure consists of the following steps:

1. Construct a succinct bounded population computer $\mathcal{P}$ deciding double(φ).
2. Convert $\mathcal{P}$ into a fixed-parameter fast and succinct population protocol $\mathcal{P}'$ deciding φ for inputs of size $\Omega(|\varphi|)$.
3. Prove that $\mathcal{P}'$ is not only fixed-parameter fast, but even fast.
4. Convert $\mathcal{P}'$ into a fast and succinct protocol deciding φ for all inputs.

Remark 7. The restriction to inputs of size $\Omega(|\varphi|)$ is mild. Indeed, in the intended applications of population protocols, like molecular programming, the number of agents is typically much larger than $|\varphi|$. In these applications the behaviour of the protocol for small inputs is irrelevant, and so Step 4 is of little interest. We include it for completeness.

We describe each of the steps in some more detail, and the results we obtain.

Step 1 (Section 6). We exhibit a procedure that constructs succinct and bounded population computers for all Presburger predicates (and so, in particular, for any predicate of the form double(φ)). More precisely, the section proves the following theorem:

Theorem 8. *For every predicate $\varphi \in \mathit{QFPA}$ there exists a bounded population computer of size $\mathcal{O}(|\varphi|)$ that decides φ.*

Step 2 (Section 7). Loosely speaking, the section shows that every succinct bounded computer for double(φ) can be transformed into a fixed-parameter fast and succinct protocol for φ. Formally, it proves the following theorem:

Theorem 9. *Every bounded population computer of size m deciding* double(φ) *can be converted into a terminating population protocol with $\mathcal{O}(m^2)$ states that decides φ within $2^{\mathcal{O}(m^2 \log m)} n^3$ interactions for all inputs of size $\Omega(m)$.*

Observe that this theorem relates boundedness, a *qualitative* property of population computers that can be proved using classical techniques like ranking functions, to the *quantitative* property of stabilising in $2^{\mathcal{O}(m^2 \log m)} n^3$ expected interactions. This greatly simplifies the task of designing fixed-parameter fast protocols. The theorem is proved by means of a sequence of conversions enforcing the conditions that make a population computer a population protocol: only binary transitions, no helpers, and consensus output.

Step 3 (Section 8). Theorem 9 does not yet prove the existence of succinct and fast protocols, because of the $2^{\mathcal{O}(m^2 \log m)}$ term. On the other hand, it holds for *arbitrary* bounded population computers, not only the ones defined in Section 6. So we trade generality against speed. We show that for the protocols of Section 6 the conversion of Section 7 yields fast protocols; more precisely, we reduce the $2^{\mathcal{O}(m^2 \log m)}$ term to m^7. Moreover, we also reduce the dependence on n. It is known that population protocols deciding majority need $\Omega(n^2)$ interactions in expectation [1][5] and so, since Theorem 9 only gives a $\mathcal{O}(n^3)$ upper bound, there is still a gap. Our refined analysis closes the gap. Formally, Section 8 proves:

Theorem 10. *For every predicate $\varphi \in \mathit{QFPA}$ there exists a terminating population protocol of size $\mathcal{O}(|\varphi|)$ that decides φ in $\mathcal{O}(|\varphi|^7 n^2)$ interactions for inputs of size $\Omega(|\varphi|)$.*

So Theorem 10 shows that our construction is optimal in n. Regarding the number of states, an $\Omega(|\varphi|^{1/4})$ lower bound was shown in [12], which leaves a polynomial gap. We conjecture that the lower bound of [12] can be improved, but this question exceeds the scope of this paper and is left for future research.

Step 4. It remains to obtain succinct protocols that are fast for all inputs not only for those of size $\mathcal{O}(m)$. This step is carried out by direct application of a technique of [11] that, given a predicate φ and a constant $\ell \in \mathcal{O}(|\varphi|^3)$, constructs a succinct

[5] In fact, $\Omega(n^2)$ interactions are required for "most" semilinear predicates [8].

protocol deciding φ for inputs of size at most ℓ (see Section 6 of [11]). This protocol can be combined with the one obtained in Step 3 to yield a succinct protocol that decides φ for all inputs (see Section 3 of [11]), and has speed $\mathcal{O}(|\varphi|^7 n^2)$ for all inputs of size $\Omega(m)$, and so $\mathcal{O}(|\varphi|^7 n^2)$ asymptotic speed. Applying Theorem 10 we directly obtain the following result:

Theorem 11. *For every $\varphi \in$ QFPA there exists a succinct terminating population protocol of size $\mathcal{O}(|\varphi|^a)$, for some constant $a \geq 0$, that decides φ in at most $\mathcal{O}(|\varphi|^7 n^2)$ interactions.*

6. Succinct bounded population computers for Presburger predicates

This section is structured as follows. Section 6.1 introduces a generic method to construct succinct computers for predicates whose coefficients are powers of 2. Section 6.2 and Section 6.3 apply the method to remainder and threshold predicates, respectively. Section 6.4 shows how to generalise the construction to remainder and threshold predicates with arbitrary integer coefficients, and how to construct computers for boolean combinations of remainder and threshold predicates.

6.1. A generic method to construct succinct computers

We introduce a method to construct computers for remainder predicates $\sum_{i=1}^{v} a_i x_i \equiv_\theta c$ and threshold predicates $\sum_{i=1}^{v} a_i x_i \geq c$. We call $\{a_1, ..., a_v\}$ the set of *coefficients* of the predicate.

The states of the computer for a predicate φ are a finite set of integers, including the coefficients of φ and 0. The initial states are the coefficients of φ, and all helpers (in a number to be determined) are initially in state 0. With this choice, a configuration C is a multiset of integers, and we define its *value* as $\mathrm{sum}(C)$. For example, a configuration that puts one agent in state 8, three agents in state 2, and two agents in state 0 has value $8 + 3 \cdot 2 + 2 \cdot 0 = 14$. Observe that helpers have value 0, and so all initial configurations $C_I + C_H$ for a given input C_I have the same value.

We introduce some terminology. A configuration C *satisfies* a remainder predicate $\sum_{i=1}^{v} a_i x_i \equiv_\theta c$ if $\mathrm{sum}(C) \equiv_\theta c$, and a threshold predicate $\sum_{i=1}^{v} a_i x_i \geq c$ if $\mathrm{sum}(C) \geq c$. For initial configurations C, this definition coincides with $\varphi(C) = 1$, hence the terminology "satisfies". However, this definition of satisfying a predicate is now also applicable if C includes states other than the coefficients of φ. While this is the obvious way to perform this extension, it is important to highlight that with this definition, whether a configuration C satisfies a predicate φ is *independent* of the coefficients of φ. Instead satisfying a predicate only depends on the modulus θ or threshold c.

Satisfying a predicate induces an equivalence relation: two configurations are *equivalent with respect to* φ if both of them satisfy φ, or none of them does. (When φ is clear from the context, we just say that the configurations are equivalent.) In particular, two configurations with the same value are equivalent with respect to any predicate.

Recall that $\mathrm{supp}(C)$ is the configuration given by $\mathrm{supp}(C)(q) = 1$ if $C(q) \geq 1$ and $\mathrm{supp}(C)(q) = 0$ otherwise. A configuration C is *well-supported* w.r.t. φ if it is equivalent to $\mathrm{supp}(C)$. Loosely speaking, whether well-supported configurations satisfy φ or not depend only on their supports. In particular, if $C = \mathrm{supp}(C)$, i.e. if C puts at most one agent in each state, then C is well-supported. However, the converse does not hold:

Example 12. Consider the predicate $\varphi = \sum_{i=1}^{v} a_i x_i \geq c$, and assume $a_1 \geq c$. The configuration $C = \{\!\{a_1, a_1\}\!\}$ is well-supported w.r.t. φ. Indeed, we have $\mathrm{sum}(C) = 2a_1$ and $\mathrm{sum}(\mathrm{supp}(C)) = a_1$, and so both of them satisfy φ. However, we have $\mathrm{supp}(C) = \{a_1\}$, and so $C \neq \mathrm{supp}(C)$.

Our generic method for constructing computers is based on the following simple fact:

Proposition 13. *Let φ be a remainder or threshold predicate. Let $\mathcal{P}$ be a computer with integers as states, the coefficients of φ as initial states, and all helpers initially in state 0. If $\mathcal{P}$ satisfies the following four properties, then it decides φ:*

1. *$\mathcal{P}$ is bounded.*
2. *Transitions preserve equivalence, i.e. if $C \mapsto C'$ then C and C' are equivalent w.r.t. φ.*
3. *Terminal configurations are well-supported.*
4. *The output function O is given by $O(S) = 1$ iff S satisfies φ.*

Proof. Since $\mathcal{P}$ is bounded, every run starting at an initial configuration $C_0 = C_I + C_H$ reaches a terminal configuration C_T, and we have:

	C_I satisfies φ	
iff	C_0 satisfies φ	($\mathrm{sum}(C_0) = \mathrm{sum}(C_I)$ because helpers have value 0)
iff	C_T satisfies φ	(C_T is equivalent to C_0 w.r.t. φ by (2))
iff	$\mathrm{supp}(C_T)$ satisfies φ	(by (3))
iff	$O(\mathrm{supp}(C_T)) = 1$	(by (4))
iff	$\mathcal{P}$ returns 1. □	

In the next two sections, we apply this method to remainder and threshold predicates whose coefficients are positive or negative powers of 2. Given such a predicate, we define a computer satisfying the properties of Proposition 13.

6.2. Population computers for remainder predicates

Since every equivalence class modulo θ has a representative r with $0 \leq r \leq \theta - 1$, every remainder predicate with integer coefficients is equivalent to a remainder predicate with coefficients in this range, hence we only consider this case. For example $7x - 2y \equiv_7 11$ can be rewritten to $5y \equiv_7 4$. Further, we assume in this section that the coefficients are powers of 2, a restriction lifted later in Section 6.4. So we let $Pow^+ = \{2^i \mid i \geq 0\}$, and for the rest of the section fix a remainder predicate

$$\varphi := \sum_{i=1}^{v} a_i x_i \equiv_\theta c \quad \text{where } \{a_1, ..., a_v\} \subseteq Pow^+ \cap \{1, ..., \theta - 1\}$$

Let $d := \lceil \log_2 \theta \rceil$. We define the computer $\mathcal{P}_\varphi = (Q, \delta, I, O, H)$ as follows.

States and initial states. The set of states of $\mathcal{P}_\varphi$ is $Q := Pow^+_d \cup \{0\}$, where $Pow^+_d = \{2^i \mid 0 \leq i \leq d\}$. The initial states are the coefficients of φ, i.e. $I := \{a_1, ..., a_v\}$.

Transitions. The transitions of $\mathcal{P}_\varphi$ transform configurations into equivalent configurations "closer" to being well-supported. Configurations C that put at most one agent in each state of Pow^+_d satisfy $\mathrm{sum}(C) = \mathrm{sum}(\mathrm{supp}(C))$, hence are well-supported.[6] So for each state $2^i \in Pow^+_d \setminus \{2^d\}$ we add to δ a transition that reduces the number of agents in 2^i, if there are more than one:

- For $2^i \in \{2^0, ..., 2^{d-1}\}$, we add to δ a transition that takes two agents from state 2^i, and puts one agent each in the states 2^{i+1} and 0:

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

We still need a transition that reduces the number of agents in 2^d. So we add to δ a transition that replaces an agent in 2^d by a multiset of agents r satisfying $\mathrm{sum}(r) = 2^d - \theta$, preserving equivalence:

- Let $b_{d-1}b_{d-2}...b_0$ be the binary encoding of $2^d - \theta$, and let $\{i_1, ..., i_j\} \subseteq [0, d-1]$ be the set of positions at which the binary encoding has a 1. We add to δ a transition

$$2^d, \underbrace{0, \ldots, 0}_{j-1} \quad \mapsto \quad 2^{i_1}, \ldots, 2^{i_j} \qquad \langle \text{modulo} \rangle$$

For example, if $\theta = 19$, then $d = 5$, $2^d - m = 13$, $b_{d-1}b_{d-2}...b_0 = 1101$, $\{i_1, ..., i_j\} = \{0, 2, 3\}$, and $\langle$modulo$\rangle$ is $2^5, 0, 0 \mapsto 2^3, 2^2, 2^0$.

As shown below, transitions $\langle$combine$\rangle$ and $\langle$modulo$\rangle$ are enough for correctness. However, in order to make the protocol faster we also add to δ a last transition that takes d agents from state 2^d and replaces them by a multiset of agents with total value $d \cdot 2^d \bmod \theta$:

- Let $b_d b_{d-1}...b_0$ be the binary encoding of $d \cdot 2^d \bmod \theta$, and let $\{i_1, ..., i_j\} \subseteq [0, d-1]$ be the set of positions at which the binary encoding has a 1, i.e. $b_{i_1} = \cdots = b_{i_j} = 1$. We introduce the transition:

$$\underbrace{2^d, \ldots, 2^d}_{d} \quad \mapsto \quad 2^{i_1}, \ldots, 2^{i_j}, \underbrace{0, ..., 0}_{d-j} \qquad \langle \text{fast modulo} \rangle$$

Note that d agents in 2^d can always represent their combined value modulo θ.

Helpers. We set $H := \{3d \cdot 0\}$, i.e. state 0 initially contains at least $3d$ helpers. (As shown in the proof of Lemma 15, with $3d$ helpers all terminal configurations are well-supported.)

Output function. For every set S of states, $O(S) := 1$ if S satisfies φ, else $O(S) := 0$.

Example 14. Fig. 1 shows the population computer for $\varphi = (8x + 2y + z \equiv_{11} 4)$.

[6] Observe that C and $\mathrm{supp}(C)$ may differ on the number of agents they put in state 0, but such agents have value 0.

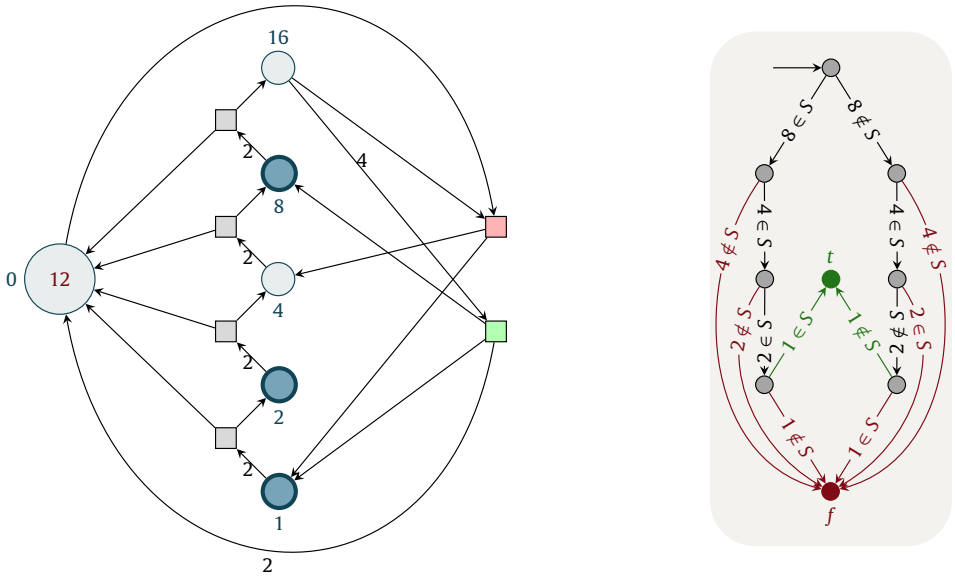


Fig. 1. Graphical Petri net representation (see Section 3) of the population computer for the predicate $\varphi = (8x + 2y + z \equiv_{11} 4)$. Recall that circles and squares represent states and transitions, respectively. Initial states are shown in darker colour: state 8 for input x, state 2 for input y, and state 1 for input z. We have $\theta = 11$, $d = \lceil \log_2 \theta \rceil = 4$, and $c = 4$. State 0 contains initially $3d = 12$ helpers. The top right transition is ⟨modulo⟩, the bottom right transition is ⟨fast modulo⟩ and the four transitions in the left column are ⟨combine⟩. The output function returns 1 for the supports S such that $\mathrm{sum}(S) \equiv_{11} 4$, i.e. $O(S) = 1$ for $S = \{4\}$ and $S = \{1, 2, 4, 8\}$. A decision diagram for this function is shown on the right. For the support $S = \{2, 4, 8\}$, the decisions are: left (because $8 \in S$), right (because $4 \in S$), right (because $2 \in S$), and down (because $1 \notin S$) leading to *false* as $2 + 4 + 8 = 14 \not\equiv_{11} 4$.

Lemma 15. *Let $\theta \in \mathbb{N}$, and let $d := \lceil \log_2 \theta \rceil$. Let $\varphi := \sum_{i=1}^{v} a_i x_i \equiv_\theta c$ be a remainder predicate such that $a_i \in Pow^+_{d-1}$ for every $1 \le i \le v$ and $0 \le c \le \theta - 1$. The computer $\mathcal{P}_\varphi$ described above satisfies the conditions of Proposition 13, and so decides φ. Further, $\mathcal{P}_\varphi$ has size $\mathcal{O}(d)$.*

Proof. Let us prove that $\mathcal{P}_\varphi$ satisfies the conditions of Proposition 13.

(1) $\mathcal{P}_\varphi$ is bounded. Let C_0 be an initial configuration with n agents. We first claim that every run starting at C_0 contains at most n occurrences of ⟨modulo⟩ or ⟨fast modulo⟩ transitions. Recall that, given a configuration C, we have $\mathrm{sum}(C) = \sum_{i=0}^{d} C(2^i) \cdot 2^i$. So, since $d := \lceil \log_2 \theta \rceil$, we have $0 \le \mathrm{sum}(C) \le \theta n$. Further, if $C \mapsto C'$, then $\mathrm{sum}(C) \ge \mathrm{sum}(C')$; moreover, if C' is obtained from C by executing a ⟨modulo⟩ or a ⟨fast modulo⟩ transition, then $\mathrm{sum}(C) \ge \mathrm{sum}(C') + \theta$, i.e. the sum decreases by at least θ. This proves the claim.

Since an occurrence of ⟨combine⟩ decreases the number of agents occupying the states $2^0, \ldots, 2^d$ by one, there are at most n occurrences of ⟨combine⟩ transitions between any two consecutive occurrences of ⟨modulo⟩ or ⟨fast modulo⟩ transitions. So, by the claim, every run starting at C_0 reaches a terminal configuration after $\mathcal{O}(n^2)$ steps, and we are done.

(2) Transitions preserve equivalence w.r.t. φ. Inspection of ⟨combine⟩, ⟨modulo⟩, and ⟨fast modulo⟩ shows that $C \mapsto C'$ implies $\mathrm{sum}(C) \bmod \theta = \mathrm{sum}(C') \bmod \theta$. So $\mathrm{sum}(C) \equiv_\theta c$ holds iff $\mathrm{sum}(C') \equiv_\theta c$ holds.

(3) Every terminal configuration C_T of $\mathcal{P}_\varphi$ is well-supported.

First of all, observe that every configuration C s.t. $C(q) \le 1$ for all $q \ne 0$ is well-supported. This follows because it would imply $\mathrm{supp}(C)(q) = C(q)$ for all $q \ne 0$, and the state 0 trivially does not influence $\mathrm{sum}(C)$ and well-supportedness. Hence our strategy to prove (3) is to prove that every terminal configuration fulfils this stronger property.

We have $C_T(2^i) \le 1$ for $0 \le i \le d - 1$, because all ⟨combine⟩ transitions are disabled in C_T. It remains to prove $C(2^d) = 0$. If it were the case that $C_T(2^d) \ge d$, then ⟨fast modulo⟩ would be enabled, hence $C_T(2^d) \le d - 1$. Since the number of agents is at least the number of helpers, i.e. at least $3d$, we have $C_T(0) + C_T(2^d) \ge 2d$, hence $C_T(0) \ge d$. This implies that if we had $1 \le C_T(2^d)$, then ⟨modulo⟩ would be enabled. So $C_T(2^d) = 0$, and the claim is proved.

(4) $O(S) = 1$ iff S satisfies φ. Holds by definition.

It remains to prove that $\mathcal{P}_\varphi$ has size $O(d)$. The computer has $\mathcal{O}(d)$ states and helpers. It has d ⟨combine⟩ transitions with size 2; further, $|\langle \text{modulo} \rangle| \le d + 2$ and $|\langle \text{fast modulo} \rangle| \le d$. So the total size of the transitions is also $\mathcal{O}(d)$. For the size of the output function, observe that for every set $S \subseteq Q$ we have $\mathrm{sum}(S) \le 2^{d+1}$. So, since $2^d \le \theta \le 2^{d+1}$ by our choice of d, either $\mathrm{sum}(S) \bmod \theta = \mathrm{sum}(S)$ or $\mathrm{sum}(S) \bmod \theta = \mathrm{sum}(S) - \theta$. Since $\mathrm{sum}(S)$ and c have $d + 1$ bits, whether $\mathrm{sum}(S) = c$ or $\mathrm{sum}(S) - \theta = c$ can be decided by a boolean circuit with $O(d)$ gates. Thus, $\mathrm{size}(\mathcal{P}_\varphi) := |Q| + |H| + \mathrm{size}(O) + \sum_{t \in \delta} |t| \in \mathcal{O}(d)$. □

6.3. Population computers for threshold predicates

We construct a population computer $\mathcal{P}_\varphi$ for a threshold predicate $\varphi := \sum_{i=1}^{v} a_i x_i \geq c$. Observe that, contrary to the case of remainder predicates, not every threshold predicate is equivalent to another one with positive coefficients. We also restrict ourselves to the case in which the coefficients are powers of 2, i.e. elements of $Pow := \{2^i, -2^i \mid i \geq 0\}$.

Let $d_0 := \max\{\lceil \log_2 c \rceil + 1, \lceil \log_2 |a_1| \rceil, ..., \lceil \log_2 |a_v| \rceil\}$. We define a protocol $\mathcal{P}_\varphi$ for each $d \geq d_0$. (This is used in Section 6.4, where we construct computers for boolean combinations of predicates.). The computer $\mathcal{P}_\varphi$ has $Pow_d \cup \{0\}$ as set of states, where $Pow_d := \{2^i, -2^i \mid 0 \leq i \leq d\}$.

The following lemma identifies a set of well-supported configurations w.r.t. φ, i.e. a set of configurations C such that $\mathrm{sum}(C) \geq c \Leftrightarrow \mathrm{sum}(\mathrm{supp}(C)) \geq c$. We design $\mathcal{P}_\varphi$ so that every terminal configuration belongs to this set.

Lemma 16. *Let φ and d_0 be defined as above and fix $d \geq d_0$. Every configuration C over states $Pow_d \cup \{0\}$ satisfying the following three conditions is well-supported w.r.t. φ:*

1. *$C(2^i) \leq 1$ and $C(-2^i) \leq 1$ for every $1 \leq i \leq d-1$;*
2. *$C(2^i) = 0$ or $C(-2^i) = 0$ for every $1 \leq i \leq d$;*
3. *$C(2^d) = 0$ or $C(-2^{d-1}) = 0$, and $C(-2^d) = 0$ or $C(2^{d-1}) = 0$.*

Before proving the lemma, observe that the third condition is necessary. Let $c = 5$ and $d = 3$. The configuration $C = \{\!|2^3, 2^3, -2^2, -2^1|\!\}$ satisfies conditions 1 and 2, but is not well-supported; indeed, $\mathrm{sum}(C) = 10 \geq 5$, but $\mathrm{sum}(\mathrm{supp}(C)) = 2 < 5$. On the contrary, the configuration $\{\!|2^3, 2^3, -2^1|\!\}$ is well-supported.

Proof. Let C be a configuration fulfilling the conditions. We prove that C satisfies φ iff $\mathrm{supp}(C)$ satisfies φ, i.e. that $\mathrm{sum}(C) \geq c$ holds iff $\mathrm{sum}(\mathrm{supp}(C)) \geq c$ holds. For clarity, in the rest of the proof we abbreviate $\mathrm{supp}(C)$ to C_S. We consider three cases:

- $C(2^d) > 0$. We prove $\mathrm{sum}(C) \geq \mathrm{sum}(C_S) \geq c$, which shows that C and C_S satisfy φ. By definition we have $C(q) \geq C_S(q)$ for every state q. Further, by conditions 1 and 2 and $C(2^d) > 0$, we have $C(q) = C_S(q)$ for every state $q \neq 2^d$. So $\mathrm{sum}(C) \geq \mathrm{sum}(C_S)$. Now we prove $\mathrm{sum}(C_S) \geq c$:

$$\begin{aligned}
\mathrm{sum}(C_S) &= \sum_{i=1}^{d} C_S(2^i) \cdot 2^i - \sum_{i=1}^{d} C_S(-2^i) \cdot 2^i \\
&\geq 2^d - \sum_{i=1}^{d} C_S(-2^i) \cdot 2^i && (C(2^d) > 0, \text{ and so } C_S(2^d) = 1) \\
&\geq 2^d - \sum_{i=1}^{d-2} C_S(-2^i) \cdot 2^i && (\text{conditions 2 and 3}) \\
&\geq 2^{d-1} && (\text{condition 1}) \\
&\geq c && (\text{definition of } d)
\end{aligned}$$

- $C(-2^d) > 0$. Symmetric.
- $C(2^d) = 0 = C(-2^d)$. By condition 1 we have $\mathrm{sum}(C) = \mathrm{sum}(C_S)$, and we are done. □

We proceed to the formal description of the computer $\mathcal{P}_\varphi = (Q, \delta, I, O, H)$ for a predicate

$$\varphi := \sum_{i=1}^{v} a_i x_i \geq c \quad \text{where } a_i \in Pow \text{ for every } 1 \leq i \leq v \text{ and } c \in \mathbb{N}$$

and for a fixed $d \geq d_0$, where $d_0 := \max\{\lceil \log_2 c \rceil + 1, \lceil \log_2 |a_1| \rceil, ..., \lceil \log_2 |a_v| \rceil\}$.

States and initial states. The set of states of $\mathcal{P}_\varphi$ is $Q := Pow_d \cup \{0\}$. The initial states are the coefficients of φ, i.e. $I := \{a_1, ..., a_v\}$.

Transitions. Since configurations that are not well-supported violate at least one of the conditions of Lemma 16, we define transitions that "repair" these violations. For every $i \in [0, d-1]$ we add to δ the following transitions, which intuitively "repair" a violation of conditions 1, 2, and 3, respectively:

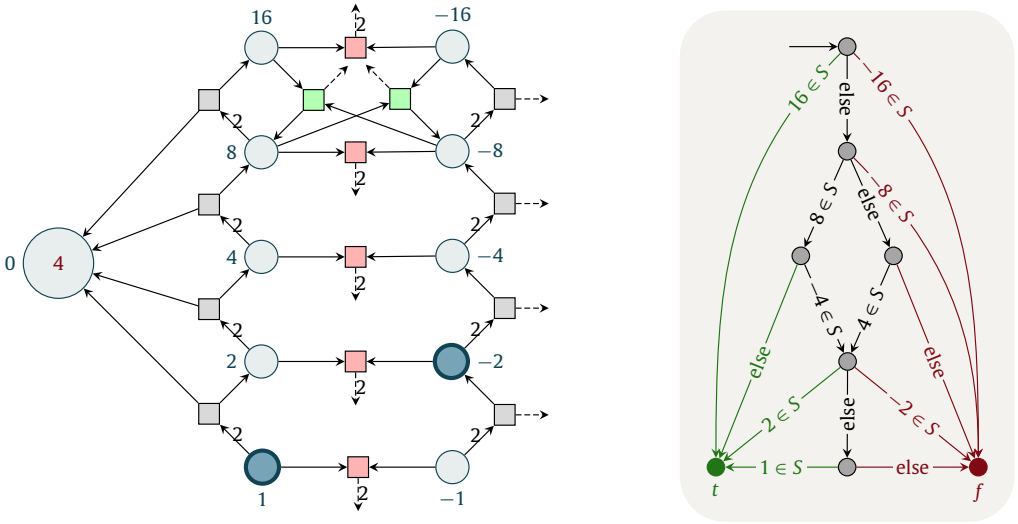


Fig. 2. Graphical Petri net representation (see Section 3) of the population computer for the predicate $\varphi = (-2x + y \geq 5)$ with $d = 4$. Initial states are shown in darker colour: state -2 for input x, and state 1 for input y. State 0 contains initially $d = 4$ helpers; all dashed edges leaving transitions implicitly lead to this state. Observe that the number of agents in 0 can only increase. The two transitions $16, -8 \mapsto 8, 0$ and $-16, 8 \mapsto -8, 0$ at the top are the ⟨cancel 2nd highest⟩ transitions. The rest of the transitions are organized in three columns. The middle column contains the ⟨cancel⟩ transitions. The left and right columns are the ⟨combine⟩ transitions. The output function O returns 1 for the supports S such that $\mathrm{sum}(S) \geq 5$. A decision diagram accepting these supports is shown on the right. For $S = \{8, -4, 2\}$ the decisions are: centre (because $16 \notin S$ and $-16 \notin S$), left (because $8 \in S$), right (because $-4 \in S$), and left (because $2 \in S$) leading to *true* as $8 - 4 + 2 = 6 \geq 5$.

$2^i, 2^i$	$\mapsto$	$0, 2^{i+1}$	$-2^i, -2^i$	$\mapsto$	$0, -2^{i+1}$	⟨combine⟩
$-2^i, 2^i$	$\mapsto$	$0, 0$	$-2^d, 2^d$	$\mapsto$	$0, 0$	⟨cancel⟩
$2^d, -2^{d-1}$	$\mapsto$	$0, 2^{d-1}$	$-2^d, 2^{d-1}$	$\mapsto$	$0, -2^{d-1}$	⟨cancel 2nd highest⟩

Helpers. We set $H := \{d \cdot 0\}$, i.e. state 0 initially contains at least d helper agents. (While the computer works correctly even with no helpers, the helpers in H are used later in Section 6.4 when constructing computers for boolean combinations of remainder and threshold predicates.)

Output function. For every set S of states, $O(S) := 1$ if S satisfies φ, else $O(S) := 0$.

Example 17. Fig. 2 shows the population computer for $\varphi = (-2x + y \geq 5)$ with $d = 4$.

Lemma 18. *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 $d \geq \max\{\lceil \log_2 c \rceil + 1, \lceil \log_2 |a_1| \rceil, \ldots, \lceil \log_2 |a_v| \rceil\}$, the computer $\mathcal{P}_\varphi$ described above satisfies the conditions of Proposition 13, and so decides φ. Further, $\mathcal{P}_\varphi$ has size $\mathcal{O}(d)$.*

Proof. We first prove that $\mathcal{P}_\varphi$ satisfies the conditions of Proposition 13.

(1) $\mathcal{P}_\varphi$ is bounded. Every transition increases the number of agents in state 0. Therefore, every run starting at an initial configuration with n agents has length at most n.

(2) Transitions preserve equivalence w.r.t. φ, i.e. if $C \to C'$, then $\mathrm{sum}(C) \geq c \iff \mathrm{sum}(C') \geq c$. In fact, even the stronger property $\mathrm{sum}(C) = \mathrm{sum}(C')$ holds by simple inspection of the transitions. For example for ⟨combine⟩ we check $-2^i + (-2^i) = 0 + (-2^{i+1})$.

(3) Every terminal configuration C_T of $\mathcal{P}_\varphi$ is well-supported. Terminal configurations satisfy all conditions of Lemma 16, because every configuration violating at least one condition enables at least one transition.

(4) $O(S) = 1$ iff S satisfies φ. Holds by definition.

It remains to prove that $\mathcal{P}_\varphi$ has size $\mathcal{O}(d)$. Observe that $\mathcal{P}_\varphi$ is binary and has $\mathcal{O}(d)$ transitions. Thus, $\mathrm{size}(\mathcal{P}_\varphi) := |Q| + |H| + \mathrm{size}(O) + \sum_{t \in \delta} |t| \in (2d + 3) + d + \mathrm{size}(O) + \mathcal{O}(d) \subseteq \mathcal{O}(d) + \mathrm{size}(O)$. So it remains to describe a boolean circuit of size $O(d)$ that decides whether a given terminal configuration C_T with support S satisfies φ, i.e. whether $\mathrm{sum}(S) \geq c$. For this, abbreviate $p_i := S(2^i)$ and $n_i = S(-2^i)$. We have $p_i, n_i \in \{0, 1\}$, and so $\mathrm{sum}(S)$ is the difference of the binary numbers $p_d p_{d-1} \ldots p_0$ and $n_d n_{d-1} \ldots n_0$. Whether this difference is bigger than or equal to c can be decided by a circuit with $\mathcal{O}(\log c) = \mathcal{O}(d)$ gates. □

6.4. Population computers for all Presburger predicates

We present a construction that, given threshold or remainder predicates $\varphi_1, \ldots, \varphi_s$ over a common set $X = \{x_1, \ldots, x_v\}$ of variables, yields a population computer $\mathcal{P}_\varphi$ deciding an arbitrary given boolean combination $\varphi = B(\varphi_1, \ldots, \varphi_s)$ of $\varphi_1, \ldots, \varphi_s$.

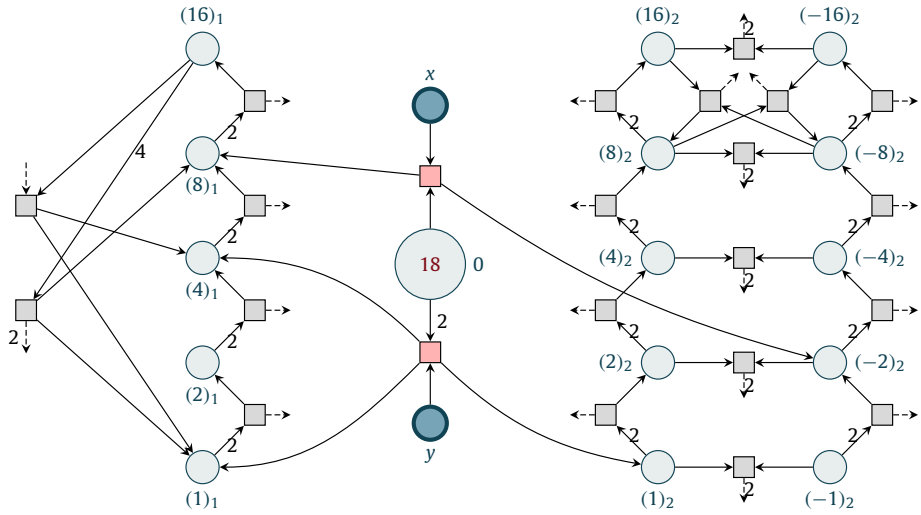


Fig. 3. Graphical Petri net representation (see Section 3) of the population computer for the predicate $\varphi = (8x + 5y \equiv_{11} 4) \vee (-2x + y \geq 5)$. The left and right parts correspond to the computers for $8x + 5y \equiv_{11} 4$, and $-2x + y \geq 5$ (see Figs. 1 and 2; while Fig. 1 depicts a population computer for $8x + 2y + z \equiv_{11} 4$, we can reuse its states and transitions). The common reservoir state is shown in the centre, and the input states above and below it. All dashed edges lead to or from it. The output function returns 1 for a support S iff $\text{sum}(S) \equiv_{11} = 4$, i.e. $O(S) = 1$ for $S = \{4\}$ and $S = \{1, 2, 4, 8\}$. A decision diagram can be obtained as the conjunction of the diagrams in Figs. 1 and 2.

The construction has a number of technical details, but its essence is simple: The computer "distributes" a given input to "subcomputers" $\mathcal{P}_1, ..., \mathcal{P}_s$ deciding $\varphi_1, ..., \varphi_s$, and lets them run concurrently. Each subcomputer, say $\mathcal{P}_j$, reaches a terminal configuration with support S_j such that $O_j(S_j) = 1$ iff the input satisfies φ_j. The output function of the computer is defined as the boolean combination of the output functions of the subcomputers, i.e. $O(S) := B(O_1(S_1), ..., O_s(S_s))$, modulo some technical details.

We remark that our notion of subcomputers is similar to the technique of *population splitting* (see e.g. [2]). However, while the eventual goal is the same (to have multiple subpopulations that work on distinct tasks), our construction differs in that *every* agent must be distributed. This must hence occur concurrently with the rest of the computation. In contrast, population splitting is usually employed as a separate phase in the beginning, and it suffices to distribute "most" agents into "roughly equal" parts.

Example 19. We use $\varphi_1 = (8x + 5y \equiv_{11} 4)$, $\varphi_2 = (-2x + y \geq 5)$ and $\varphi = \varphi_1 \vee \varphi_2$ as running example. Observe that the coefficient 5 of φ_1 is not a power of 2; in fact, the construction also shows how to deal with general remainder and threshold predicates. The Petri net representation of the computer for φ is shown in Fig. 3. The input is placed in states x and y. Intuitively, the two transitions directly below the place x and respectively directly above y distribute it to subcomputers for φ_1 (left) and φ_2 (right). Helpers help to run the subcomputers, but also to distribute the input. For example, the top pink transition takes only one agent from the input state x, but sends two agents to the subcomputers, hence it needs one helper. Careful choice of the exact set of states of each subcomputer and the number of helpers guarantee that the computer distributes all the input, i.e. that no terminal configuration puts agents in any input state.

Let us now give a more detailed, but still informal description of the construction, which proceeds in six steps:

1. Rewrite the remainder and threshold predicates. The constructions of Sections 6.2 and 6.3 only work for predicates where all coefficients are powers of 2. We transform each predicate φ_j into a new predicate φ'_j 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 $x, y \in \mathbb{N}$.

2. Construct subcomputers. For every $1 \leq j \leq s$, if φ'_j is a remainder predicate, then let $\mathcal{P}_j$ be the computer defined in Section 6.2, and if φ'_j is a threshold predicate, then let $\mathcal{P}_j$ be the computer of Section 6.3, with $d = d_0 + \lceil \log_2 s \rceil$. The computers $\mathcal{P}_1$ and $\mathcal{P}_2$ are shown in Figs. 1 and 2, respectively.

3. Combine subcomputers. Take the disjoint union of $\mathcal{P}_j$, but merge their 0 states. More precisely, rename all states $q \in Q_j$ to $(q)_j$, with the exception of state 0. Construct a computer with the union of all the renamed states and transitions. We call the combined 0 state *reservoir state*, as it holds agents with value zero needed for various tasks like input distribution.

4. Distribute the input. For each variable $x_i \in X$, add a corresponding new input state x_i and a *distribution transition* that takes one agent from state x_i and helpers from 0, and sends them to the input states of the subcomputers $\mathcal{P}_1, ..., \mathcal{P}_s$. The destinations of the agents sent to $\mathcal{P}_j$ are determined by a_i^j, the coefficient of x_i in φ_j. In Example 19, the predicates φ_1 and φ_2 have two variables x, y, and so we add to the computer two new states x and y. Further, the coefficients of x in

φ_1 and φ_2 are 8 and -2, respectively, and so we add the distribution transition $x, 0 \mapsto (8)_1, (-2)_2$. In words, this transition takes one agent from x and one helper agent, and sends them to state 8 of $\mathcal{P}_1$ and state -2 of $\mathcal{P}_2$. More interestingly, the coefficients of y in φ_1 and φ_2 are 5 and 1, and so we add the distribution transition $y, 0, 0 \mapsto (1)_1, (4)_1, (1)_2$. This transition takes one agent from x and *two* helpers, and sends two of these agents to the states 4 and 1; in this way, $\mathcal{P}_1$ receives agents with a total value of 5. The third agent is sent to state 1 of $\mathcal{P}_2$. Observe that the input is distributed to the subcomputers one agent at a time, and the distribution ends when the input states x and y become empty.

5. Set the number of helpers. As we have seen, distribution transitions need helpers. The initial number of helpers is chosen in order to guarantee that every run of the computer distributes all the input, i.e. eventually reaches a configuration where the input states are empty. Let r be the maximum arity of the distribution transitions for the input variables. In our example, the distribution transitions for x and y have arity two and three, respectively, and so $r = 3$. Intuitively, with $r - 1$ helpers the computer can distribute one agent from any of the input states $x_1, ..., x_v$. Initially, we put in the combined state 0 all helpers from all subcomputers, plus $r - 1$ additional helpers. In Example 19, the subcomputers for φ_1 and φ_2 have 12 and 4 helpers, respectively, and $r = 3$. So the final number of helpers is $12 + 4 + 2 = 18$. Lemma 21 proves that this number of helpers guarantees the complete distribution of the input.

6. Combine the output functions. Recall that $\mathcal{P}_\varphi$ combines the outputs of the subcomputers $\mathcal{P}_1, ..., \mathcal{P}_s$ according to $B(\varphi_1, ..., \varphi_s)$. In Example 19, we set the output to 1 if and only if the output of $\mathcal{P}_1$ or $\mathcal{P}_2$ is 1.

6.4.1. Formal definition

We define the population computer $\mathcal{P}_\varphi$ for a boolean combination of threshold or remainder predicates $\varphi_1, ..., \varphi_s$. Formally, $\varphi := B(\varphi_1, ..., \varphi_s)$, where $B(\varphi_1, ..., \varphi_s)$ is a boolean formula over variables $\varphi_1, ..., \varphi_s$, e.g. $(\varphi_1 \wedge \varphi_2) \vee (\varphi_3 \wedge (\varphi_1 \vee \varphi_4))$. We assume w.l.o.g. that each φ_j is a predicate over the same set $X = \{x_1, ..., x_v\}$ of variables, and that it is either a remainder predicate $\left(\sum_{i=1}^{v} a_i^j x_i \equiv_{\theta_j} c_j\right)$, where $0 \leq a_i^j < \theta_j$ and $0 \leq c_j < \theta_j$, or a threshold predicate $\left(\sum_{i=1}^{v} a_i^j x_i \geq c_j\right)$.

Rewriting the predicates We give the formal definition of the predicates φ'_j for every $1 \leq j \leq s$. We use an auxiliary function $\mathrm{bin}(x)$ that maps an integer x to the multiset over *Pow* corresponding to the binary representation of x. For example, $\mathrm{bin}(-13) = \{\!\{-2^3, -2^2, -2^0\}\!\}$ and $\mathrm{bin}(10) = \{\!\{2^3, 2^1\}\!\}$. Formally, let $\mathrm{sign}(x) : \mathbb{Z} \to \{-1, 0, 1\}$ be the function that assigns $-1, 0, 1$ to the negative integers, 0, and the positive integers, respectively, and define

$$\mathrm{bin}(x) := \{\!\{\mathrm{sign}(x) \cdot 2^i \mid i\text{-th bit in the binary encoding of } |x| \text{ is } 1.\}\!\}$$

Note that $\mathrm{sum}(\mathrm{bin}(x)) = x$ for all $x \in \mathbb{Z}$. We rewrite each predicate φ_j into:

$$\varphi'_j := \begin{cases} \sum_{i=1}^{v} \sum_{e \in \mathrm{supp}(\mathrm{bin}(a_i^j))} e \cdot x_{i,e} \geq c_j & \text{if } \varphi_j \text{ is a threshold predicate} \\ \sum_{i=1}^{v} \sum_{e \in \mathrm{supp}(\mathrm{bin}(a_i^j))} e \cdot x_{i,e} \equiv_{\theta_j} c_j & \text{if } \varphi_j \text{ is a remainder predicate} \end{cases}$$

Construction of subcomputers We define the population subcomputer $\mathcal{P}_j = (Q_j, \delta_j, I_j, O_j, H_j)$ for each φ'_j as follows:

- If φ'_j is a remainder predicate, we use the construction in Section 6.2, setting $Q_j := Pow^+_{d_j} \cup \{0\}$ for $d_j := \lceil \log_2 \theta_j \rceil$.
- If φ'_j is a threshold predicate, we use the construction in Section 6.3, setting $Q_j := Pow_{d_j} \cup \{0\}$ for $d_j := \max\{\lceil \log_2 c_j \rceil + 1, \lceil \log_2 (s \cdot a^j_{\max}) \rceil\} + 4$, where $a^j_{\max} := \max\{|a_1^j|, ..., |a_v^j|\}$. (The addition of 4 is not necessary for correctness, but we will later use it to show that the protocol is fast.)

Definition of $\mathcal{P}_\varphi$ We proceed to formally define the computer $\mathcal{P}_\varphi = (Q, \delta, I, O, H)$.

States and initial states. Define for each subcomputer $\mathcal{P}_j$ a mapping ν_j that renames the states of Q_j as follows: $\nu_j(0) = 0$ and $\nu_j(q) = (q)_j$ for every $q \neq 0$. After renaming, the states of $\mathcal{P}_1, ..., \mathcal{P}_s$ are pairwise disjoint, with the exception of the common reservoir state 0. The set of states of $\mathcal{P}_\varphi$ is $Q := X \cup \bigcup_{j=1}^{s} \nu_j(Q_j) \cup \{0\}$. The set of initial states is $I := X$.

Transitions. The set δ of transitions of $\mathcal{P}_\varphi$ contains:

- For each subcomputer $\mathcal{P}_j$, all transitions of $\mathcal{P}_j$, suitably renamed:

$$\nu_j(r) \mapsto \nu_j(s) \qquad \text{for every } (r \mapsto s) \in \delta_j \qquad \langle \text{subcomputer} \rangle$$

- Given a multiset $M \in \mathbb{N}^{Q_j}$, let $\nu_j(M)$ be the result of renaming the agents in M according to ν_j, and let $b_i := \sum_{j=1}^{s} |\mathrm{bin}(a_i^j)|$. For each variable $x_i \in X$, the computer $\mathcal{P}_\varphi$ contains a transition that distributes agents in state x_i to the states of the subcomputers:

$$\begin{aligned} x_i, \underbrace{0, ..., 0}_{b_i - 1} &\mapsto \textstyle\sum_{j=1}^{s} \nu_j(\mathrm{bin}(a_i^j)) && \text{if } b_i > 1 \\ x_i, 0 &\mapsto \textstyle\sum_{j=1}^{s} \nu_j(\mathrm{bin}(a_i^j)), 0 && \text{if } b_i = 1 \end{aligned} \qquad \langle \text{distribute} \rangle$$

Helpers. Let $r := \max_{i=1}^{v} \sum_{j=1}^{s} |\mathrm{bin}(a_i^j)|$. We set $H := (\max(r,2)-1) \cdot \langle 0 \rangle + \sum_{j=1}^{s} H_j$. So, loosely speaking, we put in state 0 at least the total number of helpers of all subcomputers plus $\max(r,2)-1$ additional helpers. This number of helpers guarantees that every run from a configuration that populates the initial states eventually enables some ⟨distribute⟩ transition, and so that terminal configurations do not populate the initial states (see Lemma 21).

Output function. The output function is the boolean combination of the output functions of the subcomputers. Formally, $O(S) := B(O_1(S \cap (\nu_1(Q_1))), ..., O_s(S \cap (\nu_s(Q_s))))$.

6.4.2. Correctness and size

We prove that $\mathcal{P}_\varphi$ decides $\varphi = B(\varphi_1, ..., \varphi_s)$. We use the method that will be described in Proposition 20. This requires to generalise some notions of Section 6.1 which were defined only for computers whose states are numbers (which is not the case for $\mathcal{P}_\varphi$, because the initial states are the variables in X and these have no value), and only for remainder and threshold predicates, not for their boolean combinations.

Given a predicate φ_j of the boolean combination, let us first define when a configuration C of $\mathcal{P}_\varphi$ satisfies φ_j. Let $C|_j$ denote the projection of C onto $\nu_j(Q_j) \cup \{0\}$ (recall that Q_j is the set of states of the subcomputer for φ_j). Define $\mathrm{sum}(C)_j := \mathrm{sum}(C|_j) + \sum_{x_i \in X} a_i^j C(x_i)$. Intuitively, this takes into account that at C the input may not have been completely distributed yet, and for the j-th subcomputer each agent in x_i has value a_i^j. We say that C *satisfies* φ_j if φ_j is a remainder predicate $\sum_{i=1}^{v} a_i^j x_i \equiv_{\theta_j} c_j$ and $\mathrm{sum}(C)_j \equiv_{\theta_j} c_j$, or if φ_j is a threshold predicate $\sum_{i=1}^{v} a_i^j x_i \geq c_j$ and $\mathrm{sum}(C)_j \geq c_j$. Similar to Section 6.1, this extends satisfying φ_j to states which are not variables. One important difference however is that in this case, $\mathrm{sum}(C)_j$ (and hence C satisfying φ_j) is *not* independent of the coefficients of φ_j, because of the $\sum_{x_i \in X} a_i^j C(x_i)$ summand.

We can now generalise the definitions of Section 6.1 as follows:

- C *satisfies* φ if $B(b_1, ..., b_s) = 1$, where $b_j = 1$ if C satisfies φ_j and $b_j = 0$ otherwise.
- Two configurations C, C' are *equivalent w.r.t.* φ if both C and C' satisfy φ, or none does.
- A configuration C is *well-supported* w.r.t. φ if C is equivalent to $\mathrm{supp}(C)$ w.r.t. φ.

We have the following result, proved exactly the same as Proposition 13:

Proposition 20. *Let $\varphi = B(\varphi_1, ..., \varphi_s)$ be a boolean combination of remainder and modulo predicates $\varphi_1, ..., \varphi_s$. If $\mathcal{P}_\varphi$ satisfies the following properties, then it decides φ:*

1. *$\mathcal{P}$ is bounded.*
2. *Transitions preserve equivalence.*
3. *Terminal configurations are well-supported.*
4. *The output function O is given by $O(S) = 1$ iff $B(O_1(S \cap Q_1), ..., O_s(S \cap Q_s)) = 1$.*

In the rest of the section we show that the protocol $\mathcal{P}_\varphi$ of Section 6.4.1 satisfies the four properties of Proposition 20. The key of the proof is the following technical lemma, showing that every terminal configuration distributes the input completely.

Lemma 21. *Let C_{term} be a terminal configuration of $\mathcal{P}_\varphi$ reachable from some initial configuration. Then $C_{\mathrm{term}}(X) = 0$.*

Proof. Let $Q_j' = \{(2^{d_j})_j, (-2^{d_j})_j, ..., (2^0)_j, (-2^0)_j\}$ be the states of $\mathcal{P}_j$ with non-zero value. Further, let Δ be the number of occurrences of ⟨distribute⟩ transitions in the run leading from C_0 to C_{term}. We proceed in three steps.

(1) $C_{\mathrm{term}}(Q_j') \leq H_j(0) + \frac{\Delta}{s}$ holds for each subcomputer $\mathcal{P}_j$. Intuitively, this states that the number of agents in Q_j is at most the number of helpers of $\mathcal{P}_j$ plus a *"fair share"* (i.e. $\frac{1}{s}$) of the processed agents. For the proof, observe that the projection $C_{\mathrm{term}}|_j$ is a terminal configuration of φ_j. If φ_j is a remainder predicate, then $C_{\mathrm{term}}(Q_j') \leq 3d_j = H_j(0)$ because otherwise $C_{\mathrm{term}}|_j$ enables some transition of $\mathcal{P}_j$ (see the proof of Lemma 15), and we are done. If φ_j is a threshold predicate, then we proceed as follows. Observe that

$$C_{\mathrm{term}}(Q_j') = \underbrace{C_{\mathrm{term}}((2^{d_j})_j) + C_{\mathrm{term}}((-2^{d_j})_j)}_{\alpha_j} + \underbrace{\sum_{i=0}^{d_j-1} C_{\mathrm{term}}((2^i)_j) + \sum_{i=0}^{d_j-1} C_{\mathrm{term}}((-2^i)_j)}_{\beta_j}$$

We show that $\alpha_j \leq \Delta/s$ and $\beta_j \leq H_j(0)$.

- $\alpha_j \leq \Delta/s$. Let $a^j_{\max}$ be the absolute value of the maximum coefficient of φ_j. Every occurrence of a ⟨distribute⟩ transition increases the total absolute value of the agents in Q'_j by at most $a^j_{\max}$. Since this absolute value is initially equal to zero, we have $\mathrm{sum}(C_{\mathrm{term}}|_j) \leq \Delta \cdot a^j_{\max}$. Further, C_{term} populates at most one of the states $(2^{d_j})_j$ and $(-2^{d_j})_j$, because otherwise it enables the ⟨cancel⟩ transition, contradicting that C_{term} is terminal. Assume C_{term} populates only $(2^{d_j})_j$ (the other case is similar). Since each agent in this state has value $2^{d_j} \geq a^j_{\max}$ and the total absolute value of $C_{\mathrm{term}}|_j$ is at most $\Delta \cdot a^j_{\max}$, at most Δ/s agents of C_{term} populate $(2^{d_j})_j$.
- $\beta_j \leq H_j(0)$. We have $\beta_j \leq d_j$ because, by Lemma 16, terminal configurations of the computer for a threshold predicate put at most d_j agents in the states of Pow_{d_j-1}, and $d_j \leq H_j(0)$ by definition of the computer for a threshold predicate.

(2) $C_{\mathrm{term}}(0) \geq \max(r,2)-1$. We start by collecting two facts:

a. By definition of $\mathcal{P}_\varphi$, the initial configuration C_0 puts at least $\max(r,2)-1+\sum_{j=1}^{s} H_j(0)$ helpers in state 0.
b. $C_{\mathrm{term}}(X) = C_{\mathrm{term}}(Q) - C_0(0) - \Delta$. This is a consequence of $C_{\mathrm{term}}(X) = C_0(X) - \Delta$, which holds because each occurrence of a ⟨distribute⟩ transition removes one agent from X, and $C_0(X) = C_0(Q) - C_0(0)$.

Now we proceed as follows:

$$
\begin{aligned}
C_{\mathrm{term}}(0) &= C_{\mathrm{term}}(Q) - \Big(C_{\mathrm{term}}(X) + \sum_{j=1}^{s} C_{\mathrm{term}}(Q'_j)\Big) && \{0\} = Q \setminus (X \cup \textstyle\bigcup_{j=1}^{s} Q'_j)\\
&= C_0(0) + \Delta - \sum_{j=1}^{s} C_{\mathrm{term}}(Q'_j) && \text{by (b)}\\
&\geq \max(r,2)-1 + \sum_{j=1}^{s} H_j(0) + \Delta - \sum_{j=1}^{s} C_{\mathrm{term}}(Q'_j) && \text{by (a)}\\
&\geq \max(r,2)-1 + \sum_{j=1}^{s} H_j(0) + \Delta - \sum_{j=1}^{s} \left(H_j(0) + \frac{\Delta}{s}\right) && \text{by (1)}\\
&\geq \max(r,2)-1
\end{aligned}
$$

(3) $C_{\mathrm{term}}(X) = 0$. By contradiction. Assume $C_{\mathrm{term}}(x_i) \geq 1$ for some $x_i \in X$. Then by (2) the ⟨distribute⟩ transition for x_i is enabled, contradicting that C_{term} is a terminal configuration. □

We are now able to prove our first main result.

Theorem 8. *For every predicate $\varphi \in$ QFPA there exists a bounded population computer of size $\mathcal{O}(|\varphi|)$ that decides φ.*

Proof. By definition, φ is a boolean combination $\varphi = B(\varphi_1, \ldots, \varphi_s)$ of remainder and threshold predicates. Let $\mathcal{P}_\varphi$ be the population computer of Section 6.4.1. We show that $\mathcal{P}_\varphi$ satisfies the conditions of Proposition 20 (and so decides φ) and has size $O(|\varphi|)$. We begin with the conditions of Proposition 20.
(1) $\mathcal{P}_\varphi$ is bounded. The execution of a ⟨distribute⟩ transition strictly reduces the number of agents in the input states, and no ⟨subcomputer⟩ transition puts agents in them. It follows that every run from an input configuration executes ⟨distribute⟩ transitions only finitely often. Further, by Lemma 15 and Lemma 18, each subcomputer $\mathcal{P}_j$ is bounded. So runs of $\mathcal{P}_\varphi$ also execute ⟨subcomputer⟩ transitions finitely often, and we are done.

(2) Transitions preserve equivalence w.r.t. φ. We have to show that if $C \mapsto C'$ then both C and C' satisfy φ, or none does. We prove a stronger property: for every $1 \leq j \leq s$, C satisfies φ_j iff C' satisfies φ_j. Let $1 \leq j \leq s$. It suffices to prove $\mathrm{sum}(C)_j = \mathrm{sum}(C')_j$. If $C \mapsto C'$ by a ⟨subcomputer⟩ transition the result follows from the corresponding result for subcomputers. If $C \mapsto C'$ by a ⟨distribute⟩ transition $x_i, 0, \ldots, 0 \mapsto \sum_{j=1}^{s} \nu_j(\mathrm{bin}(a^j_i))$, then we have

$$
\begin{aligned}
\mathrm{sum}(C')_j - \mathrm{sum}(C)_j &= (\mathrm{sum}(C'|_j) - \mathrm{sum}(C|_j)) + \sum_{x_k \in X} a^j_k (C'(x_k) - C(x_k))\\
&= \sum_{j=1}^{s} \mathrm{bin}(a^j_i) - a^j_i = 0
\end{aligned}
$$

(3) Terminal configurations are well-supported. This is the part requiring an application of Lemma 21. Let C_{term} be a terminal configuration. We prove that C_{term} and $\mathrm{supp}(C_{\mathrm{term}})$ are equivalent w.r.t. φ_j for every $1 \leq j \leq s$, which implies that C_{term} is

equivalent to $\mathrm{supp}(C_{\mathrm{term}})$ w.r.t. φ. Pick $1 \leq j \leq s$, and assume φ_j is a threshold predicate $\sum_{i=1}^{v} a_i^j x_i \geq c_j$; the remainder case $\sum_{i=1}^{v} a_i^j x_i \equiv_{\theta_j} c_j$ is analogous.

Let $C_{\mathrm{term}}|_j$ be the projection of C onto the subcomputer $\mathcal{P}_{\varphi_j}$. By definition, C_{term} satisfies φ_j iff $\mathrm{sum}(C_{\mathrm{term}})_j \geq c_j$, and $\mathrm{supp}(C_{\mathrm{term}})$ satisfies φ_j iff $\mathrm{sum}(\mathrm{supp}(C_{\mathrm{term}}))_j \geq c_j$. So it suffices to show $\mathrm{sum}(C_{\mathrm{term}})_j \geq c_j \Leftrightarrow \mathrm{sum}(\mathrm{supp}(C_{\mathrm{term}}))_j \geq c_j$. We have $\mathrm{sum}(C_{\mathrm{term}})_j = \mathrm{sum}(C_{\mathrm{term}}|_j) + \sum_{x_i \in X} a_i^j C_{\mathrm{term}}(x_i)$ by definition, and so, since $C_{\mathrm{term}}(X) = 0$ by Lemma 21, we get $\mathrm{sum}(C_{\mathrm{term}})_j = \mathrm{sum}(C_{\mathrm{term}}|_j)$. Analogously, $\mathrm{sum}(\mathrm{supp}(C_{\mathrm{term}}))_j = \mathrm{sum}(\mathrm{supp}(C_{\mathrm{term}})|_j) = \mathrm{sum}(\mathrm{supp}(C_{\mathrm{term}}|_j))$. So it suffices to show $\mathrm{sum}(C_{\mathrm{term}}|_j) \geq c_j \Leftrightarrow \mathrm{sum}(\mathrm{supp}(C_{\mathrm{term}}|_j)) \geq c_j$, i.e. that $C_{\mathrm{term}}|_j$ is a well-supported configuration of $\mathcal{P}_j$ w.r.t. φ_j. But this holds by Lemma 18.

(4) $O(S) = 1$ iff $B(O_1(S \cap Q_1), ..., O_s(S \cap Q_s)) = 1$. By definition.

Now we show that $\mathcal{P}_\varphi$ has size $O(|\varphi|)$. By definition, the size of $\mathcal{P}_\varphi$ is $\mathrm{size}(\mathcal{P}_\varphi) := |Q| + |H| + \mathrm{size}(O) + \sum_{t \in \delta} |t|$. We show that each of $|Q|$, $|H|$, $\mathrm{size}(O)$ and $\sum_{t \in \delta} |t|$ is $\mathcal{O}(|\varphi|)$.

For Q, recall that $Q := X \cup \bigcup_{j=1}^{s} \nu_j(Q_j) \cup \{0\}$. If φ_j is a remainder predicate, then $|Q_j| = d_j := \lceil \log_2 \theta_j \rceil$. If φ_j is a threshold predicate, then $|Q_j| = 2d_j = 2d_{0j} + 2\lceil \log_2 s \rceil \in \mathcal{O}(|\varphi_j| + \log_2 s)$. So $|Q| \in \mathcal{O}(|\varphi| + s \log_2 s)$. Since φ is a boolean formula over variables $\varphi_1, ..., \varphi_s$, and each variable appears at least once in the formula, φ has size $\Omega(s \log_2 s)$, and so $|Q| \in \mathcal{O}(|\varphi|)$.

For H, recall that $H := (\max(r, 2) - 1) \cdot \wr 0 \wr + \sum_{j=1}^{s} H_j$, where $r := \max_{i=1}^{v} \sum_{j=1}^{s} |\mathrm{bin}(a_i^j)|$ and $|H_j| \leq 3 \cdot d_j$ for every $1 \leq j \leq s$. So $|H| \in O(|\varphi| + s \log_2 s) = \mathcal{O}(|\varphi|)$.

For O, observe that given boolean circuits for functions $O_1, ..., O_s$, with $\gamma_1, ..., \gamma_s$ gates, and a circuit γ for a boolean formula $B(x_1, \ldots, x_s)$ with γ gates, there is a circuit for $B(O_1, \ldots, O_s)$ with $\sum_{j=1}^{s} |\gamma_j| + |\gamma| + \mathcal{O}(1)$ gates.

For $\sum_{t \in \delta} |t|$, observe that $\sum_{t \in \delta_j} |t| = \mathcal{O}(d_j)$ for every subcomputer $\mathcal{P}_j$, and that the distribution transition for x_i has arity $\sum_{j=1}^{s} |\mathrm{bin}(a_i^j)|$. So the total size is

$$\sum_{j=1}^{s} \mathcal{O}(d_j) + \sum_{x_i \in X} \sum_{j=1}^{s} |\mathrm{bin}(a_i^j)| = \sum_{j=1}^{s} \mathcal{O}(d_j) + \sum_{j=1}^{s} \sum_{x_i \in X} |\mathrm{bin}(a_i^j)|$$

$$= \mathcal{O}(|\varphi| + s \log_2 s) + \sum_{j=1}^{s} |\varphi_j| = \mathcal{O}(|\varphi| + s \log_2 s) + \mathcal{O}(|\varphi|) = \mathcal{O}(|\varphi|) \quad \square$$

7. From bounded population computers to fixed-parameter fast population protocols

We prove Theorem 9, i.e. we show that for every predicate φ, any bounded population computer of size m deciding double(φ) can be converted into a population protocol of size $\mathcal{O}(m^2)$ that decides φ in $2^{\mathcal{O}(m^2 \log m)} \cdot n^3$ interactions for inputs of size $\Omega(m)$.

We start in Section 7.1 by proving that every bounded binary computer with no helpers terminates in $2^{\mathcal{O}(m \log m)} \cdot n^3$ expected interactions. This relates to the bound in Theorem 9 by replacing m by $\mathcal{O}(m^2)$, the size of the computer we will use the statement on.

Hence it suffices to convert a bounded population computer for double(φ) into a bounded population protocol for φ, with only a quadratic blow-up in size. We achieve this by applying a sequence of five conversion steps. For most of these conversions we have to prove that they preserve the decided predicate, and for this reason before presenting the steps we introduce a technique to prove equivalence of computers in Section 7.2. The rest of Section 7 describes the five steps:

- Section 7.3 converts a bounded computer $\mathcal{P}$ for double(φ) into an equivalent bounded computer $\mathcal{P}_0$ of size $\mathcal{O}(\mathrm{size}(\mathcal{P}))$ satisfying two additional technical conditions. This is a very simple step that also serves as warm-up for the next ones.
- Section 7.4 converts $\mathcal{P}_0$ into an equivalent *binary* bounded computer $\mathcal{P}_1$ of size $\mathcal{O}(|Q_0| \cdot \mathrm{size}(\mathcal{P}_0))$, where Q_0 is the set of states of $\mathcal{P}_0$.
- Section 7.5 converts $\mathcal{P}_1$ into an equivalent binary bounded computer $\mathcal{P}_2$ with a *marked consensus output function* (a notion defined in the section) of adjusted size $\mathcal{O}(\mathrm{size}_2(\mathcal{P}_1))$.
- Section 7.6 converts $\mathcal{P}_2$ into a binary bounded computer $\mathcal{P}_3$ for φ — not double(φ) — with a marked consensus output function *and no helpers* of adjusted size $\mathcal{O}(\mathrm{size}_2(\mathcal{P}_2))$.
- Section 7.7, converts $\mathcal{P}_3$ into a binary and terminating (not necessarily bounded) computer $\mathcal{P}_4$ for φ with normal consensus output function and no helpers of adjusted size $\mathcal{O}(\mathrm{size}_2(\mathcal{P}_3))$.
- Section 7.8 puts all steps together to show that $\mathcal{P}_4$ has the number of states and number of interactions given by Theorem 9.

Sections 7.3 to 7.7 are self-contained and can be read in any order.

7.1. A $2^{\mathcal{O}(m \log m)} \cdot n^3$ bound on the expected number of interactions

We prove that a bounded binary computer with no helpers of size m terminates within $2^{\mathcal{O}(m \log m)} \cdot n^3$ expected interactions.

To prove this bound, we first introduce *potential functions* in Definition 22. A potential function assigns to every configuration a non-negative *potential*, with the property that executing any transition strictly decreases the potential. In this paper we only consider linear potential functions.

Then, we show in Lemma 23 that bounded population computers have (linear) potential functions, which allows us to show that every run of a bounded computer executes at most $2^{\mathcal{O}(m \log m)} \cdot n$ transitions. However, not every interaction between agents executes a transition. For example, consider a computer with states q, q', q'', a single transition $\{q, q'\} \mapsto \{q'', q''\}$, and a configuration with one agent in each of q and q' and $n-2$ agents in state q''. If we choose two agents uniformly at random, the probability that one of them is in state q and the other in state q' is $2/n(n-1)$. In general, all we can say is that a transition is executed after $\mathcal{O}(n^2)$ interactions in expectation. This leads to $2^{\mathcal{O}(m \log m)} \cdot n^3$ interactions in expectation for the execution of the complete run.

Definition 22. A function $\Phi : \mathbb{N}^Q \to \mathbb{N}$ is *linear* if there exist weights $w : Q \to \mathbb{N}$ s.t. $\Phi(C) = \sum_{q \in Q} w(q) \cdot C(q)$ for every $C \in \mathbb{N}^Q$. 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. In this section we consider only binary computers, but in Section 8 we will consider general ones.

If a population computer has a potential function with maximal weight $W := \max_{q \in Q} w(q)$, then every run executes at most $W \cdot n$ transitions, and so the computer is bounded. We prove that the converse holds for computers in which every state can be populated. That is, if a computer is bounded and every state can be populated, then the computer has a potential function. Observe that the condition that every state can be populated is very mild, since states that can never be populated can be deleted without changing the behaviour of the computer.

Lemma 23. *Let $\mathcal{P}$ be a computer of size m with set of states Q such that for every $q \in Q$ some reachable configuration populates q. Then $\mathcal{P}$ is bounded iff there is a potential function $\Phi(C) = \sum_{q \in Q} w(q) \cdot C(q)$ for $\mathcal{P}$ such that $W := \max_{q \in Q} w(q) \in 2^{\mathcal{O}(m \log m)}$.*

Proof. Let the incidence matrix of $\mathcal{P}$ be the matrix $A \in \mathbb{Z}^{\delta \times Q}$ s.t. the t-th row is $A_t := s - r$, for $t = (r \mapsto s) \in \delta$. In particular, given a vector $y \in \mathbb{N}^\delta$ which assigns each transition a count, $A^\top y$ is the change in the number of agents of each state after executing a sequence of transitions containing $y(t)$ times the transition t. In the following we write $\mathbf{1}$ for the all-ones vector of appropriate dimension. We prove the existence of a (linear) potential function for $\mathcal{P}$ by showing that the following statements are equivalent:

(a) $\mathcal{P}$ is bounded.
(b) $A^\top y \neq \mathbf{0}$ for all $y \in \mathbb{N}^\delta$ with $y \neq \mathbf{0}$.
(c) $A^\top y \neq \mathbf{0}$ for all $y \in \mathbb{R}^\delta_{\geq 0}$ with $y \neq \mathbf{0}$.
(d) $Ax \leq -\mathbf{1}$ for some $x \in \mathbb{R}^Q$.
(e) There is a potential function Φ for $\mathcal{P}$.

(Afterwards, we show the size bound on Φ.)

Let us first give some brief intuition. Essentially, (a) states that $\mathcal{P}$ does not have a loop, i.e. no sequence of transitions leading from a configuration to itself. This is strengthened in (b), which says that no loop exists, even when the protocol is allowed to execute transitions at any time (i.e. the number of agents is allowed to go negative). It is further strengthened in (c), where the computer is also allowed to execute transitions "fractionally"; for a transition $r \mapsto s$, the computer can now for example remove 0.5 agents from each state in r and add 0.5 agents to each state of s. Statement (d) then says that one can find real weights for each state s.t. the total weight of a configuration decreases with each transition. Finally, (e) strengthens this by requiring the weights to be natural numbers.

"(a) $\Rightarrow$ (b)": Assume that (b) does not hold, so there is a nonempty multiset $y \in \mathbb{N}^\delta$ with $A^\top y = \mathbf{0}$. Let $t_1, \ldots, t_k \in \delta$ denote an enumeration of y. Due to the definition of A, $A^\top y = \mathbf{0}$ means that executing the sequence $t_1 t_2 \ldots t_k$ has no effect. Formally, for any C, C' with $C \to_{t_1} \ldots \to_{t_k} C'$ we get $C = C'$. It suffices to find such a configuration C which is reachable; as then we can clearly construct an infinite run, contradicting (a). By assumption, for every state q there exists an initial configuration C_{Iq} and a configuration C_q reachable from C_{Iq} such that $C_q(q) > 0$. It follows that the configuration $C := \sum_{q \in Q} C_q$ is reachable from the initial configuration $C_I := \sum_{q \in Q} C_{Iq}$, and satisfies $C'(q) > 0$ for every $q \in Q$. Multiplying C and C_I by adequate constants, if necessary, we can assume w.l.o.g. that $C(q) \geq k\lambda$, where λ is the maximum arity of any transition. A single transition moves at most λ agents, and so the sequence $t_1, \ldots, t_k$ can be executed at C.

"(b) $\Rightarrow$ (c)": We argue by contraposition and assume that $\{y \in \mathbb{R}^\delta_{\geq 0} \setminus \{\mathbf{0}\} : A^\top y = \mathbf{0}\}$ is not empty. Then there is some $\varepsilon \in \mathbb{Q}$ with $\varepsilon > 0$ s.t. $\{y \in \mathbb{R}^\delta_{\geq 0} : A^\top y = \mathbf{0} \wedge \mathbf{1}^\top y \geq \varepsilon\}$ is not empty either. This is a satisfiable system of linear inequalities and thus has a rational solution $y^* \in \mathbb{Q}^\delta$. As $y^* \geq \mathbf{0}$ we can find a $\mu > 0$ with $\mu y^* \in \mathbb{N}^\delta$, showing the negation of (b).

"(c) ⇒ (d)": Due to $y \geq \mathbf{0}$ the condition $y \neq \mathbf{0}$ is equivalent to $\mathbf{1}^\top y > 0$. In other words, the system $\{y \in \mathbb{R}^\delta_{\geq 0} : A^\top y = \mathbf{0} \wedge -\mathbf{1}^\top y < 0\}$ has no solution. Applying one of the numerous versions of Farkas' lemma (in this case [20, Proposition 6.4.3iii]), we obtain that the system $\{x \in \mathbb{R}^Q : Ax \leq -\mathbf{1}\}$ does have a solution.

"(d) ⇒ (e)": Since $Ax \leq -\mathbf{1}$ is a system of linear inequalities, if it has a solution it also has a rational solution and so, after scaling with an adequate factor, also an integer solution $z^* \in \mathbb{Z}^Q$. Let $w \in \mathbb{N}^Q$ be the vector of natural numbers or *weights* given by $w := \lambda(z^* - z^*_{\min}\mathbf{1})$, where $z^*_{\min} := \min_{q \in Q} z^*(q)$ and λ is the maximum arity of a transition. We define Φ as the linear function induced by w, that is, $\Phi(x) := w \cdot x$.

We show that Φ is a potential function. Let C, C' be configurations such that $C \to C'$. Let $t = (r \mapsto s) \in \delta$ be the transition whose execution leads from C to C'. We prove $\Phi(C) - \Phi(C') > |r|$. By definition of $C' = C - r + s$, and so $\Phi(C') - \Phi(C) = \Phi(s) - \Phi(r)$. Let A_t be the t-th row of A. By the definition of the incidence matrix A we have

$$\Phi(s) - \Phi(r) = (s - r) \cdot w = A_t \cdot w = A_t \cdot \left(\lambda(z^* - z^*_{\min}\mathbf{1})\right) = \lambda\left(A_t \cdot z^* - z^*_{\min} A_t \cdot \mathbf{1}\right)$$

Since z^* is a solution of $Ax \leq -\mathbf{1}$, we have $A_t \cdot z^* \leq -1$. Further, since $|r| = |s|$, we have $A_t \cdot \mathbf{1} = \mathbf{0}$. So $\Phi(C') - \Phi(C) \leq -\lambda \leq -|r|$, and we are done.

"(e) ⇒ (a)": For any initial configuration C_0 we have $\Phi(C_0) \leq W \cdot n$ as Φ is a linear function (Definition 22). Since, by definition, $\Phi(C) \geq 0$ for all configurations C and any transition strictly reduces Φ, a run starting at C_0 can execute at most $\Phi(C_0)$ transitions.

It remains to prove $W \in 2^{\mathcal{O}(m \log m)}$. Recall that the vector of weights is $w := \lambda(z^* - z^*_{\min}\mathbf{1})$, where $z^* \in \mathbb{Z}^Q$ is a solution of the system $Ax \leq -\mathbf{1}$ of linear inequalities. By definition of A, each entry has absolute value at most λ. Using well-known results (see e.g. [23, Lemma 1]), we have $|z^*(q)| \in \mathcal{O}((\lambda|Q|)^{2|Q|}) \subseteq \mathcal{O}(2^{4m \log_2 m})$, hence $w(q) \in \mathcal{O}(2^{5m \log_2 m}) \subseteq 2^{\mathcal{O}(m \log m)}$, and we are done. □

Proposition 24. *Let $\mathcal{P}$ be a bounded binary computer with no helpers of size m. Then $\mathcal{P}$ terminates within $2^{\mathcal{O}(m \log m)} \cdot n^3$ interactions.*

Proof. Without loss of generality we assume that every state can be populated, since removing states which cannot be populated preserves boundedness, correctness and speed. Applying Lemma 23 we obtain that $\mathcal{P}$ has a linear potential function $\Phi(C) = \sum_{q \in Q} w(q) \cdot C(q)$. Let $W := \max_{q \in Q} w(q)$ be the maximal weight of Φ. Since an initial configuration C_0 with n agents fulfils $\Phi(C_0) \leq W \cdot n$, and every transition reduces Φ by at least 1, $\mathcal{P}$ terminates after executing at most $W \cdot n$ transitions. 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 $\mathcal{O}(n^2)$ expected interactions. Hence $\mathcal{P}$ terminates within $\mathcal{O}(W \cdot n^3)$ expected interactions. By Lemma 23 we have $W \in 2^{\mathcal{O}(m \log m)}$, and we are done. □

7.2. Refinement: a technique for proving equivalence

We present a general framework for proving that two computers are equivalent, that is, decide the same predicate.

Definition 25. Let $\mathcal{P} = (Q, \delta, I, O, H)$ and $\mathcal{P}' = (Q', \delta', I', O', H')$ be population computers. $\mathcal{P}'$ *refines* $\mathcal{P}$ if there is a mapping $\pi : \mathbb{N}^{Q'} \to \mathbb{N}^Q$ satisfying the following properties:

1. For all reachable configurations $C, D \in \mathbb{N}^{Q'}$, if $C \to D$ then $\pi(C) \to \pi(D)$.
2. $I = I'$; further, for every initial configuration C of $\mathcal{P}'$ the configuration $\pi(C)$ is an initial configuration of $\mathcal{P}$ such that $\pi(C)(q) = C(q)$ for $q \in I$ (i.e. C and $\pi(C)$ coincide on all initial states).
3. For every reachable terminal configuration $C \in \mathbb{N}^{Q'}$, the configuration $\pi(C)$ is terminal and $O(\mathrm{supp}(\pi(C))) = O'(\mathrm{supp}(C))$.

Usually π is chosen as a linear function of the form $\pi(C)(q) = \lambda_q C(q)$ for adequate coefficients λ_q. Intuitively, by choosing $\lambda_q = 0$ the function π *discards* information about the number of agents in q, and so in this sense $\mathcal{P}'$ is a refinement of $\mathcal{P}$: the configuration C' contains all the information of C, and more.

We prove that if $\mathcal{P}'$ is terminating and refines $\mathcal{P}$, then $\mathcal{P}$ and $\mathcal{P}'$ are equivalent. (Recall that a population computer is terminating if every fair run is finite, and bounded if every run, fair or not, is finite.)

Lemma 26 *(Refinement lemma). Let $\mathcal{P}, \mathcal{P}'$ denote population computers. If $\mathcal{P}'$ is terminating and refines $\mathcal{P}$, then $\mathcal{P}'$ decides the same predicate as $\mathcal{P}$.*

Proof. Let $C_0 \in \mathbb{N}^{Q'}$ denote an arbitrary initial configuration of $\mathcal{P}'$. We decompose $C_0 =: C_I + C_H$ into an input configuration $C_I \in \mathbb{N}^{I'}$ and a helper configuration $C_H \in \mathbb{N}^{\mathrm{supp}(H')}$, $C_H \geq H$. As $\mathcal{P}'$ is terminating, there is a terminal configuration C with $C_0 \to C$. It now suffices to show $O'(\mathrm{supp}(C)) = \varphi(C_I)$. For this, we use properties 1-3 of the definition of refinement. By

property 2, $\pi(C_0)$ is an initial configuration of $\mathcal{P}$ and $\pi(C_0) = C_I + D_H$, where $D_H \in \mathbb{N}^H$, $D_H \geq H$ is a helper configuration of $\mathcal{P}$. So we now only need to show that $\mathcal{P}$ outputs $O'(\text{supp}(C))$ on C_I, i.e. that there is a terminal configuration $D \in \mathbb{N}^Q$ reachable from $\pi(C_0)$ with $O(\text{supp}(D)) = O'(\text{supp}(C))$. We set $D := \pi(C)$. By property 1 we have $\pi(C_0) \to \pi(C)$, and by property 3 we get that $\pi(C)$ is terminal. Finally, property 3 also implies $O(\text{supp}(\pi(C))) = O'(\text{supp}(C))$. □

In the next sections we prove that some protocol $\mathcal{P}'$ is equivalent to $\mathcal{P}$ by exhibiting a suitable refinement function, and showing that $\mathcal{P}'$ is terminating or bounded (recall that bounded computers are terminating).

7.3. A preprocessing step

For translating a bounded population computer into an equivalent population protocol it is convenient to assume that the computer satisfies some technical conditions. We describe a conversion Preprocess that, given any bounded population computer, outputs an equivalent bounded computer satisfying the conditions. The conversion is particularly simple, and so we also use it to illustrate how we will proceed in the coming sections. After giving a brief high-level overview, we describe the specification of the conversion, i.e. the assumptions on the input computer and the properties that the output computer must satisfy. Then we present Preprocess, and finally we prove that Preprocess satisfies the specification. Among other properties, the specification of the conversion requires the input and output computers to be equivalent. For the equivalence proof we use the notion of refinement introduced in Section 7.2.

Let $\mathcal{P} = (Q, \delta, I, O, H)$ denote a bounded population computer deciding a predicate double(φ). We need two additional conditions, namely that states in I have no incoming transitions, and that every configuration in $\mathbb{N}^I$ is terminal. To achieve this, the idea is to add two types of information.

1. Every agent gets an additional flag. As long as the flag is not set, the agent is not allowed to perform any transition of $\mathcal{P}$.
2. Add a helper state h which gives a start signal to any agent q it meets (i.e. sets the flag), allowing q to start computing.

Without the helper, the computation cannot start, showing that $C \in \mathbb{N}^I$ is always terminal. The new input states have no incoming transitions since the flag can never be unset. The refinement π in this case removes the extra information in form of the flag, and disregards helpers in h entirely.

7.3.1. Specification

Specification: Preprocess

Input: Bounded population computer $\mathcal{P} = (Q, \delta, I, O, H)$.
Output: Equivalent bounded population computer $\mathcal{P}' = (Q', \delta', I', O', H')$ of size $\mathcal{O}(\text{size}(\mathcal{P}))$ such that

1. states in I' have no incoming transitions,
2. all configurations in $\mathbb{N}^{I'}$ are terminal and
3. $r(q) \leq 1$ for every $q \in I'$ and $(r \mapsto s) \in \delta'$.

7.3.2. *Conversion* Preprocess

Given a population computer $\mathcal{P} = (Q, \delta, I, O, H)$, we define the computer $\mathcal{P}' = (Q', \delta', I', O', H')$ as follows:

- $Q' := Q \cup \{h\} \cup \{x_* : x \in I\}$,
- $\delta' := \delta \cup \{t_x := (x_*, h \mapsto x, h) : x \in I\}$,
- $I' := \{x_* : x \in I\}$,
- $O'(S) := O(S \cap Q)$, for $S \subseteq Q'$, and
- $H'(q) := H(q)$ for $q \in Q$ and $H(h) := 1$.

7.3.3. Correctness

Proposition 27. Preprocess *satisfies its specification (page 20).*

Proof. We proceed in several steps.

Claim 1. $\mathcal{P}'$ refines $\mathcal{P}$.
We define the refinement π as follows: $\pi(q) = q$ for every $q \in Q$, $\pi(h) = 0$ and $\pi(x_*) = x$ for every $x \in I$. We prove that π

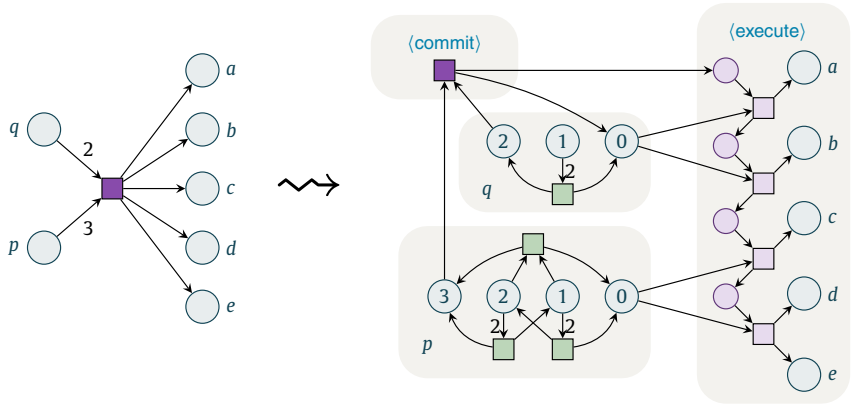


Fig. 4. Simulating the 5-way transition $\{3 \cdot p, 2 \cdot q \mapsto a, b, c, d, e\}$ by binary transitions.

satisfies the three properties of Definition 25. For property 1, observe that the new transitions $t_x = (r \mapsto s)$ we added fulfil $\pi(r) = \pi(s)$, and for all old transitions the result is clear because of $\pi(q) = q$ for all $q \in Q$. Property 2 follows immediately from the definition. For property 3, let $C \in \mathbb{N}^{Q'}$ be a reachable terminal configuration. Since the helper cannot leave h, we have $C(h) > 0$. If $C(x_*) > 0$ for any $x \in I$, then t_x is enabled, contradiction to C being terminal. So $C(x_*) = 0$ for every $x \in I$. Since $\pi(h) = 0$ and $\pi(q) = q$ for all $q \in Q$, we obtain $\pi(C) = C|_Q$. Since C is terminal in $\mathcal{P}'$, the smaller configuration $C|_Q$ is also terminal in $\mathcal{P}'$. Then $C|_Q$ is terminal in $\mathcal{P}$, since $\delta \subseteq \delta'$. The outputs agree by definition of O'.

Claim 2. $\mathcal{P}'$ is bounded.
Every occurrence of some t_x reduces the number of agents in I', therefore t_x occurs finitely often. Between any two of these occurrences, only finitely many other steps can occur, since $\mathcal{P}$ is bounded.

Claim 3. $\mathcal{P}'$ decides φ.
By claims 1 and 2, and the Refinement lemma (Lemma 26).

Claim 4. $\mathcal{P}'$ has size $\mathcal{O}(\text{size}(P))$ and satisfies properties 1.-3. of the specification.
By direct inspection of the transition function. □

7.4. 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 conversion introduced in [12, Lemma 3], which works for arbitrary transitions $r \mapsto s$, is too slow. In [12], the 3-way transition $t : q_1, q_2, q_3 \mapsto q_1', q_2', q_3'$ is simulated by the transitions

$$t_1 : q_1, q_2 \mapsto w, q_{12} \qquad t_2 : q_{12}, q_3 \mapsto c_{12}, q_3' \qquad t_3 : c_{12}, w \mapsto q_1', q_2' \qquad \overline{t}_1 : w, q_{12} \mapsto q_1, q_2$$

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 $\overline{t}_1$. The construction for a k-way transition has "revert" transitions $\overline{t}_1, \ldots, \overline{t}_{k-2}$. As in Example 4 and Example 5, 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 $|\text{supp}(r)| \leq 2$. Fig. 4 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 $(p, 0), \ldots, (p, 3)$ and $(q, 0), \ldots, (q, 2)$. 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 ⟨commit⟩. After committing, they move to the states $a, \ldots, e$ together with the other members of the group, who are "waiting" in the states $(p, 0)$ and $(q, 0)$. Note that ⟨commit⟩ 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.

The rest of this section is split into three parts. We first describe the formal specification of the conversion. Section 7.4.1 describes the conversion itself, which we call *Binarise*, formally. Section 7.4.2 shows that *Binarise* satisfies the specification.

Specification: Binarise

Input: Bounded population computer $\mathcal{P} = (Q, \delta, I, O, H)$.
Output: Equivalent bounded binary population computer $\mathcal{P}' = (Q', \delta', I', O', H')$ with adjusted size $\mathcal{O}(\beta \cdot \text{size}(\mathcal{P}))$, where $\beta \leq |Q|$. Additionally:

1. If every state of $\mathcal{P}$ but one has at most 2 outgoing transitions, then $\beta \leq 3$.
2. If no state in I has incoming transitions, then neither do states in I'.
3. If all configurations in $\mathbb{N}^I$ are terminal and $r(q) \leq 1$ for $q \in I$ and $(r \mapsto s) \in \delta$, then all configurations in $\mathbb{N}^{I'}$ are terminal.

7.4.1. Conversion Binarise

Given a bounded population computer $\mathcal{P} = (Q, \delta, I, O, H)$, we construct a binary population computer $\mathcal{P}' = (Q', \delta', I', O', H')$. Let $m(q) := \max\{r(q) : (r \mapsto s) \in \delta\}$ denote the maximum multiplicity of any outgoing transition of q. For each state q we allow up to $m(q)$ agents to "stack" in q.

Formally, we add states $\{(q, i) : q \in Q, i = 0, ..., m(q)\}$ to Q', and the following transitions, for $q \in Q$, $i, j \in \{1, ..., m(q) - 1\}$, to δ':

$$\begin{aligned} &(q, i), (q, j) \mapsto (q, i + j), (q, 0) && \text{if } i + j \leq m(q) \\ &(q, i), (q, j) \mapsto (q, m(q)), (q, i + j - m(q)) && \text{if } i + j \geq m(q). \end{aligned} \qquad \langle \text{stack} \rangle$$

Intuitively, an agent in state (q, i) "owns" i agents in state q, meaning that it certifies that $i - 1$ additional agents are in $(q, 0)$. Consider a transition $t = (r \mapsto s) \in \delta$ of $\mathcal{P}$ with $\text{supp}(r) = \{q, p\}$. Executing t in $\mathcal{P}$ requires $|r|$ agents. In $\mathcal{P}'$, the transition is simulated by a sequence of binary transitions. The simulation is started by any pair of agents that together own at least r agents. Assume these agents are in states (q, i) and (p, j) with $i + j \geq |r|$. The transition $\langle \text{commit} \rangle$ initiating the simulation designates one of the agents, say q, as primary agent, and p as secondary agent. The primary agent is responsible for executing the rest of the simulation. Transition $\langle \text{commit} \rangle$ moves the primary agent from (q, i) to $(q, i - r(q), t)$ and the secondary agent from (p, j) to $(p, j - r(p))$; intuitively, the agents together "designate" $|r|$ agents to execute t.

Formally, we add states $\{(q, i, t) : i = 0, ..., m(q)\}$ to Q'. For every q, p in Q, if $q \neq p$ we add transitions

$$(q, i), (p, j) \mapsto (q, i - r(q), t), (p, j - r(p)) \qquad \text{for } i \geq r(q), j \geq r(p) \qquad \langle \text{commit} \rangle$$

to δ. If $p = q$ then for every i, j with $i + j \geq r(q)$ we add transitions

$$\begin{aligned} &(q, i), (q, j) \mapsto (q, i + j - r(q), t), (q, 0) && \text{if } i + j - r(q) \leq m(q) \\ &(q, i), (q, j) \mapsto (q, i + j - r(q) - m(q), t), (q, m(q)) && \text{else} \end{aligned} \qquad \langle \text{commit} \rangle$$

After the execution of $\langle \text{commit} \rangle$, the primary agent transfers ownership of its remaining agents (if any) to another agent by means of a transition $\langle \text{transfer} \rangle$. Formally, we add to δ' transitions

$$(q, i, t), (q, 0) \mapsto (q, 0, t), (q, i) \qquad \text{for } i = 1, ..., m(q) \qquad \langle \text{transfer} \rangle$$

The primary agent now proceeds with the simulation of the execution of t. Formally, we add states $\{(t, i) : i = 1, ..., |r|\}$ to Q'. Let $s_1, ..., s_l$ denote an enumeration of the multiset s of t, with $l := |s|$. Intuitively, an agent in (t, i) moves one agent into $(s_i, 1)$, and then goes to $(t, i + 1)$. Instead of moving an agent into $(t, 1)$ via a transition, we identify $(q, 0, t)$ with $(t, 1)$ directly. Additionally, we identify (t, l) with $(s_l, 1)$, so that we do not have to create a special transition for the last agent. Accordingly, we formally define the last set of transitions added to δ' as follows, for $i = 1, ..., l - 1$.

$$\begin{aligned} &(t, i), (p, 0) \mapsto (t, i + 1), (s_i, 1) && \text{if } i \leq r(p) \\ &(t, i), (q, 0) \mapsto (t, i + 1), (s_i, 1) && \text{if } i > r(p) \end{aligned} \qquad \langle \text{execute} \rangle$$

(Observe that, as specified above, $\mathcal{P}'$ is not deterministic. For some of the transitions $\langle \text{stack} \rangle$ and $\langle \text{commit} \rangle$ it may be the case that e.g. $\langle \text{stack} \rangle = (r \mapsto s_1)$ and $\langle \text{commit} \rangle = (r \mapsto s_2)$ for the same r and $s_1 \neq s_2$. However, if that happens we delete all but one of these transitions to ensure that the protocol is deterministic. When choosing which transition to keep, we prefer $\langle \text{commit} \rangle$ to $\langle \text{stack} \rangle$, but otherwise pick an arbitrary one.)

We retain the original input states and helpers, by identifying each $q \in Q$ with $(q, 1)$. For the output function we define $O'(S) := O(\{q : (q, i) \in S\})$ for any S. Note, however, that a circuit for O grows by at most a factor of 3, as $(q, i) \in \text{supp}(C) \Rightarrow (q, 0) \in \text{supp}(C)$ for $i \geq 2$ and any reachable configuration C and state q, so it suffices to check for $(q, 0)$ and $(q, 1)$.

7.4.2. Correctness

Proposition 28. Binarise *satisfies its specification (page 22).*

Proof. We first show that $\mathcal{P}'$ refines $\mathcal{P}$.

Claim 1. $\mathcal{P}'$ refines $\mathcal{P}$.
To begin, let us introduce the mapping between configurations of $\mathcal{P}'$ and $\mathcal{P}$ describing the refinement. We define $\pi : Q' \to \mathbb{N}^Q$ by setting

$$\begin{aligned} \pi((q,i)) &:= q \cdot i \\ \pi((q,i,t)) &:= q \cdot i + s \qquad \text{for all } q \in Q, t = (r \mapsto s) \in \delta \text{ and } i \\ \pi((t,i)) &:= s_i + \ldots + s_l \end{aligned}$$

This uses the same enumeration of s as above. Clearly, π is well-defined, as $\pi((q,0,t)) = s = \pi((t,1))$ and $\pi((t,l)) = s_l = \pi((s_l,1))$. Finally, we extend π to a linear mapping $\pi : \mathbb{N}^{Q'} \to \mathbb{N}^Q$ in the obvious fashion. Now we prove that π fulfils the properties required by Definition 25:

1. Note that π is invariant under execution of ⟨stack⟩, ⟨transfer⟩ and ⟨execute⟩. Additionally, for a $t \in \delta$ and a corresponding ⟨commit⟩ transition t', we find that $C \to_{t'} C'$ implies $\pi(C) \to_t \pi(C')$ for all $C, C' \in \mathbb{N}^{Q'}$.
2. We identified $q \in Q$ with $(q,1)$ and set $\pi((q,1)) := q$, so $I = I'$ and $\pi(C) = C$ for all $C \in \mathbb{N}^Q$ follows.
3. Let $C \in \mathbb{N}^{Q'}$ be a reachable terminal configuration. We have to show that $\pi(C)$ is terminal and satisfies $O(\mathrm{supp}(\pi(C))) = O'(\mathrm{supp}(C))$. The latter condition follows immediately from the definition of O. To show that $\pi(C)$ is terminal, we first observe that transition ⟨transfer⟩ is always enabled if an agent is in state (q,i,t), with $t = (r,s) \in \delta$ and $i \geq 1$, as that state "owns" $i + r(q)$ agents in q. Hence there must be $i + r(q) - 1 \geq i \geq 1$ agents in $(q,0)$. By the same line of argument, ⟨execute⟩ is always enabled if an agent is in (t,i), for $t \in \delta$ and $1 \leq i < |s|$. Now, assume $\pi(C)$ is not terminal, so there is some transition $t = (r \mapsto s) \in \delta$ which is enabled at $\pi(C)$. As we have just argued, all agents of C are in states (q,i), for $q \in Q$ and $i \in \{0, \ldots, m(q)\}$. Transition ⟨commit⟩ is not enabled at C, wherefore one of the states $q \in Q$ used by t fulfils $i < r(q)$ for all i with $C((q,i)) > 0$. But t is enabled at $\pi(C)$, so $\pi(C)(q) \geq r(q)$ and, by definition of π, there are $i, j > 0$ s.t. C contains both an agent in (q,i) and one in (q,j). Due to our choice of m, we have $r(q) \leq m(q)$, and therefore $0 < i, j < m(q)$, wherefore transition ⟨stack⟩ is enabled, contradicting that C is terminal. So $\pi(C)$ is terminal.

Claim 2. $\mathcal{P}'$ is bounded.
Assume an infinite run $C_0, C_1, \ldots$ of $\mathcal{P}'$ exists. If transition ⟨commit⟩ is executed infinitely often in that run, at steps $i_0, i_1, \ldots \in \mathbb{N}$, then $\pi(C_{i_0}), \pi(C_{i_1}), \ldots$ would be an infinite run of $\mathcal{P}$, contradicting that $\mathcal{P}$ is bounded. Hence there is an infinite suffix of $C_0, C_1, \ldots$ in which ⟨commit⟩ is never fired.

In this suffix the number of agents in a state $(q,i,t) \in Q'$ with $i > 0$ cannot increase, but decreases whenever ⟨transfer⟩ is executed. Hence this also happens only finitely often and the number of agents in a state (t,i) cannot increase beyond a point. As ⟨execute⟩ increases i, it too must occur only finitely often. The only transition left is ⟨stack⟩, which always increases the number of agents in either $(q,0)$ or $(q,m(q))$, for some $q \in Q$.

Claim 3. $\mathcal{P}'$ decides φ.
By claims 1 and 2, and the Refinement lemma (Lemma 26).

Claim 4. $\mathcal{P}'$ has adjusted size $\mathcal{O}(\beta \cdot \mathrm{size}(\mathcal{P}))$ for some $\beta \leq |Q|$.
Let β_q denote the number of transitions $t \in \delta$ for which $q \in Q$ is the primary agent, and set $\beta := \max\{\beta_q : q \in Q\}$. It suffices to show $|Q'| \leq (\beta + 2)\,\mathrm{size}(\mathcal{P})$. We begin by bounding the total value of $m(q)$. Clearly, $m(q) \leq \sum_{(r \mapsto s) \in \delta} r(q)$ for $q \in Q$, and thus $\sum_{q \in Q} m(q) \leq \sum_{t \in \delta} |t|$. For every $q \in Q$ we create $m(q) + 1$ states and for every $t \in \delta$ we create $|t|$ states. Additionally, we create states for every transition t using state q as primary agent, so at most $\beta m(q)$. In total, we create at most $(\beta + 2)\sum_{t \in \delta} |t| + |Q|$ states.

Claim 5. $\mathcal{P}'$ satisfies conditions 1.-3. of the specification.
For condition 1., let $q \in Q$ denote the state with the most outgoing transitions. For our conversion, we can simply not choose q as primary agent, for all transitions that also use a different agent. There is at most one other transition (using only agents in q), so every state is chosen as primary agent of at most 3 transitions and $\beta \leq 3$. Condition 2. is obvious from the conversion. For condition 3., note that the condition implies $m(q) \leq 1$ for all $q \in I$. Therefore the only transition using an agent in I' is ⟨commit⟩, which is not enabled at C if $\pi(C)$ is terminal. □

7.5. 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).

Our starting point is some bounded and binary computer $\mathcal{P} = (Q, \delta, I, O, H)$, e.g. as constructed in Section 7.4. 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 $\mathcal{P}$.

Specification: Focalise

Input: Bounded binary population computer $\mathcal{P} = (Q, \delta, I, O, H)$.
Output: Equivalent bounded binary computer $\mathcal{P}' = (Q', \delta', I', O', H')$ with adjusted size $\mathcal{O}(\text{size}_2(\mathcal{P}))$ using a marked consensus output. Additionally:

1. If no state in I has incoming transitions, then neither do states in I'.
2. If all configurations in $\mathbb{N}^I$ are terminal, then so are all configurations in $\mathbb{N}^{I'}$.

7.5.1. Conversion Focalise

Formally, the intuition above corresponds to a partition of the state space into four parts, $Q = Q_{\text{orig}} \cup Q_{\text{supp}} \cup Q_{\text{gate}} \cup Q_{\text{reset}}$, with the crucial property that no transition allows agents to change partition. The only exceptions are leader elections producing/removing reset agents. In the following we first describe these different parts of the partition, followed by the transitions. The set $Q_{\text{orig}} := Q \times \{-, +\}$ essentially executes $\mathcal{P}$ as was already described in the intuitive explanation above.

The output 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 $\perp$.

The sets Q_{supp} and Q_{gate} are now the disjoint union of the above over all states/gates, in total we therefore obtain $Q_{\text{supp}} := Q \times \{0, 1, !\}$ and $Q_{\text{gate}} := G \times \{0, 1, \perp\}^3$.

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 $Q_{\text{reset}} := \{0, \ldots, |Q| + |G|\}$, initially in state 0. Whenever a reset is triggered, the reset-helper contacts all other $|Q| + |G|$ helpers in round-robin fashion, asking them to reset the computation.

We now specify the required transitions. First, we need to refine the original protocol, requesting to recompute the support with each transition. Different occurrences of $\pm$ need not match.

$$(q, \pm), (p, \pm) \mapsto (q', +), (p', -) \qquad \text{for } (q, p \mapsto q', p') \in T \qquad \langle\text{execute}\rangle$$

It suffices to reset once, so for the purpose of speed we clear superfluous flags.

$$(q, +), (p, +) \mapsto (q, +), (p, -) \qquad \text{for } q, p \in Q \qquad \langle\text{denotify}\rangle$$

To keep the protocol deterministic, we remove all ⟨denotify⟩ transitions which could also initiate another transition (in particular ⟨execute⟩). The support is computed by setting the stored bit to '!' once the corresponding state has been observed.

$$(q, 0), (q, -) \mapsto (q, !), (q, -) \qquad \text{for } q \in Q \qquad \langle\text{detect}\rangle$$

To define the transitions for Q_{gate}, we need to introduce some notation. First, we write nand for the NAND function, i.e. $\text{nand}(i, j) := \neg(i \wedge j)$ for $i, j \in \{0, 1\}$ and $\text{nand}(i, j) := \perp$ otherwise. We also use E_1, E_2 to denote the first and second

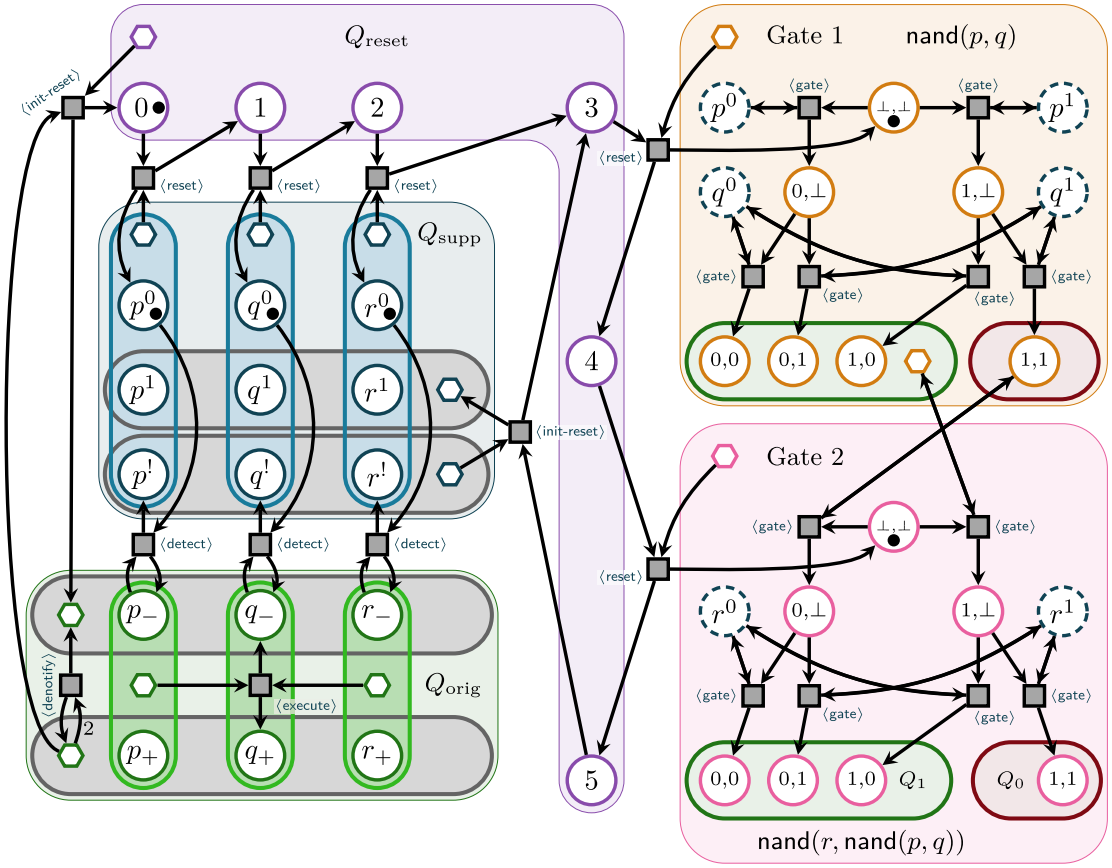


Fig. 5. Figure visualising the conversion to marked consensus output for a population computer with states $\{p, q, r\}$, a single transition $p, q \mapsto r, r$ and output function $\neg(r \wedge \neg(p \wedge q))$.
State names are abbreviated: x^y and x_y are used instead of (x, y), and only the last two components of states in Q_{gate} are shown. Transitions are drawn using Petri net notation. The same state may appear multiple times. Occurrences beyond the first are drawn with a dashed border.
Hexagons indicate wildcards. Each hexagon is associated with a group of states (indicated by shaded areas). Used as input to a transition, it denotes that any agent of that group can be used (i.e. they create multiple copies of the transition). Transition may use hexagons as both input and output, in which case the output hexagon refers to the state corresponding to the state used for the input hexagon. For example, transition ⟨denotify⟩ takes two agents x_+, y_+, with $x, y \in \{p, q, r\}$ and produces an agent in x_- and one in y_+.
We omit ⟨leader⟩ transitions and the third case of ⟨init-reset⟩ from the drawing. Please note, however, that the resulting computer is still correct under the assumption that no superfluous helpers are provided. (This is equivalent to using leaders instead of helpers.)

component of E, and write S_g^b for the set of states indicating that gate $g \in Q \cup G$ has truth value $b \in \{0, 1\}$, i.e. $S_q^b := \{(q, b)\}$ for $q \in Q$ and $S_g^b := \{g, b\} \times \{0, 1\}^2$ for $g \in G$. We add the following transitions, for any gate $g \in G$ and $b, i \in \{0, 1\}$.

$$\begin{aligned} (g, \bot, \bot, \bot), q &\mapsto (g, \bot, b, \bot), q && \text{for } q \in S^b_{E_1(g)} \\ (g, \bot, i, \bot), q &\mapsto (g, \mathsf{nand}(i, b), i, b), q && \text{for } q \in S^b_{E_2(g)} \end{aligned} \qquad \langle\text{gate}\rangle$$

These transitions perform the computation of the gate, by initialising the first and then the second input. Once the second input is initialised, the output of the gate is set accordingly. We now specify the required transitions. An example of the resulting protocol is depicted in Fig. 5.

There are two kinds of resets; depending on whether the agents in Q_{supp} are affected. (The gates are always reset.) Both resets are executed by an agent in Q_{reset}, who goes through the other agents one by one. Let $q_1, \ldots, q_{|Q|}$ denote an enumeration of Q. For G any enumeration is not enough, input gates have to be reset first. Therefore let $g_1, \ldots, g_{|G|}$ be a topological sorting of G.

$$\begin{aligned} i - 1, (q_i, b) &\mapsto i, (q_i, 0) && \text{for } (q_i, b) \in Q_{\text{supp}} \\ |Q| + i - 1, (g_i, b_1, b_2, b_3) &\mapsto |Q| + i, (g_i, \bot, \bot, \bot) && \text{for } (g_i, b_1, b_2, b_3) \in Q_{\text{gate}} \end{aligned} \qquad \langle\text{reset}\rangle$$

There are three ways to initiate a reset:

$$\begin{aligned} (q, +), i &\mapsto (q, -), 0 && \text{for } (q, +) \in Q_{\text{orig}}, i \in Q_{\text{reset}} \\ (q, !), i &\mapsto (q, 1), \min\{i, |Q|\} && \text{for } (q, !) \in Q_{\text{supp}}, i \in Q_{\text{reset}} \\ i, j &\mapsto 0, (q_h, -) && \text{for } i, j \in Q_{\text{reset}} \end{aligned} \qquad \langle\text{init-reset}\rangle$$

First, an agent in Q_{orig} may indicate that the support has changed, and everything will be reset. Second, an agent in Q_{supp} will request that all gates be reset whenever it changes its output. Third, if two agents are in Q_{reset}, the computation so far must be discarded, and we continue with only one of them. The other moves into an arbitrary state $q_h \in Q$ with $H(q_h) > 0$, so it is given back to $\mathcal{P}$ to use for its computations. Picking a state with $H(q_h) > 0$, i.e. a helper state, ensures that this does not affect the correctness of $\mathcal{P}$.

Finally, all states in Q_{supp} and Q_{gate} also participate in a leader election, to ensure that there is only one agent in Q_{supp} for each $q \in Q$, and only one agent for each gate.

$$\begin{aligned} q, q' &\mapsto q, 0 && \text{for } q, q' \in Q_{\text{supp}} \text{ s.t. } q_1 = q'_1 \\ g, g' &\mapsto g, 0 && \text{for } g, g' \in Q_{\text{gate}} \text{ s.t. } g_1 = g'_1 \end{aligned} \qquad \langle \text{leader} \rangle$$

These transitions indirectly cause a reset, by producing an agent in Q_{reset}.

It remains to define the inputs, helpers and outputs. For this, we identify a state $q \in Q$ with $(q, -) \in Q_{\text{orig}}$. We define $I' := I$, as well as $H'(q) := H(q)$ for $q \in Q$ and $H(q) := 1$ for $q \in Q \times \{0\} \cup G \times \{(\bot, \bot, \bot)\} \cup \{0\}$. To define the marked consensus output, we pick the special states $Q_0 = \{(g_{|G|}, 1)\} \times \{0, 1\}^2$ and $Q_1 := \{(g_{|G|}, 0)\} \times \{0, 1\}^2$.

7.5.2. Correctness

Proposition 29. Focalise *satisfies its specification.*

Proof. We first show that $\mathcal{P}'$ refines $\mathcal{P}$, which by Lemma 26 implies that they decide the same predicate. We introduce a mapping $\pi : \mathbb{N}^{Q'} \to \mathbb{N}^{Q}$ to describe the configuration that $\mathcal{P}'$ is representing. For all C we define

$$\pi(C) := \sum_{q \in Q} q \cdot C((q, \pm)) + q_h \cdot (C(Q_{\text{supp}} \cup Q_{\text{gate}} \cup Q_{\text{reset}}) - |Q| - |G| - 1)$$

Recall that $q_h \in Q$ is the state to which superfluous agents are moved, as defined in ⟨init-reset⟩. Eventually, we have exactly one agent for each state in Q, to detect the support, exactly one agent for each gate, and exactly one reset agent, so $|Q| + |G| + 1$ in total. Everything beyond that is superfluous and will be returned to q_h at some point. We prove the following claim:

Claim 1. Let C denote a reachable configuration of $\mathcal{P}'$. Then $C(Q_{\text{reset}}) \geq 1$, $C(\{q\} \times \{0, 1, !\}) \geq 1$ for $q \in Q$, and $C(\{g\} \times \{0, 1, \bot\}^3) \geq 1$ for $g \in G$. If C is terminal, the above hold with equality.
Let $S_q := \{q\} \times \{0, 1, !\}$ for $q \in Q$ and $G_g := \{g\} \times \{0, 1, \bot\}^3$ for $g \in G$. First, note that the sets of states Q_{reset}, S_q and G_g each contain at least one agent in an initial configuration (due to the choice of H'). Additionally, they cannot be emptied, as every transition removing agents from one of these sets also puts at least one agent back. (Using Petri net terminology, they are traps.)

If two agents are in Q_{reset}, the third part of ⟨init-reset⟩ is active and C is not terminal. Similarly, if two agents are in S_q, for some q, or two agents are in G_g, for some g, then ⟨leader⟩ can be executed.

Claim 2. $\mathcal{P}'$ refines $\mathcal{P}$.
We show that π fulfils the properties required by Definition 25. The first two are simple.

1. Observe that $\pi(C)$ is changed only via transition ⟨execute⟩, and that this happens according to a transition $t \in T$.
2. $I = I'$ holds by construction and the remainder follows from $H(q_h) > 0$ and the definition of π.

Property 3 will take up the remainder of this proof. Let C denote a reachable, terminal configuration of $\mathcal{P}'$. Using Claim 1 we get $\pi(C) = \sum_{q \in Q} q \cdot C((q, \pm))$. Therefore, if a transition $t \in \delta$ is enabled at $\pi(C)$, the corresponding ⟨execute⟩ transition is enabled at C. As C is terminal, so is $\pi(C)$.

Finally we have to argue $O'(\text{supp}(C)) = O(\text{supp}(\pi(C)))$. Due to Claim 1 we know that in C we have exactly one agent in either $(q, 0)$, $(q, 1)$, or $(q, !)$. It cannot be in $(q, !)$, as then transition ⟨init-reset⟩ would be enabled.

If it were in $(q, 0)$ but $\pi(C)(q) > 0$, then transition ⟨detect⟩ would be enabled, so that cannot be the case either. Conversely, if it were in $(q, 1)$ but $\pi(C)(q) = 0$, then we also arrive at a contradiction: after the last agent left $(q, \pm)$ via ⟨execute⟩, it must have triggered a reset, which moved the agent to $(q, 0)$ (or, if there were multiple agents in $q \times \{0, 1, !\}$, ⟨leader⟩ would have triggered another reset later). But after that reset $\pi(C)(q) = 0$, so it is impossible to leave $(q, 0)$.

Therefore we find that the agents in Q_{supp} precisely indicate the support of $\pi(C)$. Whenever an agent in Q_{supp} changes its opinion (either due to a ⟨reset⟩ or ⟨detect⟩), all gates will be reset. So there is some point at which the opinions of agents in Q_{supp} have stabilised (in particular, all inequalities of Claim 1 are tight, else there would be another reset) and the unique agent in Q_{reset} is in state $|Q|$, i.e. it is in the process of resetting all gates. As the gates are reset in order of some topological sorting (so a gate is reset after its inputs are), a gate will only assume a value after its inputs have stabilised and therefore compute the correct value according to the circuit. As the circuit computes $O(\text{supp}(\pi(C)))$, the statement follows.

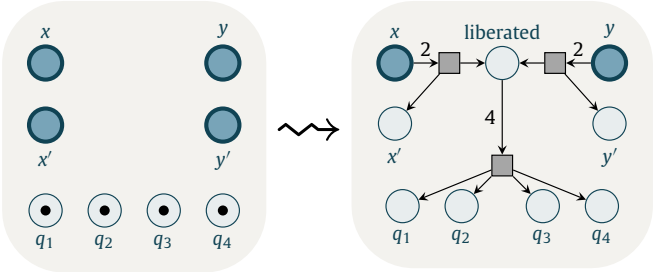


Fig. 6. Illustration in graphical Petri net notation (see Section 3) of the conversion that removes helpers. Initial states are highlighted.

Claim 3. $\mathcal{P}'$ is bounded.
Due to Claim 2 we know that $\pi(C)$ can change only finitely often, as $\mathcal{P}$ is bounded, and thus transition ⟨execute⟩ can be executed only finitely often. After that, the number of agents in a state $Q \times \{+\} \subseteq Q_{\text{orig}}$ cannot increase, but can always decrease using the first ⟨init-reset⟩ transition. So eventually no agents remain in those states.

Parallel to that, both $C(Q_{\text{supp}})$ and $C(Q_{\text{gate}})$ cannot increase. Whenever $C(Q_{\text{supp}}) > |Q|$ or $C(Q_{\text{gate}}) > |G|$, ⟨leader⟩ is enabled and decreases one of them, until $C(Q_{\text{supp}}) = |Q|$ and $C(Q_{\text{gate}}) = |G|$. Afterwards, ⟨leader⟩ cannot fire again (note Claim 1), and $C(Q_{\text{reset}})$ cannot increase. If $C(Q_{\text{reset}}) > 1$, the third case of ⟨init-reset⟩ will reduce this number, until $C(Q_{\text{reset}}) = 1$.

To summarise, eventually no agents remain in $Q \times \{+\}$ and all inequalities of Claim 1 become tight. At that point, the first and third case of ⟨init-reset⟩ are disabled, and it is not possible for the agent in Q_{reset} to lower its value to below $|Q|$. Via ⟨reset⟩, it will thus eventually arrive at $|Q|$, and the first part of ⟨reset⟩ cannot be executed again.

Once that happens, agents cannot enter states $Q \times \{0\} \subseteq Q_{\text{supp}}$, such that ⟨detect⟩ can never occur any more. This then causes ⟨init-reset⟩ to eventually be fully disabled, and then ⟨reset⟩ as well. Finally, transition ⟨gate⟩ can then fire only finitely often, and the protocol terminates.

Claim 4. $\mathcal{P}'$ decides φ.
Follows from claims 2 and 3, and the Refinement lemma (Lemma 26).

Claim 5. If no state in I has incoming transitions, then neither do states in I'.
Note that $I' = I \times \{-\}$ by definition. If no state in I has incoming transitions, then it is not possible to put an agent into $I \times \{+\}$, as that happens only via ⟨execute⟩. Therefore transitions ⟨denotify⟩ or the first part of ⟨init-reset⟩ cannot move an agent from $(q, +)$ to $(q, -)$, for $q \in I$. Finally, note that $q_h \notin I'$, as $\text{supp}(H) \cap I = \emptyset$ by the definition of population computers. (So, to be precise, the statement only holds once we modify our construction to delete unused states and transitions.)

Claim 6. If all configurations in $\mathbb{N}^I$ are terminal, then so are all configurations in $\mathbb{N}^{I'}$.
Note that the only transition which can execute from a configuration in $\mathbb{N}^{I'}$ is ⟨execute⟩, but that requires a transition in $\mathcal{P}$ which can execute in a configuration in $\mathbb{N}^I$. □

7.6. Removing helpers

We convert a bounded binary computer $\mathcal{P}$ deciding the predicate double(φ) over variables $x_1, ..., x_k, x'_1, ..., x'_k$ into a computer $\mathcal{P}'$ with no helpers deciding φ over variables $x_1, ..., x_k$. In [11], a protocol with helpers and set of states Q is converted into a protocol without helpers with states $Q \times Q$. We provide a better conversion, called Autarkify, that avoids the quadratic blowup.

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 $\text{double}(\varphi)(..., x_i, ..., x'_i, ...) = \text{double}(\varphi)(..., x_i - 2, ..., x'_i + 1, ...)$ holds by definition of double(φ).

Fig. 6 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 $\mathcal{P}'$ 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 double(φ) and $\text{double}(\varphi)(x, y, 0, 0) = \varphi(x, y)$ by the definition of double(φ), 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. This is where the assumption that all configurations in $\mathbb{N}^I$ are terminal is used: Before the first full batch of liberated agents is dispatched, $\mathcal{P}$ cannot execute any transition.

Recall the definition of double(φ) for some $\varphi \in$ *QFPA* (Definition 6): for every variable x_i in φ create copies x_i and x_i'; afterwards replace every occurrence of x_i by $x_i + 2x_i'$. The specification of the construction is as follows:

Specification: Autarkify

Input: Bounded binary population computer $\mathcal{P} = (Q, \delta, I, O, H)$ with marked consensus output deciding double(φ) for some $\varphi \in$ *QFPA*; further, states in I have no incoming transitions and every configuration in $\mathbb{N}^I$ is terminal.

Output: Bounded binary population computer $\mathcal{P}' = (Q', \delta', I', O', \emptyset)$ of adjusted size $\mathcal{O}(\text{size}_2(\mathcal{P}))$ without helpers and with a marked consensus output deciding φ for inputs of size at least $|I| + 2|H|$.

7.6.1. Conversion Autarkify

If $\mathcal{P}'$ did not have to be binary, we could just add to $\mathcal{P}$ a state h and transitions

$$x, x \mapsto x', \mathsf{h} \qquad \text{for } x \in I \tag{double}$$

$$|H| \cdot \mathsf{h} \mapsto H \tag{helper}$$

Instead, we inline the conversion from Section 7.4 (simplified slightly), and define $\mathcal{P}'$ as follows. We add to $\mathcal{P}$ states $Q_{\text{helper}} := \{\triangle_i, \triangledown_i : i = 0, ..., |H|\}$, and identify $\triangle_1$ with h and $\triangledown_1$ with h_1; here, $h_1, ..., h_m$ is an enumeration of H. For $i, j \in \{1, ..., |H| - 1\}$ we also add transitions

$$\begin{aligned} \triangle_i, \triangle_j &\mapsto \triangle_{i+j}, \triangle_0 && \text{if } i + j < |H| \\ \triangle_i, \triangle_j &\mapsto \triangledown_{|H|}, \triangle_{i+j-|H|} && \text{if } i + j \geq |H| \\ \triangledown_{i+1}, \triangle_0 &\mapsto \triangledown_i, h_{i+1} \end{aligned} \tag{helper}$$

Finally, we set $I' := I$ and $O'(S) := O(S \cap Q)$ for all $S \subseteq Q'$ (note that $Q \subseteq Q'$).

7.6.2. Correctness

Proposition 30. Autarkify *satisfies its specification (page 28).*

Proof. As for the other conversions, we define a linear map $\pi : \mathbb{N}^{Q'} \to \mathbb{N}^Q$ to translate configurations of $\mathcal{P}'$ to ones of $\mathcal{P}$. Here, we simply choose $\pi(C)(q) := C(q)$ for $q \in Q$. We do not, however, show that $\mathcal{P}'$ refines $\mathcal{P}$, as that does not hold: transitions ⟨double⟩ and ⟨helper⟩ change $\pi(C)$ in a way that is not compatible with an execution of $\mathcal{P}$. Instead, we start by showing that it suffices to consider only certain transition sequences, where the ⟨double⟩ and ⟨helper⟩ transitions occur only in the beginning. After that point, our proof proceeds just as for the refinement results.

Let $\sigma_1, \sigma_2, \sigma_3 \in (\delta')^*$ denote (finite) sequences of transitions, where σ_1 contains only ⟨double⟩ transitions, σ_2 only ⟨helper⟩ transitions, and $\sigma_3 \in \delta^*$. We call such a sequence $\sigma := \sigma_1\sigma_2\sigma_3$ *good*. We first make the following claim:

Claim. If $C \in \mathbb{N}^{Q'}$ is reachable from an initial configuration $C_0 \in \mathbb{N}^I$, then there is a good sequence $\sigma \in (\delta')^*$ with $C_0 \to_\sigma C$. Let us prove the claim. Since states $x \in I$ have no incoming transitions (precondition of the specification), the number of agents in x is monotonically decreasing during a run. Hence a ⟨double⟩ transition can always be moved to any earlier position in a transition sequence. Similarly, the states used as input by a ⟨helper⟩ transition only have incoming ⟨double⟩ or ⟨helper⟩ transitions, so if a ⟨helper⟩ transition is preceded by a δ transition, their order may be swapped. This proves the claim.

Let $C \in \mathbb{N}^{Q'}$ denote a terminal configuration reachable from an input configuration C_0 with at least $2|H| + |I|$ agents. By the claim there are configurations C_1, C_2 with $C_0 \to C_1 \to C_2 \to C$, s.t. going from C_0 to C_1 executes only ⟨double⟩ transitions, going from C_1 to C_2 only ⟨helper⟩ transitions, and from C_2 to C only transitions in δ. We now consider two cases.

If ⟨double⟩ is not enabled at C_1, then $C_1(\mathsf{h}) \geq |H|$, as there can be at most $|I|$ agents left in $C_1(I)$. It is not possible to remove agents from Q_{helper} without executing ⟨helper⟩, so at some point at least $|H|$ agents in $C_1(\mathsf{h})$ will be distributed to $h_1, ..., h_{|H|}$ and we get $C_2 \geq H$

Else, ⟨double⟩ is enabled at C_1 (but not at C), thus some transition removing agents from $C(I)$ must have occurred between C_2 and C (as ⟨helper⟩ transitions cannot do so). Since every configuration of $\mathbb{N}^I$ is terminal in $\mathcal{P}$ (precondition of the specification), we get $C_2(\mathrm{supp}(H)) > 0$, which, due to the construction of transition ⟨helper⟩, implies $C_2 \geq H$. (In particular, helpers are distributed in batches of H.)

So in both cases we have $C_2 \geq H$ and therefore find that $\pi(C_2)$ is an initial configuration (of $\mathcal{P}$). Between C_0 and C_2, only transition ⟨double⟩ affects states $I \cup I'$, and it preserves the value of $\mathrm{double}(\varphi)$. (Also note $\varphi(C_0) = \mathrm{double}(\varphi)(C_0)$, as only agents in I are present.)

As C can be reached from C_2 by executing only transitions in δ, we also get that $\pi(C)$ is a reachable configuration of $\mathcal{P}$. Moreover, C is terminal w.r.t. $\delta' \supseteq \delta$, so $\pi(C)$ is a terminal configuration of $\mathcal{P}$. We have defined O' s.t. $O'(C) = O(\pi(C))$ and thus get the correct output.

To argue that $\mathcal{P}'$ is bounded, we note that $\delta' \setminus \delta$ is acyclic. So if $\mathcal{P}'$ has arbitrarily long runs, then, by the claim, $\mathcal{P}$ has as well. But this contradicts that $\mathcal{P}$ is bounded. □

7.7. Converting to consensus output

The final step to produce a population protocol is to translate computers with marked-consensus output function into computers with standard consensus output function.

Specification: Distribute

Input: Constant k, function f, bounded binary population computer $\mathcal{P} = (Q, \delta, I, O, \emptyset)$ without helpers with marked consensus output function, and deciding a predicate φ for all inputs of size at least k in $\mathcal{O}(f(n, |\varphi|))$ interactions.

Output: Terminating population protocol $\mathcal{P}'$ with $4|Q|$ states deciding φ for all inputs of size at least k in $\mathcal{O}(n^2 + f(n, |\varphi|))$ interactions.

7.7.1. *Conversion* Distribute

Let a marked agent be an agent in a state $q \in Q_0 \cup Q_1$, where $Q_0 \subset Q$ and $Q_1 \subset Q$ are the sets of states as in the definition of marked consensus output. The obvious approach for the procedure Distribute would be to add an extra bit to every state, which will be the opinion of the agent, and is set whenever the agent meets a marked agent. As soon as the actual computation is done, all agents will be convinced of the correct opinion. However, convincing the i-th agent takes roughly $n^2/(n-i)$ steps (the inverse of the probability that the single marked agent meets one of the remaining $n-i$ agents with the wrong opinion); in total, this sums to $n^2 \log n$ steps. We give a slightly different procedure that achieves $\mathcal{O}(n^2)$. Whenever an agent meets a marked agent, besides assuming the correct opinion, it will receive a token, which it can use once, to convince another agent.

Formally, let $Q_0, Q_1 \subseteq Q$ denote the states defining the marked consensus output O, and set $Q_\perp := Q \setminus (Q_0 \cup Q_1)$. The set of states of $\mathcal{P}'$ is $Q' := Q \times \{0,1\}^2$, where the second component denotes the current opinion of the agent, and the third whether it has a token.

Let $q, p \in Q$. We want to execute $\mathcal{P}$ simultaneously to the following transitions. To write this down, we choose q', p' as the result of a transition $(q, p \mapsto q', p') \in \delta$, if such a transition exists, else we set $q' := q$, $p' := p$.

For convenience, we write $*$ if the component does not matter. A $*$ in the result of the transition indicates that this component is left unchanged. Based on our definitions the agents of a transition have no order, so $(q, p \mapsto q', p')$ and $(p, q \mapsto q', p')$ are the same transition. Let $i \in \{0,1\}$.

If an agent meets a marked agent, then the former assumes the latter's opinion and receives a token.

$$(q,*,*),(p,*,*) \mapsto (q',i,1),(p',i,1) \qquad \text{if } \{q',p'\} \cap Q_i \neq \emptyset \qquad \langle\text{certify}\rangle$$

If an agent with token meets a non-token agent of opposite opinion, the latter is convinced and the token consumed. Similarly, if two tokens held by agents with opposing opinions meet, the tokens are simply dropped.

$$(q,i,1),(p,1-i,0) \mapsto (q',i,0),(p',i,0) \qquad \langle\text{convince}\rangle$$

$$(q,i,1),(p,1-i,1) \mapsto (q',i,0),(p',1-i,0) \qquad \langle\text{drop}\rangle$$

Otherwise, nothing happens.

$$(q,*,*),(p,*,*) \mapsto (q',*,*),(p',*,*) \text{ if } (q,p) \neq (q',p') \qquad \langle\text{noop}\rangle$$

As defined above, our transitions are not deterministic. If multiple transitions are possible, we will pick a ⟨certify⟩ transition, or (if there are none) a ⟨convince⟩ or ⟨drop⟩ transition.

Finally, we choose O' as the consensus output given by the partition $O'_i := Q \times \{i\} \times \{0,1\}$, for $i \in \{0,1\}$ and $I' := I$, where we identify $q \in I$ with $(q,0,0)$.

7.7.2. Correctness

Proposition 31. Distribute *satisfies its specification (page 29).*

Proof. As for the other conversions, let $\pi : \mathbb{N}^{Q'} \to \mathbb{N}^{Q}$ denote a mapping from configurations of $\mathcal{P}'$ to ones of $\mathcal{P}$. We choose $\pi(C)(q) := C(q \times \{0,1\}^2)$, so π simply projects onto the first component.

Claim 1. $\mathcal{P}'$ refines $\mathcal{P}$ (Definition 25).
Properties 1 and 2 of Definition 25 follow immediately from the definition of Autarkify. For the third property, let C denote a reachable, terminal configuration of $\mathcal{P}'$. If a transition of $\mathcal{P}$ were enabled at $\pi(C)$, the corresponding ⟨noop⟩ transition would be enabled at C, so $\pi(C)$ must be terminal as well.

As $\mathcal{P}$ has a marked consensus output, there is an agent in a state $q \in Q_i$, where $i \in \{0,1\}$ is the output of $\mathcal{P}$ and $\pi(C)(q) > 0$. By definition of π this implies that there is a $q' \in \{q\} \times \{0,1\}^2$ with $C(q') > 0$.

We now claim $\mathrm{supp}(C) \subseteq Q'_i$, so assume the contrary and pick a $p' \in Q' \setminus Q'_{1-i}$ with $C(p') > 0$. But agents q', p' can now execute transition ⟨certify⟩, which contradicts C being terminal. Thus the claim is shown and our choice of O' yields $O'(\mathrm{supp}(C)) = i$.

Claim 2. $\mathcal{P}'$ is terminating.
Let C denote a reachable configuration of $\mathcal{P}'$. As $\mathcal{P}'$ refines $\mathcal{P}$, $\pi(C)$ is reachable in $\mathcal{P}$; as $\mathcal{P}$ is bounded, $\pi(C)$ can reach some terminal configuration D. We can execute a corresponding sequence of transitions in $\mathcal{P}'$ and find a configuration C' with $C \to C'$ and $\pi(C')$ terminal. At this point, $\pi(C)(Q_i) > 0$ and $\pi(C)(Q_{1-i}) = 0$ for an $i \in \{0,1\}$, which corresponds to the output of $\mathcal{P}$. By executing at most $n-1$ ⟨certify⟩ transitions, we can reach a configuration where all agents have opinion i and a token; the resulting configuration is terminal. To summarise, we have argued that any reachable configuration can reach a terminal configuration. Hence any infinite run can reach terminal configurations infinitely often and any fair run will eventually terminate.

Claim 3. $\mathcal{P}'$ decides φ for inputs of size at least k.
To show that $\mathcal{P}'$ decides φ for inputs of size at least k we would like to apply the Refinement lemma (Lemma 26), together with Claims 1 and 2. However, this does not work, as we would need to assume that $\mathcal{P}$ decides φ for *all* inputs. Fortunately, the proof can trivially be adapted to consider only inputs of size at least k.

Claim 4. $\mathcal{P}'$ runs in $\mathcal{O}(n^2 + f(n, |\varphi|))$ interactions for inputs of size $\Omega(k)$.
An inspection of Distribute shows that a configuration C_1 with $\pi(C_1)$ terminal is reached after $\mathcal{O}(f(n, |\varphi|))$ steps. It remains to argue that C_1 reaches a terminal configuration within $\mathcal{O}(n^2)$ steps.

Let C_1 denote a configuration s.t. $\pi(C_1)$ is terminal. In C_1, we have at least one marked agent for the correct answer $b := O(\mathrm{supp}(C_1))$, and no marked agents for the wrong answer, and this will not change during the remainder of the computation.

Let $f_{ij} := C(Q \times \{i\} \times \{j\})$, for $i, j \in \{0,1\}$ denote the number of agents with opinion i and holding j tokens. Then $F := f_{b1} - f_{(1-b)0} - 2f_{(1-b)1}$ counts the agents with correct opinion and a token, subtracting both the agents with the wrong opinion and the tokens held by agents with the wrong opinion. It is easy to see that this number cannot decrease, and will increase whenever a marked agent meets an agent that either has the wrong opinion, or does not have a token.

If $F \leq \frac{2}{5}n$, we have $f_{b1} \leq \frac{4}{5}n$, so in expectation we have to wait $5n$ steps for F to increase. As $F \geq -2n$ by definition, we need $5n(\frac{2}{5}n + 2n) \in \mathcal{O}(n^2)$ steps until we have $F \geq \frac{2}{5}n$.

As noted, F cannot decrease, so after that point we always have $f_{b1} \geq F \geq \frac{2}{5}n$. Whenever an agent with opinion $1-b$ meets an agent in f_{b1}, the value $f_{(1-b)0} + 2f_{(1-b)1}$ decreases. As long as agents with the wrong opinion exist, this has to happen after at most $\frac{5}{2}n$ steps in expectation. Noting $f_{(1-b)0} + 2f_{(1-b)1} \leq 2n$ we find that after $\mathcal{O}(n^2)$ steps no agents with opinion $1-b$ remain. □

7.8. Proof of Theorem 9

Applying the conversions of the previous sections in sequence, we obtain the final result of the section:

Theorem 9. *Every bounded population computer of size m deciding* double(φ) *can be converted into a terminating population protocol with $\mathcal{O}(m^2)$ states that decides φ within $2^{\mathcal{O}(m^2 \log m)}\, n^3$ interactions for all inputs of size $\Omega(m)$.*

Proof. Let φ be a Presburger predicate, and let $\mathcal{P}$ be a bounded population computer of size m deciding double(φ). We have:

- Applying Preprocess to $\mathcal{P}$ yields a computer $\mathcal{P}_0$ of size $\mathcal{O}(\mathrm{size}(\mathcal{P}))$ for double(φ). Moreover, $\mathcal{P}_0$ satisfies the preconditions of Binarise and has size $\mathcal{O}(\mathrm{size}(\mathcal{P}))$. Additionally, states in I have no incoming transitions, all configurations in $\mathbb{N}^I$ are terminal and $r(q) \leq 1$ for every $q \in I$ and $(r \mapsto s) \in \delta$, where I are the initial states and δ is the transition function of $\mathcal{P}_0$ (Proposition 27).

- Applying Binarise to $\mathcal{P}_0$ yields a binary computer $\mathcal{P}_1$ of adjusted size $\mathcal{O}(|Q| \cdot \text{size}(\mathcal{P})) \subseteq \mathcal{O}(\text{size}(\mathcal{P})^2)$ for double(φ) that satisfies the preconditions of Focalise, with Q being the states of $\mathcal{P}_1$. Additionally, states in I have no incoming transitions and all configurations in $\mathbb{N}^I$ are terminal, where I are the initial states of $\mathcal{P}_1$ (Proposition 28).
- Applying Focalise to $\mathcal{P}_1$ yields a binary computer $\mathcal{P}_2$ of adjusted size $\mathcal{O}(\text{size}(\mathcal{P}_2)) = \mathcal{O}(\text{size}(\mathcal{P})^2)$ for double(φ) that satisfies the preconditions of Autarkify (Proposition 29).
- Applying Autarkify to $\mathcal{P}_2$ yields a binary computer with marked consensus output $\mathcal{P}_3$ of adjusted size $\mathcal{O}(\text{size}(\mathcal{P})^2)$ that satisfies the preconditions of Distribute (Proposition 30).
- Applying Distribute to $\mathcal{P}_3$ yields a terminating population protocol $\mathcal{P}_4$ for φ with $\mathcal{O}(\text{size}(\mathcal{P})^2)$ states (Proposition 31).

By Proposition 24, $\mathcal{P}_3$ decides φ in $2^{\mathcal{O}(\text{size}(\mathcal{P}_3)\log(\text{size}(\mathcal{P}_3)))} \cdot n^3 = 2^{\mathcal{O}(m^2 \log m)} \cdot n^3$ interactions, and so $\mathcal{P}_4$ does as well. □

8. Fast and succinct population protocols

Applying Theorem 9 to any bounded population computer yields a fast population protocol stabilising within $2^{\mathcal{O}(m^2 \log m)} \cdot n^3$ expected interactions. This protocol is fixed-parameter fast, but not fast. We improve on this result for the family of bounded population computers constructed in Section 6. We show that applying the sequence of conversions Binarise-Focalise-Autarkify-Distribute defined in Section 7 to these computers yields fast protocols that stabilise in $\mathcal{O}(m^7 n^2)$ expected interactions.[7] For this we continue to use potential functions, as introduced in Section 7.1, but improve our analysis as follows:

- We introduce *rapidly decreasing* potential functions (Section 8.1). Recall that the execution of any transition decreases the potential, but not every interaction executes a transition. Indeed, interactions may be *silent*, and change neither the states of the agents involved, nor the potential. Intuitively, rapidly decreasing potential functions certify that, at every non-terminal configuration, executing a transition is not only *possible*, but also *likely*. We introduce *rapid* population computers as the computers with rapidly decreasing potential functions that also satisfy some technical conditions.
- We prove that the computers of Section 6 are rapid (Section 8.2).
- Finally, we show that applying our conversions to rapid population computers results in population protocols that stabilise within $\mathcal{O}(\alpha m^4 n^2)$ interactions, where α is a constant of a rapid computer (Sections 8.3 to 8.7). Loosely speaking, each of these sections shows that rapidness is preserved by one of the conversions.

8.1. Rapidly decreasing potential functions

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 to 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 (the time until the two agents in q meet). We therefore define the speed of a transition as $\min\{C(q), C(p)\}^2$ instead of $C(q)C(p)$.

It is important to note that this is only an approximation for the sake of analysis. Up to constant factors, it always underestimates the "true" speed of the protocol (i.e. the speed in the standard execution model for population protocols, after the conversions have been applied), but the estimate is strong enough to show $\mathcal{O}(n^2)$ stabilisation time.

For the remainder of this section, let $\mathcal{P} = (Q, \delta, I, O, H)$ denote a population computer. We define formally the speed of a configuration and rapidly decreasing potential functions.

Definition 32. 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 33. 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 $C \to C_{\text{term}}$.

Essentially, a potential function is rapidly decreasing at a configuration if the probability of reducing the potential is quadratic relative to the amount of potential which still has to be removed. In the formula, $\Phi(C_{\text{term}})$ describes the potential that will be left when the protocol terminates, the potential which still has to be removed is hence the difference to this term.

Initially, the potential is at most linear in the number of agents n. (Recall that we only consider linear potential functions in this paper.) So, if the potential function is rapidly decreasing in all configurations, we get the rough estimate $\sum_{i=1}^{\infty} \alpha n^2/i^2 \in \mathcal{O}(\alpha n^2)$ for the total amount of time until a terminal configuration has been reached.

[7] In the proof of Theorem 9 we also used the conversion Preprocess, but now it is no longer necessary.

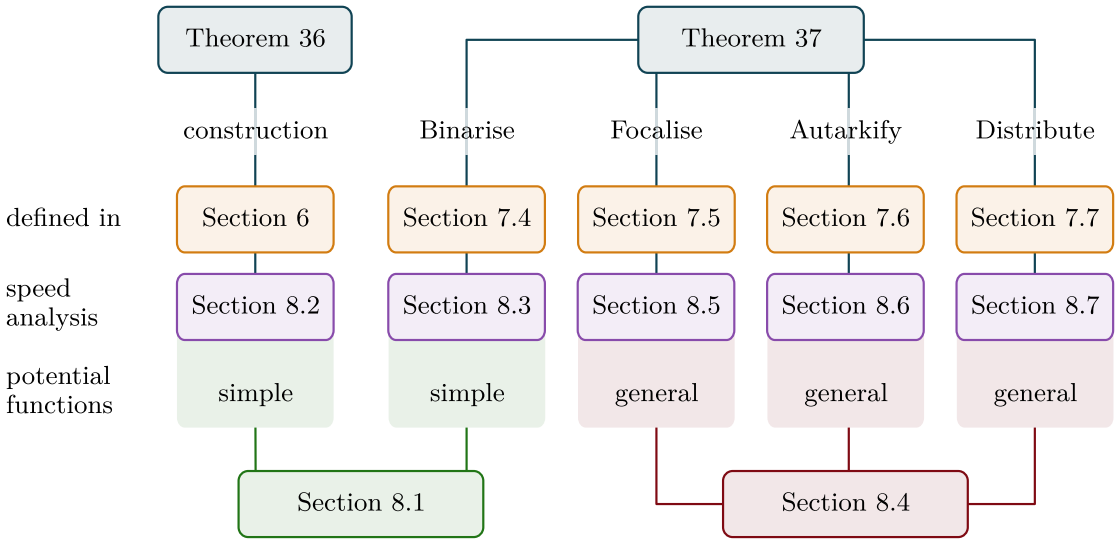


Fig. 7. Overview of Section 8.

However, no potential function is rapidly decreasing for all configurations of our protocols. Fortunately, we are able to overcome this problem. We show that, for computers satisfying some mild syntactic conditions, we only need the potential function to be rapidly decreasing for well-initialised configurations:

Definition 34. $C \in \mathbb{N}^Q$ 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 $C(\mathrm{supp}(H)) \in \Omega(C(I))$, i.e. the protocol has received a number of helpers linear in the sum of the input.)

We are now ready to present the structure of the rest of the paper. First, we introduce *rapid* population computers as those satisfying some syntactic conditions, as well as having a rapidly decreasing potential function for well-initialised configurations:

Definition 35. $\mathcal{P}$ is α*-rapid* if

1. it has a potential function Φ which is α-rapidly decreasing in all 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 $s(q) = 0$.

In the rest of the paper we prove the following two theorems:

Theorem 36. *The population computers constructed in Section 6 are* $\mathcal{O}(|\varphi|^3)$*-rapid.*

Theorem 37. *Every* α*-rapid population computer of size* m *deciding* $\mathrm{double}(\varphi)$ *can be converted into a terminating population protocol with* $\mathcal{O}(m)$ *states that decides* φ *in* $\mathcal{O}(\alpha\, m^4 n^2)$ *expected interactions for inputs of size* $\Omega(m)$.

which together immediately lead to our last main result:

Theorem 10. *For every predicate* $\varphi \in$ *QFPA there exists a terminating population protocol of size* $\mathcal{O}(|\varphi|)$ *that decides* φ *in* $\mathcal{O}(|\varphi|^7\, n^2)$ *interactions for inputs of size* $\Omega(|\varphi|)$.

Proof. By Theorem 36 there exists a construction producing population computers of size $\mathcal{O}(|\varphi|)$ which are $\mathcal{O}(|\varphi|^3)$-rapid. Concatenating this with the conversion procedure of Theorem 37 gives a construction for population protocols of size $\mathcal{O}(|\varphi|)$ which decide φ in $\mathcal{O}(|\varphi|^3 |\varphi|^4 n^2) = \mathcal{O}(|\varphi|^7 n^2)$ expected interactions for inputs of size $\Omega(|\varphi|)$. □

Fig. 7 gives a graphical overview of Section 8. Theorem 36 is proved in Section 8.2. The proof of Theorem 37 is more involved. Recall that a population computer $\mathcal{P} = (Q, \delta, I, O, H)$ is a population protocol if (a) it is binary, (b) has no helpers ($H = \emptyset$), and (c) O is a consensus output. In Sections 7.4, 7.5, 7.6, and 7.7 we introduced the Binarise, Focalise, Autarkify and Distribute conversions, which together transform a population computer deciding $\mathrm{double}(\varphi)$ into one satisfying conditions

(a), (b), and (c), and deciding φ. In Sections 8.3-8.7 we show that these conversions have small impact on α-rapidness, which proves Theorem 37. Section 8.3 proves the result for Binarise. Section 8.4 generalises the notion of potential function. Sections 8.5-8.7 apply the generalisation to prove the result for Focalise, Autarkify and Distribute, respectively.

8.2. The population computers of Section 6 are rapid

We prove Theorem 36 by showing that the four conditions of the definition of an α-rapid protocol (Definition 35) hold. The first condition, existence of a rapidly decreasing potential function for $\mathcal{P}$, is the key one. While we already know that *some* potential function exists — $\mathcal{P}$ is bounded, so Lemma 23 applies — we need to construct one that is rapidly decreasing.

The potential function Φ Let $\mathcal{P}$ be the computer for a given predicate constructed as in Section 6. We give a potential function for $\mathcal{P}$. Recall that $\mathcal{P}$ consists of a number of subcomputers, say s, deciding threshold and remainder predicates. The sets of states of these subcomputers are disjoint, apart from a shared helper state 0. A state of $\mathcal{P}$ is either an input state, a state of one of the subcomputers, or the helper state 0. For each initial state there is a distribution transition that takes one agent from the state and $s-1$ helpers from state 0, and sends one agent to an input state of each of the subcomputers. This is all we need to define the potential function:

Definition 38. Let $\mathcal{P}$ be the computer for a given predicate φ constructed as in Section 6, and let Q be its set of states. The function $\Phi : Q \to \mathbb{N}$ is defined as follows, with $q \in Q$:

- If $q = 0$, then $\Phi(q) := 0$.
- If q is an initial state and $\llparenthesis q, (k-1)\cdot 0 \rrparenthesis \mapsto r$ is its corresponding distribution transition, then $\Phi(q) := k - 1 + \sum_{q' \in r} \Phi(q')$.
- If q is a state of a subcomputer for a threshold predicate, then $\Phi(q) := 1$.
- If q is a state of a subcomputer for a remainder predicate, then $\Phi(q)$ is defined as follows. As described in Section 6, apart from the shared state 0 the states of this subcomputer are $\{2^0, 2^1, ..., 2^d\}$ for some $d \geq 0$. Let $d' := d - \lceil \log_2 6d \rceil$. For every $i \in 0, ..., d'-1$ we set $\Phi(2^i) := 2$, and for every $j \in \{0, ..., d-d'\}$ we set $\Phi(2^{d'+j}) := 2^j + 1$.

We extend Φ to configurations $\mathbb{N}^Q$ by choosing the unique linear function with weights Φ. We show that Φ is a potential function for $\mathcal{P}$.

Proposition 39. *The function Φ is a potential function for $\mathcal{P}$. Additionally, $\Phi(0) = 0$ and $\Phi(q) \in \mathcal{O}(|\varphi|)$ for all $q \in Q$.*

Proof. It follows immediately from the definition of Φ and from Section 6 that distribution transitions and transitions of threshold subcomputers decrease the potential. It remains to prove the same for transitions of remainder subcomputers. For ⟨combine⟩, we need to show $\Phi(2^i) + \Phi(2^i) > \Phi(2^{i+1})$, which, depending on i, reduces to either $2+2 > 2$ or $2^j + 1 + 2^j + 1 > 2^{j+1} + 1$ for some $j \geq 0$. In the case of ⟨modulo⟩, we note that $2\theta \geq 2^d$ (as $d = \lceil \log_2 \theta \rceil$), so $2^d - \theta \leq 2^0 + 2^1 + ... + 2^{d-2}$. It thus suffices to show $\Phi(2^d) \geq \sum_{i=0}^{d-2} \Phi(2^i) + (d-2)$, and we get

$$\sum_{i=0}^{d-2} \Phi(2^i) = 2d' + (d - d' - 2) + 2^{d-d'-1} - 1 \leq 2d + 2^{d-d'-1}$$

So $\Phi(2^d) - \sum_{i=0}^{d-2} \Phi(2^i) \geq 2^{d-d'} + 1 - (2d + 2^{d-d'-1})$ which is at least $d-2$ if $2^{d-d'-1} \geq 3d - 3$. The latter then follows from our choice of d'. Finally, ⟨fast modulo⟩ obviously reduces the potential as well. □

The potential function Φ is $\mathcal{O}(|\varphi|)^3$-rapidly decreasing Let $\mathcal{P}$ be the computer of Section 6 for a predicate φ. Recall that it consists of s subcomputers, each of which corresponds to either a remainder or a threshold predicate φ_j. Subcomputer j has a degree d_j, for $j = 1, ..., s$, corresponding to the bits of the representation it encodes. Further $\mathcal{P}$ has a helper state 0, shared by all subcomputers. We introduce some notations.

- We need to reference specifically the highest bits of threshold subcomputers, and so we define $Q_d := \{(2^{d_j})_j, (-2^{d_j})_j : \varphi_j \text{ is threshold predicate}\}$.
- Furthermore, we are interested in the states q for which a transition using only agents in q exists. The set of these states is $Q_{\text{self}} := Q \setminus (I \cup Q_d \cup \{0\})$.
- We denote by $\text{value}_j(q)$ the value of state q for the j-th subcomputer. Formally, for $q \in Q$ and a subcomputer $j \in \{1, ..., s\}$ we define $\text{value}_j((q)_j) := q$ for each $q \in Q_j$, $\text{value}_j(0) := 0$ and $\text{value}_j(x_i) := a_i^j$ for input $i \in \{1, .., v\}$, where a_i^j is the coefficient of the variable x_i in the predicate φ_j.

Note that $\sum_q \text{value}_j(q) C(q)$ is invariant for configurations C of a run, if j is a threshold predicate. (For remainder predicates it would be invariant modulo θ_j, but that is not relevant for this section.) Also, the sum $\sum_q |\text{value}_j(q)| C(q)$ is non-increasing, for all j.

We show that Φ is $\mathcal{O}(|\varphi|)^3$-rapid in all well-initialised configurations. First, we prove a lemma bounding the number of agents in the states of Q_d in reachable configurations.

Lemma 40. *Let $\mathcal{P}$ be the computer of Section 6 for a predicate φ, and let C be a reachable configuration of $\mathcal{P}$. We have:*

1. *$C(Q_d) \leq |C|/8$, and*
2. *Let $t_j := ((2^{d_j})_j, (-2^{d_j})_j \mapsto 0, 0) \in \langle\text{cancel}\rangle$, where j is the index of a threshold subcomputer, and let $C \to C'$. Then $C(Q_d) - C'(Q_d) \leq 2\sum_j \mathrm{tmin}_{t_j}(C) + C(I) + C(Q_{\mathrm{self}})$.*

Proof. For part 1., let j denote the index of the threshold predicate, $q = (\pm 2^{d_j})_j \in Q_d$ one of its largest states, and $p \in I$ an input state. We then have

$$|\mathrm{value}_j(p)| \leq a^j_{\max} \leq \frac{2^{d_j}}{16s} = \frac{\mathrm{value}_j(q)}{16s}$$

due to the choice of d_j. As C is reachable from some initial configuration C_0, and $\sum_q |\mathrm{value}_j(q)| C(q)$ cannot increase, we sum over p to get

$$|\mathrm{value}_j(q)|\, C(q) \leq \sum_{p\in I} |\mathrm{value}_j(p)|\, C_0(p) \leq \sum_{p\in I} |\mathrm{value}_j(q)|\, C_0(p)/16s = |\mathrm{value}_j(q)|\, C_0(I)/16s$$

So we have $C(q) \leq C_0(I)/16s$ and summing over q yields the desired statement.

For part 2., let $D \in \mathbb{N}^{Q_d}$ with $D(q) := \max\{C(q) - C'(q), 0\}$ for $q \in Q_d$. Note that $D(q)$ is a lower bound on how many agents leave state q in any run from C to C', and that $C(Q_d) - C'(Q_d) \leq D(Q_d)$. Let j be the index of a threshold subcomputer. As the only ways to leave Q_d are $\langle\text{cancel}\rangle$ and $\langle\text{cancel 2nd highest}\rangle$, we know that

$$\sum_{q\in Q_{j+}} |\mathrm{value}_j(q)|\, C(q) \geq 2^{d_j-1}\Big(D((2^{d_j})_j) + D((-2^{d_j})_j)\Big)$$

where $Q_{j+} := \{q \in Q : \mathrm{value}_j(q) > 0\}$. The same inequality holds when replacing Q_{j+} with $Q_{j-} := \{q \in Q : \mathrm{value}_j(q) < 0\}$. From these, we derive

$$2\, C((\sim 2^{d_j})_j) + 2 \sum_{q\in Q\setminus Q_d} 2^{-d_j} |\mathrm{value}_j(q)|\, C(q) \geq D((2^{d_j})_j) + D((-2^{d_j})_j)$$

for $\sim \in \{+, -\}$, as $Q \setminus Q_d \cup \{(\sim 2^{d_j})_j\} \supseteq Q_{j\sim}$. We can combine the two inequalities into

$$2\,\mathrm{tmin}_{t_j}(C) + 2 \sum_{q\in Q\setminus Q_d} 2^{-d_j} |\mathrm{value}_j(q)|\, C(q) \geq D((2^{d_j})_j) + D((-2^{d_j})_j)$$

Summing over j then yields

$$2\sum_{t\in T_d} \mathrm{tmin}_t(C) + 2 \sum_{q\in Q\setminus Q_d} \left[C(q) \sum_j 2^{-d_j} |\mathrm{value}_j(q)| \right] \geq D(Q_d)$$

If $\sum_j 2^{-d_j} |\mathrm{value}_j(q)| \leq \frac{1}{2}$ were to hold for all $q \in Q \setminus Q_d$, then we would get the desired statement (noting $\mathrm{value}(0) = 0$), so it remains to show this claim. For $q \in Q_j \setminus Q_d$ for some j we have $|\mathrm{value}_j(q)| \leq 2^{d_j-1}$ and $\mathrm{value}_{j'}(q) = 0$ for $j' \neq j$, and for $q \in I$ it follows from $|\mathrm{value}_j(q)| \leq a^j_{\max} \leq 2^{d_j}/16s$ for all $j \in \{1, \ldots, s\}$. □

We now show that Φ is $\mathcal{O}(|\varphi|^3)$-rapidly decreasing. The proof uses the following inequality, which follows immediately from the Cauchy-Bunyakovsky-Schwarz inequality.

Lemma 41. *Let $x_1, \ldots, x_n \in \mathbb{R}$. Then $\left(\sum_{i=1}^n x_i\right)^2 \leq n \sum_i x_i^2$.*

Proposition 42. *The function Φ is a $\mathcal{O}(|\varphi|^3)$-rapidly decreasing potential function for $\mathcal{P}$ in all well-initialised configurations.*

Proof. Let C be a well-initialised configuration and let C_{term} be a terminal configuration such that $C \to C_{\mathrm{term}}$. Let $C' := C - C_{\mathrm{term}}$ and let $W := \max_{q\in Q} \Phi(q)$ denote the largest weight of Φ. Then

$$\Phi(C) - \Phi(C_{\mathrm{term}}) = \Phi(C') \leq W \cdot C'(Q \setminus \{0\})$$

Applying Lemma 40(2), as well as $C' \leq C$, yields

$$C'(Q \setminus \{0\}) = C'(Q_d) + C'(I) + C'(Q_{\text{self}}) \leq 2 \sum_{t \in T_d} \text{tmin}_t(C) + 2C(I) + 2C(Q_{\text{self}}) \qquad (*)$$

where $T_d := \{t_j : \varphi_j \text{ threshold predicate}\}$ with t_j as for Lemma 40(2). Since C is well-initialised we have $C(I) \leq 2(C(Q_{\text{self}}) + C(0) + C(Q_d))$, and Lemma 40(1) implies $C(Q_d) \leq C(Q \setminus Q_d)/7$. We use both to derive

$$\begin{aligned} & C(I) \leq 2C(Q_{\text{self}}) + 2C(0) + \tfrac{2}{7}\big(C(I) + C(Q_{\text{self}}) + C(0)\big) \\ \Rightarrow\ & \tfrac{5}{7}C(I) \leq \tfrac{16}{7}(C(Q_{\text{self}}) + C(0)) \\ \Rightarrow\ & C(I) \leq \tfrac{16}{5}\big(C(Q_{\text{self}}) + C(0)\big) \leq 4\big(C(Q_{\text{self}}) + C(0)\big) \end{aligned}$$

It follows that $C(I) \leq 4C(Q_{\text{self}}) + 4\min\{C(0), C(I)\}$ holds. Writing $T_I := \langle \text{distribute} \rangle$, and using $\min\{C(0), C(I)\} \leq \sum_{q \in I} \min\{C(0), C(q)\}$ we get

$$C(I) \leq 4C(Q_{\text{self}}) + 4 \sum_{t \in T_I} \text{tmin}_t(C)$$

Every state $q \in Q_{\text{self}}$ has a (unique) transition using only agents in q; we use T_o to denote this set and set $T := T_I \cup T_o \cup T_d$. We can now insert the previous inequality into $(*)$ to get $C'(Q \setminus \{0\}) \leq 8 \sum_{t \in T} \text{tmin}_t(C)$. Noting $|T| \leq 2|Q|$ and applying Lemma 41 we get:

$$\Phi(C')^2 \leq \left(8\,W \sum_{t \in T} \text{tmin}_t(C)\right)^2 \leq 128\,W^2 |Q|\,\text{speed}(C)$$

The desired statement then follows from $|Q|, W \in \mathcal{O}(|\varphi|)$. □

Proof of Theorem 36 We are now ready to prove:

Theorem 36. *The population computers constructed in Section 6 are $\mathcal{O}(|\varphi|^3)$-rapid.*

Proof. We show that the computers satisfy the conditions of the definition of α-rapid protocols (Definition 35) for $\alpha \in \mathcal{O}(|\varphi|^3)$.

Condition 1 follows immediately from Proposition 42. Conditions 2-4 are easy to check: For condition 2, we note that only the reservoir state 0 (shared by all subcomputers) has more than two outgoing transitions. Condition 3 is ensured by transition $\langle \text{distribute} \rangle$ always taking at least one agent from the reservoir $0 \notin I$. Similarly, this transition is the only transition affecting the input states I, so Condition 4 is met. □

8.3. Removing multiway transitions preserves speed

We show that, loosely speaking, the Binarise conversion of Section 7.4 preserves the speed of configurations. Formally, we prove that if the input to the conversion has a rapidly decreasing potential function with parameter α, then the output also has a rapidly decreasing potential function, and its parameter is not much larger than α.

Proposition 43. *Let $\mathcal{P} = (Q, \delta, I, O, H)$ denote a bounded population computer satisfying conditions 2 and 3 of the output specification of* Binarise *(page 22). If some potential function for $\mathcal{P}$ is α-rapidly decreasing in all well-initialised configurations, then some potential function for* Binarise$(\mathcal{P})$ *is $\mathcal{O}(|Q|^2 k^2 \alpha)$-rapidly decreasing in all well-initialised configurations, where k is the maximal arity of the transitions of $\mathcal{P}$.*

Proof. Let Φ be a potential function for $\mathcal{P}$ that is α-rapidly decreasing in all well-initialised configurations, and let $\mathcal{P}' :=$ Binarise$(\mathcal{P})$. We first construct a potential function Φ' for $\mathcal{P}'$. Intuitively, the potential of a state corresponds directly to the original potential of states it owns, with some additional accounting to pay for overhead of executing a multiway transition. At this point it becomes important that our definition of potential function requires a transition of arity k to reduce the potential by $k - 1$, as this means that we have to increase the total potential by only a constant factor.

We first adjust Φ by multiplying it with 5, so that $\Phi(r) \geq \Phi(s) + 5(|s| - 1)$ for all transitions $(r \mapsto s) \in \delta$. Now, let $q, p \in Q$, $t = (r \mapsto s) \in \delta$ a transition where $\text{supp}(r) = \{q, p\}$ and q is primary, and let $s_1, \ldots, s_l$ denote the enumeration of s from Section 7.4.1. We define Φ' as

$$
\begin{aligned}
\Phi'((q,0)) &:= 0 \\
\Phi'((q,i)) &:= i\Phi(q) + 1 && \text{for } i \in \{1,\dots,m(q)-1\} \\
\Phi'((q,m(q))) &:= m(q)\Phi(q) \\
\Phi'((q,i,t)) &:= i\Phi(q) + \Phi(s) + 2l + 2 && \text{for } i \in \{1,\dots,m(q)\} \\
\Phi'((t,i)) &:= \sum_{j=i}^{l} (\Phi(s_j) + 2) && \text{for } i \in \{1,\dots,l-1\}
\end{aligned}
$$

Claim 1. Φ' is a potential function for $\mathcal{P}'$.
For most transitions, it is easy to see that Φ' decreases. However, we need to verify that ⟨commit⟩ does so as well. If $q \neq p$, we have to prove the inequality

$$i\Phi(q) + j\Phi(p) + 2 \geq (i - r(q))\Phi(q) + \Phi(s) + 2l + 2 + (j - r(p))\Phi(p) + 1$$

which boils down to $\Phi(r) \geq \Phi(s) + 2l + 1$. This then follows from $\Phi(r) \geq \Phi(s) + 5(l-1)$. The case $q = p$ is shown analogously. This concludes the proof of the claim.

The next three claims show technical properties that are needed for the proof that Φ' is rapidly decreasing. The first gives a relation between the potential of $\mathcal{P}'$ and of the refined computer $\mathcal{P}$ (recall Section 7.4.2). Let $\pi : \mathbb{N}^{Q'} \to \mathbb{N}^{Q}$ be the mapping relating configurations of $\mathcal{P}'$ and $\mathcal{P}$ defined in Section 7.4.1.

Claim 2. $\Phi(\pi(C)) \leq \Phi'(C) \leq \Phi(\pi(C)) + 2C(S)$ for $S := Q' \setminus \{(q, m(q)), (q,0) : q \in Q\}$.
For the first inequality we simply observe $\Phi(\pi(q')) \leq \Phi'(q')$ for each $q' \in Q'$. Each state $q' = (q, m(q))$, for $q \in Q$, satisfies $\Phi'(q') = \Phi(\pi(q'))$. For each other state $q' \in S$ we show $\Phi'(q') - \Phi(\pi(q')) \leq 2c$, where c is the amount of agents "owned" by an agent in state q'. E.g. $q' = (q,i)$ for $q \in Q$ and $0 < i < m(q)$ has $\Phi'(q,i) - \Phi(\pi(q,i)) = 1$ and $c = i$. For (q,i,t), we have $c = |s| + i$, and require $2c \geq 2|s| + 2$. The respective inequalities follow immediately, which concludes the proof of the claim.

In Section 8.2 we have seen that states which can initiate a transition by themselves are useful to show speed bounds. More precisely, states q such that there exists a transition $t = (r \mapsto s)$ with $\mathrm{supp}(r) = \{q\}$, implying $\mathrm{tmin}_t(C) = C(q)$ for all C. The next claim shows that most states q can be assigned a transition with the same useful property on reachable configurations (even if not all of them use only a single state).

Claim 3. Let $S := Q' \setminus \{(q, m(q)), (q,0) : q \in Q\}$. There is an injection $g : S \to \delta'$, s.t. $C(q') = \mathrm{tmin}_{g(q')}(C)$ for any reachable configuration C and $q' \in S$.
Let $q \in Q$, and $t = (r \mapsto s) \in \delta$.

If $q' = (q,i)$ for $i < m(q)$, then there is a ⟨stack⟩ transition using only agents in q, which we use as $g(q')$. If $q' = (q,i,t)$, then we know that $C(q') \leq C((q,0))$, as q' owns at least $|r| \geq 1$ agents, so we can pick the ⟨transfer⟩ transitions for $g(q')$. Finally, if $q' = (t,i)$, for $i < |s|$, then q' owns $|s| - i \geq 1$ agents other than itself, and we choose the corresponding ⟨execute⟩ transition. This proves the claim.

In the end, we want to show that Φ' is rapidly decreasing in all well-initialised configurations if Φ is. For this, we need to argue briefly that being a well-initialised configuration corresponds.

Claim 4. If C is well-initialised, then so is $\pi(C)$.
Due to Condition 3 in the specification of Binarise, $m(q) = 1$ for each $q \in I$. In combination with Condition 2 we get $C((q,1)) = \pi(C)(q)$ for all $q \in I$. This implies $C(I') = \pi(C)(I)$. Noting $|H| = |H'|$, the statement follows immediately.

Finally, we prove that Φ' is rapidly decreasing. Let k be the maximal arity of the transitions of $\mathcal{P}$. We show that Φ' is $\mathcal{O}(\alpha k^2 |Q|^2)$-rapidly decreasing in C if Φ is α-rapidly decreasing in $\pi(C)$, for all reachable configurations C and $\alpha \geq 1$. Let C_{term} denote a terminal configuration reachable from C. Using Claim 2 we get $\Phi'(C) - \Phi'(C_{\mathrm{term}}) \leq \Phi(\pi(C)) - \Phi(\pi(C_{\mathrm{term}})) + 2C(S)$. We know that $\pi(C_{\mathrm{term}})$ is reachable from $\pi(C)$, and, as shown in the proof of Proposition 28, Claim 1, it is terminal as well. Hence we can use that $\mathcal{P}$ is α-rapidly decreasing in $\pi(C)$ to get $\left(\Phi(\pi(C)) - \Phi(\pi(C_{\mathrm{term}}))\right)^2 \leq \alpha\, \mathrm{speed}(\pi(C))$.

Let us now estimate $\mathrm{speed}(\pi(C))$. Let t^* be the ⟨commit⟩ transition corresponding to $t \in \delta$ using agents in states $(q, m(q)), (p, m(p))$ for $q, p \in Q$. Additionally, let $H_q := \sum_{q' \in S} C(q')\pi(q')(q)$ for $q \in Q$ denote the contribution of agents in states S to $\pi(C)(q)$, for $q \in Q$. We have

$$\pi(C)(q) = H_q + m(q)C((q, m(q))) \leq H_q + kC((q, m(q)))\ .$$

Now, let $t = (r \mapsto s) \in \delta$ and $q, p \in Q$ with $\mathrm{supp}(r) = \{q, p\}$. Then the above (together with the well-known inequality stating that $\min\{x_1 + x_2, y_1 + y_2\} \leq \min\{x_1, y_1\} + x_2 + y_2$ holds for non-negative x_1, x_2, y_1, y_2) yields

$$\mathrm{tmin}_t(\pi(C)) = \min\{\pi(C)(q), \pi(C)(p)\} \leq k\,\mathrm{tmin}_{t^*}(C) + H_q + H_p$$

Squaring the right-hand side gives at most $3(k^2\,\mathrm{tmin}_{t^*}(C)^2 + H_q^2 + H_p^2)$, and so summing over t and applying Lemma 41 we get a first bound for $\mathrm{speed}(\pi(C))$:

$$\mathrm{speed}(\pi(C)) \leq 3k^2 \sum_{t\in\delta} \mathrm{tmin}_{t^*}(C)^2 + 6|Q| \sum_{q\in Q} H_q^2 \,.$$

Let us bound $\sum_{q\in Q} H_q^2$. We have $\sum_{q\in Q} H_q = |\pi(C_S)|$, where $C_S(q) := C(q)$ for $q \in S$ and 0 otherwise. Moreover, the definition of π yields $|\pi(q)| \leq 2k$, and so $|\pi(C_S)| \leq 2k|C_S|$. Finally, by Claim 3 we obtain $|C_S| \leq \sum_{t'\in g(S)} \mathrm{tmin}_{t'}(C)$. Putting this together we get $\sum_{q\in Q} H_q^2 \leq \big(\sum_{q\in Q} H_q\big)^2 \leq 4k^2\big(\sum_{t'\in g(S)} \mathrm{tmin}_{t'}(C)\big)^2$. Applying Lemma 41, we finally get

$$\mathrm{speed}(\pi(C)) \leq 3k^2 \sum_{t\in\delta} \mathrm{tmin}_{t^*}(C)^2 + 24k^2|Q|^2 \sum_{t'\in g(S)} \mathrm{tmin}_{t'}(C)^2 \leq 24k^2|Q|^2\,\mathrm{speed}(C)$$

We are now ready to complete the proof (note $C(S) = |C_S|$):

$$\begin{aligned}\big(\Phi'(C) - \Phi'(C_{\mathrm{term}})\big)^2 &\leq 2\big(\Phi(\pi(C)) - \Phi(\pi(C_{\mathrm{term}}))\big)^2 + 8C(S)^2\\ &\leq 2\alpha\,\mathrm{speed}(\pi(C)) + 8|Q|\,\mathrm{speed}(C)\\ &\leq 49\alpha k^2|Q|^2\,\mathrm{speed}(C) \quad \square\end{aligned}$$

8.4. Generalised potential function analysis

We generalise the notion of potential function (described in Section 7.1) to better analyse the conversions Focalise and Autarkify. We start by briefly recalling the main definitions of Sections 7.1 and 8.1.

Definition 22. A function $\Phi : \mathbb{N}^Q \to \mathbb{N}$ is *linear* if there exist weights $w : Q \to \mathbb{N}$ s.t. $\Phi(C) = \sum_{q\in Q} w(q)\cdot C(q)$ for every $C \in \mathbb{N}^Q$. A *potential function (for $\mathcal{P}$)* is a linear function Φ such that $\Phi(r) \geq \Phi(s) + |r| - 1$ for all $(r \mapsto s) \in \delta$.

Definition 32. Given a configuration $C \in \mathbb{N}^Q$ and some transition $t = (r \mapsto s) \in \delta$, we let $\mathrm{tmin}_t(C) := \min\{C(q) : q \in \mathrm{supp}(r)\}$. For a set of transitions $T \subseteq \delta$, we define $\mathrm{speed}_T(C) := \sum_{t\in T} \mathrm{tmin}_t(C)^2$, and write $\mathrm{speed}(C) := \mathrm{speed}_\delta(C)$ for convenience.

Definition 33. 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 $\mathrm{speed}(C) \geq (\Phi(C) - \Phi(C_{\mathrm{term}}))^2/\alpha$ for all terminal configurations C_{term} with $C \to C_{\mathrm{term}}$.

Definition 34. $C \in \mathbb{N}^Q$ is *well-initialised* if C is reachable and $C(I) + |H| \leq \frac{2}{3}n$.

While the above definitions can be applied to all of our conversions, they lead to large constants in the final speed. These are merely the result of a loose analysis – they do not reflect the actual speed of our protocols. Mainly, this is due to a single potential function being unable to model computations that consist of multiple phases efficiently. A concrete explanation of this problem in the context of Focalise is given in Section 8.5. In this section we introduce the formal machinery necessary to better adapt our technique to those constructions, leading to better constants and easier proofs.

We start by extending the definition of rapidly decreasing to handle linear functions which are not potentials. Here, we do not need to deal with multiway transitions, so let $\mathcal{P} = (Q, \delta, I, H, O)$ denote a binary population computer.

Definition 44. Let $\Phi : \mathbb{N}^Q \to \mathbb{N}$ be linear, let $\delta_> := \{(r \mapsto s) \in \delta : \Phi(r) > \Phi(s)\}$ be the transitions decreasing Φ, and let $\alpha > 0$. If $\mathrm{speed}_{\delta_>}(C) \geq (\Phi(C) - \Phi(C_{\mathrm{term}}))^2/\alpha$ for a configuration C and all terminal configurations C_{term} with $C \to C_{\mathrm{term}}$, we say that Φ is *α-rapidly decreasing in C*.

The only change compared to Definition 33 is that the speed considers only transitions which reduce the given linear function. If Φ is a potential function, we have $\delta_> = \delta$ and $\Phi(C) > \Phi(C_{\mathrm{term}})$ for all non-terminal configurations C, making this definition coincide with Definition 33.

To model phases, the general idea is that we construct a family of linear functions $\Phi_1, \ldots, \Phi_e$. For each configuration C, one of these will be rapidly decreasing (we refer to it as "active"). That alone would not be enough to guarantee a quadratic number of interactions (or any time bound at all), as it would not prevent the other functions from increasing their value. So we require the stronger property that a Φ_i cannot increase once it has been active. We also need that Φ_i can decrease at C, which certifies that some progress can be made. Otherwise, Φ_i might be "rapidly decreasing" but already at its lowest point.

Definition 45. A tuple $\Phi = (\Phi_1, \ldots, \Phi_e)$, where $\Phi_1, \ldots, \Phi_e : \mathbb{N}^Q \to \mathbb{N}$ denote linear maps, is a *potential group (of size e)*. A potential group Φ is *α-rapidly decreasing in a configuration C*, for $\alpha \geq 1$, if C is terminal or there is some $i \in \{1, \ldots, e\}$ s.t. Φ_i

is α-rapidly decreasing in C, some transition reducing Φ_i is enabled at C, and no transition increasing Φ_i can be executed at any configuration reachable from C. We then call Φ_i *active* at C.

The definition places no restrictions on the order in which the Φ_i are listed. However, in our proofs we will generally order them in the same fashion as they would become active in a run. Further, our potential groups have the additional property that they decrease lexicographically with each transition.

To close out the section, we show that the above notion does actually lead to a strong speed bound when applied to population protocols.

Proposition 46. *Let $\mathcal{P}$ denote a population protocol and Φ a potential group for $\mathcal{P}$ of size e which is α-rapidly decreasing in all reachable configurations with at least m agents. Then $\mathcal{P}$ reaches a terminal configuration after $\mathcal{O}(e(\sqrt{\alpha}|Q|+\alpha)n^2)$ random interactions in expectation for all initial configurations with at least m agents.*

Proof. Let $\sigma = C_0C_1\ldots$ denote a fair run of $\mathcal{P}$, and pick the smallest l s.t. C_l is terminal. We define

$$X_i^c := |\{j : \Phi_i \text{ is active at } C_j \text{ and } \Phi_i(C_j) - \Phi_i(C_l) = c\}|$$

We observe that $l \leq \sum_{i,c} X_i^c$ holds and will now proceed to prove a bound on the expected value $\mathbb{E}(X_i^c)$, for all i, c, if σ is generated via random interactions.

Consider $\mathbb{P}(X_i^c \geq k+1 \mid X_i^c \geq k)$, for $k \geq 1$. We note that σ is generated by a (homogeneous) Markov chain and the index τ of the k-th configuration counting towards X_i^c is a stopping time. By the strong Markov property, the above probability is equal to the probability that $\mathcal{P}$ reaches some configuration C counting towards X_i^c when started in the configuration C_τ. This is at most $1-\gamma$, where γ is the probability that $\mathcal{P}$ executes a transition reducing Φ_i at C_τ, as an active Φ_i cannot increase at any later point.

First, we know that Φ_i is active at C_τ, so some transition reducing Φ_i is enabled at C_τ and $\gamma \geq 1/n(n-1)$. However, if c is large enough we can get a better bound due to the fact that Φ_i is rapidly decreasing at C_τ.

Let $\delta_> \subseteq \delta$ denote the transitions reducing Φ_i. Let $\xi := \mathrm{tmin}_t(C_\tau)$ for some $t = (q, p \mapsto q', p') \in \delta_>$. By definition, we have $C_\tau(q), C_\tau(p) \geq \xi$ and thus the probability of executing t at C_τ is at least $\xi(\xi-1)/n(n-1)$ (note that $q = p$ is possible). As $\xi(\xi-1) \geq \xi^2/2 - 1$, we find $n(n-1)\gamma \geq \mathrm{speed}_{\delta_>}(C_\tau) - |\delta_>|$ by summing over t. Since $|\delta_>| \leq |Q|^2$, and rapidly decreasing implies $\mathrm{speed}_{\delta_>}(C_\tau) \geq c^2/\alpha$, we get $\gamma \geq \max\{1, c^2/\alpha - |Q|^2\}/n(n-1)$.

From $\mathbb{P}(X_i^c \geq k+1 \mid X_i^c \geq k) \leq 1-\gamma$ for all $k \geq 1$ we get $\mathbb{E}(X_i^c) \leq 1/\gamma$ (similar to the geometric distribution). Summing over i and c and using $c^2/\alpha - |Q|^2 \geq c^2/2\alpha$ for $c \geq \sqrt{2\alpha}|Q|$, we get:

$$\begin{aligned}
\mathbb{E}(l) &\leq \sum_{i=1}^{e}\sum_{c=0}^{\infty}\mathbb{E}(X_i^c) \\
&\leq en(n-1)\sum_{c=0}^{\infty}\frac{1}{\max\{1, c^2/\alpha - |Q|^2\}} \\
&\leq en(n-1)\left(\sqrt{2\alpha}|Q| + \frac{\alpha\pi^2}{3}\right) \qquad \square
\end{aligned}$$

8.5. Converting to marked consensus preserves speed

We prove that Focalise, the conversion of Section 7.5 is fast (we use the notion of potential groups introduced in Section 8.4):

Proposition 47. *Let $\mathcal{P} = (Q, \delta, I, O, H)$ be a bounded binary population computer fulfilling the specification of* Focalise *(page 24), and let Φ denote a potential function for $\mathcal{P}$ which is α-rapidly decreasing in all well-initialised configurations.*

Then $\mathcal{P}' :=$ Focalise$(\mathcal{P})$ has a potential group of size 5 which is $\mathcal{O}(\alpha + \mathrm{size}_2(\mathcal{P})^2)$-rapidly decreasing in all well-initialised configurations.

Proof. We construct a potential group $\Phi' = (\Phi'_1, \ldots, \Phi'_5)$ and show that it is rapidly decreasing in all well-initialised configurations. So let C denote such a configuration, and let C_{term} denote a terminal configuration reachable from C.

For the sake of readability we defer the definition of the Φ'_i until they are used. However, note that the definition will be independent of C.

The proof proceeds via case distinction based on the properties of C. For the i-th case we show that Φ'_i is active. We implicitly assume that prior cases are excluded, so the proof for case i assumes that the conditions for cases $1, \ldots, i-1$ are not being met.

Case 1. $C(Q_{\text{supp}} \cup Q_{\text{gate}} \cup Q_{\text{reset}}) > |Q| + \beta + 1$. For Φ'_1, the goal is to show that the "leader elections" for each state happen quickly. We set $\Phi'_1(q) := 2$ for $q \in Q_{\text{supp}} \cup Q_{\text{gate}}$, $\Phi'_1(q) := 1$ for $q \in Q_{\text{reset}}$, and $\Phi'_1(q) := 0$ for $q \in Q_{\text{orig}}$. Clearly, the only transitions that affect Φ'_1 are ⟨leader⟩ and the third part of ⟨init-reset⟩, both of these reducing the potential by 1. It is thus not possible for Φ'_1 to increase. One of these transitions is enabled, so Φ'_1 can decrease at C.

In particular, note that for each $q \in S$, where $S := \{q \in Q' : \Phi'_1(q) > 0\}$ are the states with positive potential, there is a transition reducing Φ'_1 using two agents in q. Using T to denote these transitions, we get $\Phi(C) \le 2C(S) \le 2\sum_{t\in T} \operatorname{tmin}_t(C)$ and thus (via Lemma 41), $(\Phi(C) - \Phi(C_{\text{term}}))^2 \le \Phi(C)^2 \le 4|T| \operatorname{speed}_T(C)$. Finally, we note $|T| = |Q| + \beta + 1$.

Case 2. $\pi(C)$ is not terminal. In this case, we will argue that the refined transitions of $\mathcal{P}$ are likely to occur. We define $\Phi'_2((q, \pm)) := \Phi(q)$ for $q \in Q$ (recall that Φ is the potential function of $\mathcal{P}$), and set Φ'_2 to 0 elsewhere. Φ'_2 is reduced precisely by the ⟨execute⟩ transitions, and increased only by the third case of ⟨init-reset⟩.

As we exclude Case 1, we have $C(Q_{\text{supp}} \cup Q_{\text{gate}} \cup Q_{\text{reset}}) = |Q| + \beta + 1$. This implies $\Phi(\pi(C)) = \Phi'_2(C)$ (we even get $\pi(C)(q) = C((q, \pm))$ for $q \in Q$) and ensures that the third case of ⟨init-reset⟩ cannot be executed by any configuration reachable from C. Having $\pi(C)(q) = C((q, \pm))$ for $q \in Q$ then ensures that for a transition of $\mathcal{P}$ enabled at $\pi(C)$, there is a corresponding ⟨execute⟩ transition enabled at C.

We now want to show that $\pi(C)$ is well-initialised.

$$\pi(C)(I) \overset{(1)}{=} C(I') \overset{(2)}{\le} \tfrac{2}{3}|C| - |H'| = \tfrac{2}{3}|C| - |H| - |Q| - \beta - 1 \overset{(3)}{\le} \tfrac{2}{3}C(Q_{\text{orig}}) - |H|$$

At (1), we use that states in I have no incoming transitions in $\mathcal{P}$, so $I \times \{+\}$ has no incoming transitions in $\mathcal{P}'$ and is always empty. (2) follows from C being well-initialised. For (3) we use $C(Q' \setminus Q_{\text{orig}}) = C(Q_{\text{supp}} \cup Q_{\text{gate}} \cup Q_{\text{reset}}) = |Q| + \beta + 1$. Finally, due to $C(Q_{\text{orig}}) = |\pi(C)|$ we derive that $\pi(C)$ is well-initialised.

This allows us to use that Φ is α-rapidly decreasing:

$$(\Phi'_2(C) - \Phi'_2(C_{\text{term}}))^2 = (\Phi(\pi(C)) - \Phi(\pi(C_{\text{term}})))^2 \le \alpha \operatorname{speed}(\pi(C))$$

It remains to show $\operatorname{speed}(\pi(C)) \le \operatorname{speed}_T(C)$, where T are the ⟨execute⟩ transitions. For each transition $t \in T$ we have four corresponding transitions $t_1, \dots, t_4 \in T$, one for each choice of $\pm$. The bound $\operatorname{tmin}_t(\pi(C)) \le \sum_i \operatorname{tmin}_{t_i}(C)$ then follows from the well-known inequality stating that $\min\{\sum_i x_i, \sum_i y_i\} \le \sum_{i,j} \min\{x_i, y_j\}$ holds for any non-negative numbers $x_1, \dots, x_k, y_1, \dots, y_k$.

Case 3. $C(Q \times \{+\}) > 0$. Here, we show that all "+" flags are eliminated quickly. We set $\Phi'_3((q, +)) = 1$ for $q \in Q$ and 0 elsewhere. We know that $\pi(C)$ is terminal (else we would be in Case 2), and it must remain so. Hence ⟨execute⟩ is disabled and no transition increases Φ'_3. Also, the first case of ⟨init-reset⟩ is enabled and can reduce the potential.

For every $q \in Q'$ with $\Phi'_3(q) > 0$ we have a ⟨denotify⟩ transition which decreases Φ'_3 and uses only agents in q. Similarly to Φ'_1, we use T to denote the set of these transitions, and find $(\Phi(C) - \Phi(C_{\text{term}}))^2 \le |T| \operatorname{speed}_T(C)$, noting $|T| \le |Q|$.

Case 4. $C(\{0, \dots, |Q| - 1\}) > 0$ or $C((q, 1)) \ne 1$ for some $q \in \operatorname{supp}(\pi(C))$. In this case, we show that the agents in Q_{supp} stabilise quickly. We use the potential

$$\begin{aligned} \Phi'_4((q, 0)) := \Phi'_4((q, !)) + 1 := 2 \qquad & \text{for } q \in Q \\ \Phi'_4(i) := 3(|Q| - i) \qquad & \text{for } i = 0, \dots, |Q| \end{aligned}$$

Again, Φ'_4 is 0 elsewhere. Due to the conditions on Cases 1 and 3, the only transition producing a state $\{0, \dots, |Q| - 1\}$ is the first part of ⟨reset⟩, which decreases Φ'_4. Otherwise, state $Q \times \{0\}$ cannot be produced. The only other transitions affecting the potential are ⟨detect⟩ and the second case of ⟨init-reset⟩, which both decrease Φ'_4. One of the above transitions, which we again denote by T, is always enabled, so $\operatorname{speed}_T(C) \ge 1$. Additionally, we have $C(Q_{\text{supp}}) = |Q|$ and $C(Q_{\text{reset}}) = 1$, so $\Phi'_4(C) \le 5|Q| \le 5|Q| \operatorname{speed}_T(C)$.

Case 5. C is not terminal. Finally, we consider the speed at which gates stabilise and the computer terminates.

$$\begin{aligned} \Phi'_5((g, b_1, b_2, b_3)) := \mathbf{1}_\perp(b_2) + \mathbf{1}_\perp(b_3) \qquad & \text{for } g \in G \\ \Phi'_5(|Q| + i) := 3(\beta - i) \qquad & \text{for } i = 0, \dots, \beta \end{aligned}$$

where $\mathbf{1}_\perp(\perp) := 1$ and $\mathbf{1}_\perp(0) := \mathbf{1}_\perp(1) := 0$. At this point, only transitions ⟨gate⟩ and the second part of ⟨reset⟩ are active, and both reduce Φ'_5. We denote them by T and, analogous to Phase 4, we get the estimate $\Phi'_5(C) \le 5\beta \operatorname{speed}_T(C)$ and find that one of these transitions is always enabled. □

Remark 48. While it is possible to provide a potential function for $\mathcal{P}'$ based on a potential function for $\mathcal{P}$, this results in large constants for the speed of the protocol. The reason lies in the nature of our computation, which proceeds in multiple phases. As an example, take transition ⟨execute⟩. One of the resulting agents has its flag set to +, which may initiate a reset of every agent in $Q_{\text{supp}} \cup Q_{\text{gate}}$. To pay for this work, every transition of $\mathcal{P}$ would have to reduce the potential by $|Q| + \beta$. However, most of this cost would be wasted; only the last reset needs to be executed fully, and the other resets are likely to be interrupted before completion.

8.6. Removing helpers preserves speed

We now show that Autarkify, the conversion of Section 7.6, is fast.

Proposition 49. *Let $\mathcal{P} = (Q, \delta, I, O, H)$ denote a bounded binary population computer with marked consensus output, and let Φ denote a potential group of size e for $\mathcal{P}$ which is α-rapidly decreasing in all well-initialised configurations.*

Then $\mathcal{P}' := \text{Autarkify}(\mathcal{P})$ (see Section 7.6) has a potential group of size $e+1$ which is $\mathcal{O}(\alpha + \text{size}_2(\mathcal{P})^3)$-rapidly decreasing in all reachable configurations of size at least $6|I| + 10|H|$.

Proof. Let $(\Phi_1, ..., \Phi_e) := \Phi$. We define $\Phi' := (\Phi'_1, \Phi_1, \Phi_2, ..., \Phi_e)$, where:

$$\begin{aligned} \Phi'_1(q) &:= 2 && \text{for } q \in I \\ \Phi'_1(\triangle_i) &:= i+1 && \text{for } i \in \{1, ..., |H|-1\} \\ \Phi'_1(\triangledown_i) &:= i && \text{for } i \in \{2, ..., |H|\} \end{aligned}$$

As usual, other states have potential 0, and we extend Φ_i to $\mathbb{N}^{Q'}$ by setting the weight of states in $Q' \setminus Q$ to 0.

Clearly, transitions Double and Helper decrease Φ'_1, while transitions in δ cannot increase it due to the input specification of Autarkify (page 28).

Now, let C denote a configuration reachable from an input of size at least $2|H|+|I|$. To show that Φ' is rapidly decreasing in C, we differentiate between two cases.

Case 1. If either Double or Helper is enabled at C, we show that Φ'_1 is active. It has already been shown that Φ'_1 cannot decrease. To show that Φ'_1 is rapidly decreasing, let $S := \{q \in Q' : \Phi'_1(q) > 0\}$ denote the states with positive potential. For each state $q \in I \cup \{\triangle_1, ..., \triangle_{|H|-1}\} \subseteq S$ we have a transition using only agents in q. For each other state, i.e. $q = \triangledown_i \in S$ for some i, we observe $C(\triangle_0) \geq C(\triangledown_i)$, as the construction guarantees that enough agents in $\triangle_0$ exist. So in total we have $\Phi(C) \leq |H|C(S)$ and $C(S)^2 \leq |S|\,\text{speed}_{\delta_>}(C)$ by Lemma 41. Using $|S| \leq |I| + 2|H|$, Φ'_1 is $(|I|+2|H|)|H|^2$-rapidly decreasing in C.

Case 2. Otherwise, $C(I) \leq |I|$, $C(Q_{\text{helper}}) < |H|$ and, due to our construction, $C(I') \leq |C|/2$. From the second, we derive $C(Q) > |C| - |H|$, which we combine with the other two to get $C(I \cup I') \leq |I| + |C|/2 < |I| + (C(Q) + |H|)/2$. Rearranging terms yields $C(I \cup I') + |H| \leq C(Q)/2 + |I| + \frac{3}{2}|H|$.

Now, we use $|C| \geq 6|I| + 10|H|$ to get $C(Q) > |C| - |H| \geq 6|I| + 9|H|$, so $C(I \cup I') + |H| \leq |I| + \frac{3}{2}|H| + C(Q)/2 \leq \frac{2}{3}C(Q)$. Noting $C(Q) = |\pi(C)|$, we find that $\pi(C)$ is well-initialised, so Φ is α-rapidly decreasing in $\pi(C)$. This extends directly to Φ'. □

8.7. Converting to consensus output preserves speed

Finally, we prove that speed is preserved by Distribute, the conversion of Section 7.7.

Proposition 50. *Let $\mathcal{P} = (Q, \delta, I, O, \emptyset)$ denote a bounded binary population computer with marked consensus output, and let Φ denote a potential group of size e for $\mathcal{P}$ which is α-rapidly decreasing in all reachable configurations of size at least k.*

Then on inputs of size at least k, $\mathcal{P}' := \text{Distribute}(\mathcal{P})$ stabilises in $\mathcal{O}(e(\sqrt{\alpha}|Q| + \alpha)n^2)$ interactions in expectation.

Proof. By Proposition 46, $\mathcal{P}$ reaches a terminal configuration after $\mathcal{O}(e(\sqrt{\alpha}|Q| + \alpha)n^2)$ interactions in expectation. So it suffices to show that $\mathcal{P}'$ can broadcast the result to all agents also in $\mathcal{O}(e(\sqrt{\alpha}|Q| + \alpha)n^2)$ interactions. We prove a more general result: If $\mathcal{P}'$ reaches a configuration C with $\pi(C)$ terminal within $f(\mathcal{P}', n)$ random interactions in expectation, them $\mathcal{P}'$ stabilises after $\mathcal{O}(f(\mathcal{P}', n) + n^2)$ random interactions in expectation. This was already shown in Section 7.7.2. □

8.8. Proof of Theorem 37

We collect the results of the previous sections to prove:

Theorem 37. *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 $\mathcal{O}(\alpha\, m^4 n^2)$ expected interactions for inputs of size $\Omega(m)$.*

Proof. Let $\mathcal{P} = (Q, \delta, I, O, H)$ be an α-rapid population computer of size m deciding $\text{double}(\varphi)$. Since Binarise satisfies its specification (page 22) and $\mathcal{P}$ satisfies the input specification, the computer $\mathcal{P}_1 := \text{Binarise}(\mathcal{P})$ satisfies the postcondition. In particular, $\mathcal{P}_1$ is binary. Further, the postcondition contains a conjunction of three implications, stating that if $\mathcal{P}$ satisfies additional conditions, then $\mathcal{P}'$ enjoys additional properties. By the definition of α-rapid computers (Definition 35), $\mathcal{P}$ satisfies the premises of these three implications, and so $\mathcal{P}_1$ satisfies their consequences. This shows that: $\text{size}(\mathcal{P}_1) \in \mathcal{O}(\text{size}(\mathcal{P}))$

(because we have $\beta \leq 3$); no initial state of $\mathcal{P}_1$ has incoming transitions; and all configurations that only populate the initial states of $\mathcal{P}_1$ are terminal.

Now, let $\mathcal{P}_2 := \text{Focalise}(\mathcal{P}_1)$, $\mathcal{P}_3 := \text{Autarkify}(\mathcal{P}_2)$, and $\mathcal{P}_4 := \text{Distribute}(\mathcal{P}_3)$. Proceeding exactly as in the proof in Section 7.8, we obtain that $\mathcal{P}_4$ is a population protocol of adjusted size $\mathcal{O}(m)$, and so with $\mathcal{O}(m)$ states, that decides φ for all inputs of size $\Omega(m)$. It remains to prove that $\mathcal{P}_4$ stabilises within $\mathcal{O}(\alpha m^4 n^2)$ expected interactions, which we achieve in several steps:

- By Proposition 46, there is a potential function for $\mathcal{P}_1$ that is $\mathcal{O}(|Q|^2 k^2 \alpha)$-rapidly decreasing for every well-initialised configuration, where k is the maximum arity of the transitions of $\mathcal{P}$. So, in particular, $\mathcal{P}_1$ has an $\alpha_1 \in \mathcal{O}(m^4\alpha)$-rapidly decreasing potential function.
- By Proposition 47, there is a potential group of size 5 for $\mathcal{P}_2$ that is $\mathcal{O}(\alpha_1 + m^2)$-rapidly decreasing, and so $\alpha_2 \in \mathcal{O}(m^4\alpha)$-rapidly decreasing, in all well-initialised configurations.
- By Proposition 49, there is a potential group of size $5 + 1 = 6$ for $\mathcal{P}_3$ that is $\mathcal{O}(\alpha_2 + m^3)$-rapidly decreasing, i.e. $\alpha_3 \in \mathcal{O}(m^4\alpha)$-rapidly decreasing in all reachable configurations of size $\Omega(m)$.
- By Proposition 50, $\mathcal{P}_4$ stabilises in $\mathcal{O}((\sqrt{m^4\alpha}\, m + m^4\alpha)n^2) = \mathcal{O}(\alpha m^4 n^2)$ interactions in expectation. □

9. Conclusions

We have shown that every predicate φ of quantifier-free Presburger arithmetic has a population protocol with $\mathcal{O}(|\varphi|)$ states and $\mathcal{O}(|\varphi|^7 \cdot n^2)$ expected interactions to stabilisation for all inputs of size $\Omega(|\varphi|)$. Therefore, every Presburger predicate has a protocol that is at the same time fast and succinct. 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 [12]; further, it is known that every protocol for the majority predicate requires $\Omega(n^2/\operatorname{polylog} n)$ interactions.

Our construction is very modular. We have introduced population computers, a model that extends population protocols with three very useful features: interactions of arbitrary arity, helpers, and generalised output functions. We have designed conversions that, loosely speaking, allow us to transform an arbitrary computer into an equivalent protocol by eliminating each of these features. The conversions are independent of each other. Further, we have proved a powerful theorem showing that in order to prove quantitative properties about the speed of the protocol it suffices to prove qualitative properties of the computer.

In future work we plan to study the existence of succinct protocols with $\mathcal{O}(\text{polylog}(n))$ convergence time, either in the presence of leaders, as in [6], or using the construction of [22]. As mentioned in the introduction, our work can be considered a first step in this direction.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Philipp Czerner reports financial support was provided by European Research Council. Philipp Czerner reports financial support was provided by German Research Foundation. Roland Guttenberg reports financial support was provided by European Research Council. Martin Helfrich reports financial support was provided by European Research Council. Martin Helfrich reports financial support was provided by German Research Foundation. Javier Esparza reports financial support was provided by European Research Council. Javier Esparza reports financial support was provided by German Research Foundation.

Data availability

No data was used for the research described in the article.

Acknowledgments

We thank the anonymous reviewers for many helpful remarks. In particular, one remark led to Lemma 23, which in turn led to a nicer formulation of Theorem 9, one of our main results.

References

[1] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, Ronald L. Rivest, Time-space trade-offs in population protocols, in: SODA, SIAM, 2017, pp. 2560–2579.
[2] Alistarh Dan, Rati Gelashvili, Recent algorithmic advances in population protocols, SIGACT News 49 (3) (2018) 63–73.
[3] Dan Alistarh, Rati Gelashvili, Milan Vojnovic, Fast and exact majority in population protocols, in: PODC, ACM, 2015, pp. 47–56.
[4] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, René Peralta, Computation in networks of passively mobile finite-state sensors, in: PODC, ACM, 2004, pp. 290–299.

[5] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, René Peralta, Computation in networks of passively mobile finite-state sensors, Distrib. Comput. 18 (4) (2006) 235–253.
[6] Dana Angluin, James Aspnes, David Eisenstat, Fast computation by population protocols with a leader, Distrib. Comput. 21 (3) (2008) 183–199.
[7] Dana Angluin, James Aspnes, David Eisenstat, Eric Ruppert, The computational power of population protocols, Distrib. Comput. 20 (4) (2007) 279–304.
[8] Amanda Belleville, David Doty, David Soloveichik, Hardness of computing and approximating predicates and functions with leaderless population protocols, in: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, Anca Muscholl (Eds.), 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10–14, 2017, Warsaw, Poland, in: LIPIcs, vol. 80, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 141:1–141:14.
[9] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, Tomasz Radzik, Time-space trade-offs in population protocols for the majority problem, Distrib. Comput. 34 (2) (2021) 91–111.
[10] Petra Berenbrink, George Giakkoupis, Peter Kling, Optimal time and space leader election in population protocols, in: STOC, ACM, 2020, pp. 119–129.
[11] Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, Stefan Jaax, Succinct population protocols for Presburger arithmetic, in: STACS, in: LIPIcs, vol. 154, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 40:1–40:15.
[12] Michael Blondin, Javier Esparza, Stefan Jaax, Large flocks of small birds: on the minimal size of population protocols, in: STACS, in: LIPIcs, vol. 96, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 16:1–16:14.
[13] Michael Blondin, Javier Esparza, Stefan Jaax, Expressive power of broadcast consensus protocols, in: CONCUR, in: LIPIcs, vol. 140, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 31:1–31:16.
[14] Robert Brijder, David Doty, David Soloveichik, Democratic, existential, and consensus-based output conventions in stable computation by chemical reaction networks, Nat. Comput. 17 (1) (2018) 97–108.
[15] Luca Cardelli, Attila Csikasz-Nagy, The cell cycle switch computes approximate majority, Sci. Rep. 2 (2012), https://doi.org/10.1038/srep00656.
[16] David Doty, Mahsa Eftekhari, Efficient size estimation and impossibility of termination in uniform dense population protocols, in: PODC, ACM, 2019, pp. 34–42.
[17] David Doty, Mahsa Eftekhari, Leszek Gasieniec, Eric E. Severson, Przemyslaw Uznanski, Grzegorz Stachowiak, A time and space optimal stable population protocol solving exact majority, in: 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7–10, 2022, IEEE, 2021, pp. 1044–1055.
[18] Robert Elsässer, Tomasz Radzik, Recent results in population protocols for exact majority and leader election, Bull. Eur. Assoc. Theor. Comput. Sci. 126 (2018).
[19] Nissim Francez, Fairness, Texts and Monographs in Computer Science, Springer, 1986.
[20] Bernd Gärtner, Jirí Matousek, Understanding and Using Linear Programming, Universitext, Springer, 2007.
[21] Christoph Haase, A survival guide to Presburger arithmetic, ACM SIGLOG News 5 (3) (2018) 67–82.
[22] Adrian Kosowski, Przemyslaw Uznanski, Brief announcement: population protocols are fast, in: Calvin Newport, Idit Keidar (Eds.), Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23–27, 2018, ACM, 2018, pp. 475–477, https://dl.acm.org/citation.cfm?id=3212788.
[23] Christos H. Papadimitriou, On the complexity of integer programming, J. ACM 28 (4) (1981) 765–768.
[24] David Soloveichik, Matthew Cook, Erik Winfree, Jehoshua Bruck, Computation with finite stochastic chemical reaction networks, Nat. Comput. 7 (4) (2008) 615–633.
[25] Frank Spitzer, Principles of Random Walk, vol. 34, Springer Science & Business Media, 2013.

D. The Black Ninjas and the Sniper: On Robust Population Protocols

Title	The Black Ninjas and the Sniper: On Robust Population Protocols
Authors	Benno Lossin, Philipp Czerner, Javier Esparza, Roland Guttenberg, Tobias Prehn
Venue	Principles of Verification: Cycling the Probabilistic Landscape, Part III, 2024
Publisher	Springer
DOI	10.1007/978-3-031-75778-5_10

The Black Ninjas and the Sniper: On Robust Population Protocols

Benno Lossin(✉), Philipp Czerner, Javier Esparza, Roland Guttenberg, and Tobias Prehn

Technical University of Munich, Munich, Germany
{lossin,czerner,esparza,guttenbe}@in.tum.de

Abstract. Population protocols are a model of distributed computation in which an arbitrary number of indistinguishable finite-state agents interact in pairs to decide some property of their initial configuration. We investigate the behaviour of population protocols under adversarial faults that cause agents to silently crash and no longer interact with other agents. As a starting point, we consider the property "the number of agents exceeds a given threshold t", represented by the predicate $x \geq t$, and show that the standard protocol for $x \geq t$ is very fragile: one single crash in a computation with $x := 2t - 1$ agents can already cause the protocol to answer incorrectly that $x \geq t$ does not hold. However, a slightly less known protocol is *robust*: for any number $t' \geq t$ of agents, at least $t' - t + 1$ crashes must occur for the protocol to answer that the property does not hold.

We formally define robustness for arbitrary population protocols, and investigate the question whether every predicate computable by population protocols has a robust protocol. Angluin et al. proved in 2007 that population protocols decide exactly the Presburger predicates, which can be represented as Boolean combinations of threshold predicates of the form $\sum_{i=1}^{n} a_i \cdot x_i \geq t$ for $a_1, ..., a_n, t \in \mathbb{Z}$ and modulo prdicates of the form $\sum_{i=1}^{n} a_i \cdot x_i \bmod m \geq t$ for $a_1, \ldots, a_n, m, t \in \mathbb{N}$. We design robust protocols for all threshold and modulo predicates. We also show that, unfortunately, the techniques in the literature that construct a protocol for a Boolean combination of predicates given protocols for the conjuncts do not preserve robustness. So the question remains open.

For Joost-Pieter, Honorary Sensei of the Black Ninjas.

1 Introduction

The Black Ninjas were used in [5] to provide a gentle introduction to population protocols, a model of distributed computation introduced by Angluin et al. in [3] and since then very much studied. We quote (with some changes) the first lines of the introduction to [5]:

N. Jansen et al. (Eds.): Principles of Verification: Cycling the Probabilistic Landscape, LNCS 15262, pp. 206–233, 2025.
https://doi.org/10.1007/978-3-031-75778-5_10

> The Black Ninjas are an ancient secret society of warriors, so secret that its members do not even know each other or how many they are. When there is a matter to discuss, Sensei, the founder of the society, sends individual messages to each ninja, asking them to meet.
> As it happens, all ninjas have just received a note asking them to meet in a certain Zen garden at midnight, wearing their black uniform, in order to attack a fortress of the Evil Powers. When the ninjas reach the garden in the gloomy night, the weather is so dreadful that it is impossible to see or hear anything at all. This causes a problem, because the ninjas should only attack if they are at least 64, and there are always no-shows: Some ninjas are wounded, others are under evil spells, and others still have tax forms to fill.

Is there a way for the ninjas to find out, despite the adverse conditions, if at least 64 of them have shown up?

A First Protocol. Yes, there is a way. Sensei has looked at the weather forecast, and the note sent to the ninjas contains detailed instructions on how to proceed after they reach the garden. Each ninja must bring pebbles in their pockets, and a pouch initially containing one pebble. When they reach the garden, they must start to wander *randomly* around the garden. When two ninjas with n_1 and n_2 pebbles in their pouches happen to bump into each other, they compute $n_1 + n_2$ and proceed as follows:

- If $n_1 + n_2 < 64$, then one ninja gives her pebbles to the other, that is, after the encounter their pouches contain $n_1 + n_2$ and 0 pebbles, respectively.
- If $n_1 + n_2 \geq 64$, both ninjas put 64 pebbles in their pouches.

Formally, at every moment in time each ninja is in one state from the set $Q := \{0, 1, \ldots, 64\}$, representing the number of pebbles in her pouch. The possible transitions are

$$n_1, n_2 \mapsto \begin{cases} n_1 + n_2, 0 & \text{if } n_1 + n_2 < 64 \\ 64, 64 & \text{otherwise} \end{cases}$$

The protocol can be visualized as a Petri net with one place for each state, and one Petri net transition for each transition of the protocol. The Petri net representation of the protocol for attack threshold 4 instead of 64 is shown on the left of Fig. 1.

Sensei reasons as follows: If at least 64 ninjas show up, then eventually two ninjas with at least 64 pebbles between them interact. These ninjas move to state 64, and from then on word quickly spreads, and eventually all ninjas are in state 64, and stay there forever. If, on the contrary, less than 64 ninjas show up, then no ninja ever reaches state 64. In both cases, the ninjas eventually reach a *consensus*: all of them are in state 64, or none is, and that consensus does not change, it is *stable*. Further, since Sensei knows that the society consists of 200 ninjas—a fact the ninjas themselves ignore—and she is good at math, she computes that after 10 min the consensus has been reached with high probability

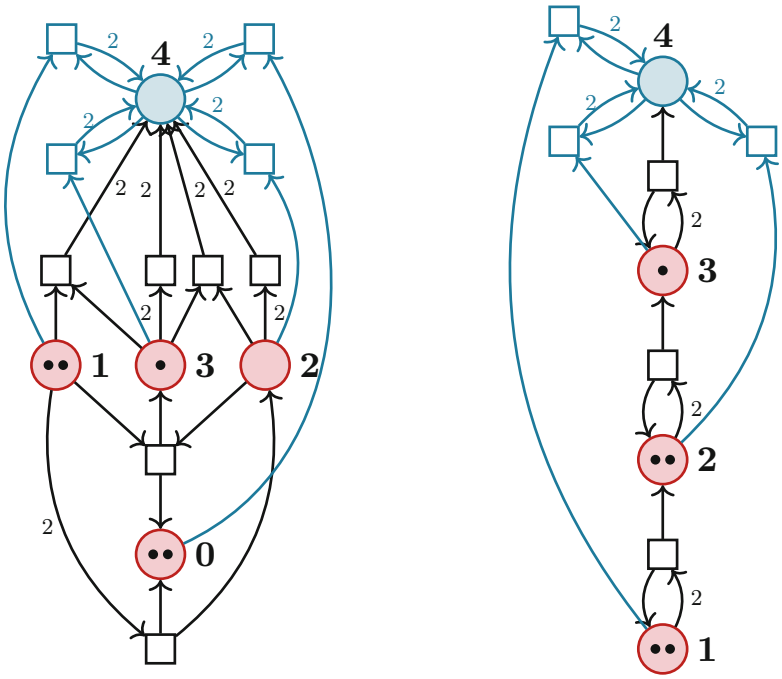


Fig. 1. Petri nets for the first (left) and second (right) protocols with attack threshold 4. Accepting states are shaded in blue, rejecting states in red. The number of pebbles (left) or the level (right) of a state are written next to its corresponding place in boldface. The tokens in the places show configurations reachable from the initial configuration with 5 ninjas. The accumulation transitions are colored in blue. (Color figure online)

(ninjas move very fast!). So her final instruction to the ninjas is: Execute the protocol for 10 min, and then attack if you are in state 64.

1.1 The Sniper and a Second Protocol

The ninjas use the protocol to successfully attack several fortresses of the Evil Powers. However, a spy finds out the point of their next gathering, and the Evil Powers send a sniper equipped with a night vision device and a powerful gun whose bullets can silently vaporize a ninja leaving no trace. No less than 126 ninjas show up at the gathering, but the sniper manages to prevent the attack by eliminating one single ninja! The sniper lets the ninjas wander around, executing the protocol, until they reach a configuration in which two ninjas have pouches with 63 pebbles each, and all other pouches are empty. Right before these two ninjas bump into each other, the sniper eliminates one of them. The other ninjas, unaware of this, are left in a configuration in which one ninja has 63 pebbles in her pouch, and the pouches of the other 124 ninjas are empty. The ninjas keep interacting, but none ever reach state 64.

When Sensei finds out about the sniper, she initially despairs. But then she realizes: munitions are expensive, and the sniper can carry only a few rounds, so perhaps there is a remedy. So she searches for a *robust* protocol. If $n \geq 64$ ninjas reach the garden, the sniper can only prevent an attack by eliminating more than $n - 64$ ninjas. Clearly, this is the best she can achieve, because if the

sniper succeeds in removing more than $n - 64$ ninjas, then the ninjas should not attack anyway.

Sensei is a capable leader, and she finds a robust protocol! The states are now the numbers 1 to 64. Sensei visualizes these numbers as the floors or levels of a tower with 64 levels, and visualizes a ninja in state ℓ as occupying the ℓ-th level. Initially all ninjas are in the first level. When two ninjas bump into each other, they interact as follows:

- if they are both in the same level, say the ℓ-th, and $\ell < 64$, then one of them moves to level $\ell + 1$, and the other stays at level ℓ.
- if one of them is in level 64, then the other moves to level 64 too (if not yet there);

Formally, at every moment in time each ninja is in a state from the set $Q := \{1, 2, ..., 64\}$. The possible transitions are

$$\ell_1, \ell_2 \mapsto \begin{cases} \ell_1 + 1, \ell_2 & \text{if } \ell_1 = \ell_2, \text{ and } \ell_1 < 64 \\ 64, 64 & \text{if } \ell_1 = 64 \text{ or } \ell_2 = 64 \\ \ell_1, \ell_2 & \text{otherwise} \end{cases}$$

Again, Sensei asks the ninjas to execute the protocol for a certain time, and then attack if they are in level 64. The protocol can again be visualized as a Petri net, shown on the right of Fig. 1.

Sensei first shows that the protocol works correctly in the absence of the sniper. Eventually, each level is populated by at most one ninja; indeed, any two ninjas in the same level eventually interact and one of them moves up. Therefore, if there are at least 64 ninjas, then one of them eventually reaches the top of the tower, and eventually brings all other ninjas to the top.

To prove that the protocol is robust, Sensei observes that the argument above works not only for the initial configuration with all ninjas in the ground level, but *for any configuration with at least 64 ninjas.* Indeed, in any such configuration either one ninja is already at the top, or, by the pigeonhole principle, eventually one ninja moves one level up. Therefore, unless the sniper brings the number of ninjas below 64, the ninjas attack.

Sensei's Question. Sensei is not satisfied yet. At some meetings, the ninjas must decide by majority if they attack or not, that is, they attack only if they are at least 64 and a majority wants to attack. Sensei has a majority protocol for this; in her jargon she says that the protocol *decides* the predicate $Q_2(n_{\text{Y}}, n_{\text{N}}) \Leftrightarrow n_{\text{Y}} + n_{\text{N}} \geq 64 \wedge n_{\text{Y}} > n_{\text{N}}$ (instead of the predicate $Q_1(n) \Leftrightarrow n \geq 64$ of the first protocol). She also has protocols for qualified majority (i.e., the ninjas attack only if at least 2/3 of them are in favour, corresponding to the predicate $Q_2(n_{\text{Y}}, n_{\text{N}}) \Leftrightarrow 2n_{\text{Y}} \geq 3n_{\text{N}}$), and for whether the number of ninjas showing up is a multiple of 7, corresponding to $Q_3(n) \Leftrightarrow \exists k\colon n = 7k$.[1] In fact, Sensei has read [4], in which

[1] Ninjas are superstitious.

Angluin et al. introduced this kind of protocols and showed how to construct a protocol for any predicate expressible in Presburger arithmetic. She has even read the improved version [6], which shows how to construct protocols slightly faster than those of [4] and, more importantly, with polynomially instead of exponentially many states in the size of the Presburger formula (assuming it is quantifier-free). However, these protocols are not robust. For example, the protocol of [6] for the predicate $x \geq 64$ is the first protocol of the introduction. Since these protocols are very dangerous in the presence of snipers, Sensei has the following question:

Is there a robust protocol for every Presburger predicate?

We do not know yet. In this paper we give a positive partial answer. We construct robust protocols for *threshold predicates* of the form $\sum_{i=1}^{k} a_i x_i \geq b$, where $a_1, \ldots, a_k, b \in \mathbb{Z}$, and for *modulo predicates* of the form $\sum_{i=1}^{k} a_i x_i \equiv_c b$, where $a_1, \ldots, a_k \in \mathbb{Z}$, $c \in \mathbb{N}$, and $0 \leq b \leq c - 1$.

Related Work. The literature on fault-tolerance and robust distributed algorithms is very large, and so we consider only work on population protocols or closely related models. In [7], Delporte-Gallet *et al.* initiated the study of population protocols with essentially the same fault model as ours. The paper provides a construction that, loosely speaking, yields robust protocols with respect to a given number of failures, independent of the number of agents. The construction we give for modulo protocols is an immediate consequence of this idea. Delporte-Gallet and collaborators have published many other papers on fault-tolerant consensus and leader election algorithm in different kinds of message-passing networks.

There also exists previous work on other fault models. In [8], Guerraoui and Ruppert study community protocols, a model that extends population protocols by assigning unique identities to agents. They present a universal construction that, given a predicate in NSPACE($\log n$), outputs a community protocol deciding it. The construction is robust with respect to Byzantine failures of a constant number of agents. More recently, Alistarh *et al.* have studied robustness of population protocols and chemical reaction networks with respect to leaks [1,2]. Loosely speaking, this is a failure model in which agents can probabilistically move to adversarially chosen states.

Structure of the Paper. Section 2 introduces the syntax of population protocols and their semantics in the presence of a sniper. Section 3 formalizes the notion of robustness, and formally proves that the first and second of Sensei's protocols are not robust and robust, respectively. Sections 4 and 5 give robust population protocols for all threshold and modulo predicates, respectively. Section 6 explains the problem with robustness of the standard construction for Boolean combinations.

2 Generalized Protocols and Population Protocols with Sniper

After some basic notations, we introduce the syntax and the semantics in the presence of a sniper of population protocols. For convenience, we introduce the model as a particular instance of a slightly more general model called generalized protocols.

Preliminaries: Multisets and Intervals. Given a finite set Q, a *multiset* is a function $M : Q \to \mathbb{N}$. We denote the set of multisets over Q by $\mathbb{N}^Q$. Multisets can be added, subtracted and compared componentwise. We define the *size* of a multiset M as $|M| := \sum_{q \in Q} M(q)$. The set of multisets of size k is denoted by $\mathbb{N}^Q_k$. The *support* of a multiset is $[\![M]\!] := \{q \in Q : M(q) \neq 0\}$. We use a set-like notation to write multisets explicitly: $\wr x, y, y, 3 \cdot z \wr$ denotes the multiset M with $M(x) = 1$, $M(y) = 2$, $M(z) = 3$ and otherwise 0. When the set Q contains natural numbers, we write the numbers from Q in bold to differentiate them from their multiplicity: $\wr \mathbf{1}, 2 \cdot \mathbf{3} \wr$.

We write $[a, b)$ to denote the set $\{x \in \mathbb{Z} : a \leq x < b\}$. We define the length of an interval as $\text{len}([a, b)) := b - a$.

Generalized Protocols: Syntax. A *generalized protocol* is a tuple: $\mathcal{P} = (Q, \delta, I, O)$ where

- Q is a finite set of states,
- $\delta \subseteq \mathbb{N}^Q_2 \times \mathbb{N}^Q_2$ is the transition relation,
- $I \subseteq Q$ is the set of initial states,
- $O : \mathbb{N}^Q \to \{1, 0, \bot\}$ the output function.

Elements of $\mathbb{N}^Q$ are called *configurations* and elements of δ *transitions*. Configurations consist of *agents*, which are in a specific state $q \in Q$. We also call agents *ninjas*. Intuitively, the output function assigns to each configuration either a decision (1 or 0), or no decision ($\bot$). We let $p, q \mapsto p', q'$ denote that $(\wr p, q \wr, \wr p', q' \wr) \in \delta$. The *preset* and *poset* of a transition $t = (p, q \mapsto p', q')$ are $\textsc{pre}(t) := \wr p, q \wr$ and $\textsc{post}(t) := \wr p', q' \wr$. We call a transition t with $\textsc{pre}(t) = \textsc{post}(t)$ *silent*.

Additionally we assume that for every pair of states $p, q \in Q$, there exists a transition t with $\textsc{pre}(t) = \wr p, q \wr$. If the protocol does not define such a transition explicitly, we add the respective identity transition (such a transition is silent and does not change the behavior of the protocol).

Steps, Snipes, and Moves. Let $C, D \in \mathbb{N}^Q$ be two configurations.

- We write $C \to D$ if $C = D$ or there exists some transition t s.t. $\textsc{pre}(t) \leq C$ and $D = C + \textsc{post}(t) - \textsc{pre}(t)$. We call $C \to D$ a *move* of the protocol.
  A configuration C is *terminal* if there is no configuration $D \neq C$ such that $C \to D$.

- We write $C \hookrightarrow D$ if $C = D + S$ for some configuration S such that $|S| = 1$. We call $C \hookrightarrow D$ a *snipe*.
- We write $C \Rightarrow D$ if either $C \to D$ or $C \hookrightarrow D$, and call $C \Rightarrow D$ a *step*. That is, a step is either a move of the protocol, or a snipe.
- We let $\to^*$, $\hookrightarrow^*$, and $\Rightarrow^*$ denote the reflexive and transitive closure of $\to$, $\hookrightarrow$ and $\Rightarrow$, respectively.
- If $C \to^* D$ and $E \leq D$, we say that C *covers* E.

i-executions, i-reachability, Fairness. Fix $i \geq 0$. Loosely speaking, an i-execution is an infinite sequence of steps containing at most i snipes. Formally, an *i-execution* is an infinite sequence $\pi = \{C_j\}_{j \in \mathbb{N}}$ of configurations such that $C_j \Rightarrow C_{j+1}$ for every $j \geq 0$, and $C_j \hookrightarrow C_{j+1}$ for at most i values of j. The *output* of π is

$$\textsc{out}(\pi) := \begin{cases} 1 & \text{if } \exists n \in \mathbb{N} : \forall i \geq n : O(\pi(i)) = 1 \\ 0 & \text{if } \exists n \in \mathbb{N} : \forall i \geq n : O(\pi(i)) = 0 \\ \bot & \text{else} \end{cases}$$

An i-execution π is *fair* iff for every configuration $D \in \mathbb{N}^Q$ the following holds: if D can be reached from infinitely many configurations of π by executing sequences of moves, then D appears in π. It is easy to see that fairness can be equivalently defined as follows: if D can be reached *in one move* from infinitely many configurations of π, then D appears *infinitely often* in π. Formally, for every configuration D, if $\pi(j) \to D$ for infinitely many $j \in \mathbb{N}$, then $\pi(j) = D$ for infinitely many $j \in \mathbb{N}$.

Predicate Decided by a Generalized Protocol. Fix $i \geq 0$. Given a configuration $C \in \mathbb{N}^Q$ we define its i-output, denoted $\textsc{out}_i(C)$, as follows:

$$\textsc{out}_i(C) := \begin{cases} 0 & \text{if } \forall \pi \text{ fair } i\text{-execution}, \pi(0) = C : \textsc{out}(\pi) = 0 \\ 1 & \text{if } \forall \pi \text{ fair } i\text{-execution}, \pi(0) = C : \textsc{out}(\pi) = 1 \\ \bot & \text{else} \end{cases}$$

A generalized protocol i-decides a predicate $\varphi \colon \mathbb{N}^I \to \{1, 0\}$ if $\textsc{out}_i(C) = \varphi(C)$ for every $C \in \mathbb{N}^I$.

Consensus Output Functions, Population Protocols. An output function $O : \mathbb{N}^Q \to \{1, 0\}$ is a *consensus function* if there is a partition of Q into two sets Q_+ and Q_- of *accepting* and *rejecting* states such that for every configuration C we have: $O(C) = 1$ if$[\![C]\!] \subseteq Q_+$, $O(C) = 0$ if $[\![C]\!] \subseteq Q_-$, and $O(C) = \bot$ otherwise. A generalized protocol is a *population protocol* if its output function is a consensus function.

Intuitively, if we interpret Q_+ and Q_- as the set of states in which ninjas have accept and reject *opinions*, respectively, then a configuration of a population protocol has output $b \in \{1, 0\}$ iff all ninjas currently have the same opinion, i.e., have currently reached a *consensus*, and $\bot$ otherwise.

Example 2.1. We restate the first protocol from the introduction using this formalism for an arbitrary threshold t:

Protocol Pebbles**.**

States.	$Q = [0, t]$
Input.	$I = \{1\}$
Output.	$O(x) = 1$ if $x = t$, and $O(x) = 0$ otherwise
Transitions.	$\delta = \{(x, y) \mapsto (x + y, 0) : x, y \in Q \wedge x + y < t\} \cup$ $\{(x, y) \mapsto (t, t) : x, y \in Q \wedge x + y \geq t\}$

3 Robustness

Recall the first protocol from the introduction. Even if the initial number of ninjas is 126, the sniper can still prevent the attack by taking one single ninja down. However, this is the case only because the sniper can intervene at any moment. Indeed, if the sniper can only intervene before the protocol starts, she must take at least 62 ninjas down to prevent the attack. We say that the initial configuration with 125 ninjas has an *initial tolerance* of 62, but a *global tolerance* of 1. It is easy to see that the initial tolerance for an arbitrary configuration with $n \geq 64$ is $n - 64$, while the global tolerance is $\lceil \frac{n}{63} \rceil - 2$. However, in the second protocol, the initial and global tolerance of any initial configuration with $n \geq 64$ ninjas coincide, and are equal to $n - 64$.

Observe that the initial tolerance of an initial configuration with n ninjas is $n - 64$ for *any* protocol deciding whether $n \geq 64$ holds. Intuitively, it gives an upper bound on how fault-tolerant a protocol for this predicate can be. On the contrary, the global tolerance depends on the protocol. This suggests to define the class of robust protocols as those protocols whose global tolerance is equal to the initial tolerance of all initial configurations. We proceed to formalize this notion.

Definition 3.1. *Let P be a well-specified generalized protocol, and let $C \in \mathbb{N}^I$ be an initial configuration of P.*

- *The* initial tolerance *of C, denoted $InTol(C)$, is the largest number i such that $\text{OUT}_0(C) = \text{OUT}_0(D)$ for every configuration D with $C \hookrightarrow^i D$.*
- *The* global tolerance *of C, denoted $Tol(C)$, is the largest number i such that $\text{OUT}_i(C) = \text{OUT}_0(C)$.*

P is robust in *C if $Tol(C) = InTol(C)$. We call P* robust *if it is robust in every initial configuration.*

We note a couple of useful facts about robustness:

Remark 3.2.
- Negating a population protocol via changing accepting states to be rejecting states and vice-versa, the protocol still remains robust. This is because neither the initial tolerance nor the global tolerance change.

– Sniping agents in initial states that have never interacted cannot change the output when considering snipes less than the initial tolerance. This follows directly from the definition: when sniping a agent in an initial state, it is as if the initial configuration was different to begin with.

3.1 Examples

We show that with our notion of robustness, the original population protocol for threshold predicates by Angluin et al. [3] is not robust.

Proposition 3.3. *The protocol of Example 2.1 is not robust for any* $t \geq 3$.

Proof. Assume the protocol is robust. Let $q := t - 1 \in Q$ denote the last non-accepting state and consider the following fair 1-execution:

$$\wr(t+1) \cdot \mathbf{1}\wr \to^* \wr\mathbf{q}, \mathbf{2}, (t-1) \cdot \mathbf{0}\wr \hookrightarrow \wr\mathbf{2}, (t-1) \cdot \mathbf{0}\wr \to ...$$

The first configuration is an initial configuration with an initial tolerance of 1. Without sniping, it is accepted. However, the last configuration is terminal and rejecting. This contradicts the assumption that the protocol is robust. □

Remark 3.4. We can say even more about the robustness of this protocol. If the sniper is only permitted to snipe a single ninja, then we know that they can at worst snipe a ninja with $t - 1$ pebbles. Thus if the input configuration has $t - 1$ additional ninjas, then the output is still correct. In other words, if the initial tolerance of a configuration is i, then the global tolerance of the protocol is $\lfloor \frac{i}{t-1} \rfloor$.

In this paper, we focus our attention on robust protocols and as such we do not need a fine-grained analysis of protocols that fail to be robust. But if one wants to analyze non-robust protocols, it is useful to define robustness with a parameter that is the ratio of initial and global tolerance.

We give the formal definition of Sensei's second protocol, which we call the Tower protocol.

Protocol Tower**.**

States.	$Q = [1, t]$
Input.	$I = \{1\}$
Output.	$O(x) = 1$ iff $x = t$
Transitions.	$\delta = \{(x, x) \mapsto (x, x+1) : x \in Q \wedge x < t\} \cup$ $\{(x, t) \mapsto (t, t) : x \in Q, x < t\}$

Let us prove that the Tower protocol is robust.

Proposition 3.5. *The* Tower *protocol is robust for every* t.

Proof. Let $C \in \mathbb{N}^I$ be an initial configuration. If $|C| < t$, then $\text{OUT}_0(C) = 0$ and C cannot *cover* the accepting state, i.e. it cannot put an agent into t. Since the transition relation is monotonic ($C_1 \to^* C_2$ implies $C_1 + D \to^* C_2 + D$ for all $C_1, C_2, D \in \mathbb{N}^Q$), sniping cannot lead to reaching the accepting state. Therefore $\text{OUT}_i(C) = 0$ for all i.

If $|C| \geq t$, then we know that $|C| \geq t + \mathit{InTol}(C)$ by definition of the initial tolerance. To show $\mathit{InTol}(C) = \mathit{Tol}(C)$, we show that the protocol accepts any configuration with at least t agents.

First, we note that each transition increases the value of $\sum_{q \in Q} q \cdot C(q)$, so only finitely many transitions can be executed. Thus the protocol eventually reaches a terminal configuration D with $|D| \geq t$. If there is an agent in state t, then all other agents must be in state t as well - otherwise the second transition could be executed. So D is accepting in this case. If there is no agent in the t state, then we have at least t agents distributed among $t - 1$ states, so there is some state $x < t$ with at least two agents. But this is a contradiction to D being terminal, since those two agents can initiate a transition. □

4 Robust Threshold Protocols

In this section we construct robust protocols for *threshold predicates*, that is, predicates of the form,

$$\varphi(x) = \left(\sum_{i=1}^{n} a_i \cdot x_i \geq t \right)$$

where $a_1, ..., a_n, t \in \mathbb{Z}$. We proceed in three steps. Section 4.1 constructs robust protocols for threshold predicates with positive coefficients a_i and an arbitrary threshold t. Section 4.2 does the same for predicates with arbitrary coefficients and threshold 1; we call them *generalized majority* predicates, because they generalize the majority predicate $x - y \geq 1 \iff x - y > 0$. Finally, Sect. 4.3 combines the protocols of the two previous sections to present protocols for arbitrary threshold predicates.

4.1 Threshold Predicates with Positive Coefficients

In this section we study the case $a_i > 0$. We can wlog assume that $t \geq a_i$ for all i: If $t < 0$, then φ is equivalent to true, and we are done. Otherwise, since all coefficients $a_i > 0$, we replace every a_i by $min(a_i, t)$ to obtain a new predicate φ': Clearly if some ninja for a coefficient a_i which was decreased occurs, then both φ and φ' are automatically true. Since changing a coefficient which does not occur does not change satisfaction either, φ and φ' are equivalent.

The Black Ninjas have recently recruited a cohort of rookie ninjas. In battle, two veterans are as good as three rookies. When the ninjas meet, the question to decide is whether their battle power is at least 192 (the power of 64 veterans). In other words, the ninjas must conduct a protocol to decide if $3v + 2r \geq 192$, where v and r are the numbers of veterans and rookies, respectively.

The protocol of [3] for this case is an easy generalization of the protocol for $n \geq 64$. Each ninja brings a pouch, but veterans put three pebbles in it, while rookies only put two. Otherwise the protocol works as before: when ninjas interact, one gives her pebbles to the other, until two interacting ninjas observe that their joint number of pebbles is at least 192. The protocol fails to be robust, for the same reason as the previous one.

Sensei comes up with a generalization of the Tower protocol. She visualizes again a Tower with 192 floors. However, veterans now occupy three consecutive floors, e.g. floors 100, 101, and 102, and rookies occupy two consecutive floors, e.g. 101 and 102 (see Fig. 2). If two ninjas that interact share at least one floor, then the ninja with the higher lowest floor moves one floor up. For example, if the two ninjas just mentioned interact, then the rookie moves one floor up, occupying now the floors 102 and 103 (if both ninjas have the same lowest floor, then any one of them moves up). The other ninja stays where she is. Initially, all veterans occupy the floors 0, 1, and 2, and all rookies occupy the floors 0 and 1.

Sensei's reasoning is similar to the previous case: eventually each floor is occupied by at most one ninja (although she has to think a bit to establish this). Therefore, a ninja eventually reaches the top floor iff $3v + 2r \geq 192$. As before, a ninja that reaches the top floor tells any other ninja she interacts with to move to the top floor too. We call this protocol InhomTower, and proceed to define it formally.

Protocol InhomTower.

The states of the protocol are intervals of natural numbers. For convenience we use semi-closed intervals $[s, e)$. A ninja in state $[s, e)$ occupies the floors $s, s+1, \ldots, e-1$. The initial states are the intervals starting at zero. Formally we set

States. $Q = \{[s, s + a_i) : i \in [1, n] \wedge 0 \leq s \wedge s + a_i \leq t + 1\}$

Input. $I = \{[0, a_i) : i \in [1, n]\}$

Output. $O([s, e]) = 1$ iff $e = t + 1$

Transitions.

– For every two non-disjoint intervals $[s_1, e_1), [s_2, e_2) \in Q$ such that $s_1 \leq s_2$ and $e_1, e_2 < t + 1$ the second interval moves one step "upwards":

$$[s_1, e_1), [s_2, e_2) \mapsto [s_1, e_1), [s_2 + 1, e_2 + 1) \qquad \langle \mathsf{step} \rangle$$

– For two intervals $[s_1, t + 1), [s_2, e_2) \in Q$ with $e_2 \leq t$, i.e. the first interval has reached the top, the second moves to the top as well.

$$[s_1, t + 1), [s_2, e_2) \mapsto [s_1, t + 1), [t + 1 - (e_2 - s_2), t + 1) \qquad \langle \mathsf{accum} \rangle$$

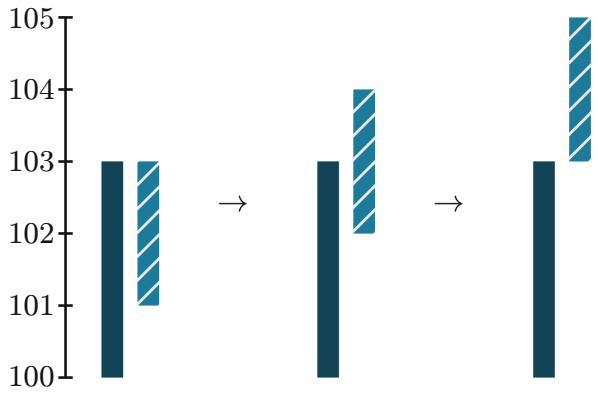


Fig. 2. Two steps using the ⟨step⟩ transition between a veteran (dark blue) and a rookie (light blue, striped) in the InhomTower protocol. (Color figure online)

Theorem 4.1. InhomTower *computes* φ *and is robust.*

Proof. Given an agent in state $[s, e) \in Q$, we say that it *occupies* levels $s, ..., e-1$.

Let $C \in \mathbb{N}^I$ denote an initial configuration and let $k := \mathit{InTol}(C)$. We show $\text{OUT}_0(C) = \varphi(C)$ (i.e. the protocol has the correct output on C) and $\text{OUT}_k(C) = \text{OUT}_0(C)$ (i.e. the protocol is robust on C).

Case 1: $\sum_{q \in Q} \text{len}(q)C(q) < t$**.** This implies $\varphi(C) = 0$. Here, we claim that no agent reaches a state $[s, t+1)$ and ⟨accum⟩ is never executed, which implies $\text{OUT}_0(C) = 0$. To see that this is true, note the invariant that the set of levels in $[0, t]$ occupied by at least one agent is contiguous and contains level 0. Formally, for any D 0-reachable from C we have $\bigcup_{D(q)>0} q = [0, i]$ for some i. We prove this invariant by induction. The base case follows from the definition of I, and for the induction step we use that ⟨accum⟩ is never executed, and that in transition ⟨step⟩ we have $s_2 \in [s_1, e_1)$, so no hole is created, and $s_1 \leq s_2$, so 0 remains occupied.

This proves $\text{OUT}_0(C) = \varphi(C)$. To show $\text{OUT}_k(C) = \text{OUT}_0(C)$ we again use a monotonicity argument similar to the proof of Theorem 3.5. We have already shown that C cannot cover a state with $t+1$ as the upper bound without snipes, and this extends to arbitrary k-executions.

Case 2: $\sum_{q \in Q} \text{len}(q)C(q) \geq t$**.** Note that any transition leaves $\sum_{q \in Q} \text{len}(q)C(q)$ invariant. Let D denote any configuration k-reachable from C.

If $\sum_q \text{len}(q)D(q) < t$, then this observation would imply that it is possible to snipe k agents from C (without executing any moves) and end up in a configuration with output 0. But this contradicts $k = \mathit{InTol}(C)$.

Again similar to the proof of Theorem 3.5, transitions ⟨step⟩ and ⟨accum⟩ can only be executed finitely often, so we assume wlog that D is terminal. We now argue that D has output 1. If level t is occupied, i.e. in D there is some agent in a state $[s, t+1)$, then all other agents must be in such a state as well (otherwise ⟨accum⟩ could be executed) and the protocol accepts. Otherwise we use $\sum_{q \in Q} \text{len}(q)D(q) \geq t$: an agent in state q occupies $\text{len}(q)$ levels and only

levels in $[0, t-1]$ are occupied, so at least one level is occupied by two agents. However, these agents could execute $\langle \mathsf{step} \rangle$, contradicting that D is terminal. □

4.2 Threshold Predicates with Threshold 1

We design a robust protocol for predicates of the form

$$\varphi(x) = \left(\sum_{i=1}^{n} a_i \cdot x_i \geq 1 \right)$$

Since they are generalizations of the majority predicate $x - y \geq 1 \iff x - y > 0$, we call them *generalized majority* predicates. In this section, we first introduce weak population protocols. Intuitively, in a weak consensus protocol not all states must be accepting or rejecting, but can can also be neutral. Then we present a very simple and yet robust weak population protocol for generalized majority. Finally, we present a conversion procedure that transforms a weak population protocol into a population protocol for the same predicate that also preserves robustness.

Weak Population Protocols for Generalized Majority. Recall that population protocols reach a decision by *consensus*. States can be split into accepting or rejecting. A configuration is a *consensus* if either all ninjas are in accepting states or all ninjas are in rejecting states. The output function assigns a decision to each consensus configuration, and only to them. We present *weak population protocols* in which states can also be *neutral*. Intuitively, ninjas in neutral states have no opinion. A configuration is a *weak consensus* if no two ninjas have conflicting opinions. The output function assigns a decision to each consensus configuration, and only to them, as follows: accept if at least one ninja wants to accept and no ninjas wants to reject, and reject otherwise. Observe that, in particular, if no ninja wants to accept, then the decision is reject.

Definition 4.2. *An output function* $O \colon \mathbb{N}^Q \to \{1, 0\}$ *is a* weak consensus *function if there is a partition of* Q *into three sets* Q_+, $Q_=$, Q_- *of* accepting, neutral, *and* rejecting *states such that for every configuration* C*:*

- $O(C) = 1$ *if* $[\![C]\!] \subseteq Q_+ \cup Q_=$ *and* $[\![C]\!] \cap Q_+ \neq \emptyset$*;*
- $O(C) = 0$ *if* $[\![C]\!] \subseteq Q_- \cup Q_=$*; and*
- $O(C) = \bot$ *otherwise (that is,* $O(C) = \bot$ *if* $[\![C]\!] \cap Q_+ \neq \emptyset \neq [\![C]\!] \cap Q_-$*).*

A generalized protocol $\mathcal{P} = (Q, \delta, I, O)$ *is a* weak population protocol *if* O *is a weak consensus function, and every* 1*-consensus is stable.*

Notice that the output function is determined by the partition Q_+, $Q_=$, Q_-. Let us now give weak population protocols for generalized majority predicates. We consider the predicate $x_1 - 2x_2 \geq 1$, which is already representative. It corresponds to the following situation. Sensei wants that the ninjas conduct a vote to decide whether they want to attack or not. The ninjas will only attack

if more than $2/3$ of them vote for it. Letting x_1 and x_2 be the number of ninjas in favor of and against attacking, respectively, the ninjas will attack if $x_1/(x_1 + x_2) > 2/3$ or, equivalently, $x_1 - 2x_2 \geq 1$. Intuitively, a negative vote has the same effect as two positive votes.

Sensei asks the ninjas to bring a pouch containing one *positive pebble* if they want to attack, and two *negative pebbles* otherwise. If two ninjas with pebbles of the same kind interact, nothing happens. If the pebbles are of opposite kinds, one ninja gives her pebbles to the other. However, when a positive and a negative pebble touch each other, they magically disappear, and so after an interaction between two ninjas with a positive and b negative pebbles one ninja has $|a - b|$ pebbles and the other 0. We visualize this protocol in Fig. 3.

The possible states of the protocol are $\{1, 0, -1, -2\}$, which gives the number of pebbles in the pouch and their kind. The partition of states is the natural one: $Q_+ = \{1\}$, $Q_= = \{0\}$, and $Q_- = \{-1, -2\}$. The only non-silent transitions are $1, -2 \mapsto 0, -1$, and $1, -1 \mapsto 0, 0$. Observe that, in particular, $1, 1 \mapsto 2, 0$ or $-2, -2 \mapsto -4$ are *not* transitions, because the pebbles of the interacting ninjas are of the same kind.

Let us informally argue that the protocol is correct and robust. Consider a configuration C in which x_1 ninjas want to attack, and x_2 ninjas do not. Observe first that the total number of pebbles never increases, and it decreases whenever two ninjas carrying pebbles of opposite kinds interact. Therefore, starting from C the protocol eventually reaches a terminal configuration C' in which all remaining pebbles (if any) are positive or all are negative. In the first case only states of $Q_+ \cup Q_=$ are populated, and in the second only states of $Q_- \cup Q_=$. So C' is a weak consensus. Further, it is easy to see that the cases occur when $x_1 - 2x_2 \geq 1$ and $x_1 - 2x_2 < 1$, respectively. To prove robustness, observe that the argument above holds for any configuration C, and so for the sniper all configurations are equally good.

Let us now give the precise definition of the protocol for the predicate $\sum_{i=1}^{n} a_i \cdot x_i \geq 1$. Let a_{min}, a_{max} be the minimum and maximum of $\{a_1, \ldots, a_n\}$. We assume $a_{min} < 0 < a_{max}$, the case of predicates in which all coefficients have the same sign was already considered in the previous section. The protocol is:

Protocol GenMajority**.**

States.	$Q = [a_{min}, a_{max}]$
Input.	$I = \{a_1, \ldots, a_n\}$
Output.	Given by $Q_+ = [1, a_{max}]$, $Q_= = \{0\}$, and $Q_- = [a_{min}, -1]$.
Transitions.	$\delta = \{x, y \mapsto x + y, 0 : x, y \in Q \wedge x < 0 < y\}$

From Weak Population to Population Protocols. In order to use weak population protocols in practice, we provide an algorithm that given a weak population protocol deciding a predicate, under the assumption that the protocol is *terminating*, i.e. every fair execution terminates, outputs a population protocol for the same predicate. Further, the algorithm preserves robustness.

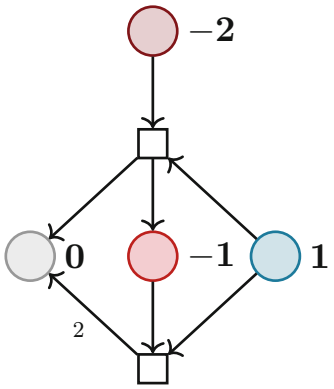


Fig. 3. Weak population protocol for $x_1 - 2x_2 \geq 1$. Red states are rejecting, the blue state is accepting and the gray state is neutral. (Color figure online)

The construction to obtain a population protocol $\mathcal{P}'$ from a weak population protocol $\mathcal{P}$ is rather simple: Every ninja remembers a state of $\mathcal{P}$, and in fact runs the protocol $\mathcal{P}$ as normal. In addition, neutral ninjas A, i.e. in a state $q \in Q_=$, get an output bit $\{+, -\}$, which they set as follows: Whenever they meet a ninja B with an active opinion, i.e. with state $p \in Q_+ \cup Q_-$, A updates her bit to the opinion of B. Since by definition of weak consensus eventually only one opinion remains, every neutral ninja eventually gets the opinion of the remaining ninja. A slight difficulty, where we need the assumption of termination of $\mathcal{P}$, is the case of only neutral ninjas remaining: We add a transition which takes neutral ninjas with output bits set to $+$ and $-$ respectively and sets both bits to $-$.

Construction 4.3. Given a weak population protocol $\mathcal{P} = (Q, \delta, I, O)$ with partition $Q_+, Q_=, Q_-$, we construct a population protocol $\mathcal{P}' = (Q', \delta', I', O')$:

Protocol WeakConvert**.**

States. $Q' = Q_+ \cup Q_- \cup Q_= \times \{+, -\}$

Input. $I' = \{q : q \in I \cap (Q_+ \cup Q_-)\} \cup \{(q, -) : q \in I \cap Q_=\}$

Output. $$O'(q') = \begin{cases} 1 & \text{if } q' \in Q_+ \text{ or } q' = (q, +) \text{ with } q \in Q_= \\ 0 & \text{if } q' \in Q_- \text{ or } q' = (q, -) \text{ with } q \in Q_= \end{cases}$$

Transitions.

– First we need a transition to simulate $\mathcal{P}$, given agents in states p, q in $\mathcal{P}'$, they 1) disregard their sign if they are in $Q_=$, 2) perform δ on the so obtained projected states p_{pr}, q_{pr}, and 3) readd a *negative sign* if they end in $Q_=$. Formally, let $\mathrm{pr}\colon Q' \to Q, q' \mapsto q'$ if $q' \in Q_+ \cup Q_-$ and $q' \mapsto q$ if $q' = (q, s) \in Q_= \times \{+, -\}$. Also let $\mathrm{inj}\colon Q \to Q', q \mapsto q$ if $q \in Q_+ \cup Q_-$ and $q \mapsto (q, -)$ otherwise. For every non-silent transition $(p_{pr}, q_{pr} \mapsto p'_{pr}, q'_{pr}) \in \delta_{\mathcal{G}}$, we add for all states p, q with $\mathrm{pr}(p) = p_{pr}, \mathrm{pr}(q) = q_{pr}$, the following transition:

$$p, q \mapsto \mathrm{inj}(p'_{pr}), \mathrm{inj}(q'_{pr}) \qquad \langle \mathsf{derived} \rangle$$

– Additionally, for $q_+ \in Q_+$ and $q_= \in Q_=$ we add the following transition:

$$q_+, (q_=, -) \mapsto q_+, (q_=, +) \qquad \langle \text{witnessPos} \rangle$$

– Dually, for $q_- \in Q_-$ and $q_= \in Q_=$ we add the following transition:

$$q_-, (q_=, +) \mapsto q_-, (q_=, -) \qquad \langle \text{witnessNeg} \rangle$$

– And for every $q, q' \in Q_=$ we add:

$$(q, -), (q', +) \mapsto (q, -), (q', -) \qquad \langle \text{convince} \rangle$$

Given an execution π on $\mathcal{P}'$, we define the induced execution $\tilde{\pi}$ on $\mathcal{P}$ via $\tilde{\pi}(i) = \mathrm{pr}(\pi(i))$ where $\mathrm{pr}((q, s)) = q$. Moving from π to $\tilde{\pi}$ intuitively replaces $\langle$derived$\rangle$ by their respective transition and $\langle$witnessPos$\rangle$/$\langle$witnessNeg$\rangle$/ $\langle$convince$\rangle$ by silent identity transitions. If π is fair, $\tilde{\pi}$ is also fair.

In the following, a positive voter is a ninja with a state in Q_+, a negative voter a ninja with a state in Q_- and an active voter is either a positive or negative voter. Essentially there is only one type of configuration of $\mathcal{P}'$ where assigning the output as above will fail: If there are no active voters, and neither is a ninja in $Q_= \times \{-\}$ to convince the others. Hence we call a configuration C *dangerous* if $[\![C]\!] \subseteq Q_= \times \{+\}$, and *harmless* otherwise.

Lemma 4.4. *Let C'_0 be a harmless configuration of $\mathcal{P}'$ such that all fair executions starting at $\mathrm{pr}(C'_0) \in \mathbb{N}^Q$ terminate with output b in $\mathcal{P}$.*

Then all fair executions starting at C'_0 terminate with output b in $\mathcal{P}'$.

Proof. Let $\pi = (\pi_0, \pi_1, \dots)$ be a fair execution in $\mathcal{P}'$ starting at C'_0. We have to show that π has output b. First consider the induced execution $\tilde{\pi}$ in $\mathcal{P}$. Since $\mathcal{P}$ is terminating and $\tilde{\pi}$ is fair, $\tilde{\pi}$ terminates in some configuration C'_{pr}. Clearly using $\langle$witnessPos$\rangle$/$\langle$witnessNeg$\rangle$ and potentially $\langle$convince$\rangle$, π then also terminates in some configuration C', and we know that $\mathrm{pr}(C') = C'_{pr}$. Because $\tilde{\pi}$ is also fair, the output of C'_{pr} is b. If an active voter exists in C'_{pr} and hence in C', it will eventually convert everyone to the correct output via the $\langle$witnessPos$\rangle$/$\langle$witnessNeg$\rangle$ transitions, and hence π again has output b.

If C' does not contain an active voter, then C'_{pr} has output 0 by definition, and we have to show the same for the execution π. We first show that the configuration C' contains an agent in $Q_= \times \{-\}$. We prove this via case distinction:

Case 1: No $\langle$derived$\rangle$ transition ever occurred**.** Then we have $\pi_i(Q_+) = 0$ for all i, and since C'_0 is harmless, initially an agent was in $Q_= \times \{-\}$. The only transition which could remove the agent from $Q_= \times \{-\}$, namely $\langle$witnessPos$\rangle$, was never enabled.

Case 2: $\langle$derived$\rangle$ did occur**.** Then consider the configuration π_i directly after the last occurrence. It contains an agent in $Q_= \times \{-\}$ by definition of $\langle$derived$\rangle$. We can now argue as in case 1.

It follows that $\langle$convince$\rangle$ moves all agents from $Q_= \times \{+\}$ to $Q_= \times \{-\}$, and π terminates with output 0. □

Theorem 4.5. *Given a terminating weak population protocol $\mathcal{P}$ deciding a predicate φ, the above construction outputs a population protocol $\mathcal{P}'$ deciding φ. Additionally, if $\mathcal{P}$ is robust, then $\mathcal{P}'$ also is.*

Proof. Applying Lemma 4.4 to all initial configurations, we conclude that $\mathcal{P}'$ decides the same predicate φ as $\mathcal{P}$. Observe that initial configurations are harmless since agents start in $Q_= \times \{-\}$ by definition.

It remains to prove that $\mathcal{P}'$ is robust. Hence let $C'_0 \in \mathbb{N}^{I'}$ be an initial configuration. We have to show that $Tol(C'_0) = InTol(C'_0)$. Hence let π be a fair i-execution starting at C'_0 with $i = InTol(C'_0)$. We have to show that π has output $\varphi(C'_0)$.

Observe that since $\mathcal{P}$ is terminating, $\mathcal{P}'$ is also terminating. Hence π' eventually becomes constant, we call the reached terminal configuration C''. Even if an agent was sniped earlier in π, we can wlog assume it was sniped directly before C'', and simply did not participate in transitions before that. I.e. there exists $C' \in \mathbb{N}^{Q'}$ such that $C'_0 \to^* C'$ and $C' \hookrightarrow^i C''$. To show that π has output $\varphi(C'_0)$, it suffices to show that C'' has output $\varphi(C'_0)$. We want to apply Lemma 4.4 on the configuration C'', hence we have to check its assumptions.

First, since $\mathcal{P}$ and $\mathcal{P}'$ compute the same predicate, we have $InTol_{\mathcal{P}'}(C'_0) = InTol_{\mathcal{P}}(\mathrm{pr}(C'_0))$. Since $\mathcal{P}$ is robust, and the projected execution $\tilde{\pi}$ contains at most $i = InTol_{\mathcal{P}}(\mathrm{pr}(C'_0))$ snipes, $\tilde{\pi}$ has output $\varphi(C'_0)$. Therefore $\mathrm{pr}(C'')$ has output $\varphi(C'_0)$. Since $\mathrm{pr}(C'')$ is terminal, this shows the assumption about all fair executions of the projection having correct output. It remains to prove that C'' is harmless.

Assume for contradiction that C'' is dangerous. In particular $\mathrm{pr}(C'')$ is rejected. Similar to the case reasoning in Lemma 4.4, observe that the only transition outputting agents in $Q_= \times \{+\}$ is $\langle \mathsf{witnessPos} \rangle$. Hence to reach a dangerous configuration, in some configuration between C_0 and C' some agent must have been in Q_+ and got sniped. We denote the latest (i.e. closest to C') configuration that has a positive voter by C_+.

Note that no agent in C'' participated in a transition since C_+, since after C_+ no positive voters exist and hence all enabled transitions set the output bit to $-$. We now consider the configuration $C''_+ = C'' + \wr q_+ \int$ where q_+ is a positive voter in C_+. We can reach C''_+ from C_+ via sniping exactly those agents that are sniped from C' to C'' except for the positive voter. Since C'' does not contain negative voters, C''_+ does not, and hence C''_+ is an accepting consensus. By assumption on weak population protocols, any accepting consensus is stable, hence C''_+ is accepted. Hence $\mathrm{pr}(C'_0)$ has an accepting execution via C''_+, and a rejecting execution via C'', despite both executions sniping $\leq InTol(\mathrm{pr}(C'_0))$ many agents. Contradiction to robustness of $\mathcal{P}$. □

Example 4.6. We now apply this construction to the protocol that we gave above for $x_1 - 2x_2 \geq 1$ and visualized in Fig. 3. We write $+0$ and -0 instead of $(0, +)$ and $(0, -)$ for clarity.

Protocol SignedNumbers**.**

States. $Q = \{1, -1, -2, +0, -0\}$

Input. $I = \{-2, 1\}$

Output. $O(q) = \begin{cases} 1 & \text{if } q = 1 \text{ or } q = +0 \\ 0 & \text{if } q = -1 \text{ or } q = -2 \text{ or } q = -0 \end{cases}$

Transitions.

$$\begin{aligned}
-2,\ \ 1 &\mapsto -1, +0 && \text{coming from } \langle\text{derived}\rangle \\
-1,\ \ 1 &\mapsto -0, -0 && \text{coming from } \langle\text{derived}\rangle \\
-0,\ \ 1 &\mapsto +0,\ \ 1 && \text{coming from } \langle\text{witnessPos}\rangle \\
+0, -2 &\mapsto -0, -2 && \text{coming from } \langle\text{witnessNeg}\rangle \\
+0, -1 &\mapsto -0, -1 && \text{coming from } \langle\text{witnessNeg}\rangle \\
+0, -0 &\mapsto -0, -0 && \text{coming from } \langle\text{convince}\rangle
\end{aligned}$$

4.3 Arbitrary Threshold Predicates

In this subsection we again consider the threshold predicate from the beginning of this section:

$$\varphi(x) = \left(\sum_{i=1}^{n} a_i \cdot x_i \geq t \right)$$

Now we also allow for $a_i, t \in \mathbb{Z}$. We again will create a tower-esque protocol, however this time it will be a weak population protocol. Ninjas representing inputs with negative coefficients will have to cancel positive ninjas. We assume $t \geq 1$ wlog, since we can rewrite $\sum_{i=1}^{n} a_i \cdot x_i \geq t$ as $\sum_{i=1}^{n} (-a_i) \cdot x_i \leq -t$, which is the negation of $\sum_{i=1}^{n} (-a_i) \cdot x_i \geq -t + 1$. Negation preserves robustness, see Remark 3.2.

Protocol InhomTowerCancel**.**

States are intervals of integers as well as non-positive integers. We identify 0 with all empty intervals. Initial states are intervals starting at zero as well as negative integers.

States. $Q = Q_{\text{TOWER}} \cup Q_{\text{CANCEL}} \cup \{0\}$ where
$Q_{\text{TOWER}} = \{[s, e) : 0 \leq s < e \leq T\}$
$Q_{\text{CANCEL}} = \{x : \min_i a_i \leq x < 0\}$
$T = \max\{t, a_1, \ldots, a_n\}$

Input. $I = \{[0, a_i) : a_i > 0\} \cup \{a_i : a_i < 0\}$

Output. Given by the following partition:
$Q_+ = \{[s, e) \in Q_{\text{TOWER}} : t \leq e\}$,
$Q_= = \{[s, e) \in Q_{\text{TOWER}} : e < t\} \cup \{0\}$,
$Q_- = Q_{\text{CANCEL}}$

Transitions.

– For every two non-disjoint intervals $[s_1, e_1), [s_2, e_2) \in Q_{\text{TOWER}}$ with $e_1, e_2 < T$ and $s_1 \leq s_2$ the transition

$$[s_1, e_1), [s_2, e_2) \mapsto [s_1, e_1), [s_2 + 1, e_2 + 1) \qquad \langle \mathsf{step} \rangle$$

– When a negative agent meets an agent above level t, they both reduce their absolute value by one. For all $0 \leq s < e \leq T$ with $t \leq e$ and $x \in Q_{\text{CANCEL}}$ the transition

$$[s, e), x \mapsto [s, e - 1), x + 1 \qquad \langle \mathsf{cancel} \rangle$$

note that we identify empty intervals i.e. $[s, s)$ with the state 0.

Theorem 4.7. InhomTowerCancel *computes* φ *and is robust.*

Proof. This proof is similar to that of Theorem 4.1. Given an agent in state $[s, e) \in Q_{\text{TOWER}}$, we say that it *occupies* levels $s, ..., e - 1$.

Let $C_0 \in \mathbb{N}^I$ be an initial configuration and let $k = InTol(C_0)$. Additionally, let $Sum(C) := \sum_{q \in Q_{\text{TOWER}}} \text{len}(q)C(q) + \sum_{q \in Q_{\text{CANCEL}}} qC(q)$ for any configuration C. Observe that any transition leaves Sum invariant. Since $Sum(C) \leq t$ is φ, initial tolerance ensures that sniping does not change $Sum(C) \leq t$.

Additionally, we note that the protocol produces an output for every fair n-execution for arbitrary n. Since eventually no agent reaches a level above or equal to t, or no negative states contain agents. We now show $\text{OUT}_0(C_0) = \varphi(C_0)$ and $\text{OUT}_k(C_0) = \text{OUT}_0(C_0)$.

Case 1: $Sum(C_0) < t$**.** This implies $\varphi(C_0) = 0$. For contradiction, we consider a fair i-execution starting at C_0 that is accepting with minimal $i \leq k$. Since the protocol is bounded, we can equivalently consider configurations $C_0 \to^* C \hookrightarrow^* D \to^* C_{term}$ where C_{term} is terminal.

We now show that we can assume that the snipes from C to D do not snipe agents that are in Q_{TOWER}. Let Δ denote the agents in Q_{TOWER} that were sniped in $C \hookrightarrow^* D$. Now consider $C' = D + \Delta$: we have $C \hookrightarrow^* C' \to^* C_{term} + \Delta$. The last configuration is again accepting, since it does not contain any agents in Q_{CANCEL}. While it is not terminal, it cannot become rejecting, since agents in Q_{CANCEL} can never again exist. If $|\Delta| > 0$, then we have a smaller counterexample.

In the same manner as in the proof of Theorem 4.1, we show that the set of occupied levels in a configuration D k-reachable from C without sniping in Q_{TOWER} is contiguous and contains 0. We show by induction that $\bigcup_{D(q)>0} q = [0, i]$ for some i. The base case follows from the definition of I, in the step case we must consider the two transitions: $\langle \mathsf{step} \rangle$ does not introduce holes, since $s_2 \in [s_1, e_1)$ and $s_1 \leq s_2$, so 0 remains occupied. $\langle \mathsf{cancel} \rangle$ either reduces the upper bound of an interval at or above t by one, this also does not create a hole and leaves 0 occupied.

Now we consider the output of C_{term}. If there are agents in Q_{CANCEL}, then the levels $[t, T)$ cannot be occupied, since otherwise $\langle \mathsf{cancel} \rangle$ could be executed. If no agents are in Q_{CANCEL}, then we also know that $[t, T)$ cannot be occupied, since $Sum(C_{term}) = \sum_{q \in Q_{\text{TOWER}}} \text{len}(q)C_{term}(q) < t$ and the occupied levels are contiguous and include 0.

Case 2: $Sum(C_0) \geq t$**.** This implies $\varphi(C_0) = 1$. Let D be a configuration k-reachable from C_0.

If $Sum(D) < t$, then by the observation above it would be possible to snipe k agents in C_0 and obtain an initial configuration with $Sum < t$. By the previous case, it would reject, thus contradicting that $k = InTol(C_0)$.

We again, similar to the proofs Theorem 3.5 and Theorem 4.1 use the fact that $\langle \mathsf{step} \rangle$ and $\langle \mathsf{cancel} \rangle$ can only be executed finitely often. So wlog D is terminal. We show that D has output 1: if level t or any above is occupied, then since D is terminal, there cannot exist any agent in D that is in a state from Q_{CANCEL}. Therefore D has output 1. In the case that no agent occupies level t or above, we know from $Sum(D) \geq t$ that $\sum_{q \in Q_{\text{TOWER}}} \text{len}(q) D(q) \geq t$ and thus at least one level is occupied by two agents. This means that $\langle \mathsf{step} \rangle$ could be executed contradicting that D is terminal. □

Corollary 4.8. *Giving* InhomTowerCancel *as input to Construction 4.3 we obtain a robust population protocol computing* φ*.*

Proof. Clearly InhomTowerCancel is terminating, and Theorem 4.7 shows it is robust. Every accepting consensus is stable: Namely no agent is in Q_{CANCEL} since these are negative voters, wherefore $\langle \mathsf{cancel} \rangle$ is permanently disabled. On the other hand $\langle \mathsf{step} \rangle$ can only cause additional agents to become positive voters. Hence we can apply Theorem 4.5. □

5 Robust Modulo Protocols

Throughout this section, fix a modulo predicate not equivalent to true or false:

$$\varphi(x) = \left(\sum_{i=1}^{n} a_i \cdot x_i \bmod m \geq t \right)$$

With $a_i, t \in \mathbb{Z}$ and $m \in \mathbb{N}$. Without loss of generality, we can assume that $0 < a_i, t < m$, because the sum is calculated modulo m.

When the Black Ninjas attack fortresses of the Evil Powers on nights with a full moon, they only do so if the number of ninjas modulo 7 is equal to $4, 5$ or 6. Therefore, Sensei is searching for a robust population protocol for $x \bmod 7 \geq 4$. Sensei finds an important difference between threshold and modulo predicates: Threshold predicates allow for arbitrarily large initial tolerance, for modulo predicates however there exists an upper bound on the initial tolerance.

Proposition 5.1. *For any population protocol computing* φ*, we have for any initial configuration* $C \in \mathbb{N}^I$ *that* $InTol(C) < m$*.*

Proof. Using at most $m - 1$ snipes, the value of the sum can be changed by at least $m - 1$, thus allowing to change the output. □

Using this fact, we will first construct a protocol for sufficiently large inputs in Sect. 5.1. Then we will combine that protocol in Sect. 5.2 with an InhomTowerand a protocol for small inputs to obtain a robust population protocol for all inputs.

5.1 A Modulo Protocol for Big Inputs

After proving Proposition 5.1, Sensei comes up with a robust population protocol that can handle sufficiently large initial configurations. The sniper snipes at most $m-1$ ninjas, allowing Sensei to take advantage of a principle common with error-correcting codes: to achieve a fault tolerance of r, $2r+1$ copies of a fault intolerant code are used in parallel. Then one can decide the correct value via majority. Sensei applies this principle to protocols: she thinks that it will be sufficient to have $2(m-1)+1 = 2m-1$ copies of any protocol for a modulo predicate running in parallel. For combining with small inputs, which will become clear later, we need an additional copy, so we will use $2m$ copies in total.

To compute $x \bmod 7 \geq 4$, Sensei gives the following instructions to the ninjas: Each ninja must bring 14 differently colored pouches, each containing a magic pebble. Whenever 7 magic pebbles are in the same pouch, they annihilate each other. Thus the pebbles are added modulo 7. In the beginning, each ninja must choose a color. After doing so, they become a *leader* for that color. When a leader and any other ninja that is not a leader for the same color interact, the leader steals all pebbles from the other's pouch with the chosen color. When two leaders for the same color interact, one of them retires and gives all the pebbles of that color to the other. The retired leader then again chooses a color where she still has some pebbles left and can again become a different leader. If she has no pebbles left, she remains a non-leader. Additionally, every ninja remembers for every color the number of pebbles in the pouch of the last seen leader of that color. The ninjas decide to attack via majority: if a majority of leaders agree that there at least 4 pebbles in their pouch, then they attack.

Sensei argues that the protocol works for sufficiently large configurations (where $x \geq 14$): every ninja can at most be a leader for a given color once. At least one leader for every color will remain and since there are more than 14 ninjas, every color will have a leader. Every leader correctly computes the value $x \bmod 7$ via the magic pebbles and without a sniper, they all will agree. In the case of a sniper, for every snipe at most one color can be disturbed. Since a sniper can at most snipe 6 times, the majority of 14 leaders will remain correct.

We now give the formal definition of the BigModulo population protocol, where we write $\mathbb{1}_A$ for the indicator function of the set A:

Protocol BigModulo**.**

States.	$Q = [0, 2m] \times \mathbb{Z}_m^{2m} \times \{0,1\}^{2m}$
Input.	$I = \{(0, a_i \cdot \mathbb{1}_{[1,2m]}, \mathbb{0}) : 1 \leq i \leq n\}$
Output.	$O(\_, \_, r) = 1 \iff \lvert\{j : r_j = 1\}\rvert > m$
Transitions.	

– We need to distribute the agents in the different copies. For this, we use non-determinism, for all $(0, v, r), q \in Q$ and $1 \leq i \leq 2m$ with $1 \leq v(i)$ we add the following transition:

$$(0, v, r), q \mapsto (i, v, r), q \qquad \langle \mathsf{distrib} \rangle$$

– When two agents meet that are in different copies, both steal the tokens from their respective copy from the other. For all $(i, v, r), (j, w, s) \in Q$ with $1 \leq i, j \leq 2m$ and $i \neq j$ we add the following transition:

$$(i, v, r), (j, w, s) \mapsto (i, v', r), (j, w', s) \qquad \langle \mathsf{steal} \rangle$$

Where

$$v' = v - v_j \cdot \mathbb{1}_{\{j\}} + w_i \cdot \mathbb{1}_{\{i\}} \qquad w' = w - w_i \cdot \mathbb{1}_{\{i\}} + v_j \cdot \mathbb{1}_{\{j\}}$$

– When two agents from the same copy meet, one of them takes all tokens from the other and the other retires. For all $(i, v, r), (i, w, s) \in Q$ with $1 \leq i \leq 2m$ we add the following transition:

$$(i, v, r), (i, w, s) \mapsto (i, v + w_i \cdot \mathbb{1}_{\{i\}}, r), (0, w - w_i \cdot \mathbb{1}_{\{i\}}, s) \qquad \langle \mathsf{retire} \rangle$$

– Lastly we need a transition that updates the last component to keep track of the overall output of the protocol. For all $(i, v, r), (j, w, s) \in Q$ with $1 \leq i \leq 2m$ and $0 \leq j \leq 2m$ we add the following transition:

$$(i, v, r), (j, w, s) \mapsto (i, v, r'), (j, w, s') \qquad \langle \mathsf{result} \rangle$$

Where

$$r'(l) = \begin{cases} v_i \geq t & \text{if } l = i \\ r(l) & \text{else} \end{cases} \qquad s'(l) = \begin{cases} v_i \geq t & \text{if } l = i \\ s(l) & \text{else} \end{cases}$$

We call agents with a non-zero first component *leaders*, and call the j-th copy of $\mathbb{Z}^m$ the j-th subprotocol.

Proposition 5.2. *For every initial configuration $C_0 \in \mathbb{N}^I$ with $|C_0| \geq 2m$,* BigModulo *decides φ.*

Proof. Let C_0 be an initial configuration. We first claim that every subprotocol will eventually have exactly one associated leader.

Proof of claim: When two leaders meet via ⟨retire⟩, only one of them remains a leader and the other gives away their tokens in that subprotocol. Since ⟨distrib⟩ only makes agents leaders that have tokens in the respective subprotocol, that agent will never again be a leader for that subprotocol. Since we have at least $2m$ agents, every subprotocol will eventually end up with a leader.

Now it is easy to see that each leader will eventually hold all the tokens in their respective subprotocol. Namely they accumulate them via ⟨steal⟩ .

Since the j-th sum $Sum_j(C) := \sum_{q=(i,v,r)\in Q} v(j) \cdot C(q)$ is invariant modulo m, the leader for every subprotocol has exactly $Sum_j(C_0)$ mod m many tokens, hence obtaining the correct output. This output is then broadcasted via the ⟨result⟩ transition to every non-leader. □

Proposition 5.3. *For every initial configuration $C_0 \in \mathbb{N}^I$ with $|C_0| \geq 3\,m$,* BigModulo *is robust.*

Proof. Let C_0 be such an initial configuration. First observe that even after sniping we then have at least $2\,m$ agents left. Intuitively the argument will be along the following lines: Case 1: If an agent has not interacted in a copy j yet, then sniping it falls into the initial tolerance. Case 2: If an agent has value 0 in a copy, it can also be sniped without issue. Case 3: Neither case 1 nor 2 is the case. We have to show that for every agent, case 3 happens for at most one copy. Then the number of copies damaged is at most the number of sniped agents, i.e. at most $m - 1$, and the majority of copies remains correct.

Hence let $\rho = (C_0, C_1, \dots)$ be a fair k-execution with $k = \mathit{InTol}(C_0)$. For every agent a we define

$$J(a,n) := \{j \in [1, 2m] : \\ a \text{ has interacted in the } j\text{-th copy of } \mathbb{Z}^m \text{ before step } n \\ \wedge\, v_j(a) \neq 0\}$$

$$L(a,n) := \begin{cases} \{j\} & \text{if } a \text{ is a leader in the } j\text{-th copy of } \mathbb{Z}^m \text{ at step } n \\ \emptyset & \text{else} \end{cases}$$

We claim that if C_n fulfills that for all agents $J(a,n) \subseteq L(a,n)$, then also C_{n+1} does. Comparing with the outline, this states exactly that case 3 occurs only for the copy j in which agent a is the leader. To see that this property is actually inductive, we inspect every transition: ⟨steal⟩ and ⟨retire⟩ set the value of agent a in a copy j to 0 on the non-leader, ⟨result⟩ does not influence these values. ⟨distrib⟩ can only be taken if $v_j(a) \neq 0$ and we thus have $J(a, n+1) = \emptyset$. Importantly, the condition is in fact even preserved by snipes, since properties of the form "For all agents ..." become easier by removing agents. Additionally note that the condition holds in any initial configuration.

Next we prove that cases 1 and 2 are non-problematic. We first claim that ρ ends in a terminal configuration.

Proof of claim: wlog we reorder ρ s.t. ρ is of the form $C_0 \to^* C \hookrightarrow^* D \to^* C_{term}$, i.e. such that all snipes happen at once and it reaches a terminal configuration. To ensure this, we commute the snipes past other transitions towards the last snipe. This is possible since a non-interacting agent might as well not exist. Transitions ⟨steal⟩ and ⟨retire⟩ reduce the total number of non-zero entries in the copies of $\mathbb{Z}_m$ and this number cannot be increased. Hence they become permanently disabled eventually. Then all non-leaders have all entries equal to 0, and ⟨distrib⟩ is hence also disabled. Now ⟨result⟩ provides every agent with some output for every subprotocol, and afterwards becomes disabled.

Hence ρ is of the form $C_0 \to^* C \hookrightarrow^* D \to^* C_{term}$. Let j be one of the copies such that no leader for this copy was sniped (i.e. such that no case 3 occurred for this copy). We prove that the projection pr_j of ρ to the j-th subprotocol has the correct output. For all agents which were sniped before ever interacting, we assume they were sniped at the start, and hence consider the initial configuration C_0' with those agents removed. C_0' still has at least $2m$ agents. Consider the execution ρ' starting at C_0' and otherwise mirroring $\mathrm{pr}_j(\rho)$, which is possible

since the removed agents never interacted in the j-th subprotocol. We prove that the j-th subprotocol has the correct output in ρ', and hence also in ρ.

By choice of j, the only agents which are sniped in ρ' fulfilled $v(j) = 0$, and were non-leaders for this copy. In particular after the first ⟨distrib⟩, every configuration in ρ' always had at least one leader, in particular the terminal configuration $\mathrm{pr}(C_{term})$. Since the configuration is terminal, we have exactly one leader. The j-th sum $Sum_j(C) := \sum_{q=(i,v,r)\in Q} v(j)\cdot C(q)$ is invariant modulo m, and is not impacted by sniping of agents with $v(j) = 0$. Hence we in particular have $Sum_j(\mathrm{pr}(C_{term})) \equiv Sum_j(C'_0) \mod m$. In $\mathrm{pr}(C_{term})$ this value is fully stored by the leader of the j-th subprotocol, since otherwise ⟨steal⟩ would be enabled. Hence the leader has the correct output, and therefore using transition ⟨result⟩ every agent eventually has the correct output in the j-th subprotocol.

We have therefore seen that only copies j where a leader gets sniped are damaged. Since every agent is leader in at most one subprotocol, and at most $InTol(C_0)$ many agents are sniped, it follows that at most $k \leq m-1$ subprotocols have the wrong output. The majority $m+1$ of subprotocols hence remain correct. Therefore the whole protocol has the correct output. □

5.2 Handling Small Inputs

Sensei is not fully satisfied with her modulo protocol. At the moment the ninjas have a lot of members (thanks to the many new rookies), but that can change at any moment. Therefore it is important for the modulo protocol to also handle small inputs. She thinks of the following idea: a tower computes the predicate $\psi' := \sum_{i=1}^{n} x_i \geq 2m$ to detect if there are enough ninjas to successfully use the BigModulo protocol. If there are not enough ninjas, she would want to determine the exact modulo value by the highest ninja in the tower, which is not correct with this choice of ψ', since the coefficients were ignored. Hence instead she uses $\psi := \sum_{i=1}^{n} a_i x_i \geq 3\,m^2$, which is a sufficient condition that there are at least $3\,m$ ninjas, though not a necessary condition.

Since the ninjas do not know how many of them will show up, in particular they do not know whether their number will be large or small. We solve this problem by suitably combining the protocols for the two cases. Intuitively, the ninjas execute both protocols *simultaneously*, that is, at every moment in time they are in a state for each protocol. Further, each ninja maintains an estimate of their remainder modulo m. Formally, we let $\mathcal{P}_T = (Q_T, \delta_T, \_, O_T) = \textsc{InhomTower}$ for the predicate ψ and $\mathcal{P}_M = (Q_M, \delta_M, \_, O_M) = \textsc{BigModulo}$. The ModuloCombined population protocol is defined as follows:

Protocol ModuloCombined**.**

States. $Q = Q_T \times [0, 3m^2] \times Q_M$

Input. $I = \{([0, a_i), a_i, (0, a_i \cdot \mathbb{1}_{[1,2m]}, 0)) : 1 \leq i \leq n\}$

Output. $O((q_T, h, q_M)) = \begin{cases} h \bmod m \geq t & \text{if } h < 3m^2 \\ O_M(q_M) & \text{else} \end{cases}$

Transitions.

– For all $(q_1, q_2 \mapsto q_1', q_2') \in \delta_T$ and $(p_1, p_2 \mapsto p_1', p_2') \in \delta_M$ and $h_1, h_2 \in [0, 3m^2]$ we add the following transition:

$$(q_1, h_1, p_1), (q_2, h_2, p_2) \mapsto (q_1', h, p_1'), (q_2', h, p_2') \qquad \langle\mathsf{parallel}\rangle$$

Where $h = \max\{h_1, h_2, \textsc{end}(q_1'), \textsc{end}(q_2')\}$ with $\textsc{end}([s, e)) = e$.

Theorem 5.4. ModuloCombined *computes* φ *and is robust.*

Proof. Let C_0 be an initial configuration, and $\rho = (C_0, C_1, \dots)$ a fair k-execution with $k = \mathit{InTol}(C_0)$. Since every fair execution of both BigModulo and InhomTower terminates, every fair execution of ModuloCombined terminates. Let C_r be the terminal configuration reached. Since $\langle\mathsf{parallel}\rangle$ in particular updates the values of h_1 and h_2 to the maximum, the fact that C_r is terminal implies that every agent has the same value h in this component. Since we chose the threshold of ψ to be $3m^2$, which is a multiple of m, we have that $\mathit{InTol}_{\mathcal{P}_T}(C_0) \geq \mathit{InTol}_{\mathcal{P}_M}(C_0)$. By this argument the following case distinction does not depend on snipes:

Case 1: $h < 3m^2$**.** Then no agent in the tower component is at the top, and the output of ModuloCombined is determined by O_T. Let C_0' be the initial configuration if the agents sniped in ρ had been sniped at the start. The height h has to be in the interval $\left[\sum_{q\in Q} \mathrm{len}(q_T)\cdot C_0'(q), \min(\sum_{q\in Q} \mathrm{len}(q_T)\cdot C_0(q), 3m^2)\right]$. By definition of initial tolerance, in fact φ is constant even on the larger interval $\left[\sum_{q\in Q} \mathrm{len}(q_T)\cdot C_0'(q), \sum_{q\in Q} \mathrm{len}(q_T)\cdot C_0(q)\right]$. Hence no matter which of these values h actually has, using it we obtain the correct value of φ.

Case 2: $h = 3m^2$**.** Then the output is fully given by $O_M(q_M)$. Since $h = 3m^2$, and every coefficient a_i is at most m, initially we must have had at least $3m$ agents. Hence C_0 is a large enough initial configuration that we can use Proposition 5.3 which states BigModulo is robust for C_0, wherefore the projection of ρ to Q_M and therefore also ρ itself gives the correct output. □

6 Future Work: Boolean Combinations

Recall that every Presburger predicate can be represented as a boolean combination of threshold and modulo predicates. Given protocols for two predicates φ_1, φ_2, there is a simple construction that yields protocols deciding $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$. Intuitively, agents execute both protocols simultaneously. In particular, their states are pairs of states of the protocols for φ_1 and φ_2, and their output function is the conjunction of disjunction of the output functions for φ_1 and φ_2 [3]. If this construction would preserve robustness, that is, if the protocol for $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ would be robust whenever the protocols for φ_1 and φ_2 are, then we would have proved that every predicate has a robust protocol.

Unfortunately, the construction does not preserve robustness. While applying the construction for conjunction to our InhomTower protocol from Sect. 4.1

yields a robust protocol, this is not the case for the InhomTowerCancel protocol from Sect. 4.3.

Consider an initial configuration where the first protocol accepts and the second protocol rejects. Additionally, via sniping within the initial tolerance, it is possible to make the first protocol reject and the second accept. Now the sniper can wait until the first protocol accepts and then snipe such that the second protocol also accepts.

Example 6.1. We consider the two predicates $\varphi_1(x, y) = (2x - y \geq 3)$ and $\varphi_2(x, y) = (y - x \geq 1)$ and their conjunction $\varphi(x, y) = (\varphi_1(x, y) \wedge \varphi_2(x, y))$. The reachable states of our InhomTowerCancel protocol instantiated with the two predicates are:

$$Q_1 = \{[0, 2), [1, 3), -1, 0\} \qquad Q_2 = \{[0, 1), -1, 0\}$$

Where the initial states corresponding to x are $[0, 2)$ for φ_1 and -1 for φ_2. For y we have -1 and $[0, 1)$ respectively.

In order to show that the construction is not robust, we search for an initial configuration that is rejected by φ_1 and accepted by φ_2, and which can be sniped such that φ_1 accepts and φ_2 rejects. From such a configuration we can have an execution that first stabilizes in the second protocol, then a snipe follows, and then the first protocol also stabilizes, leading to an overall accepting configuration that should have been rejected. To find such a configuration, we draw the plot in Fig. 4. Colors indicate the values of both predicates (see the legend of the figure). Sniping corresponds to moving left or down in the plot, and so we search for a blue square from which we can reach a red square by moving down or left, and from which we cannot reach a black square with the same number of moves. This last condition corresponds to the tolerance condition: the sniper is only allowed to snipe when the output of the initial configuration does not change.

The plot shows that $C = \wr 3 \cdot x, 4 \cdot y \wr$ is a suitable configuration. Its initial tolerance is 6, since all configurations $D \leq C$ are rejected. We now consider the following fair 1-execution (state changes are highlighted in bold):

$$\wr 3 \cdot ([0, 2), -1), 4 \cdot (-1, [0, 1)) \wr \tag{1}$$

$$\to^* \wr 3 \cdot ([0, 2), \mathbf{0}), 3 \cdot (-1, \mathbf{0}), \mathbf{1} \cdot (-1, [0, 1)) \wr \tag{2}$$

$$\hookrightarrow \wr 3 \cdot ([0, 2), 0), \mathbf{2} \cdot (-1, 0), (-1, [0, 1)) \wr \tag{3}$$

$$\to^* \wr \mathbf{1} \cdot ([0, 2), 0), 2 \cdot ([\mathbf{1}, \mathbf{3}), 0), 2 \cdot (-1, 0), (-1, [0, 1)) \wr \tag{4}$$

$$\to^* \wr ([0, 2), 0), 2 \cdot ([1, \mathbf{2}), 0), 2 \cdot (\mathbf{0}, 0), (-1, [0, 1)) \wr \tag{5}$$

$$\to^* \wr ([0, 2), 0), 2 \cdot ([\mathbf{2}, \mathbf{3}), 0), 2 \cdot (0, 0), (-1, [0, 1)) \wr \tag{6}$$

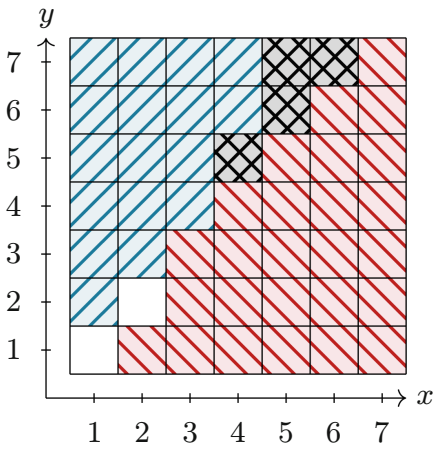


Fig. 4. Plot of the predicates $\varphi_1(x, y) = (2x - y \geq 3)$ and $\varphi_2(x, y) = (y - x \geq 1)$ on initial configurations. The color indicates the value of the predicates: it is red if φ_1 accepts, blue if φ_2 accepts, black if both accept and white if both reject. (Color figure online)

$$\rightarrow^* \wr([0, 2), 0), \mathbf{1} \cdot ([2, 3), 0), \mathbf{3} \cdot (0, 0), (\mathbf{0}, [0, 1))\wr \tag{7}$$

From (1) to (2) only the second protocol executes a non-silent transition: three of the $(-1, [0, 1))$ agents annihilate with $([0, 2), -1)$ in the second protocol. Then in (3), we proceed to snipe a single $(-1, 0)$ agent. This does not influence the second protocol, since it already is finished. In (4), the $([0, 2), 0)$ agents move upwards in the first protocol and after that in (5), we annihilate $([2, 3), 0)$ with $(-1, 0)$ twice. In (6) we again move the agents in the first protocol upwards and in (7) we do a final annihilation.

The last configuration is accepting and terminal, thus we have shown that the product construction applied to our protocol is not robust.

This example only shows that the simple construction for conjunction does not preserve robustness. But there could be a more sophisticated construction, or a procedure that directly constructs a robust protocol for any predicate, for example from a finite automaton for the predicate. Currently, the simplest predicate for which we have not been able to find a robust protocol is

$$\varphi(x, y) = (x \geq y \vee (x + 1 \geq y \wedge x \bmod 5 = 0))$$

The problematic initial configurations are of the form $\wr 5k \cdot x, n \cdot y\wr$ with $5k > n$. The initial tolerance of these configurations is $5k - n$, but the robustness of the majority protocol only gives correct results for up to $5k - n - 1$ snipes.

References

1. Alistarh, D., Dudek, B., Kosowski, A., Soloveichik, D., Uznanski, P.: Robust detection in leak-prone population protocols. CoRR abs/1706.09937 (2017). http://arxiv.org/abs/1706.09937
2. Alistarh, D., Töpfer, M., Uznanski, P.: Comparison dynamics in population protocols. In: Miller, A., Censor-Hillel, K., Korhonen, J.H. (eds.) PODC 2021: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, 26–30 July 2021, pp. 55–65. ACM (2021). https://doi.org/10.1145/3465084.3467915
3. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distrib. Comput. **18**(4), 235–253 (2006). https://doi.org/10.1007/s00446-005-0138-3
4. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. **20**(4), 279–304 (2007)
5. Blondin, M., Esparza, J., Jaax, S., Kucera, A.: Black ninjas in the dark: formal analysis of population protocols. In: LICS, pp. 1–10. ACM (2018)
6. Czerner, P., Guttenberg, R., Helfrich, M., Esparza, J.: Fast and succinct population protocols for presburger arithmetic. J. Comput. Syst. Sci. **140**, 103481 (2024)
7. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Ruppert, E.: When birds die: making population protocols fault-tolerant. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 51–66. Springer, Heidelberg (2006). https://doi.org/10.1007/11776178_4
8. Guerraoui, R., Ruppert, E.: Names trump malice: tiny mobile agents can tolerate byzantine failures. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 484–495. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02930-1_40

E. Brief Announcement: The Expressive Power of Uniform Population Protocols with Logarithmic Space

Title	Brief Announcement: The Expressive Power of Uniform Population Protocols with Logarithmic Space
Authors	Philipp Czerner, Vincent Fischer, Roland Guttenberg
Venue	DISC, 2024
Publisher	Dagstuhl
DOI	10.4230/LIPIcs.DISC.2024.44

Brief Announcement: The Expressive Power of Uniform Population Protocols with Logarithmic Space

Philipp Czerner
Technical University of Munich, Germany

Vincent Fischer
Technical University of Munich, Germany

Roland Guttenberg
Technical University of Munich, Germany

Abstract

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

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Population Protocols, Uniform, Expressive Power

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.44

Related Version *Full Version*: `https://arxiv.org/abs/2408.10027` [15]

1 Introduction

Population protocols are a model of computation in which indistinguishable mobile agents randomly interact in pairs to decide whether their initial configuration satisfies a given property. The decision is taken by *stable consensus*; eventually all agents agree on whether the property holds or not, and never change their mind again. While originally introduced to model sensor networks [4], population protocols are also very close to chemical reaction networks [23], a model in which agents are molecules and interactions are chemical reactions.

Originally agents were assumed to have a finite number of states [4, 5, 6], however many predicates then provably require at least $\Omega(n)$ time to decide [21, 7, 1], as opposed to recent breakthroughs of $\mathcal{O}(\log n)$ time using $\mathcal{O}(\log n)$ number of states (in some cases even $\mathcal{O}(\log \log n)$ states) for important tasks like leader election [9] and majority [19]. Limiting the number of states to logarithmic is important in most applications, especially the chemical reaction setting, since a linear in n number of states would imply the unrealistic number of approximately 10^{23} different chemical species. Therefore most recent literature focusses on the polylogarithmic time and space setting, and determines time-space tradeoffs for various important tasks like majority [3, 1, 2, 22, 8, 19], leader election [1, 22, 9] or estimating/counting the population size [20, 16, 10, 17, 18].

This leads to the interesting open problem of characterizing the class of predicates which can be computed in polylogarithmic time using a logarithmic or polylogarithmic number of states. There is however a fundamental problem with working on this question: Despite the focus on $\mathcal{O}(\log n)$ number of states in recent times, the expressive power for this number of states (regardless of time) is still unknown.

More precisely, there is a gap in the existing literature: protocols with $f(n) \in \Omega(n)$ states are known to have expressive power $\mathsf{SNSPACE}(n \log f(n))$ [14], i.e. symmetric predicates in $\mathsf{NSPACE}(n \log f(n))$, while a subclass of protocols with $o(\log n)$ states can only compute semilinear predicates [6, 14]. The latter result applies only to *uniform* population protocols, i.e. protocols where the transitions are independent of the size of the population.

However, many constructions in the literature have e.g. $\Theta(\log n)$ or $\Theta(\operatorname{polylog} n)$ states. This important case is not covered by the existing results. To the best of our knowledge, the only research in this direction is [12], where the expressive power is characterised for polylog(n) number of states for a similar model – not population protocols themselves. Most importantly, their results do not lead to a complete characterization for $\Theta(\log n)$ states since they lose some log factors in their characterization of polylog(n).

In this paper, we fill the gap by proving that for functions $f(n) \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$, where $\varepsilon > 0$, population protocol with $f(n)$ states compute exactly $\mathsf{SNSPACE}(f(n) \cdot \log n)$, i.e. the symmetric predicates computable by a non-deterministic Turing machine using $\mathcal{O}(f(n) \cdot \log n)$ space. This result applies to both uniform and non-uniform protocols. (The function f needs to fulfil some technical conditions.)

With this result, the expressive power of uniform population protocols is characterised in all cases, and for non-uniform protocols it is characterised in the case of $\Omega(\log n)$ states. (A slight gap between $\mathcal{O}(n^{1-\varepsilon})$ and $\Omega(n)$ remains.)

2 Preliminaries

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

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

A *configuration* of $\mathcal{P}$ is a finite multiset $C \in \mathbb{N}^Q$, which represents a collection of agents with states in Q. A step $C \to C'$ in $\mathcal{P}$ occurs by choosing two agents from C and letting them interact via δ, i.e. if their states are p, q in C, then their new states in C' will be $\delta(p, q)$.

We write $\to^*$ for the reflexive and transitive closure of $\to$, and say that a configuration C' is *reachable* from C if $C \to^* C'$. The input to $\mathcal{P}$ consists of a multiset $w \in \mathbb{N}^\Sigma$. Every input w can be mapped to its corresponding *initial* configuration by applying I to every letter in w

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

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

An input $w \in \mathbb{N}^\Sigma$ has *output* b if every fair run starting at its corresponding initial configuration $\hat{I}(w)$ has output b. The protocol scheme $\mathcal{P}$ *computes* a predicate $\varphi\colon \mathbb{N}^\Sigma \to \{0,1\}$ if every input w has output $\varphi(w)$.

Let us provide an example which also shows how to treat infinite sets Q.

▶ **Example 2.** Consider $Q := \{0\} \cup \{2^i \mid i \in \mathbb{N}\}$, and define $\delta(2^i, 2^i) = (2^{i+1}, 0)$. Let $\Sigma = \{x\}$, and let $x \mapsto 2^0$ be the input mapping. Then a configuration is initial if every agent is in state 2^0. Intuitively this protocol will eventually end up with the binary representation of the number of agents. Namely each transition preserves the total sum of all agents' values, and every transition increases the number of agents in 0, so this protocol in fact always reaches a terminal configuration.

Regarding the infinite state space, intuitively the protocol uses $\lfloor \log n \rfloor + 2$ states, namely $\lfloor \log n \rfloor + 1$ powers of two and 0. The other states cannot be reached with n agents.

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

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

As defined so far, protocol schemes are not necessarily computable. Hence actual population protocols require some uniformity condition.

▶ **Definition 4.** *A* uniform population protocol $\mathcal{P} = (Q, \Sigma, \delta, I, O)$ *is a protocol scheme together with a bijection $Q \to \{0,1\}^*$ to represent Q via binary strings, such that the functions δ, I, O are computable by linear space Turing-machines (TMs).*

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

In the literature on uniform population protocols, e.g. [13, 14, 20, 16], often agents are defined as TMs and states hence automatically assumed to be represented as binary strings. We avoid talking about the exact implementation of a protocol via TMs because it introduces an additional logarithm in the number of states and potentially confuses the reader, while most examples are clearly computable.

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

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

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

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

$\mathcal{P}$ is weakly uniform *if there exist TMs M_δ, M_I, M_O using $\mathcal{O}(S(n))$ space which compute δ_n, I_n and O_n, respectively, taking n as additional input.*

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

The protocol for a given population size n is allowed to differ completely from the protocol for $n-1$ agents, as long as TMs are still able to evaluate transitions, input and output. Usually this is not fully utilised, with the most common case of a non-uniform protocol being that $\log n$ is encoded into the transition function [19].

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

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

3 Main Result

We give a characterisation for the expressive power of both uniform and weakly uniform population protocols with $f(n)$ states, where $f \in \Omega(\log n) \cap \mathcal{O}(n^{1-\varepsilon})$, for some $\varepsilon > 0$. For technical reasons, we must place a few limitations on $f(n)$ (see the full paper [15] for details). We will refer to a function f fulfilling these requirements as *reasonable*.

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

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

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

Proof. This will follow from Proposition 8 and Theorem 9. ◀

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

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

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

Proof (sketch). $\mathsf{UPP}(f(n)) \subseteq \mathsf{WUPP}(f(n))$ is trivial/was explained in Section 2. $\mathsf{WUPP}(f(n)) \subseteq \mathsf{SNSPACE}(f(n) \log n)$ can be shown using a reduction to a reachability problem in the configuration graph as in [11]. ◀

Our main contribution is the proof of the lower bound:

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

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

Proof (sketch). We construct a uniform population protocol $\mathcal{P} = (Q, \Sigma, \delta, I, O)$, simulating a counter machine with $|\Sigma|$ input counters using space $\mathcal{O}(2^{f(n)\log n})$, which is equivalent to a $\mathcal{O}(f(n)\log n)$ space-bounded Turing machine.

Initialisation. Our first goal is to reach a configuration where the number of agents n is known. By this we mean, that we want to have $\lfloor \log n \rfloor + 1$ uniquely indentifiable "counter agents", each of which stores one bit of the binary representation of n. We use a similar approach as in Example 2 to achieve this:

$$\begin{aligned} &(\mathsf{Ctr}, i, 1), (\mathsf{Ctr}, i, 1) \mapsto (\mathsf{Ctr}, i+1, 1), (\mathsf{Ctr}, i, 0) && \text{for } i \in \mathbb{N} \\ &(\mathsf{Ctr}, i, 0), (\mathsf{Ctr}, i, b) \mapsto (\mathsf{Ctr}, i, b), (\mathsf{Ldr}, i+1) && \text{for } i \in \mathbb{N} \end{aligned} \qquad \langle \mathsf{counter} \rangle$$

Here $(\mathsf{Ctr}, i, 1)$ encodes that the i-th bit in the binary representation of n is set, while $(\mathsf{Ctr}, i, 0)$ represents an unset bit. The first transition is analogous to Example 2, but instead of simply sending the second agent to state 0, it also remembers which bit it represents. The second transition gets rid of additional agents storing the same bit.

Among the remaining agents we now want to elect one leader, who knows how many bits n has. We will refer to all agents which are neither counters nor a leader as *free*:

$$\begin{aligned} &(\mathsf{Ldr}, i), (\mathsf{Ldr}, j) \mapsto (\mathsf{Ldr}, j), \mathsf{Free} && \text{for } i, j \in \mathbb{N}, i \leq j \\ &(\mathsf{Ldr}, i), (\mathsf{Ctr}, j, b) \mapsto (\mathsf{Ldr}, j), (\mathsf{Ctr}, j, b) && \text{for } i, j \in \mathbb{N}, i \leq j \end{aligned} \qquad \langle \mathsf{leader} \rangle$$

The first transition here is a standard leader election, the second informs the leader of the number of bits required to store n.

At some point a configuration will be reached, where the binary counting and leader elections have been completed. Note that there is no way of telling for certain when this is the case. For now, we will assume that we have reached such a configuration and describe how to solve this problem later on.

Once n is known, we gain the ability to loop over all agents: each of the counter agents stores an additional, initially unset, bit. Every agent stores a marker flag. The Leader can then apply operations to all agents by sequentially interacting with them and setting the marker flag. Each time the leader interacts with an agent which has the marker flag unset, it increments the second value stored in the counter agents. If at some point both values match, then, as the first value is n, all of the agents must have been marked. In particular this allows the leader to check if an agent with a certain state does not exist. Normally this is quite difficult for population protocols, as agents in the queried state might not take part in any interactions for an arbitrarily long time.

Simulating Counter Machines. Often, when population protocols need to simulate some type of counter, either a unary [5], or binary encoding [12], is used. Neither approach works for us, as we need to be able to count up to $2^{f(n)\log n}$, but a unary encoding with n agents is bounded by n, and a binary encoding with $f(n)$ distinguishable digits is bounded by $2^{f(n)}$. Instead we use a mixed-radix positional encoding with the base $b_i \in \Omega(\frac{n}{f(n)})$ for every digit i. To achieve this, the leader evenly divides the remaining free agents into $\Omega(f(n))$ groups, each encoding one digit. Recall that the leader can detect when no free agents remain, so it will know when this process is finished. Within each digit unary counting is used, that is, each agent in that digit stores one counter bit and the overall value of the digit is the sum of all counter bits. The commands of the counter machine involve manipulating these digits, by either incrementing, or decrementing the encoded values, as well as checking whether they are zero. For the latter, the leader again uses the ability to detect whether a state is present in the population.

Resets. The counter machine simulation described in the previous section relies on the looping and absence checks enabled by the data structures set up during initialisation. However, there is no way of being certain that the initialisation has finished. We solve this by raising a dirty flag each time a transition from the initialisation phase occurs. When seen by the leader, this will trigger a reset, where the leader will move all agents back to state Free, once again relying on being able to count the number of agents. When the last reset occurs, the counter agents must encode the correct value of n, and the leader is thus able to iterate over all agents. Care must be taken s.t. other agents do not interact with agents in Free while the reset is ongoing, e.g. when only half of the agents have moved to Free, and the others are still some intermediate states. ◀

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

15 Philipp Czerner, Vincent Fischer, and Roland Guttenberg. The expressive power of uniform population protocols with logarithmic space, 2024. arXiv:2408.10027.

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

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

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

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

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

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

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

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

F. Running Time Analysis of Broadcast Consensus Protocols

Title	Running Time Analysis of Broadcast Consensus Protocols
Authors	Philipp Czerner, Stefan Jaax
Venue	FOSSACS, 2021
Publisher	Springer
DOI	10.1007/978-3-030-71995-1_9

Running Time Analysis of Broadcast Consensus Protocols* **

Philipp Czerner[1] [✉] and Stefan Jaax[1]

Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
{czerner,jaax}@in.tum.de

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

Keywords: broadcast protocols · complexity theory · distributed computing

1 Introduction

In recent years, models of distributed computation following the *computation-by-consensus* paradigm attracted considerable interest in research (see for example [9,25,26,8,13]). In such models, network agents compute number predicates, i.e. Boolean-valued functions of the type $\mathbb{N}^k \to \{0,1\}$, by reaching a stable consensus whose value determines the outcome of the computation. Perhaps the most prominent model following this paradigm are *population protocols* [5,6], a model in which anonymous, identical, finite-state agents interact randomly in pairwise rendezvous to agree on a common Boolean output.

Due to anonymity and locality of interactions, it is an inherent property of population protocols that agents are generally unable to detect with absolute

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

** The full version of this paper can be found at https://arxiv.org/abs/2101.03780 .

S. Kiefer and C. Tasson (Eds.): FOSSACS 2021, LNCS 12650, pp. 164–183, 2021.
https://doi.org/10.1007/978-3-030-71995-1_9

certainty when the computation has stabilized. This makes sequential composition of protocols difficult, and further complicates the implementation of control structures such as loops or branching statements. To overcome this drawback, two kinds of approaches have been suggested in the literature: 1.) Let agents guess when the computation has stabilized, leading to composable, but merely *approximately correct* protocols [7,24], or 2.) extend population protocols by global communication primitives that enable agents to query global properties of the agent population [13,8,26].

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

In [13], Blondin, Esparza, and one of the authors of this paper introduced *broadcast consensus protocols* (BCPs), an extension of population protocols by reliable, global, and atomic broadcasts. BCPs find their precursor in the broadcast protocol model introduced by Emerson and Namjoshi in [17] to describe bus-based hardware protocols. This model has been investigated intensely in the literature, see e.g. [18,19,15,28]. Broadcasts also arise naturally in biological systems. For example, Uhlendorf *et al.* analyse applications of broadcasts in the form of an external, global light source for controlling a population of yeasts [12].

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

An epidemics-based approach was used in [7] to show that population protocols can simulate with high probability a step of a virtual register machine with expected $\mathcal{O}(n \log^5(n))$ interactions, where n is the number of agents. This result stimulated further research into time bounds for classical problems such as leader election (see e.g. [21,1,16,29,11]) and majority (see e.g. [4,2]). In their seminal paper [5], Angluin *et al.* already showed that population protocols can stably compute Presburger predicates with $\mathcal{O}(n^2 \log n)$ interactions in expectation. Belleville *et al.* further showed that leaderless protocols require a quadratic number of interactions in expectation to stabilize to the correct output for a wide class of predicates [10]. The aforementioned bounds apply to *stabilisation time*: the time it takes to go from an initial configuration to a stable consensus that cannot be destroyed by future interactions. In [24], Kosowski and Uznanski considered the weaker notion of *convergence time*: the time it takes on average to ultimately transition to the correct consensus (although this consensus could

in principle be destroyed by future interactions), and they show that sublinear convergence time is achievable.

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

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

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

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

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

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

2 Preliminaries

Complexity classes. As is usual, we define NL as the class of languages decidable by a nondeterministic log-space TM. Additionally, by ZPL we denote the set of languages decided by a randomised log-space TM A, s.t. A only terminates with the correct result (zero-error) and that it terminates within $\mathcal{O}(\text{poly}\, n)$ steps in expectation, as defined by Nisan in [27].

Multisets. A *multiset* over a finite set E is a mapping $M\colon E \to \mathbb{N}$. The set of all multisets over E is denoted $\mathbb{N}^E$. For every $e \in E$, $M(e)$ denotes the number of occurrences of e in M. We sometimes denote multisets using a set-like notation, e.g. $\{\!\{f, g, g\}\!\}$ is the multiset M such that $M(f) = 1$, $M(g) = 2$ and

$M(e) = 0$ for every $e \in E \setminus \{f, g\}$. Addition, comparison and scalar multiplication are extended to multisets componentwise, i.e. $(M + M')(e) \stackrel{\text{def}}{=} M(e) + M'(e)$, $(\lambda M)(e) \stackrel{\text{def}}{=} \lambda M(e)$ and $M \leq M' \stackrel{\text{def}}{\iff} M(e) \leq M'(e)$ for every $M, M' \in \mathbb{N}^Q$, $e \in E$, and $\lambda \in \mathbb{N}$. For $M' \leq M$ we also define componentwise subtraction, i.e. $(M - M')(e) \stackrel{\text{def}}{=} M(e) - M'(e)$ for every $e \in E$. For every $e \in E$, we write $\boldsymbol{e} \stackrel{\text{def}}{=} \{\!|e|\!\}$. We lift functions $f \colon E \to E'$ to multisets by defining $f(M)(e') \stackrel{\text{def}}{=} \sum_{f(e)=e'} M(e)$ for $e' \in E'$. Finally, we define the *support* and *size* of $M \in \mathbb{N}^E$ respectively as $[\![M]\!] \stackrel{\text{def}}{=} \{e \in E : M(e) > 0\}$ and $|M| \stackrel{\text{def}}{=} \sum_{e \in E} M(e)$.

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

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

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

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

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

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

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

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

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

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

Executions. An *execution* is an infinite sequence $\pi = C_0C_1C_2...$ of configurations with $C_i \to C_{i+1}$ for every i. It has some fixed number of agents $n \stackrel{\text{def}}{=} |C_0| = |C_1| = ...$. Given a BCP and an initial configuration $C_0 \in \mathbb{N}^Q$, we generate a random execution with the following Markov chain: to perform a step at configuration C_i, a state $q \in Q$ is picked at random with probability distribution $p(q) = C_i(q)/|C_i|$, and the (uniquely defined) transition $\delta(q)$ is executed, giving the successor configuration C_{i+1}. We refer to the random variable corresponding to the trace of this Markov chain as *random execution*.

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

Finally, for an execution $\pi = C_0C_1C_2...$ we let T_π denote the smallest i s.t. all configurations in $C_iC_{i+1}...$ are $\varphi(X)$-consensuses, or ∞ if no such i exists. We say that a BCP $\mathcal{P}$ *computes* φ *within* $f(n)$ *interactions*, if for all initial configurations C_0 with n agents the random execution π starting at C_0 has $\mathbb{E}(T_\pi) \leq f(n) < \infty$, i.e. $\mathcal{P}$ stabilises within $f(n)$ steps in expectation. If $f \in \mathcal{O}(\text{poly}(n))$, then we call $\mathcal{P}$ a *polynomial-time* BCP.

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

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

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

3 Example: Majority

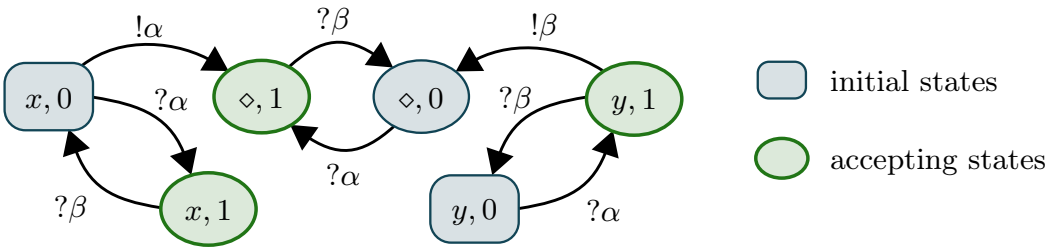


Fig. 1. A fast broadcast consensus protocol computing the majority predicate.

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

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

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

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

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

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

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

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

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

Considering these alternations with shrinking factions, the expected number of steps of the protocol until stabilization can be bounded by $2\sum_{k=1}^{n} n/k =$

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

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

4 Comparison with other Models

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

4.1 Non-Deterministic Broadcast Protocols

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

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

We implement this using a synthetic coin, i.e. we are utilising randomness provided by the scheduler to enable individual agents to make random choices. This idea has also been used for population protocols [1,3]. Compared to these implementations, broadcasts allow for a simpler approach.

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

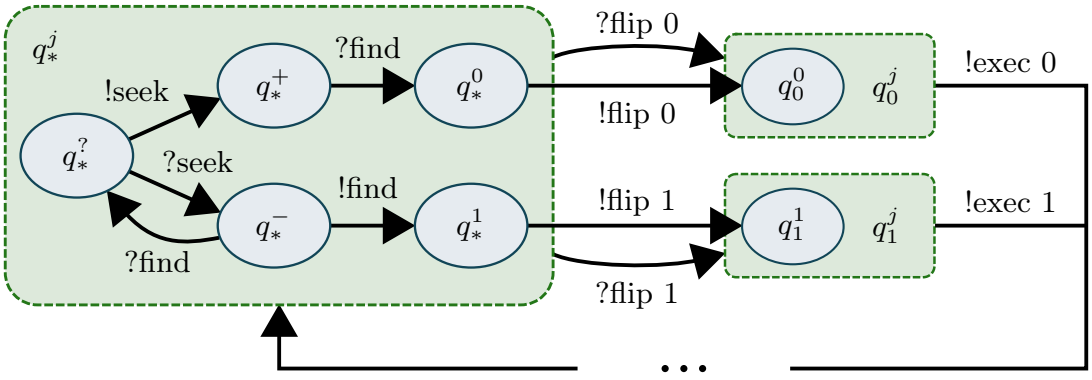


Fig. 2. Transition diagram for implementing multiple broadcasts per state, for $q \in Q$, with (q, i, j) written as q_j^i. Dashed nodes represent multiple states, with $j \in T$. Transitions resulting from executing the broadcasts in δ_0, δ_1 are not shown.

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

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

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

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

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

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

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

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

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

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

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

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

4.2 Population Protocols

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

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

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

5 Protocols for Presburger Arithmetic

While Blondin, Esparza and Jaax [14] show that BCPs are more expressive than population protocols, they leave the question open whether BCPs provide a run-time speed-up for the class of Presburger predicates computable by population protocols. We already saw that Majority can be computed within $\mathcal{O}(n \log n)$ interactions in BCPs. This also holds in general for Presburger predicates:

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

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

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

5.1 Linear Inequalities

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

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

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

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

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

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

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

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

denote the sum of all agents' contributions and the current value of the counter. Every initial configuration C_0 has $\mathsf{ctr}(C) = 0$ and thus $\mathsf{sum}(C) = \sum_i \alpha x_i$. Each

transition α increases the counter by α but sets the agent's contribution to 0 (from α), so $\mathsf{sum}(C)$ is constant throughout the execution.

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

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

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

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

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

5.2 Modulo Predicates and Boolean Combinations

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

Proof (sketch). The idea is the same as for Proposition 1, but instead of taking care not to overflow the counter we simply perform the additions modulo l.

Proposition 3 (Boolean combination of predicates). *Let φ be a Boolean combination of predicates $\varphi_1, \ldots, \varphi_k$, which are computed by BCPs $\mathcal{P}_1, \ldots, \mathcal{P}_k$, respectively, within $\mathcal{O}(n \log n)$ interactions. Then there is a protocol computing φ within $\mathcal{O}(n \log n)$ interactions.*

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

6 Protocols for all Predicates in ZPL

BCPs compute precisely the predicates in NL with input encoded in unary, which corresponds to $\mathsf{NSPACE}(n)$ when encoded in binary. The proof of the NL

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

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

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

Lemma 1. *Polynomial-time BCPs compute at most the predicates in* ZPL *with input encoded in unary.*

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

Theorem 2. *Polynomial-time BCPs compute exactly the predicates in* ZPL *with input encoded in unary.*

The proof of Theorem 2 will take up the remainder of this section.

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

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

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

- $\text{step}(\mathsf{mul}_2, x) \stackrel{\text{def}}{=} (\mathsf{done}_0, 2x)$,
- $\text{step}(\mathsf{inc}, x) \stackrel{\text{def}}{=} (\mathsf{done}_0, x + 1)$,

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

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

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

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

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

Theorem 3. *Let φ be a predicate decidable by a log-space bounded RTM within $\mathcal{O}(f(n))$ steps in expectation with unary input encoding. There exists an n-bounded CM that accepts φ within $\mathcal{O}(f(n)\log(n))$ steps in expectation.*

Proof (sketch). This can be shown by first representing the Turing machine by a stack machine with two stacks that contain the tape content to the left/right of the current machine head position. In this representation, head movements and tape updates amount to performing pop/push operations on the stack. Moreover, we can simulate an $c \cdot n$-bounded stack by c many n-bounded stacks. An n-bounded stack, in turn, can be represented in a counter machine with a constant number of 2^n-bounded counters. The stack content is represented as the base-2 number corresponding to the binary sequence stored in the stack. Popping then amounts to a divmod_2 operation, and pushing amounts to doubling the counter value, followed by adding 1 or 0, respectively.

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

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

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

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

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

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

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

Lemma 2. *Let* $n \geq 3$ *and* $X_1, ..., X_n$ *denote independent random variables with sum* X *and* $X_i \sim \mathcal{G}(i/n)$*. Then for any* $k \geq 1$ *there is an* l *s.t.*

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

Proof. See the full version of this paper.

Lemma 3. *There is a CM-simulating BP s.t. starting from an initial configuration it reaches a final configuration within* $\mathcal{O}(n \log n)$ *steps with high probability.*

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

$$(b, \mathsf{mul}_2) \mapsto (2b, \mathsf{done}_0),\ \{1 \mapsto 2, 0 \mapsto 0\} \tag{α_1}$$

$$(b, \mathsf{divmod}_2) \mapsto (\tfrac{b}{2}, \mathsf{done}_0),\ \{1 \mapsto \tfrac{1}{2}, 0 \mapsto 0\} \tag{α_2}$$

$$(b, \mathsf{inc}) \mapsto (b, \mathsf{high}),\ \emptyset \tag{α_3}$$

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

$$(0, \mathsf{high}) \mapsto (1, \mathsf{done}_0),\ \emptyset \tag{β_1}$$

$$(2, \mathsf{done}_0) \mapsto (1, \mathsf{high}),\ \emptyset \tag{β_2}$$

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

$$(\tfrac{1}{2}, \mathsf{done}_1) \mapsto (1, \mathsf{done}_0),\ \emptyset \tag{β_4}$$

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

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

$$(1, \mathsf{iszero}) \mapsto (1, \mathsf{done}_1),\ \emptyset \qquad (\alpha_4)$$

$$(0, \mathsf{iszero}) \mapsto (0, \mathsf{done}_0),\ \{1 \mapsto *\} \qquad (\alpha_5)$$

$$(*, \mathsf{done}_0) \mapsto (1, \mathsf{done}_1),\ \{* \mapsto 1\} \qquad (\beta_5)$$

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

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

It remains to be shown that starting from a configuration C_0 we reach a final configuration within $\mathcal{O}(n \log n)$ steps with high probability. Note that transitions (α_1-α_5) are executed at most once. Moreover, these are the only transitions enabled at C_0, so let C_1 denote the successor configuration after executing (α_1-α_5), i.e. $C_0 \to C_1$. From now on, we consider only transitions (β_1-β_5).

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

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

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

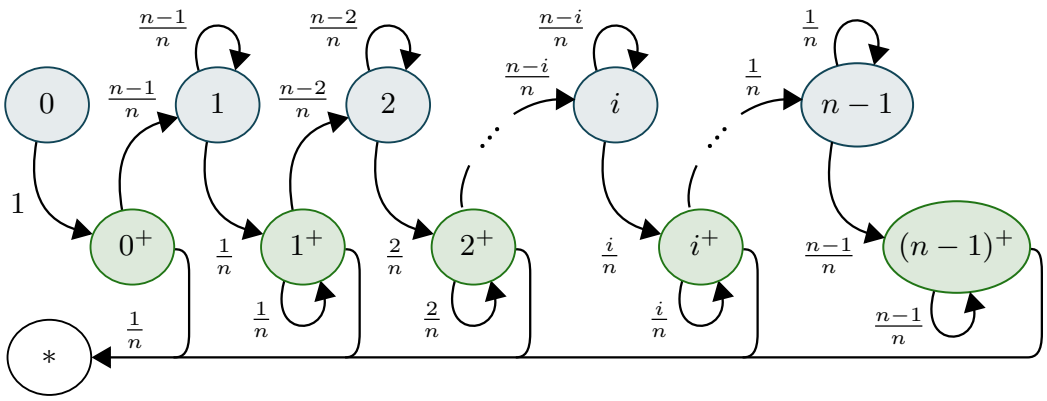


Fig. 3. State diagram of the clock implementation. Nodes with i agents in state c_3 are labelled i or i^+, the latter denoting that the other agents are in states c_1^+ and c_2^+. The final state $*$ has all agents in state 1. Arcs are labelled with transition probabilities.

Let X_k denote the number of silent transitions before executing (β_j) for the k-th time, $k = 1, ..., l$, and let r_k denote the number of agents in state q at that time. Then $n \geq r_1 > r_2 > ... > r_l \geq 1$ and X_k is distributed according to $\mathcal{G}(r_k/n)$. So we can use Lemma 2 to show that the sum of X_k is $\mathcal{O}(n \log n)$ with high probability. There are only 5 transitions (β_j), so the same holds for the total number of steps until reaching a final state.

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

Lemma 4. *There is a clock-BP* $\mathcal{P} = (Q, \delta)$ *s.t.* $\mathbb{E}(\text{Time}(\mathcal{P})) \in \mathcal{O}(n \log n)$ *and* $\text{Time}(\mathcal{P}) \in \Omega(n \log n)$ *with probability* $1 - \mathcal{O}(n^{-1/2})$.

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

$$0 \mapsto c_1^+, \{0 \mapsto c_2^+\} \qquad (\alpha)$$

$$c_2^+ \mapsto c_3, \{c_2^+ \mapsto c_2, c_1^+ \mapsto c_1\} \qquad (\beta)$$

$$c_3 \mapsto c_3, \{c_2 \mapsto c_2^+, c_1 \mapsto c_1^+\} \qquad (\gamma)$$

$$c_1^+ \mapsto 1, \{c_2^+ \mapsto 1, c_3 \mapsto 1\} \qquad (\omega)$$

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

It remains to show that this protocol fulfils the stated time bounds. We prove $\mathbb{E}(\text{Time}(\mathcal{P})) \in \mathcal{O}(n \log n)$ by using that, in expectation, the protocol spends at

most n/j steps in state j and at most $n/(n-j)$ in state j^+. For the lower bound we make a case distinction: either state $\lfloor\sqrt{n}\rfloor$ is not visited (i.e. the leader is one of the first $\sqrt{n}$ agents to be infected), or the total number of steps is at least $X_1 + ... + X_{\lfloor\sqrt{n}\rfloor}$, where X_j is the number of steps the protocol spends in state i. As X_j is geometrically distributed with mean n/j, we apply a tail bound from Janson [23] to get the desired result.

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

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

Lemma 5. *Let $k \in \mathbb{N}$ denote some constant. Then there is a clock-BP $\mathcal{P}$ s.t. $\mathbb{E}(\mathrm{Time}(\mathcal{P})) \in \mathcal{O}(n \log n)$, and $\mathrm{Time}(\mathcal{P}) < kn \log n$ with probability $\mathcal{O}(n^{-k})$.*

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

As mentioned earlier, we combine the clock with the construction in Lemma 3. While we cannot reliably determine whether the operation has finished, we can use a clock to measure an interval of time long enough for the protocol to terminate with high probability. The next construction does just that. In particular, in contrast to Lemma 3, it uses its global state to indicate that it is done.

Lemma 6. *There is a CM-simulating BP s.t. starting from an initial configuration it reaches either a final or a failing configuration C almost surely and within $\mathcal{O}(n \log n)$ steps in expectation, and C is final with high probability. Additionally, all reachable configurations with global state in* Ret *are final or failing.*

Proof. Fix some $k \in \mathbb{N}$ and let $\mathcal{P} = (Q \times G, \delta)$ denote the BP we want to construct. Further, let $\mathcal{P}_1 = (Q_1 \times G_1, \delta_1)$ denote the BP from Lemma 3 and choose some c s.t. $\mathcal{P}_1$ reaches a final configuration after at most $cn \log n$ steps with probability at least $1 - n^{-k}$.

Now we use Lemma 5 to get a clock $\mathcal{P}_2 = (Q_2, \delta_2)$ that runs for at least $cn \log n$ steps with probability at least $1 - n^{-k}$.

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

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

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

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

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

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

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

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

Lemma 7. *Fix some predicate $\varphi : \mathbb{N}^k \to \{0, 1\}$ computable by an n-bounded l-CM within $\mathcal{O}(f(n)) \subseteq \mathcal{O}(\mathrm{poly}\, n)$ steps. Then there is a BCP computing φ in $\mathcal{O}(f(n)\, n \log n)$ steps.*

Proof (sketch). For each counter we need n agents, so ln in total, but we can simply have each agent simulate a constant number of agents. To execute a step of the CM, we use the BP from Lemma 6. It succeeds only with high probability, but in the case of failure at least one agent will have local state $\bot$, from which that agent initiates a restart of the whole computation.

As the CM takes only a polynomial number of steps, we can fix a k s.t. a computation of our BCP without failures (i.e. one that succeeds on the first try) takes $\mathcal{O}(n^k)$ steps. A single step succeeds with high probability, so we can require it to fail with probability at most $\mathcal{O}(n^{-k-1})$. In total, the restarts increase the running time by a factor of $1/(1 - \mathcal{O}(n^{-1}))$, which is only a constant overhead.

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

This completes the proof of Theorem 2. By Theorem 3, each predicate in ZPL (with input encoded in unary) is computable by a bounded l-CM. Lemma 7 then yields a polynomial-time BCP for that predicate.

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

References

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

2. Alistarh, D., Aspnes, J., Gelashvili, R.: Space-optimal majority in population protocols. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 2221–2239. SIAM (2018)
3. Alistarh, D., Gelashvili, R.: Recent algorithmic advances in population protocols. ACM SIGACT News **49**(3), 63–73 (2018)
4. Alistarh, D., Gelashvili, R., Vojnović, M.: Fast and exact majority in population protocols. In: Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing. pp. 47–56 (2015)
5. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed computing **18**(4), 235–253 (2006), https://www.cs.yale.edu/homes/aspnes/papers/podc04passive-dc.pdf
6. Angluin, D., Aspnes, J., Eisenstat, D.: Stably computable predicates are semilinear. In: Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing. pp. 292–299. PODC '06, ACM, New York, NY, USA (2006). https://doi.org/10.1145/1146381.1146425
7. Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. Distributed Computing **21**(3), 183–199 (2008), https://www.cs.yale.edu/homes/aspnes/papers/disc2006-journal.pdf
8. Aspnes, J.: Clocked population protocols. In: Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing (PODC). pp. 431–440 (2017). https://doi.org/10.1145/3087801.3087836
9. Aspnes, J., Ruppert, E.: An introduction to population protocols. In: Middleware for Network Eccentric and Mobile Applications, pp. 97–120. Springer (2009)
10. Belleville, A., Doty, D., Soloveichik, D.: Hardness of computing and approximating predicates and functions with leaderless population protocols. In: 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
11. Berenbrink, P., Giakkoupis, G., Kling, P.: Optimal time and space leader election in population protocols. In: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing. pp. 119–129 (2020)
12. Bertrand, N., Dewaskar, M., Genest, B., Gimbert, H.: Controlling a population. In: Proc. 28$^{\text{th}}$ International Conference on Concurrency Theory (CONCUR). vol. 85, pp. 12:1–12:16 (2017). https://doi.org/10.4230/LIPIcs.CONCUR.2017.12
13. Blondin, M., Esparza, J., Jaax, S.: Expressive Power of Broadcast Consensus Protocols. In: Proceedings of the 30th International Conference on Concurrency Theory (CONCUR). pp. 31:1–31:16 (2019). https://doi.org/10.4230/LIPIcs.CONCUR.2019.31, http://drops.dagstuhl.de/opus/volltexte/2019/10933
14. Blondin, M., Esparza, J., Jaax, S.: Expressive power of broadcast consensus protocols (2019), https://arxiv.org/abs/1902.01668.pdf
15. Delzanno, G., Raskin, J., Begin, L.V.: Towards the automated verification of multithreaded Java programs. In: Proc. 8$^{\text{th}}$ International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 173–187 (2002)
16. Doty, D., Soloveichik, D.: Stable leader election in population protocols requires linear time. Distributed Computing **31**(4), 257–271 (2018)
17. Emerson, E.A., Namjoshi, K.S.: On model checking for non-deterministic infinite-state systems. In: Proc. Thirteenth Annual IEEE Symposium on Logic in Computer Science (LICS). pp. 70–80 (1998). https://doi.org/10.1109/LICS.1998.705644

18. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: Proc. 14th Annual IEEE Symposium on Logic in Computer Science (LICS). pp. 352–359 (1999). https://doi.org/10.1109/LICS.1999.782630
19. Finkel, A., Leroux, J.: How to compose Presburger-accelerations: Applications to broadcast protocols. In: Proc. 22nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS). pp. 145–156 (2002)
20. Flajolet, P., Gardy, D., Thimonier, L.: Birthday paradox, coupon collectors, caching algorithms and self-organizing search. Discrete Applied Mathematics **39**(3), 207–229 (1992)
21. Gąsieniec, L., Staehowiak, G.: Fast space optimal leader election in population protocols. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 2653–2667. SIAM (2018)
22. Ginsburg, S., Spanier, E.H.: Semigroups, presburger formulas, and languages. Pacific J. Math. **16**(2), 285–296 (1966), https://projecteuclid.org:443/euclid.pjm/1102994974
23. Janson, S.: Tail bounds for sums of geometric and exponential variables. Statistics & Probability Letters **135**, 1–6 (2018), https://arxiv.org/pdf/1709.08157.pdf
24. Kosowski, A., Uznanski, P.: Brief announcement: Population protocols are fast. In: Newport, C., Keidar, I. (eds.) Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018. pp. 475–477. ACM (2018), https://dl.acm.org/citation.cfm?id=3212788
25. Michail, O., Chatzigiannakis, I., Spirakis, P.G.: Mediated population protocols. Theoretical Computer Science **412**(22), 2434–2450 (2011)
26. Michail, O., Spirakis, P.G.: Terminating population protocols via some minimal global knowledge assumptions. Journal of Parallel and Distributed Computing pp. 1–10 (2015). https://doi.org/10.1016/j.jpdc.2015.02.005
27. Nisan, N.: On read once vs. multiple access to randomness in logspace. Theoretical Computer Science **107**(1), 135–144 (1993)
28. Schmitz, S., Schnoebelen, P.: The power of well-structured systems. In: Proc. 24th International Conference on Concurrency Theory (CONCUR). pp. 5–24 (2013)
29. Sudo, Y., Masuzawa, T.: Leader election requires logarithmic time in population protocols. Parallel Processing Letters **30**(01), 2050005 (2020)

G. Decision Power of Weak Asynchronous Models of Distributed Computing

Title	Decision Power of Weak Asynchronous Models of Distributed Computing
Authors	Philipp Czerner, Roland Guttenberg, Martin Helfrich, Javier Esparza
Venue	PODC, 2021
Publisher	ACM
DOI	10.1145/3465084.3467918

Decision Power of Weak Asynchronous Models of Distributed Computing

Philipp Czerner
czerner@in.tum.de
Technische Universität München
München, Germany

Roland Guttenberg
guttenbe@in.tum.de
Technische Universität München
München, Germany

Martin Helfrich
helfrich@in.tum.de
Technische Universität München
München, Germany

Javier Esparza
esparza@in.tum.de
Technische Universität München
München, Germany

ABSTRACT

Esparza and Reiter have recently conducted a systematic comparative study of models of distributed computing consisting of a network of identical finite-state automata that cooperate to decide if the underlying graph of the network satisfies a given property. The study classifies models according to four criteria, and shows that twenty-four initially possible combinations collapse into seven equivalence classes with respect to their decision power, i.e. the properties that the automata of each class can decide. However, Esparza and Reiter only show (proper) inclusions between the classes, and so do not characterise their decision power. In this paper we do so for *labelling* properties, i.e. properties that depend only on the labels of the nodes, but not on the structure of the graph. In particular, majority (whether more nodes carry label a than b) is a labelling property. Our results show that only one of the seven equivalence classes identified by Esparza and Reiter can decide majority for arbitrary networks. We then study the expressive power of the classes on bounded-degree networks, and show that three classes can. In particular, we present an algorithm for majority that works for all bounded-degree networks under adversarial schedulers, i.e. even if the scheduler must only satisfy that every node makes a move infinitely often, and prove that no such algorithm can work for arbitrary networks.

CCS CONCEPTS

• **Theory of computation → Distributed computing models**.

KEYWORDS

Distributed computing; graph automata; weak asynchronous models; labelling properties; communication graphs; decision power

PODC '21, July 26–30, 2021, Virtual Event, Italy

ACM ISBN 978-1-4503-8548-0/21/07…$15.00
https://doi.org/10.1145/3465084.3467918

ACM Reference Format:
Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. 2021. Decision Power of Weak Asynchronous Models of Distributed Computing. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC '21), July 26–30, 2021, Virtual Event, Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3465084.3467918

1 INTRODUCTION

A common feature of networks of natural or artificial devices, like molecules, cells, microorganisms, or nano-robots, is that agents have very limited computational power and no identities. Traditional distributed computing models are often inadequate to study the power and efficiency of these networks, which has led to a large variety of new models, including population protocols [3, 4], chemical reaction networks [30], networked finite state machines [16], the weak models of distributed computing of [21], and the beeping model [1, 14] (see e.g. [18, 28] for surveys and other models).

These new models share several characteristics [16]: the network can have an arbitrary topology; all nodes run the same protocol; each node has a finite number of states, independent of the size of the network or its topology; state changes only depend on the states of a bounded number of neighbours; nodes do not know their neighbours, in the sense of [2]. Unfortunately, despite such substantial common ground, the models still exhibit much variability. In [17] Esparza and Reiter have recently identified four fundamental criteria according to which they diverge:

- *Detection.* In some models, nodes can only detect the *existence* of neighbours in a certain state, e.g., [1, 21], while in others they can *count* their number up to a fixed threshold, e.g., [16, 21].
- *Acceptance.* Some models compute by *stable consensus*, requiring all nodes to eventually agree on the outcome of the computation, e.g. [3, 4, 30]; others require the nodes to produce an output and halt, e.g. [21, 23].
- *Selection.* Some models allow for *liberal selection*: at each moment, an arbitrary subset of nodes is selected to take a step [16, 29]. *Exclusive* models (also called *interleaving* models) select exactly one node (or one pair of neighbouring nodes) [3, 4, 30]. *Synchronous* models select all nodes at each step e.g.,[21] or classical synchronous networks [25].
- *Fairness.* Some models assume that selections are *adversarial*, only satisfying the minimal requirement that each node is

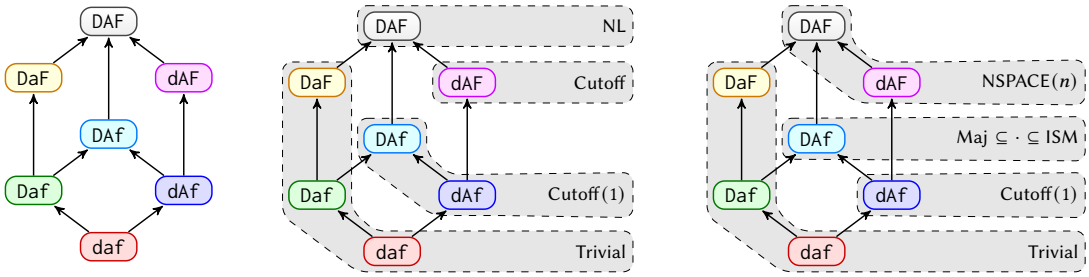


Figure 1: The seven distributed automata models of [17]; their decision power w.r.t. labelling predicates for arbitrary networks, and for bounded-degree networks. ISM **stands for *invariant under scalar multiplication*. The other complexity classes are defined in Section 5.**

> selected infinitely often [19, 24]. Others assume *stochastic* or *pseudo-stochastic* selection (meaning that selections satisfy a fairness assumption capturing the main features of a stochastic selection) [3, 4, 30]. In this case, the selection scheduler is a source of randomness that can be tapped by the nodes to ensure e.g. that eventually all neighbours of a node will be in different states.

In [17], Esparza and Reiter initiated a comparative study of the computational power of these models. They introduced *distributed automata*, a generic formalism able to capture all combinations of the features above. A distributed automaton consists of a set of rules that tell the nodes of a labelled graph how to change their state depending on the states of their neighbours. Intuitively, the automaton describes an algorithm that allows the nodes to decide whether the graph satisfies a given property. The decision power of a class of automata is the set of graph properties they can decide, for example whether the graph contains more red nodes than blue nodes (the *majority* property), or whether the graph is a cycle. The main result of [17] was that the twenty-four classes obtained by combining the features above collapse into only seven equivalence classes w.r.t. their decision power. The collapse is a consequence of a fundamental result: the selection criterion does not affect the decision power. That is, the liberal, exclusive, or synchronous versions of a class with the same choices in the detection, acceptance, and fairness categories, have the same decision power. The seven equivalence classes are shown on the left of Figure 1, where D and d denote detection with and without the ability to count; A and a denote acceptance by stable consensus and by halting; and F and f denote pseudo-stochastic and adversarial fairness constraints. So, for example, DAf corresponds to the class of distributed automata in which agents can count, acceptance is by stable consensus, and selections are adversarial. (As mentioned above, the selection component is irrelevant, and one can assume for example that all classes have exclusive selection.) Intuitively, the capital letter corresponds to the option leading to higher decision power.

The results of [17] only prove inclusions between classes and separations, but give no information on which properties can be decided by each class, an information available e.g. for multiple variants of population protocols [3, 6, 7, 11, 20, 26]. In this paper, we characterise the decision power of all classes of [17] w.r.t. *labelling* properties, i.e. properties that depend only on the labels of the nodes. Formally, given a labelled graph G over a finite set Λ of labels, let $L_G : \Lambda \to \mathbb{N}$ be the *label count* of G that assigns to each label the number of nodes carrying it. A labelling property is a set $\mathcal{L}$ of label counts. A graph G satisfies $\mathcal{L}$ if $L_G \in \mathcal{L}$, and a distributed automaton decides $\mathcal{L}$ if it recognises exactly the graphs that satisfy $\mathcal{L}$. For example, the majority property is a labelling property, while the property of being a cycle is not.

Our first collection of results is shown in the middle of Figure 1. We prove that all classes with halting acceptance can only decide the trivial labelling properties $\emptyset$ and $\mathbb{N}^\Lambda$. More surprisingly, we further prove that the computational power of DAf, dAf, and dAF is very limited. Given a labelled graph G and a number K, let $\lceil L_G \rceil_K$ be the result of substituting K for every component of L_G larger than K. The classes DAf, dAf can decide a property $\mathcal{L}$ iff membership of L_G in $\mathcal{L}$ depends only on $\lceil L_G \rceil_1$, and dAF iff membership depends only on $\lceil L_G \rceil_K$ for some $K \geq 1$. In particular, none of these classes can decide majority. Finally, moving to the top class DAF causes a large increase in expressive power: DAF can decide exactly the labelling properties in the complexity class NL, i.e. the properties $\mathcal{L}$ such that a nondeterministic Turing machine can decide membership of L_G in $\mathcal{L}$ using logarithmic space in the number of nodes of G. In particular, DAF-automata can decide majority, or whether the graph has a prime number of nodes.

In the last part of the paper, we obtain our second and most interesting collection of results. Molecules, cells, or microorganisms typically have short-range communication mechanisms, which puts an upper bound on their number of communication partners. So we re-evaluate the decision power of the classes for bounded-degree networks, as also done in [3] for population protocols on graphs. Intuitively, nodes know that they have at most k neighbours for some fixed number k, and can exploit this fact to decide more properties. Our results are shown on the right of Figure 1. Both DAF and dAF boost their expressive power to NSPACE(n), where n is the number of nodes of the graph. This is the theoretical upper limit since each node has a constant number of bits of memory. Further, the class DAf becomes very interesting. While we are not yet able to completely characterise its expressive power, we show that it can only decide properties *invariant under scalar multiplication* (ISM), i.e. labelling properties $\mathcal{L}$ such that $L_G \in \mathcal{L}$ iff $\lambda \cdot L_G \in \mathcal{L}$ for every $\lambda \in \mathbb{N}$, and that it can decide all properties satisfied by a graph G iff L_G is a solution to a system of homogeneous linear inequalities. In particular, DAf can decide majority, and we have

the following surprising fact. If nodes have no information about the network, then they require stochastic-like selection to decide majority; however, if they know an upper bound on the number of their neighbours, they can decide majority even with adversarial selection. In particular, there is a synchronous majority algorithm for bounded-degree networks.

Related work. Decision power questions have also been studied in [21] for a model similar to Daf, and in [13] for a graph version of the mediated population protocol model [26]. The distinguishing feature of our work is the systematic study of the influence of a number of features on the decision power.

Further, there exist numerous results about the decision power of different classes of population protocols. Recall that agents of population protocols are indistinguishable and communicate by rendez-vous; this is equivalent to placing the agents in a clique and selecting an edge at every step. Angluin *et al.* show that standard population protocols compute exactly the semilinear predicates [6]. Extensions with absence detectors or cover-time services [27], consensus-detectors [7], or broadcasts [11] increase the power to NL (more precisely, in the case of [27] the power lies between L and NL). Our result DAF = NL shows that these features can be replaced by a counting capability. Further, giving an upper bound on the number of neighbours increases the decision power to NSPACE(n), a class only reachable by standard population protocols (not on graphs) if agents have identities, or channels have memory [20, 26].

Structure of the paper. Section 2 recalls the automata models and the results of [17]. Section 3 presents fundamental limitations of their decision power. Section 4 introduces a notion of simulating an automaton by another, and uses it to show that distributed automata with more powerful communication mechanisms can be simulated by standard automata. Section 5 combines the results of Sections 3 and 4 to characterise the decision power of the models of [17] on labelling properties (middle of Figure 1). Section 6 does the same for bounded-degree networks (right of Figure 1).

Due to the nature of this research, we need to state and prove many results. To stay within the page limit, each section concentrates on the most relevant result; all others are only stated, and their proofs are given in the full version of this paper [15].

2 PRELIMINARIES

Given sets X, Y, we denote by 2^X the power set of X, and by Y^X the set of functions $X \to Y$. We define a closed interval $[m:n] := \{i \in \mathbb{Z} : m \leq i \leq n\}$ and $[n] := [0:n]$, for any $m, n \in \mathbb{Z}$ such that $m \leq n$.

A *multiset* over a set X is an element of $\mathbb{N}^X$. Given a multiset $M \in \mathbb{N}^X$ and $\beta \in \mathbb{N}$, we let $\lceil M \rceil_\beta$ denote the multiset given by $\lceil M \rceil_\beta(x) := M(x)$ if $M(x) < \beta$ and $\lceil M \rceil_\beta(x) := \beta$ otherwise. We say that $\lceil M \rceil_\beta$ is the result of *cutting off* M at β, and call the function that assigns $\lceil M \rceil_\beta$ to M the *cutoff function* for β.

Let Λ be a finite set. A *(Λ-labelled, undirected) graph* is a triple $G = (V, E, \lambda)$, where V is a finite nonempty set of *nodes*, E is a set of undirected *edges* of the form $e = \{u, v\} \subseteq V$ such that $u \neq v$, and $\lambda : V \to \Lambda$ is a *labelling*.

Convention. Throughout the paper, all graphs are labelled, have at least three nodes, and are connected.

2.1 Distributed automata

Distributed automata [17] take a graph as input, and either accept or reject it. We first define distributed machines.

Distributed machines. Let Λ be a finite set of *labels* and let $\beta \in \mathbb{N}_+$. A *(distributed) machine* with *input alphabet* Λ and *counting bound* β is a tuple $M = (Q, \delta_0, \delta, Y, N)$, where Q is a finite set of *states*, $\delta_0 : \Lambda \to Q$ is an *initialisation function*, $\delta : Q \times [\beta]^Q \to Q$ is a *transition function*, and $Y, N \subseteq Q$ are two disjoint sets of *accepting* and *rejecting* states, respectively. Intuitively, when M runs on a graph, each node v (or *agent*) with label γ is initially in state $\delta_0(\gamma)$ and uses δ to update its state, depending on the number of neighbours it has in each state; however v can only detect if it has $0, 1, ..., (\beta - 1)$, or at least β neighbours in a given state. We call β the *counting bound* of M.

Transitions given by δ are called *neighbourhood transitions*. We write $q, \mathcal{N} \mapsto q'$ for $\delta(q, \mathcal{N}) = q'$. If $q = q'$ the transition is *silent* and may not be explicitly specified in our constructions. Sometimes δ_0, Y, N are also irrelevant and not specified, and we just write $M = (Q, \delta)$.

Selections, schedules, configurations, runs, and acceptance. A *selection* of a graph $G = (V, E, \lambda)$ is a set $S \subseteq V$. A *schedule* is an infinite sequence of selections $\sigma = (S_0, S_1, S_2, ...) \in (2^V)^\omega$ such that for every $v \in V$, there exist infinitely many $t \geq 0$ such that $v \in S_t$. Intuitively, S_t is the set of nodes activated by the scheduler at time t, and schedules must activate every node infinitely often.

A *configuration* of $M = (Q, \delta_0, \delta, Y, N)$ on G is a mapping $C : V \to Q$. We let $N_v^C : Q \to [\beta]$ denote the *neighbourhood function* that assigns to each $q \in Q$ the number of neighbours of v in state q at configuration C, up to threshold β; in terms of the cutoff function, $N_v^C = \lceil M_v^C \rceil_\beta$, where $M_v^C(q) = \left|\{u : \{u, v\} \in E \wedge C(u) = q\}\right|$. The *successor configuration* of C via a selection S is the configuration $succ_\delta(C, S)$ obtained from C by letting all nodes in S evaluate δ simultaneously, and keeping the remaining nodes idle. Formally, $succ_\delta(C, S)(v) = \delta\big(C(v), N_v^C\big)$ if $v \in S$ and $succ_\delta(C, S)(v) = C(v)$ if $v \in V \setminus S$. We write $C \to C'$ if $C' = succ(C, S)$ for some selection S, and $\to^*$ for the reflexive and transitive closure of $\to$. Given a schedule $\sigma = (S_0, S_1, S_2, ...)$, the *run* of M on G scheduled by σ is the infinite sequence $(C_0, C_1, C_2, ...)$ of configurations defined inductively as follows: $C_0(v) = \delta_0(\lambda(v))$ for every node v, and $C_{t+1} = succ_\delta(C_t, S_t)$. We call C_0 the *initial configuration*. A configuration C is *accepting* if $C(v) \in Y$ for every $v \in V$, and *rejecting* if $C(v) \in N$ for every $v \in V$. A run $\rho = (C_0, C_1, C_2, ...)$ of M on G is *accepting* resp. *rejecting* if there is $t \in \mathbb{N}$ such that $C_{t'}$ is accepting resp. rejecting for every $t' \geq t$. This is called acceptance by *stable consensus* in [4].

Distributed automata. A *scheduler* is a pair $\Sigma = (s, f)$, where s is a *selection constraint* that assigns to every graph $G = (V, E, \lambda)$ a set $s(G) \subseteq 2^V$ of *permitted selections* such that every node $v \in V$ occurs in at least one selection $S \in s(G)$, and f is a *fairness constraint* that assigns to every graph G a set $f(G) \subseteq s(G)^\omega$ of *fair schedules* of G. We call the runs of a machine with schedules in $f(G)$ *fair runs* (with respect to Σ).

A *distributed automaton* is a pair $A = (M, \Sigma)$, where M is a machine and Σ is a scheduler satisfying the *consistency condition*: for every graph G, either all fair runs of M on G are accepting, or all fair runs of M on G are rejecting. Intuitively, whether M accepts or rejects G is independent of the scheduler's choices. A *accepts* G if some fair run of A on G is accepting, and *rejects* G otherwise. The language $L(A)$ of A is the set of graphs it recognises. The *property decided* by A is the predicate φ_A on graphs such that $\varphi_A(G)$ holds iff $G \in L(A)$. Two automata are equivalent if they decide the same property.

2.2 Classifying distributed automata.

Esparza and Reiter classify automata according to four criteria: detection capabilities, acceptance condition, selection, and fairness. The first two concern the distributed machine, and the last two the scheduler.

Detection. Machines with counting bound $\beta = 1$ or $\beta \geq 1$ are called *non-counting* or *counting*, respectively (abusing language, non-counting is considered a special case of counting).

Acceptance. A machine is *halting* if its transition function does not allow nodes to leave accepting or rejecting states, i.e. $\delta(q, P) = q$ for every $q \in Y \cup N$ and every $P \in [\beta]^Q$. Intuitively, a node that enters an accepting/rejecting state cannot change its mind later. Halting acceptance is a special case of acceptance by stable consensus.

Selection. A scheduler $\Sigma = (s, f)$ is *synchronous* if $s(G) = \{V\}$ for every $G = (V, E, \lambda)$ (at each step all nodes make a move); *exclusive* if $s(G) = \{\{v\} \mid v \in V\}$ (at each step exactly one node makes a move); and *liberal* if $s(G) = 2^V$ (at every step some set of nodes makes a move).

Fairness. A schedule $\sigma = (S_0, S_1, \ldots) \in s(G)^\omega$ of a graph G is *pseudo-stochastic* if for every finite sequence $(T_0, \ldots, T_n) \in s(G)^*$ there exist infinitely many $t \geq 0$ such that $(S_t, \ldots, S_{t+n}) = (T_0, \ldots, T_n)$. Loosely speaking, every possible finite sequence of selections is scheduled infinitely often. A scheduler $\Sigma = (s, f)$ is *adversarial* if for every graph G, the set $f(G)$ contains all schedules of $s(G)^\omega$ (i.e. we only require every node to be selected infinitely often), and *pseudo-stochastic* if it contains precisely the pseudo-stochastic schedules.

Whether or not a schedule σ of a graph $G = (V, E, \lambda)$ is pseudo-stochastic depends on $s(G)$. For example, if $s(G) = \{V\}$, i.e. if the only permitted selection is to select all nodes, then the synchronous schedule V^ω is pseudo-stochastic, but if $s(G) = 2^V$, i.e. if all selections are permitted, then it is not.

This classification yields 24 classes of automata (four classes of machines and six classes of schedulers). It was shown in [17] that the decision power of a class is independent of the selection type of the scheduler (liberal, exclusive, or synchronous). This leaves 8 classes, which we denote using the following scheme:

Detection	*Acceptance*	*Fairness*
d: non-counting	a: halting	f: advers. scheduling
D: counting	A: stable consensus	F: pseudo-stoch. sched.

Intuitively, the uppercase letter corresponds to the more powerful variant. Each class of automata is denoted by a string $xyz \in \{\mathsf{d}, \mathsf{D}\} \times \{\mathsf{a}, \mathsf{A}\} \times \{\mathsf{f}, \mathsf{F}\}$. Finally, it was shown in [17] that daf and daF have the same decision power, yielding the seven classes on the left of Figure 1.

In the rest of the paper, we generally assume that selection is exclusive (exactly one node is selected at each step). Since for synchronous automata there is only one permitted selection, adversarial and pseudo-stochastic scheduling coincide, and we therefore denote synchronous classes by strings $xy\$$; for example, we write DA$.

3 LIMITATIONS

Our lower bounds on the decision power of the seven classes follow from several lemmata proving limitations of their *discriminating* power, i.e. of their ability to distinguish two graphs by accepting the one and rejecting the other. We present four limitations. We state the first three, and prove the last one, a non-trivial limitation of dAF-automata. Recall that φ_A denotes the property decided by the automaton A.

Automata with halting acceptance cannot discriminate cyclic graphs. Automata with halting acceptance necessarily accept all graphs containing a cycle, or reject all graphs containing a cycle. Intuitively, given two graphs G and H with cycles, if one is accepted and the other rejected, one can construct a larger graph in which some nodes behave as if they were in G, others as if they were in H. This makes some nodes accept and others reject, contradicting that for every graph the automaton accepts or rejects.

Lemma 3.1. *Let A be a DaF-automaton. For all graphs G and H containing a cycle, $\varphi_A(G) = \varphi_A(H)$.*

Automata with adversarial selection cannot discriminate a graph and its covering. Given two graphs $G = (V_G, E_G, \lambda_G)$ and $H = (V_H, E_H, \lambda_H)$, we say that H *covers* G if there is a *covering map* $f : V_H \to V_G$, i.e. a surjection that preserves labels and neighbourhoods by mapping the neighbourhood of each v in H bijectively onto the neighbourhood of $f(v)$ in G. Automata with adversarial selection cannot discriminate a graph from another one covering it. Intuitively, if H covers G then a node u of H and the node $f(u)$ of G visit the same sequence of states in the synchronous runs of A on G and H. Since these runs are fair for adversarial selection, both nodes accept, or both reject.

Lemma 3.2. *Let A be a DAf-automaton. For all graphs G and H, if H is a covering of G, then $\varphi_A(G) = \varphi_A(H)$.*

Let $L_G : \Lambda \to \mathbb{N}$ assign to each label $\ell \in \Lambda$ the number of nodes $v \in V$ such that $\lambda(v) = \ell$. We call L_G the *label count* of G. Recall that a labelling property depends only on the label count of a graph, not on its structure. Based on the existence of a λ-fold covering graph for every G and $\lambda \in \mathbb{N}$, we immediately get the following.

Corollary 3.3. *Let A be a DAf-automaton deciding a labelling property. For all graphs G and H, if $L_H = \lambda L_G$ for some $\lambda \in \mathbb{N}_{>0}$, then $\varphi_A(G) = \varphi_A(H)$. This also holds when restricting to k-degree-bounded graphs.*

Automata with adversarial selection and non-counting automata cannot discriminate beyond a cutoff. Our final results show that for every DAf- or dAF-automaton deciding a labelling property there is a number K such that whether the automaton accepts a graph G or

not depends only on $\lceil L_G \rceil_K$, and not on the "complete" label count L_G. In such a case we say that the property admits a cutoff. For DAf-automata, the cutoff K is simply $\beta + 1$, where β is the counting bound.

Lemma 3.4. *Let A be a DAf-automaton with counting bound β that decides a labelling property. For all graphs G and H, if $\lceil L_G \rceil_{\beta+1} = \lceil L_H \rceil_{\beta+1}$ then $\varphi_A(G) = \varphi_A(H)$, i.e. φ_A admits a cutoff.*

The proof that dAF-automata also cannot discriminate beyond a cut-off is more involved, and the cutoff value K is a complex function of the automaton. The proof technique is similar to that of Theorem 39 of [6].

Lemma 3.5. *Let A be a dAF-automaton that decides a labelling property. There exists $K \geq 0$ such that for every graph G and H, if $\lceil L_G \rceil_K = \lceil L_H \rceil_K$ then $\varphi_A(G) = \varphi_A(H)$, i.e. φ_A admits a cutoff.*

Proof (sketch). Let A be a dAF-automaton, and let Q be its set of states. In this proof we consider the class of *star graphs*. A star is a graph in which a node called the *centre* is connected to an arbitrary number of nodes called the *leaves*, and no other edges exist. Importantly, for every graph G, there is a star G' with the same label count. We consider labelling properties (which do not depend on the graph), so if the property has a cutoff for star graphs, then the property has a cutoff in general. A configuration of a star graph G is completely determined by the state of the centre and the number of leaves in each state. So in the rest of the proof we assume that such a configuration is a pair $C = (C_{\text{ctr}}, C_{\text{sc}})$, where C_{ctr} denotes the state of the centre of G, and C_{sc} is the *state count* of C, i.e. the mapping that assigns to each $q \in Q$ the number $C_{\text{sc}}(q)$ of leaves of G that are in state q at C. We denote the *cutoff* of C at a number m as $\lceil C \rceil_m := (C_{\text{ctr}}, \lceil C_{\text{sc}} \rceil_m)$.

Given a configuration C of A, recall that C is rejecting if all nodes have rejecting states. We say that C is *stably rejecting* if C can only reach configurations which are rejecting. Given an initial configuration C_0, it is clear that A must reject if it can reach a stably rejecting configuration C from A. Conversely, if it cannot reach such a C, then A will not reject C_0, as there is a fair run starting at C_0 which contains infinitely many configurations that are not rejecting.

In the full version we now use Dickson's Lemma to show that there is a constant m s.t. a configuration C of A on a star is stably rejecting iff $\lceil C \rceil_m$ is. For this it is crucial that for stars stable rejection is *downwards closed* in the following sense: if such a C is stably rejecting and has at least two leaves in a state q, then the configuration C' that results from removing one of these leaves is still stably rejecting.

Now, let $C = (C_{\text{ctr}}, C_{\text{sc}})$ denote a configuration of A on a star $G = (V, E)$, and let q denote a state with $C_{\text{sc}}(q) \geq |Q|(m-1)+1$. We will show: if A rejects C then it must also reject the configuration $C' = (C'_{\text{ctr}}, C'_{\text{sc}})$ which results from adding a leaf v_{new} in state q to G, i.e. $C'_{\text{ctr}} := C_{\text{ctr}}$, $C'_{\text{sc}}(q) := C_{\text{sc}}(q) + 1$, and $C'_{\text{sc}}(r) := C_{\text{sc}}(r)$ for states $r \neq q$.

We know that A rejects C, so there is some stably rejecting configuration D reachable from C. Our goal is to construct a configuration D' reachable from C' which fulfils $\lceil D \rceil_m = \lceil D' \rceil_m$, implying that D' would also be stably rejecting. For this, let $S \subseteq V$ denote the leaves of G which are in state q in C. There are $|Q|$ states and $(m-1)|Q| + 1$ nodes in S, so by the pigeonhole principle there is a state $r \in Q$ s.t. in configuration D at least m nodes in S are in state r. Let v_{old} denote one of these nodes.

To get D', we construct a run starting from C', where v_{new} behaves exactly as v_{old}, until D' is reached. Afterwards, the nodes may diverge because of the pseudo-stochastic scheduler. However, this does not matter as D' is stably rejecting.

Let $\rho = (v_1, ..., v_\ell) \in V^*$ denote a sequence of selections for A to go from C to D. We construct the sequence $\sigma \in V^*$ by inserting a selection of v_{new} after every selection of v_{old}, and define D' as the configuration which A reaches after executing σ from C'. We claim that D' is the same as D, apart from having an additional leaf in the same state as v_{old}.

This follows from a simple induction: v_{old} and v_{new} start in the same state and see only the root node. As they are always selected subsequently, they will remain in the same state as each other. For the centre we use the property that A cannot count: it cannot differentiate between seeing just v_{old}, or seeing an additional node in the same state. We remark that G being a star is crucial for this argument, which does not extend to e.g. cliques.

To summarise, we have shown that for every rejected star G and state q with $L_G(q) \geq (m-1)|Q| + 2$ (note the centre), the input H obtained by adding a node with label q to G is still rejected. An analogous argument shows that the same holds for acceptance, and by induction we find that $K := m(|Q| - 1) + 2$ is a valid cutoff. □

Since the majority property does not admit a cutoff, in particular we obtain:

Corollary 3.6. *No DAf- or dAF-automaton can decide majority.*

4 EXTENSIONS

We introduce automata with more powerful communication mechanisms, and show that they can be simulated by standard automata with only neighbourhood transitions. We first present our notion of simulation (Definitions 4.1-4.3), and then in Sections 4.1-4.3 extend automata with weak versions of broadcast (a node sends a message to all other nodes) and absence detection (a node checks globally if there exists a node occupying a given state), and with communication by rendezvous transitions (two neighbours change state simultaneously).

Definition 4.1. Let $G = (V, E, \lambda)$ be a labelled graph and let Q, Q' denote sets of states, with $Q \subseteq Q'$. For configurations $C_1, C_2 : V \to Q'$ we define the relation $\sim_Q$ as $C_1 \sim_Q C_2$ iff $C_1(v) = C_2(v)$ for all v with $C_1(v) \in Q$ and $C_2(v) \in Q$. Let π, π' denote runs over states Q and Q', respectively. We say that π' is an *extension* of π if there exists a monotonically increasing $g : \mathbb{N} \to \mathbb{N}$ with $\pi(i) = \pi'(g(i))$ for all $i \in \mathbb{N}$, and $\pi'(j) \sim_Q \pi'(g(i))$ or $\pi'(j) \sim_Q \pi'(g(i+1))$ for all $g(i) \leq j \leq g(i+1)$.

To implement complicated transitions in an automaton without extensions, we decompose them into multiple standard neighbourhood transitions. Instead of performing, say, a broadcast atomically in one step, agents perform a sequence of neighbourhood transitions, moving into intermediate states in the process. As mentioned in Section 2, the results of [17] allow us to use liberal or exclusive selection without changing the decision power. We assume that selection is exclusive, unless stated otherwise.

Definition 4.2. Let $G = (V, E, \lambda)$ be a labelled graph. Let π, π' denote runs of an automaton induced by schedules $s, s' \in V^\omega$, respectively. Let I, I' denote the set of indices where π or π', respectively, execute non-silent transitions, i.e. $I := \{i \colon \pi_i \neq \pi_{i+1}\}$. We say that π' is a *reordering* of π if there exists a bijection $f \colon I \to I'$ s.t. $s(i) = s'(f(i))$ for all $i \in \mathbb{N}$, and $f(i) < f(j)$ for all $i < j$ where the nodes $s(i)$ and $s(j)$ are adjacent or identical. If that is the case, we also write $\pi_f := \pi'$ for the reordering induced by f.

While an extension of a run can execute a single complicated transition in many steps instead of atomically, steps of different transitions, or of different phases of a transition, should not "interfere". Ideally all the neighbourhood transitions simulating, say, a broadcast, should be executed before any of the transitions simulating the next one. However, in distributed automata this cannot be guaranteed. This is where we make use of reorderings: We will guarantee that every run can be reordered into an equivalent run in which transitions do not "interfere". We will only allow reordering of nodes that are not adjacent, thus ensuring that the reordered run yields the same answer as the original one.

Lastly, we now introduce a generic model encompassing all of our extended automata, which allows us to define our notion of simulation for all extensions simultaneously.

Definition 4.3. We say that $P = (Q, \mathrm{Run}, \delta_0, Y, N)$ is a *generalised graph protocol*, where Q are states, δ_0, Y, N are initialisation function, accepting states and rejecting states, respectively, and Run is a function mapping every labelled graph $G = (V, E, \lambda)$ over a given alphabet Λ to a subset $\mathrm{Run}(G) \subseteq (Q^V)^\omega$ of fair runs. We define accepting/rejecting runs and the statement "P decides a predicate φ" analogously to distributed automata. Further, let P' be an automaton with states $Q' \supseteq Q$. We say that P' *simulates* P, if for every fair run π' of P' there is a reordering π'_f of π' and a fair run $\pi \in \mathrm{Run}$ of P, s.t. π'_f is an extension of π. If P' simulates P, we refer to the states in $Q' \setminus Q$ as *intermediate* states.

We will apply this general definition to simulate broadcast, absence-detection, and rendezvous transitions by automata with only neighbourhood transitions. In the full version [15] we show that if π' is a reordering of π and v is the node satisfying $s(i) = v = s'(f(i))$, then the neighbourhood of v in π at time i and the neighbourhood of v in π' at time $f(i)$ coincide. Furthermore, we show that an automaton P' that simulates P can be easily transformed into an automaton P'' that also simulates P *and* is equivalent to P, i.e., decides the same property as P.

Lemma 4.4. *Let $P = (Q, \mathrm{Run}, \delta_0, Y, N)$ denote a generalised graph protocol deciding a predicate φ, and P' an automaton simulating P. Then there is an automaton P'' simulating P which also decides φ.*

The automaton P'' constructed in the proof of this lemma is basically P', except that nodes remember the last state $q \in Q$ they visited, in addition to their current state $q' \in Q'$. This allows us to define the accepting/rejecting states of Q'' as the pairs $(q', q) \in Q' \times Q$ such that q is an accepting/rejecting state of P.

Notation. Because of Lemma 4.4, in simulation proofs we often leave out the accepting and rejecting states of generalised graph protocols and automata.

4.1 Weak Broadcasts

Intuitively, a broadcast transition $q \mapsto r, f$ models that an agent in state q, called the *initiating* agent or *initiator*, sends a signal to the other agents, and moves to state r; the other agents react to the signal by moving to new states, determined by their current state and by f, a mapping from states to states. Broadcasts are weak, meaning that multiple broadcasts can occur at the same time. When this happens, all initiators send their signals and move to their new states, and for every other agent the scheduler decides which signal it receives and reacts to. It is only guaranteed that every non-initiator receives exactly one signal, and that this signal has been sent.

Definition 4.5. A *distributed machine with weak broadcasts* is defined as a tuple $M = (Q, \delta_0, \delta, Q_B, B, Y, N)$, where $(Q, \delta_0, \delta, Y, N)$ is a distributed machine, $Q_B \subseteq Q$ is a set of *broadcast-initiating* states, and $B \colon Q_B \to Q \times Q^Q$ describes a set of *weak broadcast transitions*, one for each state of Q_B. In particular, B maps a state q to a pair (q', f), where q' is a state and $f \colon Q \to Q$ is a *response function*. We write broadcast transitions as $q \mapsto r, f$, where f is usually given as a set $\{r \mapsto f(r) : r \in Q\}$. (Mappings $r \mapsto r$, and silent transitions $q \mapsto q$, id, id being the identity function, may be omitted.) Given a configuration C and a selection $S \subseteq V$ of initiators such that $C(v) \in Q_B$ for every $v \in S$, the machine can move to any configuration C' satisfying the following conditions:

- If $v \in S$, then $C'(v) = q'$, where q' is the state such that $B(C(v)) = (q', f)$.
- If $v \notin S$, then $C'(v) = f(C(v))$, where $B(C(u)) = (q', f)$ for some $u \in S$, i.e., f is the response function of an initiator u.

A valid selection is a nonempty independent set of nodes of V. The set of valid selections is denoted $\mathcal{I}$. A *schedule* of M is a sequence $\sigma \in (\{n, b\} \times \mathcal{I})^\omega$; intuitively, $\sigma(i) = (n, S)$ means that at time i the scheduler asks the agents of S to perform a neighbourhood transition, and $\sigma(i) = (b, S)$ that it asks the agents of S to initiate weak broadcasts. Given a schedule σ, we generate a *run* $\pi = (C_0, C_1, \ldots)$ as follows. For each step $i \geq 1$ either $\sigma(i) = (n, S)$ for $S \subseteq V$ and we execute a neighbourhood transition for $S' := S \setminus C_i^{-1}(Q_B)$, or $\sigma(i) = (b, S)$ for $S \subseteq V$ and we execute a weak broadcast transition on $S' := S \cap C_i^{-1}(Q_B)$. In either case, if S' is empty we set $C_{i+1} := C_i$.

A schedule σ is *adversarial* if there are infinitely many i with $\sigma(i) = (b, S)$ for some S, or for all $v \in V$ there are infinitely many i with $\sigma(i) = (n, S)$ and $v \in S$. It is *pseudo-stochastic*, if every finite sequence of selections $w \in (\{n, b\} \times \mathcal{I})^*$ appears infinitely often in σ. Given $xyz \in \{\mathsf{d}, \mathsf{D}\} \times \{\mathsf{a}, \mathsf{A}\} \times \{\mathsf{f}, \mathsf{F}\}$, an *xyz-automaton with weak broadcasts* is a tuple (M, Σ) defined as for an xyz-automaton, except that M is a distributed machine with weak broadcasts. In particular, we extend the definitions of fair runs, consensuses, and acceptance to automata with weak broadcasts.

A *strong broadcast protocol* is a tuple $P = (Q, \delta_0, B, Y, N)$ that is defined analogously to a dAF-automaton with weak broadcasts $(Q, \delta_0, \emptyset, Q, B, Y, N)$, except that the set of valid selections is $\mathcal{I} := \{\{v\} : v \in V\}$. In other words, only one agent can broadcast at a given time. This model corresponds to the broadcast consensus protocols of [11].[1]

[1] The protocols of [11] also contain rendez-vous transitions, but they can be removed without affecting expressive power.

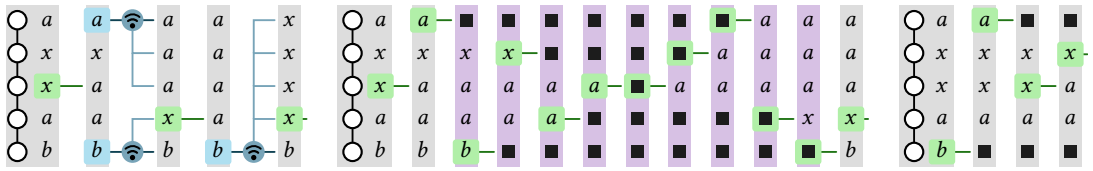


Figure 2: (a) Prefix of a run of the automaton of Example 4.6. on a line with five nodes. (b) An extension of the same run, where ■ denotes intermediate states; only the first 12 steps are shown. (c) A reordering of the run of (b); only four steps are shown.

Additionally, to simplify our proofs we assume that all selections (n, S) satisfy $|S| = 1$, i.e. at each step the scheduler selects one single agent to execute a neighbourhood transition. Observe that we can assume $|S| = 1$ without loss of generality. Indeed, since S is an independent set by definition, it only contains non-adjacent nodes, and so after the agents of S execute a neighbourhood transition, be it simultaneously or sequentially, they reach the same states.

Example 4.6. Consider a dAF-automaton P with states $\{a, b, x\}$, a neighbourhood transition $x, N \mapsto a$ for every neighbourhood $N\colon Q \to [1]$ with $N(a) > 0$ (i.e. an agent moves from x to a if it has at least one neighbour in a), and weak-broadcast transitions

$$a \mapsto a, \{x \mapsto a\} \qquad \text{and} \qquad b \mapsto b, \{b \mapsto a, a \mapsto x\}\ .$$

Figure 2 shows sample runs of P on the graph consisting of a line with five nodes. Note that the simultaneous broadcasts at both ends of the line are executed simultaneously, and are received by three and two nodes, respectively. However, the next (and last) broadcast, which is initiated by the bottom node, reaches all nodes. The reordering depicted in (c) shows the interleaving of two different transitions: while the two ends have already initiated broadcasts, the information has not reached the middle node, and it can execute a neighbourhood transition.

Of course, our model of weak broadcasts would be of limited use if we were not able to simulate it. For this we use a construction similar to the three-phase protocol of Awerbuch's alpha-synchroniser [8]. Instead of simply using it to synchronise, we will propagate additional information, allowing the agents to perform the local update necessary to execute the broadcast.

Lemma 4.7. *Every automaton with weak broadcasts is simulated by some automaton of the same class without weak broadcasts.*

Proof (sketch). Let $P = (Q, \delta_0, \delta, Q_B, B)$ denote an automaton with weak broadcasts. We define an automaton $P' = (Q', \delta_0', \delta')$ simulating P. The automaton P' has three phases, called 0,1, and 2. A node moves to the next phase (modulo 3) only if every neighbour is in the same phase or in the next. The states of P' are $Q' := Q \cup Q \times \{1, 2\} \times Q^Q$. Intuitively, an agent of P' in state $q \in Q$ is in phase 0, and simulates an agent of P in state q; an agent of P' in state $(q, i, f) \in Q \times \{1, 2\} \times Q^Q$ is in phase i, and simulates an agent executing P in state q, and initiating or responding to a broadcast with response function f.

Let β denote the counting bound of P. To specify the transitions, for a neighbourhood $N : Q' \to [\beta]$ we write $N[i] := \sum_{q,f} N((q, i, f))$ for $i \in \{1, 2\}$ and $N[0] := \sum_{q \in Q} N(q)$ to denote the number of adjacent agents in a particular phase, and choose a function $g(N) \in Q^Q \cup \{\square\}$ s.t. $g(N) = f \neq \square$ implies $N((q, 1, f)) > 0$, and $g(N) = \square$ implies $N[1] = 0$. The function g is used to select which broadcast to execute, if there are multiple possibilities. We define the following transitions for δ', for all states $q \in Q$ and neighbourhoods $N : Q \to [\beta]$.

$$q, N \mapsto \delta(q, N) \qquad \text{if } q \notin Q_B \text{ and } N[0] = |N| \tag{1}$$

$$q, N \mapsto (q', 1, f) \qquad \text{if } q \in Q_B \text{ and } N[0] = |N|, \text{ with } (q', f) := B(q) \tag{2}$$

$$q, N \mapsto (f(q), 1, f) \qquad \text{if } g(N) = f \neq \square \tag{3}$$

$$(q, 1, f), N \mapsto (q, 2, f) \qquad \text{if } N[0] = 0 \tag{4}$$

$$(q, 2, f), N \mapsto q \qquad \text{if } N[1] = 0 \tag{5}$$

If all neighbours are in phase 0, the agent either executes a neighbourhood transition via (1) or it initiates the broadcast in (2), depending on the state of the agent. For the latter, the agent immediately performs the local update. Once there is a phase 1 neighbour, the agent instead executes the broadcast of one of its neighbours via (3) (if there are multiple, g is used to select one). Note that (2) and (3) are indeed well-defined, as $N[0] = |N|$ holds iff $g(N) = \square$. Finally, transitions (4) and (5) move agents to the next phase, once all of their neighbours are in the same or the next phase. □

4.2 Weak Absence Detection

Absence detection, introduced in [27], enables agents to determine the *support* of the current configuration, defined as the set of states currently populated by at least one agent. More precisely, an agent that executes an absence-detection transition moves to a new state that depends on the current support. We consider a weaker mechanism where, as for weak broadcasts, multiple absence-detection transitions may occur at the same time. In this case, each agent executing an absence-detection transition moves according to the support of a *subset* of the agents. However, it is ensured that every agent belongs to at least one of these subsets.

While it is possible to define and implement a more general model involving absence-detection, we limit ourselves to a special case to simplify our proofs. In particular, we define a model in which scheduling is synchronous. Further, we implement a simulation only for graphs of bounded degree.

Definition 4.8. A *distributed machine with weak absence-detection* is defined as a tuple $(Q, \delta_0, \delta, Q_A, A, Y, N)$, where $(Q, \delta_0, \delta, Y, N)$ is a distributed machine, Q_A is a set of *initiating* states or *initiators*, and $A\colon Q_A \times 2^Q \to Q$ a set of *(weak) absence-detection transitions*. Given a configuration C, a selection $S \subseteq V$ of initiators such that $C(v) \in Q_A$ for every $v \in S$, and a set $S_v \subseteq V$ for every $v \in S$

satisfying $v \in S_v$ and $\bigcup_{v \in S} S_v = V$, the machine can move to any configuration C' with $C'(v) := A(v, C(S_v))$ for $v \in S$ and $C'(v) := C(v)$ for $v \notin S$. (Notice that the S_v need not be pairwise disjoint.) We write $q, S \mapsto q'$ to denote that $A(q, S) = q'$ for $q \in Q_A$, $q' \in Q$ and $S \subseteq Q$.

We use the synchronous scheduler, so the only valid selection is V. A step at a configuration C is performed by having each agent execute a neighbourhood transition simultaneously, moving to C', followed by an absence-detection with $S := C^{-1}(S_A)$ as set of initiators, to go from C' to C''. If S is empty, the computation hangs, and we instead set $C'' := C$. A DA$-*automaton with (weak) absence-detection* is defined analogously to a DA$-automaton.

As for broadcasts, absence detection is implemented using a three phase protocol. To allow the information to propagate back, we use a distance labelling that effectively embeds a rooted tree for each initiating agent.

Lemma 4.9. *Every* DA$*-automaton with weak absence detection is simulated by some* DAf*-automaton, when restricted to bounded-degree graphs.*

4.3 Rendez-vous transitions

In rendez-vous transitions two neighbours interact and move to new states according to a joint transition function. They are the communication mechanism of population protocols [4]. In fact, population protocols on graphs have also been studied previously [3], and we use exactly the same model.

A rendez-vous transition $p, q \mapsto p', q'$ allows two neighbouring nodes u and v in states p and q to interact and change their states to p' and q', respectively. Like neighbourhood transitions, rendez-vous transitions are local, i.e., they only involve adjacent nodes. They are useful to model transactions such as transferring a token from one node to another. A population protocol on graphs, or graph population protocol, is a pair (Q, δ) where q is a set of states and $\delta \colon Q^2 \to Q^2$ is a set of rendez-vous transitions, and $p, q \mapsto p', q'$ denotes $\delta(p, q) = (p', q')$. The formal definition can be found in the full version [15].

Lemma 4.10. *Every graph population protocol is simulated by some* DAF*-automaton.*

5 UNRESTRICTED COMMUNICATION GRAPHS

We prove the characterisation of the decision power of the different classes as presented in the introduction. The classes are defined as follows. For a labelling property $\varphi \colon \mathbb{N}^\Lambda \to \{0, 1\}$ we have

- $\varphi \in$ Trivial iff φ is either always true or always false,
- $\varphi \in$ Cutoff(1) iff $\varphi(L) = \varphi(\lceil L \rceil_1)$ for all multisets $L \in \mathbb{N}^\Lambda$,
- $\varphi \in$ Cutoff iff there exists a $K \in \mathbb{N}$ s.t. $\varphi(L) = \varphi(\lceil L \rceil_K)$ for all multisets $L \in \mathbb{N}^\Lambda$, and
- $\varphi \in$ NL iff φ is decidable by a non-deterministic Turing-Machine using logarithmic space.

The proof proceeds in the following steps:

(1) DaF and therefore all automata-classes with weak acceptance have an upper bound of Trivial and thus decide exactly Trivial. This proof also works when restricted to degree-bounded graphs.
(2) DAf and therefore also dAf can decide at most Cutoff(1).
(3) dAf and therefore also DAf can decide at least Cutoff(1).
(4) dAF can decide exactly Cutoff.
(5) DAF can decide exactly the labelling propertis in NL.

In this section we sketch the hardest proof, the characterisation for DAF. All other proofs can be found in the full version [15]. We start with some conventions and notations.

Conventions and notations. When describing automata of a given class (possibly with weak broadcasts or weak absence detection) we specify only the machine; the scheduler is given implicitly by the fairness condition and selection criteria of the class. Further, when the initialisation function and the accepting/rejecting states are straightforward, which is usually the case, we only describe the sets of states and transitions. So, for example, we speak of the automaton (Q, δ), or the automaton with weak broadcasts (Q, δ, Q_B, B). We even write $(Q, \delta) + B$; in this case Q_B is implicitly given as the states of B initiating non-silent broadcasts, i.e. $Q_B := \{q : B(q) \neq (q, \mathrm{id})\}$.

Given an automaton P (possibly with weak broadcasts or absence detection) with set of states Q and a set Q', we let $P \times Q'$ denote the automaton with set of states $Q \times Q'$ whose transitions leave the Q' component of the state untouched. In other words, if a transition makes a node move from state (q, q') to state (p, p'), then $q' = p'$. The definition of the transitions is straightforward, and we omit it.

We often combine the two notations above. Given an automaton P', we write for example $P = P' \times Q' + B$ to denote the automaton with weak broadcasts obtained by first constructing $P' \times Q'$, and then adding the set B of weak broadcast transitions.

Lemma 5.1. DAF*-automata decide exactly the labelling properties in* NL.

Proof (sketch). First, we argue why DAF-automata can decide only labelling properties in NL. Let P be a DAF-automaton deciding a labelling property φ. We exhibit a log-space Turing machine that given a labelled graph $G = (V, E, \lambda)$ decides whether P accepts G. Since φ is a labelling property, $\varphi(G) = \varphi(\hat{G})$ for the unique clique $\hat{G}$ with set of nodes V and labelling λ. The Turing machine therefore ignores G and simulates M on $\hat{G}$. A configuration of $\hat{G}$ is completely characterized up to isomorphism by the *number* of agents in each state; in particular, it can be stored using logarithmic space. In [11, Proposition 4] it is shown that any class of automata whose configurations have this property, and whose step relation is in NL (i.e., there is a log-space Turing machine that on input (C, C') decides if the automaton can move from C to C'), can only decide properties in NL. Since the step relation of DAF-automata on cliques is certainly in NL, the result follows.

Now we show the other direction. It is known that strong broadcast protocols decide exactly the predicates in NL [11, Theorem 15]. Therefore, it suffices to show that for every strong broadcast protocol there is an equivalent DAF-automaton. By Lemma 4.7 DAF-automata can simulate weak broadcasts, and so, loosely speaking, the task is to simulate strong broadcasts with weak ones.

Let $P = (Q, \delta, I, O)$ be a strong broadcast protocol. We start with a graph population protocol $P_{\text{token}} := (Q_{\text{token}}, \delta_{\text{token}})$, with states $Q_{\text{token}} := \{0, L, L', \bot\}$ and rendez-vous transitions δ_{token} given by

$$(L, L) \mapsto (0, \bot), \quad (0, L) \mapsto (L, 0), \quad (L, 0) \mapsto (L', 0) \quad \langle \text{token} \rangle$$

Now we construct a DAF-automaton $P'_{\text{token}} = (Q'_{\text{token}}, \delta'_{\text{token}})$ simulating P_{token} using Lemma 4.10, and combine it with P by setting $P_{\text{step}} := P'_{\text{token}} \times Q + \langle \text{step} \rangle$, where $\langle \text{step} \rangle$ is a weak broadcast defined as

$$(L', q) \mapsto (L, q'), \begin{Bmatrix} (t, r) \mapsto (t, f(r)) \\ \text{for } (t, r) \in Q'_{\text{token}} \times Q \end{Bmatrix} \qquad \langle \text{step} \rangle$$

for each broadcast $q \mapsto q', f$ in δ. Finally, let $P'_{\text{step}} = (Q'_{\text{step}}, \delta'_{\text{step}})$ be a DAF-automaton simulating P_{step}, which exists by Lemma 4.7.

Intuitively, agents in states L, L' have a *token*. If we could ensure that initially there is only one token in L, L', then we would be done. Indeed, in this case at each moment only the agent with the token can move; if in L, it initiates a (simulated) rendez-vous transition, and if in L', a weak broadcast. Since no other agent is executing a weak broadcast at the same time, the weak broadcast is received by all agents, and has the same effect as a strong broadcast.

We cannot ensure that initially there is only one token, but if the computation starts with more than one, then two tokens eventually meet using transition $\langle \text{token} \rangle$ and an agent moves into the *error state* $\perp$. We design a mechanism to restart the computation after this occurs, now with fewer agents in state $(L, \cdot)$, guaranteeing that eventually the computation is restarted with only one token. For this we again add an additional component to each state and consider the protocol $P_{\text{reset}} := P'_{\text{step}} \times Q + \langle \text{reset} \rangle$, where $\langle \text{reset} \rangle$ are the following broadcast transitions, for each $q, q_0 \in Q$.

$$((\perp, q), q_0) \mapsto ((L, q_0), q_0), \begin{Bmatrix} (r, r_0) \mapsto ((0, r_0), r_0) \\ \text{for } r \in Q'_{\text{step}}, r_0 \in Q \end{Bmatrix} \qquad \langle \text{reset} \rangle$$

For P_{reset} we define the input mapping $I_{\text{reset}}(x) := ((L, I(x)), I(x))$ and the set of accepting states $O_{\text{reset}} := \{((r, q), q_0) : q \in O, q_0 \in Q, r \in \{0, L\}\}$. Using Lemma 4.7 (and Lemma 4.4) we get a DAF-Automaton equivalent to P_{reset}, so it suffices to show that P_{reset} is equivalent to P.

In the full version [15], we show that a run of P_{reset} starting with more than one token will eventually reset and restart the computation with strictly fewer tokens, until only one token is left. After this moment, $\langle \text{reset} \rangle$ is never executed again, and so we are left with a run of P_{step}, which stabilises to a correct consensus. □

6 BOUNDED-DEGREE COMMUNICATION GRAPHS

We characterise the decision power of the models when the degree of the input graphs is at most k for some constant $k \in \mathbb{N}$. Many results for the unrestricted set of graphs continue to hold, in particular Corollary 3.3, proving that DAf-automata can only compute properties invariant under scalar multiplication (called ISM in Figure 1), as well as the result that automata with halting acceptance can only decide trivial properties. The new results are:

(1) For every $k \geq 3$ the expressive power of dAf is precisely Cutoff(1).
(2) DAF- and dAF-automata decide exactly the labelling properties in NSPACE(n).
(3) DAf can decide all homogeneous threshold predicates, in particular majority. This is a proper subset of ISM (the latter contains e.g. the divisibility predicate $\varphi(x, y) \Leftrightarrow x | y$), so there is a gap between our upper and lower bounds for DAf.

We describe the DAf-automata for homogeneous threshold predicates. All other proofs can be found in the full version [15].

6.1 DAf decides all homogeneous threshold predicates on bounded-degree graphs

Let $\varphi : \mathbb{N}^l \to \{0, 1\}$, $\varphi(x_1, ..., x_l) \Leftrightarrow a_1 x_1 + ... + a_l x_l \geq 0$ denote an arbitrary homogeneous threshold predicate, with $a_1, ..., a_l \in \mathbb{Z}$, and let k denote the maximum degree of the communication graph.

Local Cancellation. We first define a protocol that performs local updates. Each agent stores a (possibly negative) integer contribution. If the absolute value of the contribution is large, then the agent will try to distribute the value among its neighbours. In particular, if a node v has contribution x with $x > k$, then it will "send" one unit to each of its neighbours with contribution $y \leq k$. Those neighbours increment their contribution by 1, while v decrements its contribution accordingly. (This happens analogously for $x < -k$, where -1 units are sent.) Agents may receive multiple updates in a single step, or may simultaneously send and receive updates.

We define a DA$-automaton with weak absence detection $P_{\text{cancel}} := (Q_{\text{cancel}}, \delta_{\text{cancel}}, \emptyset, \emptyset)$, but use only neighbourhood transitions for the moment. We use states $Q_{\text{cancel}} := \{-E, ..., E\}$. Here $E := \max\{|a_1|, ..., |a_l|\} \cup \{2k\}$ is the maximum contribution an agent must be able to store: any agent with contribution x s.t. $|x| \leq k$ may receive an increment or decrement from up to k neighbours, so $E \geq k + k$. The transitions δ_{cancel} are

$$\begin{array}{lll} x, N \mapsto & x - N[-E, -k{-}1] + N[k{+}1, E] & \text{for } x = -k, ..., k \\ x, N \mapsto & x - N[-E, k] & \text{for } x = k{+}1, ..., E \\ x, N \mapsto & x + N[-k, E] & \text{for } x = -E, ..., -k{-}1 \end{array} \qquad \langle \text{cancel} \rangle$$

Here we write $N[a, b] := \sum_{i=a}^{b} N(i)$ for the total number of adjacent agents with contribution in the interval $[a : b]$. As we use the synchronous scheduler, at each step all agents make a move. It is thus easy to see that $\langle \text{cancel} \rangle$ preserves the sum of all contributions $\sum_v C(v)$ for a configuration C, and that it does not increase $\sum_v |C(v)|$.

We can now show that the above protocol converges in the following sense:

Lemma 6.1. *Let $\pi = (C_0, C_1, ...)$ denote a run of P_{cancel} with $\sum_v C_0(v) < 0$. Then there exists $i \geq 0$ such that either all configurations $C_i, C_{i+1}, ...$ only have states in $\{-E, ..., -1\}$, or they only have states $\{-k, ..., k\}$.*

Convergence and Failure Detection. The overall protocol waits until P_{cancel} converges, i.e. either all agents have "small" contributions, or all contributions are negative. In the latter case, we can safely reject the input, as the total sum of contributions is negative. In the former case we perform a broadcast, doubling all contributions. As we only double once all contributions are small, each agent can always store the new value. This idea of alternating cancelling and doubling phases has been used extensively in the population protocol literature [5, 9, 10, 22].

To detect whether P_{cancel} has already converged, and to perform the doubling, we elect a subset of agents as leaders. A "true" leader election, with only one leader at the end, is impossible due to weak fairness, but we can elect a "good enough" set of leaders: whenever two leaders disagree, we can eliminate one of them and restart the computation with a non-empty, proper subset of the original set of leaders.

We use weak absence-detection transitions to determine whether P_{cancel} has converged. Set $Q_L := \{0, L, L_{\text{double}}, L_\square\}$ and let $(Q, \delta) := P_{\text{cancel}} \times Q_L$. (Recall the notation from Section 5.) We define $P_{\text{detect}} := (Q \cup \{\bot, \square\}, \delta, Q_{\text{cancel}} \times \{L\}, A)$, where A are the following absence-detection transitions, for $x \in Q_{\text{cancel}}, s \subseteq Q \cup \{\bot, \square\}$.

$$\begin{aligned}
&(x, L), s \mapsto \bot && \text{if } \square \in s \\
&(x, L), s \mapsto (x, L_{\text{double}}) && \text{if } s \subseteq \{-k, ..., k\} \times \{0\} \\
&(x, L), s \mapsto (x, 0) && \text{if } \bot \in s \\
&(x, L), s \mapsto (x, L_\square) && \text{if } s \subseteq \{-E, ..., -1\} \times \{0\}
\end{aligned} \qquad \langle \text{detect} \rangle$$

Intuitively, $\bot$ and $Q_{\text{cancel}} \times \{L, L_{\text{double}}, L_\square\}$ are leader states, and $\square$ is the (only) rejecting state. State $\bot$ is an error state: an agent in that state will eventually restart the computation. Via Lemma 4.9 we get a DAf-automaton $P'_{\text{detect}} = (Q'_{\text{detect}}, \delta'_{\text{detect}})$ simulating P_{detect}.

We want our broadcasts to interrupt any (simulated) absence-detection transitions of P'_{detect}, by moving agents in intermediate states $Q'_{\text{detect}} \setminus Q_{\text{detect}}$ to their last "good" state in Q_{detect}. To this end, we introduce the mapping $\text{last} : Q'_{\text{detect}} \to Q_{\text{detect}}$, which fulfils $\text{last}(C_i(v)) \in \{\text{last}(C_{i-1}(v)), C_i(v)\}$ for all runs $\pi = C_0 C_1 ...$ of P'_{detect} and $i > 0$, where C_0 has only states of Q_{detect}. It is, of course, not true that last exists for *any* simulation P'_{detect} of P_{detect}. However, one can extend any simulation which does not, by having each agent "remember" its last state in Q_{detect}.

We construct a DAf-automaton with weak broadcasts P_{bc} by adding the following transitions to P'_{detect}.

$$\begin{aligned}
&(x, L_{\text{double}}) \mapsto (2x, L), \big(\\
&\quad \{(y, 0) \mapsto (2y, 0) : y \in \{-k+1, ..., k-1\}\} \\
&\quad \cup \{q \mapsto \bot : q \in Q_{\text{cancel}} \times \{L, L_{\text{double}}, L_\square\}\} \\
&\big) \circ \text{last}
\end{aligned} \qquad \langle \text{double} \rangle$$

$$\begin{aligned}
&(x, L_\square) \mapsto \square, \big(\\
&\quad \{(y, 0) \mapsto \square : y \in \{-E, ..., -1\}\} \\
&\quad \cup \{q \mapsto \bot : q \in Q_{\text{cancel}} \times \{L, L_{\text{double}}, L_\square\}\} \\
&\big) \circ \text{last}
\end{aligned} \qquad \langle \text{reject} \rangle$$

These transitions are written somewhat unintuitively. Recall that we write a weak broadcast transition as $q \mapsto q', f$, where $q, q' \in Q'_{\text{detect}}$ are states and $f : Q'_{\text{detect}} \to Q'_{\text{detect}}$ is the transfer function. Usually, we specify f as simply a set of mappings $\{r \mapsto f(r) : r \in Q'_{\text{detect}}\}$. Here, our transition essentially is $q \mapsto q'$, $(f \circ \text{last})$, where $\circ$ denotes function composition, and f is given as a set of mappings. This means that broadcasts first move all agents to their last state in Q_{detect}, and then apply the other mappings as specified.

Before extending P_{bc} with resets that restart the computation from an error state, we analyse the behaviour of P_{bc} in more detail. To talk about accepting/rejecting runs, we define the set of rejecting states as $\{\square\}$. (All other states are accepting.) Let $\pi := (C_0, C_1, ...)$ denote a fair run of P_{bc} starting in a configuration C_0 where all agents are in states $\{-E, ..., E\} \times \{0, L\}$, and at least one agent is in a state $(\cdot, L)$. We refer to the agents starting in $(\cdot, L)$ as *leaders*. Note that it is not possible to enter a state in $Q \times \{L, L_{\text{double}}, L_\square\} \cup \{\bot\}$ without being a leader. We usually disregard the first component (if any) while referring to states of leaders.

To argue correctness, we state two properties of P_{bc}. First, it is not possible for *all* leaders to enter $\bot$, which ensures that a reset restarts the computation with a proper subset of the leaders. Second, P_{bc} works correctly if no agent enters an error state. Here, $L_G : X \to \mathbb{N}$ denotes the label count of the input graph, i.e. $L_G(x_i) = |C_0^{-1}(a_i)|$ for $i = 1, ..., l$.

Lemma 6.2. *Assuming that no agent enters state $\bot$, π is accepting iff $\varphi(L_G) = 1$. Additionally, π cannot reach a configuration with all leaders in state $\bot$.*

Resets. Finally, we can add resets to the protocol, to restart the computation in case of errors. We use Lemma 4.7 to construct a DAf-computation $P'_{\text{bc}} = (Q'_{\text{bc}}, \delta'_{\text{bc}})$ simulating P_{bc}, and then set $P_{\text{reset}} := P'_{\text{bc}} \times Q_{\text{cancel}} + \langle \text{reset} \rangle$, where the broadcasts are defined as follows, for $q_0 \in Q_{\text{cancel}}$.

$$(\bot, q_0) \mapsto ((q_0, L), q_0), \left\{ \begin{array}{l} (r, r_0) \mapsto ((r_0, 0), r_0) \\ \text{for } (r, r_0) \in Q'_{\text{bc}} \times Q_{\text{cancel}} \end{array} \right\} \qquad \langle \text{reset} \rangle$$

To actually compute φ, we add the initialisation function $I(x_i) := ((a_i, L), a_i)$ and the set of rejecting states $N := \{\square\}$ to P_{reset} (all other states are accepting).

Proposition 6.3. *For every predicate $\varphi : \mathbb{N}^l \to \{0, 1\}$ such that $\varphi(x_1, ..., x_l) \Leftrightarrow a_1 x_1 + ... + a_l x_l \geq 0$ with $a_1, ..., a_l \in \mathbb{Z}$ there is a bounded-degree DAf-automaton computing φ.*

7 CONCLUSION

We have characterised the decision power of the weak models of computation studied in [17] for properties depending only on the labelling of the graph, not on its structure. For arbitrary networks,the initially twenty-four classes of automata collapse into only four; further, only DAF can decide majority. For bounded-degree networks (a well-motivated restriction in a biological setting, also used in e.g. in [3, 12]), the picture becomes more complex. Counting and non-counting automata become equally powerful, an interesting fact because biological models are often non-counting. Further, the class DAf, which uses adversarial scheduling, substantially increases its power, and becomes able to decide majority. So, while majority algorithms require (pseudo-)random scheduling to work correctly for arbitrary networks, they can work correctly under adversarial scheduling for bounded-degree networks. In particular, there exist a synchronous deterministic algorithm for majority in bounded-degree networks.

ACKNOWLEDGMENTS

We thank the anonymous referees for helpful comments.

This work is partly funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement PaVeS (No 787367), and by the German Research Foundation (DFG) project Continuous Verification of Cyber-Physical Systems (GRK 2428).

REFERENCES

[1] Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haeupler, and Fabian Kuhn. 2013. Beeping a maximal independent set. *Distributed Comput.* 26, 4 (2013), 195–208.

[2] Dana Angluin. 1980. Local and Global Properties in Networks of Processors (Extended Abstract). In *STOC*. ACM, 82–93.

[3] Dana Angluin, James Aspnes, Melody Chan, Michael J Fischer, Hong Jiang, and René Peralta. 2005. Stably computable properties of network graphs. In *International Conference on Distributed Computing in Sensor Systems*. Springer, 63–74.

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

[5] Dana Angluin, James Aspnes, and David Eisenstat. 2008. Fast computation by population protocols with a leader. *Distributed Comput.* 21, 3 (2008), 183–199.

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

[7] James Aspnes. 2017. Clocked Population Protocols. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*. 431–440.

[8] Baruch Awerbuch. 1985. Complexity of Network Synchronization. *J. ACM* 32, 4 (1985), 804–823. https://doi.org/10.1145/4221.4227

[9] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. 2018. A Population Protocol for Exact Majority with O(log5/3 n) Stabilization Time and Theta(log n) States. In *DISC (LIPIcs, Vol. 121)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 10:1–10:18.

[10] Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. 2017. Brief Announcement: Population Protocols for Leader Election and Exact Majority with $O(\log^2 n)$ States and $O(\log^2 n)$ Convergence Time. In *PODC*. ACM, 451–453.

[11] Michael Blondin, Javier Esparza, and Stefan Jaax. 2019. Expressive Power of Broadcast Consensus Protocols. In *CONCUR (LIPIcs, Vol. 140)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 31:1–31:16.

[12] Olivier Bournez and Jonas Lefèvre. 2013. Population Protocols on Graphs: A Hierarchy. In *UCNC (Lecture Notes in Computer Science, Vol. 7956)*. Springer, 31–42.

[13] Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, and Paul G. Spirakis. 2013. The computational power of simple protocols for self-awareness on graphs. *Theor. Comput. Sci.* 512 (2013), 98–118.

[14] Alejandro Cornejo and Fabian Kuhn. 2010. Deploying Wireless Networks with Beeps. In *DISC (Lecture Notes in Computer Science, Vol. 6343)*. Springer, 148–162.

[15] Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. 2021. Decision Power of Weak Asynchronous Models of Distributed Computing. *CoRR* abs/2102.11630 (2021). arXiv:2102.11630 https://arxiv.org/abs/2102.11630

[16] Yuval Emek and Roger Wattenhofer. 2013. Stone age distributed computing. In *PODC*. ACM, 137–146.

[17] Javier Esparza and Fabian Reiter. 2020. A Classification of Weak Asynchronous Models of Distributed Computing. In *CONCUR (LIPIcs, Vol. 171)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 10:1–10:16.

[18] Ofer Feinerman and Amos Korman. 2013. Theoretical Distributed Computing Meets Biology: A Review. In *ICDCIT (Lecture Notes in Computer Science, Vol. 7753)*. Springer, 1–18.

[19] Nissim Francez. 1986. *Fairness*. Springer.

[20] Rachid Guerraoui and Eric Ruppert. 2009. Names Trump Malice: Tiny Mobile Agents Can Tolerate Byzantine Failures. In *ICALP (2) (Lecture Notes in Computer Science, Vol. 5556)*. Springer, 484–495.

[21] Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. 2015. Weak models of distributed computing, with connections to modal logic. *Distributed Computing* 28, 1 (2015), 31–53.

[22] Adrian Kosowski and Przemyslaw Uznanski. 2018. Brief Announcement: Population Protocols Are Fast. In *PODC*. ACM, 475–477.

[23] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. 2010. Distributed computation in dynamic networks. In *STOC*. ACM, 513–522.

[24] Daniel Lehmann, Amir Pnueli, and Jonathan Stavi. 1981. Impartiality, Justice and Fairness: The Ethics of Concurrent Termination. In *ICALP (Lecture Notes in Computer Science, Vol. 115)*. Springer, 264–277.

[25] Nancy A. Lynch. 1996. *Distributed Algorithms*. Morgan Kaufmann.

[26] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. 2011. Mediated population protocols. *Theor. Comput. Sci.* 412, 22 (2011), 2434–2450.

[27] Othon Michail and Paul G. Spirakis. 2015. Terminating population protocols via some minimal global knowledge assumptions. *J. Parallel Distributed Comput.* 81-82 (2015), 1–10.

[28] Saket Navlakha and Ziv Bar-Joseph. 2015. Distributed information processing in biological and computational systems. *Commun. ACM* 58, 1 (2015), 94–102.

[29] Fabian Reiter. 2017. Asynchronous Distributed Automata: A Characterization of the Modal Mu-Fragment. In *ICALP (LIPIcs, Vol. 80)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 100:1–100:14.

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