Fakultät für Informatik
Technische Universität München

Population Protocols: Expressiveness, Succinctness and Automatic Verification.

Stefan Jaax

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:
Prof. Dr. Helmut Seidl

Prüfende der Dissertation:
1. Prof. Dr. Francisco Javier Esparza Estaun
2. Prof. Rupak Majumdar, Ph.D.,
   Technische Universität Kaiserslautern

Die Dissertation wurde am 03.03.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 04.06.2020 angenommen.

Abstract

Population protocols (Angluin *et al.*, PODC, 2004) are a model of distributed computation in which identical, finite-state, passively mobile agents interact in pairs to achieve a common goal. In the basic model of population protocols, agents compute number predicates by reaching a stable consensus. It is well known that population protocols compute precisely the semilinear predicates, or, equivalently, the predicates definable in Presburger arithmetic, the first-order theory of the natural numbers equipped with addition and the standard linear order.

This thesis investigates three fundamental questions of the theory of population protocols: Space complexity, verification complexity, and expressiveness of reasonable extensions.

Space Complexity. We show that every quantifier-free Presburger predicate φ augmented with remainder predicates is computable by a population protocol with $\text{poly}(|\varphi|)$ states, where $|\varphi|$ denotes the size of φ in binary encoding. Further, the protocol can be constructed in polynomial time. This is a major improvement to the previously known construction, which requires $2^{\text{poly}(|\varphi|)}$ states. As a special case, we consider predicates of the form $\varphi_c(x) = x \geq c$, where c is a positive integer constant. We provide upper and lower bounds for the number of states needed to represent $\varphi_c(x)$.

Verification Complexity. Verification complexity is concerned with automatic solutions to the *correctness problem*: Given a population protocol $\mathcal{P}$ and some Presburger formula φ, decide whether $\mathcal{P}$ computes φ. Esparza *et al.* showed that the correctness problem is decidable, albeit of very high complexity: It is at least as hard as the Petri net reachability problem, which has non-elementary complexity (Esparza *et al.*, Acta Informatica, 2017). The high complexity of the correctness problem is essentially due to its parametricity: The verifying procedure has to decide whether the protocol stabilizes to the correct consensus for *all* inputs (of which there are infinitely many). We exhibit a subclass of population protocols that is efficiently decidable and suitable for automatic verification. We implemented our approach in the first fully-automatic and parametric verification tool of its kind, and show its viability with benchmarks from the literature.

Expressiveness of Extensions. We show that under very weak assumptions, extensions of the basic model of population protocols compute at most the number predicates in NL, when the input is given in unary. As a very natural example of an extension matching this upper bound, we introduce *broadcast protocols*, which extend population protocols by the ability to transmit reliable broadcast signals simultaneously received by all agents.

Zusammenfassung

Populationsprotokolle (Angluin *et al.*, PODC, 2004) sind ein distribuiertes Rechenmodell von baugleichen, passiv-mobilen Agenten mit endlichem Speicher, welche in paarweisen Interaktionen Rechnungen durchführen. In dem Standardmodell berechnen Agenten Zahlenprädikate durch Erlangen eines stabilen Konsenses. Bekanntermaßen berechnen Populationsprotokolle genau die semilinearen Prädikate; dies sind die Prädikate definierbar in Presburger-Arithmetik, der Theorie der Prädikatenlogik erster Stufe der ganzen Zahlen mit Addition und der gewöhnlichen linearen Ordnung.

Die vorliegende Arbeit untersucht Populationsprotokolle aus dreierlei Blickwinkeln: Platzkomplexität, Verifikationskomplexität, und Ausdrücksstärke von Erweiterungen.

Speicherkomplexität. Wir zeigen, dass sich jedes Quantoren-freie Presburger-Prädikat φ (erweitert um Modulo-Prädikate) durch ein Protokoll mit poly($|\varphi|$) Zuständen berechnen lässt, wobei $|\varphi|$ die Länge von φ in Binärkodierung bezeichnet. Ferner lässt sich das Protokoll in Polynomialzeit konstruieren. Dies ist eine deutliche Verbesserung zu der bekannten Vorläuferkonstruktion, welche $2^{\text{poly}(|\varphi|)}$ Zustände benötigt. Als Spezialfall betrachten wir Prädikate der Form $\varphi_c(x) = x \geq c$ mit $c \in \mathbb{N}$. Wir geben obere und untere Schranken für die Anzahl an benötigten Zuständen zur Berechnung von $\varphi_c(x)$.

Verifikationskomplexität. Verifikationskomplexität bezieht sich auf automatische Lösungsansätze für das *Korrektheitsproblem*: Entscheide, ob ein gegebenes Protokoll $\mathcal{P}$ ein gegebenes Prädikat φ berechnet. Esparza *et al.* wiesen Entscheidbarkeit des Korrektheitsproblems nach, wenngleich mit hoher Komplexität: Das Problem ist mindestens so schwer wie das Erreichbarkeitsproblem für Petrinetze, welches nicht-elementare Komplexität aufweist (Esparza *et al.*, Acta Informatica, 2017). Diese hohe Komplexität lässt sich im Wesentlichen auf die Parametrizität des Problems zurückführen: Die verifizierende Prozedur muss entscheiden, ob das Protokoll zum richtigen Konsens konvergiert, und zwar für *alle* Eingaben (wovon es unendlich viele gibt). Wir stellen eine Teilklasse von Populationsprotokollen vor, welche sich für automatische Verifikation eignet. Wir implementieren den daraus resultierenden Ansatz in dem ersten vollautomatischen und parametrischen Verifikationswerkzeug seiner Art, und führen eine Machbarkeitsstudie mit Benchmarks aus der Literatur durch.

Ausdrucksstärke von Erweiterungen. Wir zeigen, dass Erweiterungen von Populationsprotokollen unter relativ schwachen Annahmen höchstens die Prädikate in NL berechnen, wobei die Eingabe als unär-kodiert angenommen wird. Als natürliche Erweiterung, welche an diese obere Schranke heranreicht, stellen wir Broadcast-Protokolle vor; diese erweitern Populationprotokolle um globale, verlässliche Broadcasts, die simultan von allen Agenten empfangen werden.

Acknowledgments

First and foremost, I would like to thank my doctoral advisor, Javier Esparza, for his invaluable guidance and support. Javier gave me plenty of opportunities for personal growth; his focus on clarity and precision helped me become a better researcher, and a better communicator of ideas.

With sincere gratitude, I would like to thank my Canadian co-author Michael Blondin, who I now consider a good friend. Even by Canadian standards, Michael is exceptionally nice and considerate; it is always a joy to work with him. In autumn 2018, Michael hosted me at his home university in Sherbrooke, Canada. Together with his partner Fannie he went out of his way to make my research stay pleasant and memorable.

I would also like to thank Stefan Kiefer for hosting me at the University of Oxford in spring 2019. Stefan introduced me to the academic culture of Oxford (with all its arcane customs and traditions), and he reserved a good chunk of his time for a rewarding exchange of ideas.

The two research stays in Oxford and Sherbrooke were made possible by my graduate school IGSSE, which provided extensive funding. I would like to thank the team of IGSSE for their great support. I am also grateful for the support I received by an ERC Advanced Grant (787367: PaVeS).

Furthermore, I would like to thank my other co-authors for their fruitful collaboration: Blaise Genest, Martin Helfrich, Antonín Kučera, Philipp Meyer, Michael Raskin, and Chana Weil-Kennedy. Special thanks go to Philipp, who implemented our verification approach quickly and efficiently. I would also like to thank all my colleagues from Lehrstuhl VII for the insightful discussions and the welcoming atmosphere; in particular, I would like to thank Salomon Sickert for always surprising me with neat presents, Michael Luttenberger for his efforts to make organizational matters easier for us, and Stefan Kugele for bringing a breath of fresh air.

Finally, I would like to thank friends and family for their support. I am grateful for being surrounded by a circle of friends that call themselves Dehaan-Bros; their quirky and whimsical humor never fails to cheer me up. I would like to thank my friends Fari, Hannah, Jens, Niklas, Vincent, Pauline and Rasty for giving me strength, especially in difficult times.

The biggest thanks of all go to my parents for their limitless love, patience and support.

Contents

List of Figures

List of Tables

Acronyms

BSCC	Bottom Strongly Connected Component.
QFPA	Quantifier Free Presburger Arithmetic.
VASS	Vector Addition System with States.

1 Introduction

In a never-ending quest to condense units of computation to ever smaller units, researchers have entertained the idea of a *chemical computer* [MCdFD05, AC02, DDPM19]. In this unconventional computer, input data are represented by differences in chemical concentrations, and computations are implemented by local interactions between chemical particles. From the perspective of distributed computing, the interacting particles are network nodes, and interactions between these nodes result in a global system configuration representing the distributed output of the computation.

If chemical particles are to be considered computational devices in a distributed network, they must be devices of a particular kind: They are *passively mobile* in the sense that they merely follow a Brownian motion pattern with no control over their movement (they swirl around in a "chemical soup", so to speak); they possess *limited memory*; they follow the *same protocol* (molecules of the same kind behave identically in the chemical soup); they are *anonymous* (they cannot identify each other); they have *limited communication primitives* and *interact locally* with their immediate surroundings; finally, their movements and interactions are *stochastic* and *asynchronous.* Similar constraints apply to sensor networks [AAD$^+$06] and robot swarms [SVR17] composed of tiny computational devices with limited mobility.

Applications in chemical computing and sensor networks inspired theoretical models of distributed computing which take these limitations into account. In the 1960s, chemists started modeling the interactions between chemical components in continuous-time models called *chemical reaction networks* [CCDS14]. In 2004, Angluin *et al.* introduced *population protocols* [AAD$^+$06], a model of distributed computing that may be regarded as a discrete-time variant of chemical reaction networks. Population protocols are the central object of study of this thesis.

Since their introduction in 2004, population protocols have attracted substantial research interest[1]. The popularity of population protocols in research can in part be explained by applications in sensor networks and chemical computing, and in part by the elegant simplicity of the model: network nodes are replicas of the same finite state machine; state transitions occur through pair-wise rendez-vous; the occurrence of such rendez-vous is governed by a scheduler subject to a global fairness condition. The fairness condition, in turn, serves as a substitute for a probabilistic scheduler, and frees the theoretician from the complications that usually come with stochastic models, while capturing the essential properties of random interactions.

In the standard variant of population protocols, network agents compute predicates (i.e. boolean-values functions) by reaching a stable consensus; the value of the consensus (*yes* or *no*) determines the outcome of the computation. The predicates computable by population protocols are precisely the *Presburger-definable* predicates, that is, the predicates definable in the first-order theory of the natural numbers with addition and

[1]A search for population protocols (in quotes) yields 1140 results on Google Scholar for the years 2004-2020.

the usual linear order. The deep and surprising connection with Presburger arithmetic indicates that population protocols are a formalism worthy of close consideration.

Population protocols exhibit essential properties of the applications that inspired their invention: anonymity and uniform behavior of network participants, limited memory, and asynchronous local interactions that give rise to convergent collective behavior. On the other hand, population protocols in their most basic form are incapable of expressing the behavior of most real-world systems in sufficient detail; the basic model misses essential properties such as the ability to specify dynamic network topologies, failure probabilities, or differences in latencies. However, this apparent shortcoming is more of a feature than a bug: Population protocols lie firmly at the boundary of decidability [EGLM17, AR09], and seemingly innocuous extensions of the basic model quickly lead to undecidability of many relevant decision problems, like whether some agent eventually reaches a certain state for all input configurations, whether a protocol computes a certain predicate, or whether a protocol stabilizes to a unique output [AR09]. In this sense population protocols are quite economical: they allow us to reason about distributed computations of tiny devices in a very abstract setting.

In this publication-based thesis, we investigate population protocol from the viewpoint of *complexity*. When it comes to reasoning about the complexity of any theoretical model of computation, the following questions are central:

1. How succinctly can computations be represented in the model?
2. What is the complexity of verifying properties of the model?
3. What is the expressiveness of the model and of its extensions?

We investigate all three questions for the model of population protocols.

Space complexity. Space complexity measures the size of the memory required per agent, relative to the description size of the function computed by the protocol. Since agents are assumed to be tiny in applications of population protocols, memory requirements are crucial, both in terms of implementation costs and in terms of practical realizability. This holds particularly true for molecular computing, where technical constraints limit the number of states representable by a molecule [CDS$^+$13]. Verifiability provides another incentive for small protocols, because larger protocols are usually harder to verify due to the state space explosion problem [Val96].

Automatic Verification. Population protocols, as a model of distributed computing, specify interactions between nodes in a distributed system. In general, debugging and testing of distributed systems is particularly difficult [MH89]: interleaved interactions lead to inherent nondeterminism, and reproducibility of errors is reduced as a consequence. Limited system transparency further restrains debugging capabilities: errors may arise from global system properties whose identification is difficult from a local view on the system.

These difficulties in debugging and testing make *automatic verification methods* attractive. A formal verification algorithm takes as input a specification of the protocol $\mathcal{P}$ and a formal specification φ of a system property, and determines whether $\mathcal{P}$ satisfies φ. Automatic verification helps avoid design errors at the protocol level, and may provide stronger correctness guarantees than empirical testing techniques.

Automatic verification techniques for population protocols can be divided into two categories: 1.) *Finite-state model checking techniques*, and 2.) *Parametric verification* *techniques* (see e.g. [BK08], Chapter 21, for a comparison of the two categories). The first category is limited to verifying networks of bounded size, while methods of the second category verify a protocol independent of the number of network nodes. In algorithms of the second category the size of the system is a parameter of the verification problem, hence the name parametric verification technique [BK08].

Expressiveness. Population protocols were invented with applications in low-resource environments in mind. The basic model is therefore quite restrictive in terms of communication primitives: asynchronous interactions between nodes occur locally in pairs, and a global clock or some other form of synchronization mechanism is absent. However, population protocols are easily extendible, which is evidenced by the voluminous literature on extensions of the basic model [ISV17, GR07, MCS11, Asp17, MS15, BEJ19a, AAER07, BBB13, CFQS12]. This raises the question how expressive these extensions are. Here we are particularly interested in extensions of population protocols by non-local communication primitives. These extensions are motivated by implementations of *in vivo* computations in population of microorganisms, which may be stimulated by an external global signal, such as a source of light [UMD$^+$15, BDG$^+$19].

1.1 Literature Overview

The population protocol model was introduced by Angluin, Aspnes, Diamadi, and Peralta in 2004 in a seminal paper in [AAD$^+$06]. Aspnes and Ruppert provide a survey of population protocols in [AR09].

Related models and applications. The development of population protocols was motivated by work on networks of passively mobile finite-state sensors [AAD$^+$06] and by distributed models of trust propagation due to Diamadi and Fischer [DF01]. The model of population protocols bears a strong resemblance to models used in theoretical chemistry [Gil77, Gil92, AR09]. Population protocols are also closely related to multiset rewrite systems [BT05, BLM86], vector addition systems [Ler11] or, equivalently, Petri nets [DA94]. Various different extensions and variations of the standard model of population protocols have been suggested in the literature, among these protocols that admit faulty interactions [ISV17], protocols with identifiers [GR07, MCS11], protocols with global synchronization mechanisms [Asp17, MS15, BEJ19a], protocols with unbounded memory [AAER07], protocols with different fairness assumptions [CDFS11, SLDP09a], and protocols acting on varying network topologies [BBB13, CFQS12].

Expressiveness. The expressive power of the basic model of population protocols is well understood. In [AAE06, AAER07], Angluin, Aspnes and Eisenstat showed that the basic model of population protocols computes precisely the *semilinear* predicates, which is the class of predicates definable in *Presburger arithmetic*, the first-order theory of the natural number with addition and the usual linear order (for a survey of Presburger arithmetic, see e.g. [Haa18]). Semilinearity had earlier been shown to be sufficient in [AAD$^+$06].

For a discussion of research on population protocols related to space complexity, verification, and expressiveness of extensions we refer the reader to chapters 4-6.

1.2 Summary of Contributions

Here we only give a brief and high-level account of the contributions of the thesis. For a technical summary, we refer the reader to the chapters 4-6.

We show new results for three aspects of complexity in population protocols: Space complexity, complexity of automatic verification, and expressiveness of extensions.

Space complexity. In [BEJ18a, BEG$^+$20], we consider the number of states needed per agent to compute a given predicate. We assume that predicates are represented by formulas in the quantifier-free fragment of Presburger arithmetic, augmented with remainder predicates of the form $\varphi(x_1, \ldots, x_n) = \sum_i \alpha_i x_i = c \bmod m$, where the α_i, c, and m are integer constants. This fragment is equivalent in expressiveness to Presburger arithmetic [Haa18, AAE06]. We show that any formula φ can be computed by a population protocol with $\text{poly}(|\varphi|)$ states, where $|\varphi|$ denotes the binary length of the formula, provided that numerical constants are encoded in binary. This is a major improvement to the previously known construction which requires $2^{\text{poly}(|\varphi|)}$ states. As a special case, we consider predicates of the form $\varphi(x) = x \geq c$, where c is a positive integer constant. We provide upper and lower bounds for the number of states needed to represent these predicates.

Automatic verification. Given a population protocol $\mathcal{P}$ and a predicate φ, we would like to determine algorithmically whether $\mathcal{P}$ computes φ, independent of the size of the input. This is a very hard problem, which has no practical solution for the entire class of population protocols [EGLM17]. In [BEJM17] we exhibit a subclass of population protocols suitable for automatic verification. We implemented our approach in the first fully-automatic and parametric verification tool of its kind. We show the viability of our approach with benchmarks using examples of population protocols from the literature [BEJM17, BEJ18b].

Expressiveness of extensions. In [BEJ19a], we consider the expressive power of *broadcast consensus protocols*, an extension of population protocols by reliable broadcasts which are simultaneously received by the entire population. We show that broadcast consensus protocols compute precisely the number predicates in the complexity class NL; these are the predicates computable by a nondeterministic Turing machine with logarithmic space, where the input is given in unary. We further establish an NL upper bound for the expressive power of conservative extensions of protocols whose reachability relation is computable in NL.

1.3 Publication Summary

This is a publication-based thesis. The papers are included in the appendix.
In Part I of Appendix, we present the following papers:

A Michael Blondin, Javier Esparza, Stefan Jaax. Large Flocks of Small Birds: on the Minimal Size of Population Protocols. STACS, 2018. [BEJ18a]

B Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, Stefan Jaax. Succinct Population Protocols for Presburger Arithmetic, STACS, 2020. [BEG$^+$20]

In Part II of Appendix, we present the following papers:

C Michael Blondin, Javier Esparza, Stefan Jaax, Philipp J. Meyer. Towards Efficient Verification of Population Protocols. PODC, 2017. [BEJM17]

D Michael Blondin, Javier Esparza, Stefan Jaax. Peregrine: A Tool for the Analysis of Population Protocols. CAV, 2018. [BEJ18b]

In Part III of Appendix, we present the following paper:

E Michael Blondin, Javier Esparza, Stefan Jaax. Expressive Power of Broadcast Consensus Protocols. CONCUR, 2019. [BEJ19a]

The papers appeared in peer-reviewed conference proceedings, and are therefore self-contained. Each paper is prefaced with a summary page, containing the full citation of the original publication, a short synopsis of the publications content and the thesis author contributions.

1.4 Outline of the Thesis

Chapter 2 introduces mathematical notation, the basic model of population protocols, and models related to population protocols.

Chapters 4-6 provide a technical overview of our contributions with respect to space complexity, automatic verification, and expressiveness of extensions, in that order. Each of the three chapters is divided into four sections: 1. Problem Statement, 2. New Contributions, 3. Related Work, and 4. Open Research Questions. The first section introduces the problem under consideration, the second section states our major new contributions and provides proof sketches, the third section relates our results to existing research, and the fourth section closes with an outlook on open questions.

The appendix has three parts. In Part I, II, and III we present papers A-B, C-D, and E, respectively.

2 Preliminaries

2.1 Basic Notation

Sets of numbers. By $\mathbb{N}$ we denote the set $\{0, 1, 2, \ldots\}$ of positive integers including 0, and by $\mathbb{Z}$ we denote the set $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ of integers.

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

Vector notation. We write vectors of numbers in bold face. For number vectors $\boldsymbol{x} = (x_1, \ldots, x_k), \boldsymbol{y} = (y_1, \ldots, y_k)$ of equal dimension k, by $\boldsymbol{x} \cdot \boldsymbol{y}$ we denote the dot product $(x_1 \cdot y_1 + x_2 \cdot y_2 + \ldots + x_k \cdot y_k)$. Addition/substraction is defined componentwise.

Complexity. By log we denote the logarithm to the base 2. A function $f \colon \mathbb{N} \to \mathbb{N}$ is bounded by poly(n), if there exists a polynomial p such that $f(n) \leq p(n)$ for every n. Similarly, we say f is bounded by polylog(n), if there exists a polynomial p such that $f(n) \leq p(\log(n))$ for every $n > 0$.

2.2 Vector Addition Systems

Population protocols are related to *vector addition systems.*

Definition. A vector addition system with states (VASS) of some fixed dimension $k \in \mathbb{N}$ can be described as a pair (Q, T) where Q is a finite set of states, and $T \subseteq Q \times \mathbb{Z}^k \times Q$ is a transition relation. VASS with $|Q| = 1$ are called *vector addition systems* or *Petri nets.* We write $q \xrightarrow{\boldsymbol{v}} r$ whenever $(q, \boldsymbol{v}, r) \in T$. Furthermore, for two vectors $\boldsymbol{w}, \boldsymbol{w}' \in \mathbb{N}^k$ and states $q, q' \in Q$, we write $(q, \boldsymbol{w}) \to (q', \boldsymbol{w}')$ whenever there exists a vector $\boldsymbol{v}$ such that $q \xrightarrow{\boldsymbol{v}} q'$ and $\boldsymbol{w}' = \boldsymbol{w} + \boldsymbol{v}$. If $|Q| = 1$, we may omit the states and simply write $\boldsymbol{v} \to \boldsymbol{w}$ for a single step from $\boldsymbol{v}$ to $\boldsymbol{w}$. We call $\to$ *one-step relation*, and by $\xrightarrow{*}$ we denote the reflexive-transitive closure of $\to$.

Reachability. The *reachability problem for VASS* is the following problem: Given vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{N}^k$ in the dimension k of a given VASS, and given states q, r, does $(q, \boldsymbol{v}) \xrightarrow{*} (r, \boldsymbol{w})$ hold? The reachability problem is already TOWER-hard for vector addition systems (without state) [CLL+19].

Forward-inductive sets. Let $\rightarrow \subseteq \mathbb{N}^m \times \mathbb{N}^m$ be the one-step relation of a vector addition system. We say a set $P \subseteq \mathbb{N}^m$ is forward-inductive (w.r.t $\rightarrow$), if $\boldsymbol{v} \rightarrow \boldsymbol{w}$ and $\boldsymbol{v} \in P$ implies $\boldsymbol{w} \in P$.

2.3 Presburger Arithmetic and Semilinear Sets

Presburger Arithmetic. Presburger arithmetic is the first-order theory of the natural numbers with addition, inequality, and the standard axioms of arithmetic. We say a first-order formula is *Presburger* if it is compatible with the structure $(\mathbb{Z}, 0, 1, +, <)$, where $+$ is the usual addition, and $<$ denotes the usual linear order on the integers.

A *number predicate of arity n* is a subset of $\mathbb{Z}^n$. A Presburger formula φ with freely occuring variables $x_1, \ldots, x_n$ defines a number predicate $\varphi(x_1, \ldots, x_n)$ of arity n:

$$\varphi(x_1, \ldots, x_n) \overset{\text{def}}{=} \{(a_1, \ldots, a_n) \in \mathbb{Z}^n \mid \varphi[x_1 := a_1; \ldots; x_n = a_n]\}$$

where $\varphi[x_1 := a_1; \ldots; x_n = a_n]$ denotes the formula that is obtained by replacing all free occurrences of x_i by a_i in φ.

If $\boldsymbol{a} = (a_1, \ldots, a_n) \in \varphi(x_1, \ldots, x_n)$, we write this more concisely as $\varphi(\boldsymbol{a})$. In this case we say that $\boldsymbol{a}$ satisfies φ. Furthermore, we write $\varphi(\boldsymbol{a}) = 1$, if $\boldsymbol{a}$ satisfies φ, and $\varphi(\boldsymbol{a}) = 0$ otherwise. Two formulas φ, ψ are *semantically equivalent* if their sets of freely occuring variables are identical, and $\varphi(\boldsymbol{a}) = \psi(\boldsymbol{a})$ holds for every $\boldsymbol{a}$.

Example 2.3.1. The Presburger formula $\exists y\colon y + y = x$ expresses the predicate "x is even".

It is well known that Presburger arithmetic admits *quantifier elimination*:

Theorem 1. *[Haa18, AAD⁺06] Every Presburger predicate $\varphi(x_1, \ldots, x_n)$ is semantically equivalent to a boolean combination of predicates of the form*

$$\begin{aligned} &\alpha_1 x_1 + \ldots + \alpha_n x_n \geq c \qquad\qquad , \textit{ and} \\ &\alpha_1 x_1 + \ldots + \alpha_n x_n = c \ (\textit{mod } m) \end{aligned}$$

where $\alpha_1, \ldots, \alpha_n, c, m$ are integer constants.

We call predicates of the upper kind *threshold predicates*, and predicates of the lower kind *remainder predicates.* We refer to threshold or remainder predicates as *atomic predicates.*

If a Presburger predicate φ is given as boolean combination of threshold and remainder predicates, we say that φ is in *QFPA* (quantifier-free Presburger arithmetic)[1].

Semilinear Sets. A *linear set* (of dimension $d > 0$) is a any set of the form

$$\{\boldsymbol{b} + \lambda_1 \boldsymbol{p}_1 + \ldots + \lambda_n \boldsymbol{p}_n \mid \lambda_i \geq 0, 1 \leq i \leq n\}$$

where $\boldsymbol{b} \in \mathbb{Z}^d$ is a *base vector* and $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ is a set of *period vectors.* A *semilinear* set is a finite union of *linear sets.*

The following result relates semilinear sets and Presburger-definable predicates:

Theorem 2. *[GS64] Semilinear sets and Presburger-definable predicates coincide.*

[1]This is a slight abuse of notation; technically, remainder predicates are not expressible in Presburger arithmetic without quantifiers.

3 The Basic Model of Population Protocols

In the basic model of population protocols, anonymous agents carry out computations in populations of arbitrary, but fixed size. This excludes the possibility of crashes on the level of individual agents, and the ability to spawn new agents after initialization. The size of the population is determined by the size of the input: each agent is initially given one token of the input that is mapped to some internal state, e.g. if the input is $x = 2$ and $y = 3$, this is represented by two agents in, say, state q_x and three agents in, say, state q_y. At any given point in time, each agent is in one out of finitely many states. The global distribution of states is referred to as *configuration*. Agents may change their state in discrete time steps through pairwise rendez-vous. How interactions between agents affect their states is governed by a transition relation, and the occurrence of interactions is determined by a scheduler, which picks at every time step a pair of agents to interact. The scheduler's choices determine an infinite trace of successive configurations, called *execution* of the protocol. The scheduler is assumed to be adversarial, but it must obey a certain fairness condition. The condition roughly states that the scheduler may not avoid a configuration forever when it is within reach infinitely many times.

By assuming a state, agents cast a vote: each state maps to a unique boolean output in the set $\{0, 1\}$. A configuration is in a *consensus* of some value $b \in \{0,1\}$ if the state of each agent maps to output b, and a consensus is *stable* if it cannot be destroyed by future interactions. An execution rejects or accepts the input when a stable consensus of value 0 or 1, respectively, is reached.

3.1 Formal Definition

A population protocol is formally specified by a tuple (Q, T, X, I, O), where

- Q is a finite set of states,
- $T \subseteq Q^4$ is a set of *transitions* satisfying $(q, r, q, r) \in T$ for every $q, r \in Q$,
- X is a finite set of *input variables*,
- $I \colon X \to Q$ is the *input mapping*,
- $O \colon Q \to \{0, 1\}$ is the *output mapping*.

Configurations and input. Since agents are anonymous, the current state of a population is exhaustively described by the number of agents in each state. Mathematically, we represent this by a multiset $C \in \mathbb{N}^Q$, and we call such a multiset C a *configuration*. Every input $\boldsymbol{v} \in \mathbb{N}^X$ is mapped to an *initial configuration* $C_{\boldsymbol{v}} \in \mathbb{N}^Q$ given by:

$$C_{\boldsymbol{v}}(q) \overset{\text{def}}{=} \sum_{x \in I^{-1}(q)} \boldsymbol{v}(x) \qquad \text{for every } q \in Q.$$

Intuitively, every agent receives one token of the input.

For a given configuration $C \in \mathbb{N}^Q$, its *output* $O(C) \in \{0, 1, \bot\}$ is given by:

$$O(C) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } O(q) = 1 \text{ for every } q \in [\![C]\!], \\ 0 & \text{if } O(q) = 0 \text{ for every } q \in [\![C]\!], \\ \bot & \text{otherwise.} \end{cases}$$

If $O(q) = b \in \{0, 1\}$, we also say C *is in a consensus.*

Transitions. For every $t = (q_1, q_2, r_1, r_2) \in T$, we write $q_1, q_2 \to r_1, r_2$ to denote t, and we define $\text{pre}(t) \overset{\text{def}}{=} \wr q_1, q_2 \wr$ and $\text{post}(t) \overset{\text{def}}{=} \wr r_1, r_2 \wr$. We say $t\colon q_1, q_2 \to r_1, r_2 \in T$ is *enabled* at a given configuration C if $\wr q_1, q_2 \wr \leq C$, and *disabled* otherwise. If t is enabled at C, then it can be *fired*, leading to a configuration $C' = C - \wr q_1, q_2 \wr + \wr r_1, r_2 \wr$, which we denote $C \xrightarrow{t} C'$. Intuitively, firing t can be thought of as the scheduler picking two agents in states q_1 and q_2, respectively, and putting them into the successor states r_1, r_2, resulting in the successor configuration C'.

Executions. An *execution* starting in some configuration C is an infinite sequence $C_1C_2C_3C_4\ldots$ such that $C = C_1$ and $C_i \xrightarrow{t_i} C_{i+1}$ for every $i \in \mathbb{N}$. If the transitions are important, we write $C_1 \xrightarrow{t_1} C_2 \xrightarrow{t_2} C_3 \xrightarrow{t_3} \ldots$ instead, and call such a sequence a *labeled execution.* By $\text{Inf}(\pi) \overset{\text{def}}{=} \{C \mid C_i = C \text{ for infinitely many } i\}$ we denote the set of of configurations occurring infinitely often in the execution $\pi = C_1C_2C_3\ldots$. We lift the output function to the execution π in the following way:

$$O(\pi) \overset{\text{def}}{=} \begin{cases} b & \text{if } O(C) = b \text{ for every } C \in \text{Inf}(\pi), \\ \bot & \text{otherwise.} \end{cases}$$

Global fairness. The basic model of population protocols imposes a global fairness assumption on the scheduler. An execution π is (globally) fair if the set of configurations occurring infinitely often in π is closed under reachability, that is, if the following equality holds:

$$\{C' \mid \exists C \in \text{Inf}(\pi)\colon C \xrightarrow{*} C'\} = \text{Inf}(\pi)$$

In other words, every configuration that *can* be reached infinitely often, *must* be reached infinitely often in a fair execution.

Well-Specified protocols. We call a protocol *well specified* if its executions stabilize to a unique output for every input. More formally, a protocol is well specified if it satisfies the following: For every input $\boldsymbol{v} \in \mathbb{N}^X$, there exists a boolean $b \in \{0, 1\}$ such that $O(\pi) = b$ for every fair execution π starting in the initial configuration $C_{\boldsymbol{v}}$.

Stable computations. Every well-specified protocol *stably computes* a number predicate $\varphi\colon \mathbb{N}^X \to \{0, 1\}$ given by $\varphi(\boldsymbol{v}) = O(\pi_{\boldsymbol{v}})$, where $O(\pi_{\boldsymbol{v}})$ is the unique output of any fair execution $\pi_{\boldsymbol{v}}$ starting in $C_{\boldsymbol{v}}$. To reduce notational clutter, we sometimes tacitly assume that the input mapping I is the identity function (this does not limit expressiveness); in this case, we have $\boldsymbol{v} = C_{\boldsymbol{v}}$, and we simply write $\varphi(C)$ for the value of the predicate computed for the initial configuration C.

3.2 Examples of Population Protocols

Example 3.2.1. A simple counting protocol [AAD⁺06, BEJK18]. Let $c \in \mathbb{N}$ be some integer constant, and let $\varphi_c(x)$ be the one-arity predicate $\varphi_c(x) = x \geq c$. We define a protocol $\mathcal{P}_c = (Q, T, X, I, O)$ with $X = \{x\}$ that computes $\varphi_c(x)$. This means any fair execution of $\mathcal{P}_c$ shall stabilize to a consensus that decides the question: *Are there at least c agents in the population?*

We set $Q \overset{\text{def}}{=} \{0, 1, \ldots, c\}$, $I(x) = 1$, $O(q) = 1 \Leftrightarrow q = c$, and let T consist of the following transitions:

$$\begin{aligned} &t_{a,b}\colon a, b \to 0, \min(a+b, c) && \text{for every } 0 \leq a, b < c, \\ &t_a\colon a, c \to c, c \end{aligned}$$

Intuitively, $\mathcal{P}_c$ works as follows. Each agent stores a number. When two agents interact due to transition $t_{a,b}$, one agents stores the sum, capped at c, and the other agent is sent to 0. The transition t_a and fairness ensures that once an agent reaches c, this value is propagated throughout the population, and the computation stabilizes to consensus 1. For example, let $c = 3$, let $x = 4$, and let the initial population be $C_x = \wr 1, 1, 1, 1 \int$. The following labeled execution is fair and stabilizes to the correct consensus $O(\wr 3, 3, 3, 3 \int) = 1 = (x > 3)$:

$$\wr 1,1,1,1 \int \xrightarrow{t_{1,1}} \wr 2,0,1,1 \int \xrightarrow{t_{2,1}} \wr 3,0,0,1 \int \xrightarrow{t_0} \wr 3,3,0,1 \int \xrightarrow{t_0} \wr 3,3,3,1 \int \xrightarrow{t_1} \wr 3,3,3,1 \int \xrightarrow{t_1} \ldots$$

Example 3.2.2. Majority protocol [AAD+06]. We define the protocol $\mathcal{P}_{x \geq y} = (Q, T, X, I, O)$ that computes the *majority predicate* $\varphi(x, y) \equiv x \geq y$ as follows.

$$\begin{aligned} Q &= \{\mathtt{X}, \mathtt{Y}, \mathtt{x}, \mathtt{y}\} & O(q) &= 1 \Leftrightarrow q \in \{\mathtt{X}, \mathtt{x}\} & T = \{&\mathtt{X}, \mathtt{Y} \to \mathtt{x}, \mathtt{y}; \\ X &= \{x, y\} & I(x) &= \mathtt{X} & &\mathtt{X}, \mathtt{y} \to \mathtt{X}, \mathtt{x}; \\ & & I(y) &= \mathtt{Y} & &\mathtt{Y}, \mathtt{x} \to \mathtt{Y}, \mathtt{y}; \\ & & & & &\mathtt{x}, \mathtt{y} \to \mathtt{x}, \mathtt{x}\} \end{aligned}$$

Intuitively, initially all agents are in "big" states X or Y. Whenever X and Y interact, they are turned into their smaller counterparts. This happens until only X or Y remains among the big states. The big states may dominate the small states, and convert them to the majority opinion. If a tie occurs, i.e. the number of X initially equals the number of Y, then the bottom-most transition serves as a tie breaker by converting all agents to x, yielding a stable 1-consensus. Agents can be converted back and forth for an arbitrary amount of time, but fairness ensures that the majority eventually wins.

3.3 Population Protocols compute precisely the Semilinear Predicates

In [AAD+06], Angluin *et al.* show that for every semilinear predicate, there exists a protocol that stably computes this predicate. By Theorem 1, it suffices to show that boolean combinations of predicates of the form $\sum_{i=1}^{n} \alpha_i x_i > c$ (*threshold*) and $\sum_{i=1}^{n} \alpha_i x_i \equiv a \pmod{b}$ (*remainder*) are computable by population protocols, where α_i, c, a, b are integer constants. Angluin *et. al* gave protocols for any remainder and threshold predicate; the protocols are generalizations of the protocols from Example 3.2.1 and Example 3.2.2.

Negation can be implemented by flipping the output values. Finally, the conjunction/disjunction of two predicates can be implemented through a synchronous *product*

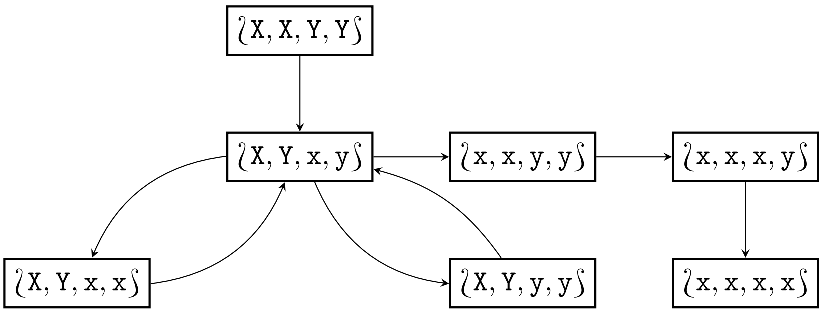


Figure 3.1: Drawing of the reachability graph of the configuration ⦃X, X, Y, Y⦄ for the majority protocol from Example 3.2.2. Self loops are omitted for clarity. The configuration ⦃x, x, x, x⦄ is the only bottom configuration. Note, however, that in general BSCCs of reachability graphs need not be singletons.

construction: Given predicates φ_1 and φ_2, and protocols $\mathcal{P}_i = (Q_i, T_i, \Sigma, I_i, O_i)$ that compute φ_i, the protocol $\mathcal{P}_\circ = (Q, T, \Sigma, I, O)$ computes $\varphi_1 \circ \varphi_2$ for $\circ \in \{\wedge, \vee\}$:

$$\begin{aligned}
Q &= Q_1 \times Q_2,\\
T &= \{(q_1, r_1), (q_2, r_2) \to (q_1', r_1'), (q_2', r_2') \mid q_1, q_2 \to q_1', q_2' \in T_1, r_1, r_2 \to r_1', r_2' \in T_2\},\\
I(x) &= (I_1(x), I_2(x))\\
O((q, r)) &= O_1(q) \circ O_2(r)
\end{aligned}$$

The protocol $\mathcal{P}_\circ$ executes $\mathcal{P}_1$ and $\mathcal{P}_2$ in parallel: $\mathcal{P}_1$ is executed in the first component of each state, and $\mathcal{P}_2$ is executed in the second component. The output of $\mathcal{P}$ is given by applying the operator $\circ \in \{\vee, \wedge\}$ to the outputs of $\mathcal{P}_1$ and $\mathcal{P}_2$.

In [AAD+06], Angluin *et al.* show that the converse also holds: predicate stable computable by a population protocol is semilinear. The proof for the converse direction is more involved; for details, see [AAD+06, AR09].

3.4 Global Fairness and other Fairness Notions

Since global fairness is an essential part of population protocol semantics, we provide alternative characterizations of fairness. These characterizations provide further intuition. They are also a prerequisite for later chapters.

Characterization via bottom configurations. The one-step reachability relation $\to$ of a population protocol induces for every configuration C a finite, directed *reachability graph* $\mathcal{G}(C) = (V, E)$ with nodes $V = \{C' \mid C \xrightarrow{*} C'\}$ and edges $E = \{(C_1, C_2) \in V \times V \mid C_1 \to C_2\}$. We call a strongly connected component $\mathcal{C}$ of $\mathcal{G}(C)$ a *bottom strongly connected component* (BSCC), if $\mathcal{C}$ is closed under the reachability relation $\xrightarrow{*}$. We call a configuration $C \in V$ *bottom*, if C belongs to a BSCC. Figure 3.1 illustrates a reachability graph for the majority protocol from Example 3.2.2.

Using the notion of BSCCs, we may characterize fair executions as precisely those executions whose infinitely occurring configurations form a BSCC:

Proposition 1. *[EGLM17] An execution $\pi = C_1 C_2 \ldots$ is fair if and only if there exists a BSCC $\mathcal{B}$ of $\mathcal{G}(C_1)$ satisfying Inf$(\pi) = \mathcal{B}$.*

Proof sketch. We only sketch the proof for the left-to-right direction; the converse direction can be proven analogously. Assume $\pi = C_1C_2\dots$ is fair. We have to show that there exists some BSCC $\mathcal{B}$ of $\mathcal{G}(C_1)$ such that $\mathrm{Inf}(\pi) = \mathcal{B}$. Clearly, for every $i \in \mathbb{N}$ it is possible to reach a bottom configuration of $\mathcal{G}(C_1)$ from C_i. By finiteness of $\mathcal{G}(C_1)$, at least one bottom configuration C is reachable from C_i for infinitely many i. Then by fairness there exists a bottom configuration C such that $C_i = C$ for infinitely many i. Fix such a C. Since C belongs to the BSCC $\mathcal{B}$ and BSCCs are closed under reachability, every extension of an execution starting in C strictly contains configurations from $\mathcal{B}$. This establishes $\mathcal{B} \subseteq \mathrm{Inf}(\pi)$. Moreover, since $C \in \mathrm{Inf}(\pi)$ by assumption and every $C' \in \mathcal{B}$ is reachable from C, we have that every $C' \in \mathcal{B}$ is reachable from C_i for infinitely many i. From this we obtain $\mathcal{B} \supseteq \mathrm{Inf}(\pi)$ by fairness. Finally, from $\mathcal{B} \supseteq \mathrm{Inf}(\pi)$ and $\mathcal{B} \subseteq \mathrm{Inf}(\pi)$ we conclude $\mathrm{Inf}(\pi) = \mathcal{B}$ for some BSCC $\mathcal{B}$ of $\mathcal{G}(C_1)$, and the claim follows.

□

Characterization via one-step reachability. There is an alternative characterization that relaxes transitive reachability to one-step reachability: An execution $\pi = C_0C_1C_2\dots$ is one-step (globally) fair, if every configuration that can be reached infinitely often *in one step* is reached infinitely often in π, formally: if $\{i \in \mathbb{N} \mid C_i \to D\}$ is infinite, then $\{i \in \mathbb{N} \mid C_i = D\}$ is infinite for every configuration D.

Proposition 2. *Every fair execution is one-step fair, and vice versa.*

Proof. Every fair execution π is clearly one-step fair. Conversely, assume $\pi = C_0C_1C_2\dots$ is one-step fair, and let D be such that $\{i \in \mathbb{N} \mid C_i \xrightarrow{*} D\}$ is infinite. To complete the proof, we must show that $D \in \mathrm{Inf}(\pi)$ holds. By finiteness of the number of reachable configurations in any given execution, there must be some configuration C such that $C = C_i$ and $C_i \xrightarrow{*} D$ for infinitely many i. If $C_i \to D$ for infinitely many i, we are done by one-step fairness. Otherwise $C \xrightarrow{*} D$ entails the existence of some $m \geq 1$ and some configurations $C'_1, \dots, C'_m$ such that $C_i \to C'_1 \to C'_2 \to \dots \to C'_m \to D$. Then by one-step fairness, C'_1 occurs infinitely often in π, and by iterative application of one-step fairness, so does $C'_2, \dots, C'_m$, and D. Thus, if D is reachable infinitely often in π, it is indeed reached infinitely often, which proves the claim. □

Notice that the latter argument in the proof of Proposition 2 crucially depends on finiteness of the number of configurations in any given execution; in infinite-state extensions of population protocols the two notions of global fairness generally do not coincide [AAER07].

Relation to probabilistic scheduling. In order to make sense of convergence time of population protocols, the fair scheduler must be replaced by probabilistic semantics. The obvious choice is to replace the fair scheduler by a probabilistic scheduler [AAE08b, MNRS14, AGV15] that picks a pair of agents to interact uniformly at random at every discrete time step. Using the theory of finite-state Markov chains [AAOW15], and the characterization established in Proposition 1, one can show that executions induced by such a uniformly probabilistic scheduler are almost surely fair, and thus every protocol that is well specified under a fair scheduler remains well specified under a uniformly probabilistic scheduler.

Time complexity in population protocols is usually measured in *parallel time* [AG15, BEK18, AGV15], which is defined as the average number of steps the scheduler needs until stabilization, divided by the population size.

Local Fairness. In some publications (*e.g.* [CDFS11, SLDP09a]), global fairness is replaced by a fairness notion called *local fairness.* A labeled execution $\pi = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} C_2 \dots$ is *locally fair* if every transition that can be fired infinitely often in π must be fired infinitely often, formally: If $\{i \in \mathbb{N} \mid \exists D \colon C_i \xrightarrow{t} D\}$ is infinite, then $\{i \in \mathbb{N} \mid C_i \xrightarrow{t} C_{i+1}\}$ is also infinite for every transition t.

Global fairness and local fairness are incomparable notions: To see that local fairness does not imply global fairness, consider the majority protocol from Example 3.2.2. The execution $\wr \mathtt{Y}, \mathtt{x}, \mathtt{y} \wr \to \wr \mathtt{X}, \mathtt{x}, \mathtt{x} \wr \to \wr \mathtt{Y}, \mathtt{y}, \mathtt{x} \wr \to \wr \mathtt{Y}, \mathtt{x}, \mathtt{x} \wr \to \dots$ that alternates between the transitions $\mathtt{Y}, \mathtt{x} \to \mathtt{Y}, \mathtt{y}$ and $\mathtt{x}, \mathtt{y} \to \mathtt{x}, \mathtt{x}$ is locally fair, but not fair. To see why global fairness does not imply local fairness, consider the protocol with states a, b, c and transitions $t_{x,y} \colon x, x \to y, y$ for every pair $(x, y) \in \{(a, b), (a, c), (b, c), (c, b), (c, a)\}$. The execution that repeats the cycle $\wr a, a \wr \xrightarrow{t_{a,c}} \wr c, c \wr \xrightarrow{t_{c,b}} \wr b, b \wr \xrightarrow{t_{b,c}} \wr c, c \wr \xrightarrow{t_{c,a}} \wr a, a \wr$ ad infinitum is globally fair, but not locally fair, since $t_{a,b}$ is enabled infinitely often and never fired.

Observe, however, that for every bottom configuration C there is a locally fair execution π such that C occurs infinitely often in π. Thus, every protocol that is well specified under local fairness is also well specified under global fairness.

3.5 Protocols with Leaders

Leaders are auxiliary agents in some designated state that are provided to the input population. The number of leaders and their initial states are fixed, independent of the input. Formally, a *protocol with leaders* can be specified as a tuple $\mathcal{P} = (Q, T, X, L, I, O)$, where Q, T, X, I, O are defined as in leaderless population protocols, and $L \in \mathbb{N}^Q$ is the initial leader population. The predicate computed by $\mathcal{P}$ for every input $\boldsymbol{v}$ is given by the output of every fair execution starting in $C_{\boldsymbol{v}} + L$.

Example 3.5.1. Consider the protocol with leaders $\mathcal{P} = (Q, T, X, L, I, O)$ where Q, T, X, I, and O are defined as in the majority protocol provided in Example 3.2.2 , and the leader configuration is given by $L = \wr \mathtt{X}, \mathtt{X}, \mathtt{X}, \mathtt{X} \wr$. The protocol $\mathcal{P}$ computes the predicate $x + 4 \geq y$; the offset of 4 is provided by the leader population (recall that $I(x) = \mathtt{X}$ and $I(y) = \mathtt{Y}$).

Leaders do not increase the expressiveness of population protocols, but they may speed up computation: Angluin *et al.* showed that for every predicate there exists a protocol with leaders that runs in polylogarithmic parallel time [AAE08a]. This stands in contrast to leaderless protocols, which require expected linear parallel time for some predicates [AAE08a, DS18].

Leaders also facilitate the implementation of control states which enable or disable transitions. This allows for bounded simulations of stateful token-based transition systems, like vector addition systems with states [HH14, Ler11], or, equivalently, counter machines without zero tests.

To see why leaders do not increase the expressive power of the model, observe that every protocol with leaders that computes some predicate $\varphi(\boldsymbol{x})$ with initial leader population L, can be transformed into a leaderless protocol that computes $\varphi'(\boldsymbol{x}, \boldsymbol{y}) \equiv (\boldsymbol{y} = L) \Rightarrow \varphi(\boldsymbol{x}) \equiv (\boldsymbol{y} \neq L \vee \varphi(\boldsymbol{x}))$ via synchronous product with a protocol computing $\boldsymbol{y} \neq L$. Since $\varphi'(\boldsymbol{x}, \boldsymbol{y})$ is computable by a population protocol without leaders, it must be semilinear (every predicate computable by population protocol is semilinear), and thus so is the projection $\varphi(\boldsymbol{x}) \equiv \forall \boldsymbol{y} \colon \varphi'(\boldsymbol{x}, \boldsymbol{y})$. Every semilinear predicate is computable by a leaderless protocol, and thus so is $\varphi(\boldsymbol{x})$ in particular.

Leader election. In some applications, leaders are only needed in so far that it must be guaranteed that *eventually* some agent is in some designated state. This can be achieved in leaderless protocols by means of a *leader election*: Initially, each agent starts in a *leader state* L. Interactions of the form $L, L \to L, N$ eliminate duplicate leaders by turnig one leader into a non-leaders state N, until precisely one agent in the leader state L remains. This form of leader election by coalescence comes with a caveat: The leader can never be certain that it got elected. Computations that require, say, *precisely* one leader cannot be implemented by leader election, since the computation would have to be started while the leader election might still be ongoing.

4 Space Complexity

In this chapter, we consider the question of the number of states needed to represent any given predicate in a population protocol. This is the question of the *space complexity* of population protocols.

4.1 Problem Statement

For a given representation of a semilinear predicate φ, by $|\varphi|$ we denote the length of its binary encoding. We investigate upper and lower bounds on the number of states required to represent any given QFPA formula φ in a population protocol, relative to $|\varphi|$. By the construction of Angluin *et al.* in [AAD$^+$06], we have that every QFPA formula φ can be computed by a leaderless protocol with $2^{\text{poly}(|\varphi|)}$ states. The exponentiality of this construction raises the question whether population protocols can be more succinct: Is it e.g. possible to compute any φ with only poly($|\varphi|$) states?

We investigate two cases: the case where φ belongs to the class $\{x \geq c \mid c \in \mathbb{N}\}$ of counting predicates, and the more general case where φ is an arbitrary QFPA formula.

4.2 New Contributions

The following ideas were developed in two separate papers ([BEJ18a] and [BEG$^+$20]) that build on one another. Some results of [BEJ18a] were subsumed by later results in [BEG$^+$20]; we only discuss the theorems in their most general form. We first give an overview of our contributions. After this, in subsections 4.2.3 - 4.2.6, we give further explanations for some results that are of independent interest.

4.2.1 Overview for $\mathbf{x \geq c}$

In the first part of [BEJ18a] we established upper and lower bounds on the number of states needed to compute $\{x \geq c \mid c \in \mathbb{N}\}$.

Lower bound for leaderless protocols. We established the following lower bound for leaderless protocols:

Theorem 3. *[BEJ18a] For every family $\{\mathcal{P}_c : c \in \mathbb{N}\}$ of leaderless population protocols such that $\mathcal{P}_c$ computes $x \geq c$, there exist infinitely many c such that $\mathcal{P}_c$ has at least $(\log c)^{1/4}$ states.*

Theorem 3 follows from a simple counting argument that bounds the number of unary predicates computable by leaderless protocols with a certain number of states.

A $\mathbf{O(\log\log c)}$ upper bound for some c. Theorem 3 merely states that *in general*, at least $\log(c)^{\frac{1}{4}}$ states are needed to compute $x \geq c$. This raises the question whether this lower bound holds for *all* infinite families of thresholds ($c_1 < c_2 < \ldots$). Could it e.g. be possible, that $x \geq c$ is computable with $\mathcal{O}(\log\log c)$ states for infinitely many c? – At first glance, this appears counterintuitive. After all, the double-logarithmic bound implies that a single agent does not have enough memory to store even a *single bit* of c. However, we managed to exhibit a family of thresholds $\{c_i\}_{i\in\mathbb{N}}$ s.t. $x \geq c_i$ can be computed with $O(\log\log c_n)$ states and two leaders:

Theorem 4. *[BEJ18a] There exists a family $\{\mathcal{P}_n : n \in \mathbb{N}\}$ of population protocols with two leaders, and values $c_0 < c_1 < \ldots \in \mathbb{N}$, such that $\mathcal{P}_n$ has $O(\log\log c_n)$ states and computes the predicate $x \geq c_n$ for every $n \in \mathbb{N}$.*

The proof of Theorem 4 is based on a construction in [Huy85], which exhibits a family of commutative semigroup representations with a designated start symbol s and a final symbol f, such that the smallest word beginning with f that is derivable from s is double-exponential in length. Using the auxiliary two leaders, a population protocol simulates possible derivations of a word beginning with f, starting from the symbol s. If the population size is $\geq 2^{2^n}$, then the protocol has enough tokens at its disposal to always eventually reach the final symbol f. Conversely, if there are not enough agents, then f is never reached. This establishes double-logarithmic succinctness for some $x \geq c_n$.

Upper bound for 1-aware protocols. Is it possible to improve the lower bound of Theorem 4 even further to $O(\log\log\log c_n)$ states? – This question is still open, but we managed to establish a negative result for the class of protocols where the occurrence of a single state of opinion 1 ensures stabilization to 1. We call these protocols *1-aware* protocols. *E.g.* in Example 3.2.1, as soon as an agent transitions to the state c of opinion 1, the execution is guaranteed to stabilize to consensus 1. In [BEJ18a], we show:

Theorem 5. *[BEJ18a] Every 1-aware protocol (leaderless or not) computing $x \geq c$ has at least $(\log\log(c)/151)^{1/9}$ states.*

The proof uses the fact that in 1-aware protocols, stabilization reduces to *coverability*: A configuration C *covers* a state q if there is a configuration C' such that $C'(q) > 1$ and $C \xrightarrow{*} C'$. By definition of 1-awareness, an input configuration C is accepting in a 1-aware protocol if and only if there exists a state q covered by C such that $O(q) = 1$. We then obtain the lower bound via Rackoff's procedure for coverability in vector addition systems [Rac78].

4.2.2 Overview for QFPA

Our central contribution in [BEG+20] is the following theorem:

Theorem 6. *[BEG+20] Every QFPA predicate φ can be computed by a leaderless protocol $\mathcal{P}$ with $\mathrm{poly}(|\varphi|)$ states. Moreover, $\mathcal{P}$ can be constructed in polynomial time.*

The proof of Theorem 6 is based on a construction that was first introduced in [BEJ18a] for the special case of systems of linear inequalities. The proof is quite involved. The construction of the protocol requires assembling a host of gadgets into a succinct protocol. However, the core ideas of the construction are simple. We now sketch the basic ingredients; in the following explanations, we simply call a translation from a subset of QFPA to population protocols *succinct*, if the resulting protocol has $\mathrm{poly}(|\varphi|)$ states per input formula φ.

The standard translation from QFPA to population protocols introduced by Angluin *et al.* is exponential due to the following two factors:

1. Counting in unary like in Example 3.2.1 results in an exponential blow-up. For example, the standard protocol for $x \geq c$ has $|\{0, 1, 2, \ldots, c\}| = c + 1$ states, which is exponential in $|\varphi|$ (recall that c is encoded in binary). For this reason, a protocol has $2^{\mathrm{poly}(|\varphi|)}$ states per atomic predicate φ in the standard translation.

2. Boolean combinations like e.g. $\varphi_1 \wedge \ldots \wedge \varphi_n$ are implemented through n-fold application of the product construction in the standard translation, which blows up exponentially: If $\mathcal{P}_i$ computes φ_i with m_i states, then the product has $(m_1 \cdot m_2 \cdot \ldots \cdot m_n)$ states.

To overcome these two difficulties, we provide succinct constructions for any QFPA formula φ in a sequence of steps in [BEG$^+$20]: We first provide a succinct protocol for φ with leaders; then we show how to eliminate leaders, provided that the population size is large enough; in a separate step, we show how to compute φ in small populations succinctly in a leaderless protocol; finally, the protocols for large and small populations are assembled into a succinct, leaderless protocol that computes φ.

Succinct protocols with leaders. We start by relaxing the requirement of Theorem 6 to be leaderless, and allow $\mathcal{O}(|\varphi|^3)$ leaders. This amount of leaders suffices to overcome the two difficulties above:

Theorem 7. *[BEG$^+$20] Every QFPA formula φ is computable by a population protocol with $\mathcal{O}(|\varphi|^3)$ leaders and* $\mathrm{poly}(|\varphi|)$ *states.*

We avoid the first difficulty by representing each number in binary with bits scattered across the population, and by performing all additions and subtractions in binary. *E.g.* instead of a single agent representing, say, $5 = 2^2 + 2^0$, we may represent the number 5 with two agents, one in state 2^2 and one in state 2^0. This way, only $\mathcal{O}(\log(|c|))$ states are needed per constant c. Decomposing numbers into bits requires a reservoir of spare tokens beyond the original input; tokens of numerical value 0 can be reused to split input tokens into their binary representation, and thus only a moderate number of additional tokens is needed. These additional tokens can be supplied by $l \in \mathcal{O}(|\varphi|^3)$ leaders (see Subsection 4.2.5 for a detailed explanation).

The second difficulty could in principle be avoided by putting the protocols for the atomic predicates "side by side" (instead of composing them in parallel), by dispatching identical copies of the input tokens to each individual protocol, and by collecting the outcome of each protocol in a bottom-up fashion, following the structure of the syntax tree of φ. However, copies of the input would require a non-constant amount of extra agents beyond the agents that are initialized with the input. So we have to modify this approach a little bit: The individual n protocols do not each receive a full copy of the input; instead, each protocol receives (roughly) the fraction $\frac{1}{n}$ of the input, and the individual protocols work under the assumption that input tokens carry n times their original value. To put this in vivid terms, we deflate the currency in order to pay more with the same number of coins (see Subsection 4.2.4 for a detailed explanation).

Succinct leaderless protocols for large populations. The leaders in Theorem 7 merely provide additional tokens for the computation. This entails the following monotonicity property: if the protocol works with l leaders, then it also works correctly with $l' > l$ leaders. The monotonicity property enables the simulation of the leaders in a leaderless protocol, by means of a leader election (recall that the leader election works by elimination of leaders). As long as there are *enough* agents in the population, it is possible to simulate the $l \in \mathcal{O}(|\varphi|^3)$ leaders with constant blow-up in memory per agent. This gives the following theorem:

Theorem 8. *[BEG⁺20] For every predicate φ, there exists some $l \in \mathcal{O}(|\varphi|^3)$, such that $|\boldsymbol{x}| \geq l \Rightarrow \varphi(\boldsymbol{x})$ can be computed by a succinct, leaderless protocol.*

Succinct leaderless protocols for small populations. For small population sizes, we must use a different approach. Under the assumption that the population size is smaller than $l \in \mathcal{O}(|\varphi|^3)$, agents may eventually learn their number through simple counting in states $1, 2, \ldots$, up to at most l. Knowledge of the population size empowers agents to simulate a restricted form of register machine, which performs computations on inputs of size at most l. Moreover, it can be shown that every QFPA formula can be represented succinctly in such a register machine. From these insights, we derive the following theorem (see Subsection 4.2.6 for details):

Theorem 9. *[BEG⁺20] For every QFPA formula φ and every positive integer l, there is a succinct, leaderless protocol that computes $\boldsymbol{x} < l \Rightarrow \varphi(\boldsymbol{x})$.*

Succinct leaderless protocols for QFPA. Finally, the synchronous product of the succinct protocols for $\boldsymbol{x} \geq\Rightarrow \varphi(\boldsymbol{x})$ and $\boldsymbol{x} < l \Rightarrow \varphi(\boldsymbol{x})$ gives a succinct, leaderless protocol that computes $\varphi(\boldsymbol{x})$.

Figure 4.1 outlines the structure of the proof.

4.2.3 Multi-Way Protocols

In the standard model of population protocols, agents are only allowed to interact in pairs. For the presentation of our succinct constructions, it is useful to let $k > 2$ agents interact *simultaneously.* This means we would like to use k-way transitions of the form $t\colon q_1, q_2, \ldots, q_{k-1}, q_k \rightarrow r_1, r_2, \ldots, r_{k-1}, r_k$. The usual notions from 2-way transitions carry over to k-way transitions: t is enabled at C if $\wr q_1, \ldots, q_k \int \leq C$, and by $C \xrightarrow{t} C'$ we denote that t is enabled at C and $C' = C - \wr q_1, \ldots, q_k \int + \wr r_1, \ldots, r_k \int$.

Call a protocol of maximum transition arity k a k-way protocol. The restriction to 2-way protocols in the standard model is inessential. In [BEJ18a], we show that every well-specified k-way protocol can be translated to an equivalent 2-way protocol with polynomial blow-up:

Theorem 10. *[BEJ18a] Let $\mathcal{P} = (Q, T, I, L, O)$ be a well-specified k-way population protocol. For every $3 \leq i \leq k$, let n_i be the number of i-way transitions of $\mathcal{P}$. There exists a 2-way population protocol $\mathcal{P}'$, with at most $|Q| + \sum_{3 \leq i \leq k} 3i \cdot n_i$ states, which is well-specified and computes the same predicate as $\mathcal{P}$.*

As Theorem 10 suggests, every k-way transitions of the form $q_1, q_2, \ldots, q_{k-1}, q_k \rightarrow r_1, r_2, \ldots, r_{k-1}, r_k$ can be implemented using only pairwise transitions and $\mathcal{O}(k)$ auxiliary states. The idea is to "collect" agents in the predecessor states $q_1, q_2, \ldots, q_{k-1}, q_k$, one by one, through a chain of pairwise interactions, and to temporarily "freeze" collected agents so that they do not participate in multiple transitions simultaneously. As long as the state q_k has not been collected, we ensure that this chain can be reversed, so that the computation does not get stuck if the k-way transition is not enabled in the first place. Finally, when the last state q_k is collected, the agent in states $q_k, q_{k-1}, \ldots, q_1$, are

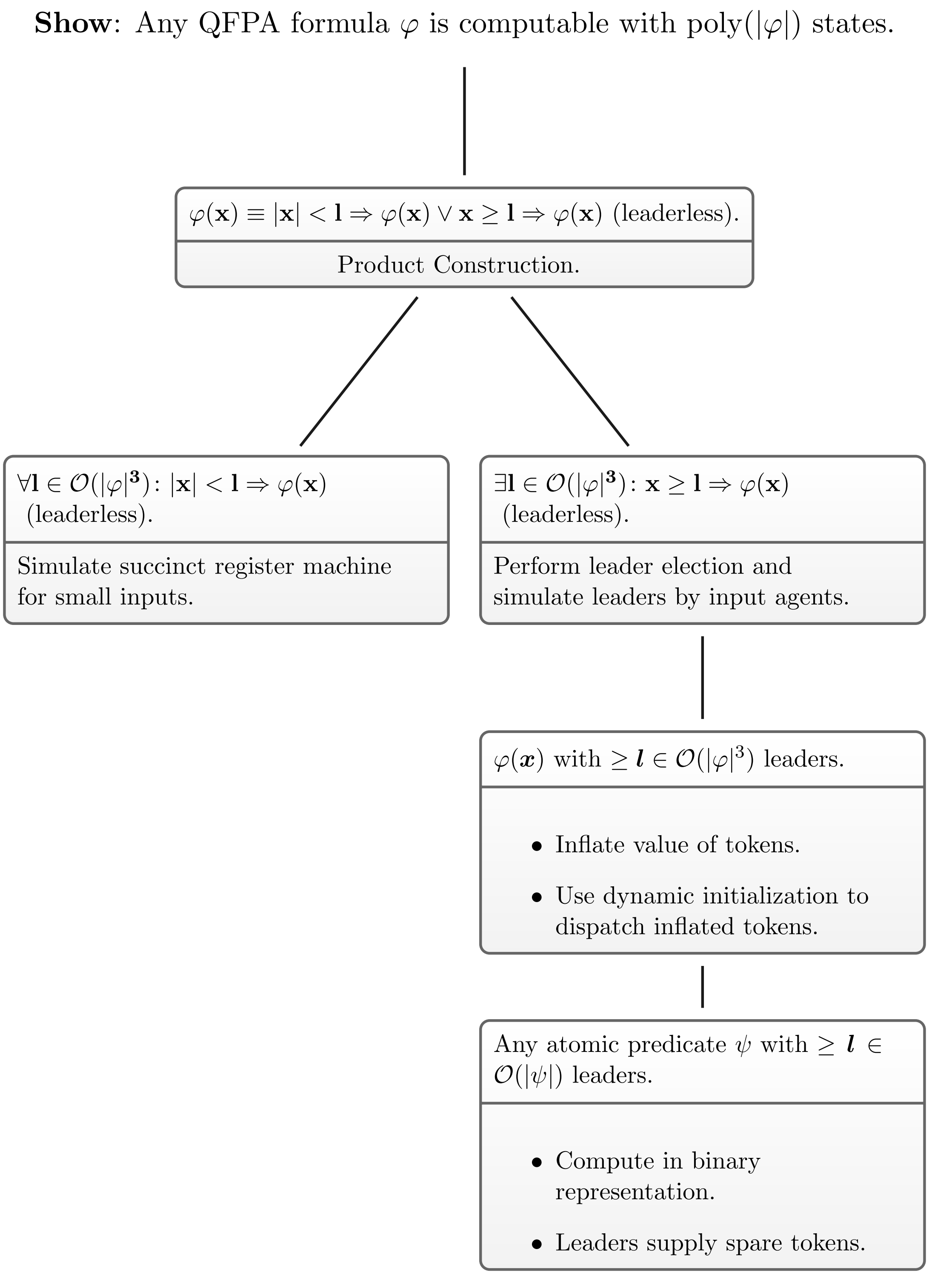

Show: Any QFPA formula φ is computable with poly($|\varphi|$) states.
$\varphi(\mathbf{x}) \equiv |\mathbf{x}| < \mathbf{l} \Rightarrow \varphi(\mathbf{x}) \vee \mathbf{x} \geq \mathbf{l} \Rightarrow \varphi(\mathbf{x})$ (leaderless).
Product Construction.
$\forall \mathbf{l} \in \mathcal{O}(|\varphi|^{\mathbf{3}})$: $|\mathbf{x}| < \mathbf{l} \Rightarrow \varphi(\mathbf{x})$ (leaderless).
Simulate succinct register machine for small inputs.
$\exists \mathbf{l} \in \mathcal{O}(|\varphi|^{\mathbf{3}})$: $\mathbf{x} \geq \mathbf{l} \Rightarrow \varphi(\mathbf{x})$ (leaderless).
Perform leader election and simulate leaders by input agents.
$\varphi(\boldsymbol{x})$ with $\geq \boldsymbol{l} \in \mathcal{O}(|\varphi|^3)$ leaders.
• Inflate value of tokens.
• Use dynamic initialization to dispatch inflated tokens.
Any atomic predicate ψ with $\geq \boldsymbol{l} \in \mathcal{O}(|\psi|)$ leaders.
• Compute in binary representation.
• Leaders supply spare tokens.

Figure 4.1: Top-down tree view of the proof structure for Theorem 6. The upper part of each node states what kind of predicate can be computed in a succinct protocol (with or without leaders). The lower part of each node sketches the implementation, assuming the claims of its children hold.

turned into the successor states $r_k, r_{k-1}, \dots, r_1$, one by one, through a reverse chain of pairwise interactions.

4.2.4 Dynamic Initialization for Succinct Parallelism

For succinct representation of boolean combinations of atomic predicates, we introduce a sophisticated method for asynchronous, parallel computation of protocols in [BEG+20]. We illustrate the idea in a simple example. To this end, let φ_1 and φ_2 be atomic predicates defined over a single variable. Further assume φ_i is computable with m_i states. We would like to compute $\varphi(x) = \varphi_1(x) \vee \varphi_2(x)$ using $\mathcal{O}(m_1 + m_2)$ states, in contrast to the $\Omega(m_1 \cdot m_2)$ states required by the product construction. Naively, we might "dispatch" the input tokens to the protocols computing φ_1 and φ_2, respectively, but doing so in parallel would require twice as many tokens: the input tokens for the first protocol, and a copy of the input tokens for the second protocol. However, instead of computing $\varphi_1(x) \vee \varphi_2(x)$, the protocol may equivalently compute

$$\bigvee_{i=1,2} \varphi_i'((x \div 2), (x \bmod 2)),$$

where the φ_i' are predicates satisfying $\varphi_i(x) \iff \varphi_i'((x \div 2), (x \bmod 2))$ for every i. Assume the existence of protocols $\mathcal{P}_i$ that compute φ_i' using $\mathcal{O}(m_i)$ states, and further assume there is a way to dispatch every second input token as left input to $\mathcal{P}_i$ for every i, and to dispatch the remainder $x \bmod 2$ as second input to each protocol $\mathcal{P}_i$. Then only two additional tokens are needed to represent the remainder, which solves our previous issue of requiring a copy of the input tokens. The constant number of additional tokens needed in the new construction can be easily dealt with.

There is one caveat: When the protocol starts dispatching the input to the protocols $\mathcal{P}_i$, the remainder $(x \bmod 2)$ is not known. At some point, the protocol needs to *guess* whether the remainder equals 1 or 0, and it must dispatch the remainder to the protocols $\mathcal{P}_i$ accordingly. This guess may be incorrect. If the guess is incorrect, then the input will never be emptied. Thus, as long as there are still tokens in the input state, the protocols must be able to "rewind" the computation of the protocols $\mathcal{P}_i$, and return tokens to the input, in order to make a new guess. However, not every protocol can be safely "rewinded" this way: it is additionally required that all transitions of $\mathcal{P}_i$ can be safely reversed, and that removing input tokens and putting tokens back from the input between computations is safe. In [BEG+20], we formalize these requirements in a class of protocols we call *protocols with reversible dynamic initialization.* We further show that for every atomic predicate ψ, one may construct a succinct protocol with reversible dynamic initialization, using only $\mathcal{O}(|\psi|)$ leaders. We generalize this idea to arbitrary boolean combinations of threshold and remainder predicates in order to derive Theorem 7.

4.2.5 Computing Atomic Predicates with $l \in \mathcal{O}(|\varphi|)$ Leaders

We now sketch how to construct for every atomic predicate φ a succinct protocol with $l \in \mathcal{O}(|\varphi|)$ leaders. We illustrate the construction in a sequence of examples with increasing difficulty:

1. The family of predicates $x \geq c$ with parameter $c > 0$,

2. The family of predicates $\alpha x \geq c$ with parameters $\alpha, c > 0$,

3. The family of threshold predicates $\sum_i \alpha_i \cdot x_i > c$ with $\alpha_i, c \in \mathbb{Z}$.

Protocol for $\mathbf{x \geq c}$. Let us first construct a protocol that computes $\varphi_c \overset{\text{def}}{=} x \geq c$ with $\text{poly}(|\varphi_c|) = \text{polylog}(c)$ states. If $c = 2^i$ for some i, then the protocol may count up to c in binary powers, using states $\{0\} \cup \{2^0, 2^1, \ldots, 2^i\}$, and transitions $2^j, 2^j \to 2^{j+1}, 0$ for every $0 < j < i$, and $c, q \to c, c$ for every $q \in Q$. The input/output functions I and O are defined as in Example 3.2.1. The resulting leaderless protocol has size polynomial in $\mathcal{O}(|\varphi_c|) = \mathcal{O}(\log c)$ and computes φ_c.

However, if c is not a power of 2, the construction for a protocol with $\mathcal{O}(\text{polylog}(c))$ states needs to be modified. Let $c > 0$ be an arbitrary positive integer, and let $0 \leq i_1 < \ldots < i_m$ be such that $c = 2^{i_1} + 2^{i_2} + \ldots + 2^{i_m}$. We define a (tentative) construction of a protocol $\mathcal{P}_c = (Q, \Sigma, T, I, O)$ that computes φ_c using $\mathcal{O}(\text{poly}(|\varphi|)) = \mathcal{O}(\text{polylog}(c))$ states as follows. Set

$$Q \overset{\text{def}}{=} \{0, c\} \cup \{2^0, 2^1, \ldots, 2^{i_m}\}$$

For every $0 \leq i < i_m$, let

$$\texttt{add}_i \colon 2^i, 2^i \to 2^{i+1}, 0.$$

Since c is not a power of 2, a transition is needed that "collects" the binary representation of c, scattered across multiple agents, into a single agent representing c.

$$\texttt{collect}_c \colon 2^{i_1}, 2^{i_2}, \ldots, 2^{i_m} \to c, \underbrace{0, \ldots, 0}_{(m-1) \text{ times}}.$$

The transition $\texttt{collect}_c$ is i_m-way. Technically, only 2-way transitions are allowed in the standard model. However, the transition $\texttt{collect}_c$ is a k-way transition with $k \in \mathcal{O}(\log c)$, and can by Theorem 10 be converted to pairwise transitions using $\text{poly}(k) = \text{polylog}(c)$ states. Like in the old protocol, I maps x to 1, the threshold state c converts every other state to c, and c is the only state of opinion 1. The construction thus far is not correct: As a counter example consider *e.g.* threshold $c = 5 = 2^2 + 1$, and input $x = 8 > c$. Given these values for x and c, the terminal configuration $\llbracket 2^2, 2^2 \rrbracket$ of consensus 0 is reachable from the initial configuration via the sequence $\texttt{add}_0; \texttt{add}_0; \texttt{add}_1; \texttt{add}_1$, but the protocol should stabilize to consensus 1. However, it is readily seen that whenever $x \geq c$ holds, there is a run starting in the initial configuration C_x that stabilizes to 1 after firing $\texttt{collect}_c$. Conversely, if $x < c$, then $\texttt{collect}_c$ is disabled in all configurations reachable from the initial configuration, and consequently all runs starting in C_x stabilize to 0. In order to fix the protocol, it suffices to ensure the execution can always return to the initial configuration, as long as $\texttt{collect}_c$ has not been fired. This property can be implemented by adding a reverse transition $\texttt{add}_i^{-1} \colon 0, 2^{i+1} \to 2^i, 2^i$ for every transition $\texttt{add}_i \colon 2^i, 2^i \to 2^{i+1}, 0$.

The full protocol computing $x > c$ with polylog(c) states can thus be specified as [BEJ18a]:

$$
\begin{aligned}
Q &= \{0\} \cup \{2^0, 2^1, \ldots, 2^{i_m}\},\\
T &= \{\mathtt{add}_i, \mathtt{add}_i^{-1} \mid 0 \le i < i_m\} \cup \{\mathtt{collect}_c\} \cup \{c, q \to c, c \mid q \in Q\},\\
I(x) &= 1\\
O(q) &= \begin{cases} 1 & \text{if } q = c,\\ 0 & \text{otherwise.}\end{cases}
\end{aligned}
$$

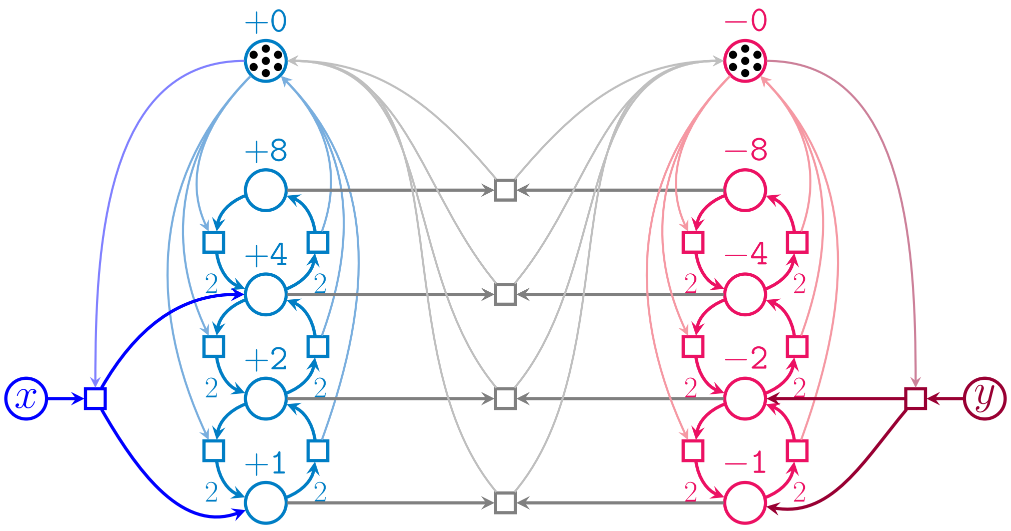


Figure 4.2: Graphical representation of a population protocol with 14 leaders computing $5x - 3y > 0$. Outer circles represent states, squares represent transitions, and filled circles represent number of agents in each state (image source: [BEJK18]).

Protocol for $\alpha \cdot \mathbf{x} \ge \mathbf{c}$. Now let us consider a slightly more general family of predicates: Given some some threshold $c > 0$ and some positive integer coefficient $1 < \alpha < c$, we would like to compute the predicate $\varphi_{\alpha,c} = \alpha x \ge c$ in a population protocol $\mathcal{P}_{\alpha,c}$ with $\mathcal{O}(|\varphi_{\alpha,c}|)$ states and leaders. For every input x, the corresponding initial population of $\mathcal{P}_{\alpha,c}$ must represent the value $\alpha \cdot x$ using x agents, thus initially each agent carries the value $\alpha > 1$ in $\mathcal{P}_{\alpha,c}$. The value α may not be a power of 2, and thus we cannot use the approach for the construction of $\mathcal{P}_c$, where all agents carry a power of 2 up to the point where $\mathtt{collect}_c$ is fired. Even if α is a power of 2, the reverse transitions $\mathtt{add}_i^{-1}$ may no longer avoid getting stuck prematurely, as there may not be enough agents to always go back to a configuration that enables $\mathtt{collect}_c$. However, if leaders are allowed, this issue can be avoided by adding a sufficiently large supply of leaders in state 0 that provide “spare tokens”: Let m denote the length of the binary representation of c. Define the states of $\mathcal{P}_{\alpha,c}$ as $\{0, c, \alpha\} \cup \{2^i \mid 0 \le i \le m+1\}$. Further let $\mathtt{add}_i, \mathtt{add}_i^{-1}$ and $\mathtt{collect}_c$ be defined as before. Assuming a sufficient supply of additional agents in state 0, it is possible to translate every agent from the initial state α to its binary representation $2^{i_1} + \ldots + 2^{i_n} = \alpha$ using $n-1$ additional agents in state 0, by means of the n-way transition

$$t_\alpha \colon \alpha, \underbrace{0, \ldots, 0}_{(n-1)\text{ times}} \to 2^{i_1}, \ldots, 2^{i_n}.$$

It is relatively straightforward to see that the following holds for every configuration C satisfying $\sum_{0\leq i\leq m} C(2^i) \geq c$: Given a sufficiently large supply of agents in state 0, there is a sequence σ of transitions of the form $\texttt{add}_i$ and $\texttt{add}_i^{-1}$ such that $C \xrightarrow{\sigma} C'$ and C' enables $\texttt{collect}_c$. Moreover, this property is forward-inductive, as the transitions $\texttt{add}_i$ and their reverse transitions leave the quantity $\sum_{0\leq i\leq m} C(2^i)$ invariant.

Thus, given enough leaders in state 0, the protocol computes $\varphi_{\alpha,c}$ succinctly. How many leaders are needed to obtain a correct protocol? – It can be shown that the number of leaders is independent of the size of the input, and that $2m$ leaders suffice for correctness. To see this, observe that a reservoir of $2m$ agents in state 0 is enough to decompose two agents in the input state α into the constituent binary powers of α via the transition t_α. Moreover, whenever two identical powers of 2 interact, one token is released. Thus, it is indeed possible to compute $\alpha x \geq c$ in a succinct protocol with $m \in \mathcal{O}(|\varphi|)$ leaders.

Protocol for $\sum_\mathbf{i} \alpha_\mathbf{i} \cdot \mathbf{x} > \mathbf{c}$. In [BEJ18a], we generalize the last construction to obtain succinct protocols for threshold predicates of the form with $l \in \mathcal{O}(|\varphi|)$ leaders serving as spare tokens. The underlying idea for threshold is similar, except that the powers of 2 are signed in order to distinguish between negative and positive coefficients, and transitions are added which guarantee stabilization to the final majority.

We now sketch how to construct for a given threshold predicate φ a protocol $\mathcal{P}$ that computes φ, using $\mathcal{O}(|\varphi|)$ states and $\mathcal{O}(|\varphi|)$ leaders. Consider for example the threshold predicate $\varphi = 5x - 3y > 0$. The protocol $\mathcal{P}$ is partially represented in Figure 4.2. The initial states x and y are translated into the (signed) binary representation of their coefficients: x is translated into $4 + 1 = 5$, and y is translated into $-3 = -2 + (-1)$. Like in the previous protocol, there is an initial reservoir of $\mathcal{O}(|\varphi|)$ zeros, serving as spare tokens, only this time there is one for each sign ($+$ or $-$). The transitions in the middle allow positive and negative values of identical absolute value to cancel out. Like in the previous protocol, binary powers can be promoted and demoted. By fairness, either all positive or all negative values are eventually eliminated through promotions/demotions and canceling. Transitions are added that ensure stabilization to the remaining majority consensus (positive or negative). For the sake of clarity, these additional majority transitions are missing from the graphical representation in Figure 4.2.

In [BEG$^+$20] we extended this approach to remainder predicates. The construction is similar.

4.2.6 Achieving Succinctness in Small Populations

Now we outline our approach for the construction of a succinct protocol under the assumption that the population size is small. Concretely, we sketch the succinct construction of a protocol $\mathcal{P}$ that computes $|\boldsymbol{x}| \leq l \Rightarrow \varphi(\boldsymbol{x})$ for any given formula φ and positive integer $l \in \mathcal{O}(|\varphi|^3)$. Let us first weaken the requirement on $\mathcal{P}$, aiming for a protocol $\mathcal{P}_l$ that computes $\varphi(\boldsymbol{x})$ in the case where $\boldsymbol{x} = l$ holds, and that may behave arbitrarily whenever $\boldsymbol{x} \neq l$. We denote this requirement by saying that $\mathcal{P}_l$ computes $(\varphi(\boldsymbol{x}) \mid |\boldsymbol{x}| = l)$. Notice that $\mathcal{P}_l$ need not be well-specified.

Construction for $\mathbf{x} = \mathbf{l}$. Since $\mathcal{P}_l$ is only required to work correctly for inputs of size l, we may assume that every initial configuration is of size l. Given this assumption, $\mathcal{P}_l$ may elect a leader by counting up to l like in the simple counting protocol from Example 3.2.1; the agent that transitions to the threshold state l becomes the leader.

Since the elected leader knows the population size l, it may label the remaining agents with unique identities in the range $[1, l-1]$ through pairwise interactions. The identities enable the leader to iterate over the population via sequences of rendez-vous with agents of identity $1, 2, \dots l-1$, in that order, keeping track of interactions in a counter ranging from 1 to $(l-1)$. This global iteration mechanism enables $\mathcal{P}_l$ to simulate a virtual register machine: The leader stores non-input registers and the control state of the register machine, while each agent stores one bit of the input. Register updates and register queries can be implemented by the iteration mechanism. Every register machine with a description of size $\mathrm{poly}(l)$ can be simulated this way by a protocol $\mathcal{P}_l$ with $\mathrm{poly}(l)$ states for all inputs of size l.

To finish the description of $\mathcal{P}_l$, it thus suffices to show how any QFPA predicate φ can be represented succinctly in a register machine for any input of size $l \leq \mathrm{poly}(|\varphi|)$. Boolean combinations of atomic predicates can be represented succinctly through sequential composition, given that the representation of atomic predicates is succinct. Among the atomic predicates, it suffices to only consider threshold predicates; remainder predicates can be succinctly represented by a disjunction of threshold predicates for inputs of size l [BEG⁺20]. Here we only consider threshold predicates of the form

$$\varphi(\boldsymbol{x}, \boldsymbol{y}) = \alpha_1 x_1 + \ldots + \alpha_n x_n > \beta_1 y_1 + \ldots + \beta_m y_m.$$

where $\alpha_i > 0$ and $\beta_j > 0$ for all i and j. The more general threshold predicate can be implemented similarly.

The obvious approach to the computation of $\varphi(\boldsymbol{x}, \boldsymbol{y})$ would be to compute the sums $A = \boldsymbol{\alpha} \cdot \boldsymbol{x}$ and $B = \boldsymbol{\beta} \cdot \boldsymbol{y}$, and to finally compare A and B. But summation would require $\Theta(2^{|\varphi|})$ different register machine configurations, which cannot be represented succinctly. Another approach would be to store the result of all comparisons $\boldsymbol{\alpha} \cdot \boldsymbol{x} > \boldsymbol{\beta} \cdot \boldsymbol{y}$ in a lookup table. But there are $\Theta(2^{|\varphi|})$ possible pairings of $\boldsymbol{x}$ and $\boldsymbol{y}$, hence this approach also does not yield a succinct machine.

However, in order to decide the inequality $\boldsymbol{\alpha} \cdot \boldsymbol{x} > \boldsymbol{\beta} \cdot \boldsymbol{y}$, it is not necessary to store all digits of $\boldsymbol{\alpha} \cdot \boldsymbol{x}$ and $\boldsymbol{\beta} \cdot \boldsymbol{y}$ *simultaneously*: Since $|\boldsymbol{x}|$ and $|\boldsymbol{y}|$ are bounded by l, the sums $\boldsymbol{\alpha} \cdot \boldsymbol{x}$ and $\boldsymbol{\beta} \cdot \boldsymbol{y}$ can be represented by a fixed-length binary sequence $a_1, \dots, a_m$ and $b_1, \dots, b_m$ such that $m = \mathcal{O}(l)$. Without loss of generality, assume 1 is the least-significant bit position. Then $\boldsymbol{\alpha} \cdot \boldsymbol{x} > \boldsymbol{\beta} \cdot \boldsymbol{y}$ holds if and only if there is some $1 \leq k \leq m$ such that $a_k > b_k$ and $a_i = b_i$ for all $i < k$. Consequently, the inequality $\boldsymbol{\alpha} \cdot \boldsymbol{x} > \boldsymbol{\beta} \cdot \boldsymbol{y}$ can be decided by searching for the smallest $k = 1, 2, \dots, m$, such that $a_k < b_k$ or $a_k > b_k$ holds. If no such k exists, then $\boldsymbol{\alpha} \cdot \boldsymbol{x}$ and $\boldsymbol{\beta} \cdot \boldsymbol{y}$ are equal. This approach requires the machine to store the current index $k \leq m$, and to probe the bits a_k and b_k. Probing of the bits a_k and b_k can be implemented succinctly via a "forgetful" variant of the standard algorithm for binary addition, where only the carry is kept when moving from one bit position to the next, and the result of the summation for previous digits of lower significance is discarded. This gives a succinct register machine for inputs of size l.

Construction for $\mathbf{x} \leq \mathbf{l}$. Given that for every $2 \leq i \leq l$, there is a succinct protocol $\mathcal{P}_i$ that computes $(\varphi(\boldsymbol{x}) \mid |\boldsymbol{x}| = i)$, we construct a succinct protocol $\mathcal{P}$ that computes $|\boldsymbol{x}| < l \Rightarrow \varphi(\boldsymbol{x})$. The protocol $\mathcal{P}$ works as follows: Agents start by counting the population size from 1, up to at most l. Whenever the counter is incremented to some value i, an agent resets a subpopulation of size i to the initial configuration of $\mathcal{P}_i$, and initiates a simulation of $\mathcal{P}_i$. If $|\boldsymbol{x}| \geq l$, then some agent's counter reaches l, and this is propagated to stabilize to the correct value 1. If $|\boldsymbol{x}| = i < l$, then $\mathcal{P}$ eventually executes $\mathcal{P}_i$, and stabilizes to the output of $\mathcal{P}_i$, which equals $\varphi(\boldsymbol{x})$.

4.3 Related Work

In our analysis of space complexity, we assume a uniform model of computation: Every protocol computes a predicate for all (infinitely many) inputs. In a variety of publications [AAE+17, AGV15, AG15, ABBS16], time/space trade-offs have been considered for non-uniform computation of population protocols, where the number of states available to each agent depends on the size of the input. The research on time/space trade-offs was motivated by the result that leader election can be solved in $\mathcal{O}(\log^3(n))$ time in a protocol with $\mathcal{O}(\log^3(n))$ states in populations of size n [AG15]. This stands in contrast to leader election with constant memory, which requires linear time [DS18].

The space complexity of counting in population protocols has been studied in a number of papers [BBCS15, ABBS16, BCM+07, BBC14]. In [BBCS15], counting is considered in the setting where agents may be initialized arbitrarily, and there is an additional leader in a designated initial state referred to as the *base station*. The authors show that counting can be implemented in this setting with only one bit per regular agent under global fairness. They also give a space-optimal protocol for counting with a base state under local fairness. In [ABBS16], a space and time optimal protocol under uniformly probabilistic scheduling is presented for arbitrarily initialized agents with base station. The optimal protocol requires one bit per agent and converges in $\mathcal{O}(n \log n))$ expected time. A number of recent publications [DLBBC14, MCS13, MM15] consider the space complexity of the counting problem in networks of anonymous agents in a *synchronous* model of computation.

Counting protocols notwithstanding, space complexity in the uniform model of computation has attracted surprisingly little attention by the research community. In [MNRS14], the majority protocol from Example 3.2.2 is shown to be space-optimal. To the best of our knowledge, we are the first to improve the exponential construction presented by Angluin *et al.* in [AAD+06] for the entire class of QFPA.

4.4 Open Research Questions

No lower bound is known for the number of states required to represent predicates in population protocols with constant number of leaders. In particular, the following question is still open: Is there an increasing sequence of positive integers $c_1, c_2, \ldots$, such that $x \geq c_i$ can be computed by protocol with a constant number of leaders and $\mathcal{O}(\log \log \log c_i)$ states for every i? This would be an improvement from the $\mathcal{O}(\log \log c_i)$ bound in Theorem 4. The existence of such a sequence seems implausible, but as of now no proof could be found that establishes the opposite.

5 Parametric Verification

This chapter is concerned with automatic verification of population protocols. For a *fixed* input, a verification procedure can be obtained via exhaustive exploration of the state space: Given some input $\boldsymbol{v}$, some protocol $\mathcal{P}$ and some predicate φ, the procedure may decide whether every fair run starting in the initial configuration $C_{\boldsymbol{v}}$ stabilizes to $\varphi(\boldsymbol{v})$ by inspecting the BSSCs of the reachability graph of $C_{\boldsymbol{v}}$.

Here we are primarily interested in the *parameterized* variant of the verification problem, *i.e.* we would like to know whether a given protocol computes the correct output for *all, infinitely many* inputs (instead of a single input $\boldsymbol{v}$). The problem is thus parameterized over all inputs.

5.1 Problem Statement

We call the parameterized verification problem the *correctness problem.* Formally, this is the following decision problem.

Correctness
Input: A population protocol $\mathcal{P}$ and a QFPA formula φ.
Question: Does $\mathcal{P}$ compute the predicate specified by φ?

By the characterization of fairness via BSSCs, deciding whether a protocol $\mathcal{P}$ computes a predicate φ amounts to deciding whether the following formula is *unsatisfiable* [BEJK18]:

$$\begin{aligned}\exists C, D : \quad & C \xrightarrow{*} D && (5.1)\\ & \wedge\ C \text{ is initial} && (5.2)\\ & \wedge\ D \text{ is bottom} && (5.3)\\ & \wedge\ O(D) \neq \varphi(C) && (5.4)\end{aligned}$$

In words: $\mathcal{P}$ does *not* compute φ, if there exists a bottom configuration D reachable from some initial population C, such that the output of D does not equal $\varphi(C)$.
A related problem is the *well-specification problem.*

WellSpecification
Input: A population protocol $\mathcal{P}$.
Question: Is $\mathcal{P}$ well-specified?

By the characterization of fairness via BSSCs, deciding WellSpecification amounts to deciding whether the following formula is *unsatisfiable*:

$$\begin{aligned}\exists C, D_1, D_2 : \quad & C \xrightarrow{*} D_i \text{ for every } i \in \{1,2\} && (5.5)\\ & \wedge\ C \text{ is initial} && (5.6)\\ & \wedge\ D_i \text{ is bottom for every } i \in \{1,2\} && (5.7)\\ & \wedge\ (O(D_1) = \bot \vee O(D_1) \neq (D_2)) && (5.8)\end{aligned}$$

In words: $\mathcal{P}$ is *not* well-specified, if two bottom configurations of differing output are reachable from the same initial configuration, or a non-consensus bottom configuration is reachable from an initial configuration.

WellSpecification and Correctness are polynomially interreducible.

Reduction from Correctness to WellSpecification. Given $\mathcal{P}$ and φ for Correctness, a polynomial-time procedure may by Theorem 6 construct a protocol $\mathcal{P}_\varphi$ that computes φ . The procedure then constructs a protocol $\mathcal{P}'$ that executes $\mathcal{P}_0 = \mathcal{P}$ and $\mathcal{P}_1 = \mathcal{P}_\varphi$ in parallel while alternating nondeterministically between the output of $\mathcal{P}_0$ and $\mathcal{P}_1$. A state in $\mathcal{P}'$ is of the form $(q_0, q_1, b) \in Q^{\mathcal{P}_0} \times Q^{\mathcal{P}_1} \times \{0, 1\}$: each agent carries a pair of states from $\mathcal{P}_0$ and $\mathcal{P}_1$ for the parallel execution of the two protocols, and a boolean flag $b \in \{0, 1\}$ which governs the output. The output is given by $O((q_0, q_1, b)) = O_b(q_b)$. Transitions are added to $\mathcal{P}'$ that may flip the flag b any time, and leave the other components unchanged. By construction $\mathcal{P}'$ is well-specified if and only if $\mathcal{P}_0$ and $\mathcal{P}_1$ compute the same predicate, which is precisely the case if $\mathcal{P}$ computes φ.

Reduction from WellSpecification to Correctness. The reduction from WellSpecification to Correctness is similar: Given an instance $\mathcal{P}$ for WellSpecification, a polynomial time procedure may construct a protocol $\mathcal{P}'$ that executes two copies of $\mathcal{P}$ in parallel by means of the product construction. The output of a product state (q_1, q_2) is set to 1 if and only if the output of q_1 and q_2 is identical in $\mathcal{P}$. It is readily seen that $\mathcal{P}'$ computes the constant predicate *true* if and only if $\mathcal{P}$ is well-specified.

5.2 New Contributions

In [BEJM17] we presented the first fully automatic procedure for solving a multitude of instances of Correctness and WellSpecification. We sketch the construction for Correctness; the approach for WellSpecification only requires mild modifications.

The obvious idea is to express the constraints (5.1)-(5.4) in a suitable logic and to automatically test for unsatisfiability with the help of a constraint solver. Here we face two difficulties:

1. The reachability relation $\xrightarrow{*}$ is not semilinear, and not expressible in any logic of reasonable complexity [AAER07, EGLM17].

2. Subformula (5.3) (D *is bottom*) is effectively Presburger, but there is no known upper bound on the norm of the corresponding semilinear set [EGLM17]. Known approaches for construction of a formula equivalent to (5.3) are impractical.

We resolve the first issue by replacing $\xrightarrow{*}$ by some overapproximation $\overset{*}{\dashrightarrow} \supseteq \xrightarrow{*}$; the overapproximation $\overset{*}{\dashrightarrow}$ will be expressible in the existential fragment of Presburger logic.

We resolve the second issue by confining our approach to *silent* protocols. A protocol is silent if every fair run ends up in a *terminal* configuration; D is terminal if $D \rightarrow D'$ implies $D = D'$ for every D'. In the case of silent protocols, we may replace Subformula (5.3) (D *is bottom*) by the constraint (D *is terminal*). The constraint (D *is terminal*) is definable in the quantifier-free fragment of Presburger arithmetic – we only need to verify that effectful transitions are disabled at D, which is expressible as a boolean combination of inequalities.

Example 5.2.1. The majority protocol from Example 3.2.2 is silent. The reachable terminal configurations are of the type $\wr\mathtt{X},\ldots,\mathtt{X},\mathtt{x},\ldots,\mathtt{x}\wr$ (if $x > y$), $\wr\mathtt{Y},\ldots,\mathtt{Y},\mathtt{y},\ldots,\mathtt{y}\wr$ (if $y > x$), or $\wr\mathtt{x},\ldots,\mathtt{x}\wr$ (if $x = y$).

The majority of protocols in the literature is naturally silent. However, switching to silent protocols only seems to shift the problem: to guarantee soundness of our approach, we now have to verify that the protocol is silent, which is also hard. To bypass this issue, we replace the property ($\mathcal{P}$ *is silent*) by a simpler sufficient criterion we call LayeredTermination. Deciding LayeredTermination is in NP, and hence of reasonable complexity.

LayeredTermination. A protocol $\mathcal{P}$ satisfies LayeredTermination, if the set of effectful transitions of $\mathcal{P}$ can be partitioned into $T_1 \uplus T_2 \uplus \ldots \uplus T_m$ for some m, such that the following holds for every configuration $C \in \mathbb{N}^Q$:

1. For every $1 \leq k \leq m$, every execution $C \xrightarrow{t_1} C_1 \xrightarrow{t_2} C_2 \xrightarrow{t_3} \ldots$, such that $t_i \in T_k$ for every i, is finite.
2. The following property is forward-inductive (*i.e.* closed under reachability) for every $1 \leq k \leq m$: All transitions in $T_1 \cup \ldots \cup T_k$ are disabled.

Example 5.2.2. The majority protocol from Example 3.2.2 satisfies Layered Termination. It is easily verified that the following partition $T_1 \uplus T_2$ of its transition relation satisfies the required properties:

$$T_1 = \{(\mathtt{X},\mathtt{Y}) \to (\mathtt{x},\mathtt{y}), (\mathtt{Y},\mathtt{x}) \to (\mathtt{Y},\mathtt{y})\}$$
$$T_2 = \{(\mathtt{X},\mathtt{y}) \to (\mathtt{X},\mathtt{x}), (\mathtt{x},\mathtt{y}) \to (\mathtt{x},\mathtt{x})\}$$

Given a partition $T' = T_1 \uplus T_2 \uplus \ldots \uplus T_m$ that satisfies the items in the definition of LayeredTermination, it is always possible to reach from every configuration a terminal configuration via some sequence in $T_1^* T_2^* \ldots T_m^*$; by fairness, a terminal configuration must eventually be reached. Thus, we obtain:

Theorem 11. *[BEJM17] Every protocol that satisfies* LayeredTermination *is silent.*

Moreover, we show the following:

Theorem 12. *[BEJM17] Deciding whether a given protocol satisfies* LayeredTermination *is in* NP.

The NP procedure first guesses m and a partition $T' = T_1 \uplus \ldots \uplus T_m$. The procedure then checks whether the partition satisfies Properties 1 and 2 in the definition of LayeredTermination. It is well-known that the first property can be checked in polynomial time for vector addition systems, and thus also for population protocols. For the second property, the procedure has to verify that once the first k blocks of the partition are disabled in any execution, they cannot be re-enabled at a later point. This reduces to a simple syntactic check on the presets and postsets of the transitions, which can be carried out in polynomial time.

Overapproximating Reachability. For the approximation of the reachability relation, we use techniques familiar from the theory of Petri nets. Let $T = \{t_1, \ldots, t_n\}$ be a set of transitions. Given a finite sequence $\sigma \in T^*$, let the *Parikh vector* of σ be defined as $\boldsymbol{x}_\sigma \overset{\text{def}}{=} (|\sigma|_{t_1}, \ldots, |\sigma|_{t_n}) \in \mathbb{N}^n$ where $|\sigma|_{t_i}$ denotes the number of occurrences of t_i in σ.

For every pair of configurations C and C', $C \xrightarrow{\sigma} C'$ implies the following equality for $\boldsymbol{x} = \boldsymbol{x}_\sigma$:

$$C'(q) = C(q) + \sum_{t \in T} \boldsymbol{x}(t) \cdot (\text{post}(t)(q) - \text{pre}(t)(q)) \tag{5.9}$$

If C' is reachable from C, then there is some counting vector $\boldsymbol{x}$ that satisfies (5.9) for the fixed configurations C and C'. However, the converse need not hold: From the existence of a vector $\boldsymbol{x}$ satisfying (5.9), it does not follow that C' is reachable from C, as (5.9) only takes the cumulative effect of $\boldsymbol{x}$ into account, and consequently there may be no realizable transition sequence σ such that $\boldsymbol{x}_\sigma = \boldsymbol{x}$ for a given solution $\boldsymbol{x}$. Thus, the following relation $\overset{*}{\dashrightarrow}$ is an *overapproximation* of the reachability relation $\xrightarrow{*}$:

$$C \overset{*}{\dashrightarrow} C' \Leftrightarrow \exists \boldsymbol{x} \text{ s.t. } C, C', \boldsymbol{x} \text{ satisfy (5.9).}$$

For given C and C', checking $C \overset{*}{\dashrightarrow} C'$ amounts to solving a linear system over the integers, which can be done in nondeterministic polynomial time [BEJM17].

The relation $\overset{*}{\dashrightarrow}$ is too crude an overapproximation for our purposes. We refine it using the established concepts *traps* and *siphons* from the Petri net literature.

Definition 5.2.1. Let U be a set of transitions. A set of states P is a U-trap if the following holds for every transition $q_1, q_2 \to r_1, r_2 \in U$: $\{q_1, q_2\} \cap P \neq \emptyset \Rightarrow \{r_1, r_2\} \cap P \neq \emptyset$.

Intuitively, P is a U-trap if every transition from U that removes an agent from P must also put an agent into P.

Siphons are the dual concept to traps:

Definition 5.2.2. Let U be a set of transitions. A set of states P is a U-siphon if the following holds for every transition $q_1, q_2 \to r_1, r_2 \in U$: $\{r_1, r_2\} \cap P \neq \emptyset \Rightarrow \{q_1, q_2\} \cap P \neq \emptyset$.

Intuitively, P is a U-siphon if every transition from U that puts an agent into P must also remove an agent from P.

We make the following observation:

Observation 5.2.1. *Let $\sigma \in T^*$ and $U \overset{\text{def}}{=} \{t \mid \sigma(t) > 0\}$. Further let C, C' be configurations such that $C \xrightarrow{\sigma} C'$. Then the following holds:*

$$[\![C']\!] \cap P = \emptyset \implies [\![C]\!] \cap P = \emptyset \textit{ for every } U\textit{-trap } P, \tag{5.10}$$

$$[\![C]\!] \cap P = \emptyset \implies [\![C']\!] \cap P = \emptyset \textit{ for every } U\textit{-siphon } P. \tag{5.11}$$

We refine $\overset{*}{\dashrightarrow}$ by additionally requiring that (5.10) and (5.11) hold for all U-traps and U-siphons, respectively, where U is defined as $U \overset{\text{def}}{=} U_{\boldsymbol{x}} = \{t \in T \mid \boldsymbol{x}(t) > 0\}$ for the solution vector $\boldsymbol{x}$ of the state equation. We denote the refined relation by $\overset{*}{\dashrightarrow}_U$.

The relation $\xrightarrow{*}_U$ is our replacement for the reachability relation $\xrightarrow{*}$.

Putting the pieces together. Our approach executes the following steps:

1. Verify LayeredTermination.
2. Verify Strong-φ-Consensus: Verify that the following formula is *unsatisfiable.*

$$\begin{aligned} \exists C, D: \quad & C \xrightarrow{*}_U D && (5.12)\\ & \land\ C \text{ is initial} && (5.13)\\ & \land\ D \text{ is terminal} && (5.14)\\ & \land\ O(D) \neq \varphi(C) && (5.15) \end{aligned}$$

If both steps succeed, then $\mathcal{P}$ computes φ, otherwise the question whether $\mathcal{P}$ computes φ remains undecided. This gives a sound, albeit incomplete, decision procedure for Correctness.

It can further be shown that every Presburger predicate φ can be computed by a protocol that is verifiable by our procedure:

Theorem 13. *[BEJM17] Every predicate given by a QFPA formula φ can be computed by a population protocol that satisfies* LayeredTermination *and* Strong-φ-Consensus[1].

Complexity. Deciding the two steps has complexity DP $= \{L_1 \cap L_2 \mid L_1 \in \mathsf{NP}, L_2 \in \mathsf{coNP}\}$. The first step is in NP by Theorem 12. The unsatisfiability constraint for the second step is expressible in existential Presburger arithmetic, the syntactic fragment of Presburger arithmetic of formulas of the form $\exists \boldsymbol{x}\colon \varphi(\boldsymbol{x})$, where $\varphi(\boldsymbol{x})$ is a boolean combination of atomic predicates. It can further be shown that the size of the formula does not blow up, and that unsatisfiability of a formula in the existential fragment is in coNP. This gives membership in coNP for deciding Strong-φ-Consensus.

5.2.1 Experimental Results

We implemented our approach in a tool called Peregrine[2]. The tool takes a protocol $\mathcal{P}$ as input and constructs two formulas: one for LayeredTermination, and one for Strong-φ-Consensus. The former is satisfiable if and only if $\mathcal{P}$ satisfies LayeredTermination, the latter is *un*satisfiable if and only if $\mathcal{P}$ satisfies Strong-φ-Consensus. The constraints are solved with the help of the SMT solver Z3 [DMB08]. The traps and siphon constraints for Strong-φ-Consensus are not added all at once. Instead, a refinement loop adds trap- and siphon constraints incrementally, until the set of constraints for Strong-φ-Consensus is rendered unsatisfiable. Iterative refinement helps us avoid a quadratic blow-up in the number of variables, which would result from a direct translation of the coNP-approach into a set of constraints (see [BEJM17] for details). Usually, only few refinement steps are required, and the iterative refinement approach is expected to perform well in practice [BEJM17].

Table 5.1 illustrates how our tool performs on a set of benchmarks. The set consists of protocols introduced in the literature: the threshold and remainder protocol of [AAD$^+$06], the majority protocol of [AAE06], the broadcast protocol of [CDFS11],

[1]In [BEJM17], this result is presented in terms of a related property, called StrongConsensus; however, the proof can be easily adapted to Strong-φ-Consensus.

[2]`https://peregrine.model.in.tum.de/`

Threshold			
α	$\lvert Q\rvert$	$\lvert T\rvert$	Time
3	28	288	8.0
4	36	478	26.5
5	44	716	97.6
6	52	1002	243.4
7	60	1336	565.0
8	68	1718	1019.7
9	76	2148	2375.9
10	84	2626	timeout

Remainder			
m	$\lvert Q\rvert$	$\lvert T\rvert$	Time
10	12	65	0.4
20	22	230	2.8
30	32	495	15.9
40	42	860	79.3
50	52	1325	440.3
60	62	1890	3055.4
70	72	2555	3176.5
80	82	3320	timeout

Flock of birds [CMS10]			
c	$\lvert Q\rvert$	$\lvert T\rvert$	Time
20	21	210	1.5
25	26	325	3.3
30	31	465	7.7
35	36	630	20.8
40	41	820	106.9
45	46	1035	295.6
50	51	1275	181.6
55	56	1540	timeout

Flock of birds [CDFS11]			
c	$\lvert Q\rvert$	$\lvert T\rvert$	Time
50	51	99	11.8
100	101	199	44.8
150	151	299	369.1
200	201	399	778.8
250	251	499	1554.2
300	301	599	2782.5
325	326	649	3470.8
350	351	699	timeout

Majority		
$\lvert Q\rvert$	$\lvert T\rvert$	Time
4	4	0.1

Broadcast		
$\lvert Q\rvert$	$\lvert T\rvert$	Time
2	1	0.1

Table 5.1: Results of the experimental evaluation where $|Q|$ denotes the number of states, $|T|$ denotes the number of non silent transitions, and the time to verify WellSpecification in seconds (this experimental evaluation first appeared in [BEJM17]).

and two versions of the simple counting protocol from Example 3.2.1. For families of protocols such as threshold, we gradually increased the primary parameter until a timeout was reached. We set the the timeout to one hour. All experiments were performed on the same machine equipped with an Intel Core i7-4810MQ CPU and 16GB of RAM. Checking Correctness was generally faster than checking WellSpecification.

5.2.2 Peregrine

In[BEJ18b], we introduced a new version of Peregrine. The new version has a graphical user interface that offers the following features:

- Specification of population protocols in a graphical interface,
- Specification of families of population protocols in a text editor,
- Various modes of visualization for executions of protocols,
- Gathering Statistics of properties such as convergence speed,
- Automatic verification of correctness, with limited support for error diagnostics.

Support for error diagnostics is implemented with the help of the Petri net reachability tool LoLa [Sch00]. Since Strong-φ-Consensus is defined with respect to the overapproximation $\xrightarrow{*}_U$, a silent protocol may violate Strong-φ-Consensus, although the protocol computes φ. This means every counter example to Strong-φ-Consensus needs to be independently verified, using $\xrightarrow{*}$ instead of $\xrightarrow{*}_U$. Given configurations C and C', LoLa tries to determine whether $C \xrightarrow{*} C'$ holds. Whenever a counter example to Strong-φ-Consensus is returned during verification, the counter example is passed to LoLa to check whether it is realizable by a corresponding finite execution. The counter example is displayed to the user whenever LoLa manages to verify its validity.

5.3 Related Work

Decidability of WellSpecification. In [EGLM17] Esparza *et al.* showed that Well-Specification, and equivalently, Correctness are decidable. They further show that the reachability problem for vector addition systems is polynomially reducible to Well-Specification. The reachability problem is TOWER-hard [CLL$^+$19]. Thus any complete decision procedure for WellSpecification or Correctness is bound to be practically infeasible.

The authors obtain a decision procedure for WellSpecification by parallel composition of two semi-decision procedures, one for the problem and one for its complement. The semi-decision procedure for the complement can be obtained by enumeration of all initial configurations, and by testing for each input configuration if the protocol stabilizes to a unique output via inspection of its reachability graph. The procedure accepts the protocol when such a test fails.

The proof for semi-decidability of WellSpecification is more involved; it exploits the rich theory of vector additon systems and their correspondence to semilinear sets. Since decidability of WellSpecification is a relatively surprising result, we now explain the proof structure in greater detail.

Let $\mathcal{P}$ be a population protocol. Let $\mathcal{I}$ be the set of input configurations of $\mathcal{P}$, let B denote the set of bottom configurations of $\mathcal{P}$, and let B_b be the set of all bottom configurations of $\mathcal{P}$ that can only reach configurations of output $b \in \{0,1\}$. Further let By Proposition 1, $\mathcal{P}$ is well-specified if and only if there exist $\mathcal{I}_0$ and $\mathcal{I}_1$ such that $\mathcal{I}_0$ and $\mathcal{I}_1$ form a partition of $\mathcal{I}$, and the bottom configurations reachable from $\mathcal{I}_b$ are all in consensus b. Formally, there must be some $\mathcal{I}_1, \mathcal{I}_0$ such that:

$$\mathcal{I}_0 \uplus \mathcal{I}_1 = \mathcal{I} \wedge \forall b \in \{0,1\} \forall C \in \mathcal{I}_b, \forall C' \in B \setminus B_b \colon C \not\xrightarrow{*} C'. \tag{5.16}$$

The authors establish the following [EGLM17]:

1. The partition $\mathcal{I}_0 \uplus \mathcal{I}_1$ of $\mathcal{I}$ in rejecting and accepting input configurations exists and is Presburger-definable.

2. B and B_b are effectively Presburger for every $b \in \{0,1\}$.

The first item follows from the assumption that $\mathcal{P}$ is well specified and from the fact that well-specified protocols compute Presburger predicates. The second result can be derived from fact that the mutual reachability relation $\{(C,C') \mid C \xrightarrow{*} C' \wedge C' \xrightarrow{*} C\}$ is effectively Presburger, which was previously established for the more general vector addition systems in [Ler11] by Leroux.

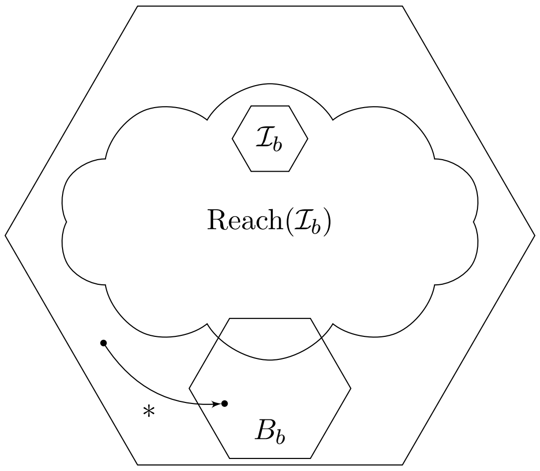


Figure 5.1: Schematic drawing of the overapproximation $\widetilde{\text{Reach}}(\mathcal{I}_b)$. Semilinear sets are indicated by hexagons. The outermost hexagon represents the forwards-inductive set $\widetilde{\text{Reach}}(\mathcal{I}_b)$. The set of configurations reachable from $\mathcal{I}_b$ is represented by a cloud, indicating that it may not be semilinear. The set $\widetilde{\text{Reach}}(\mathcal{I}_b)$ contains B_b, but no configuration in $(B \setminus B_b)$.

Let $\text{Reach}(\mathcal{I}_b) \overset{\text{def}}{=} \{C' \mid \exists C \in \mathcal{I}_b \colon C \xrightarrow{*} C'\}$. It can be shown ([EGLM17]) that for every $b \in \{0,1\}$ there exist semilinear overapproximations $\widetilde{\text{Reach}}(\mathcal{I}_b) \supseteq \text{Reach}(\mathcal{I}_b)$ of the configurations reachable from $\mathcal{I}_b$, satisfying:

1. $\widetilde{\text{Reach}}(\mathcal{I}_b)$ is closed under reachability,
2. $B_b \subseteq \widetilde{\text{Reach}}(\mathcal{I}_b)$,
3. $(B \setminus B_b) \cap \widetilde{\text{Reach}}(\mathcal{I}_b) = \emptyset$.

Figure 5.1 illustrates these three properties. Properties 1-3 are effectively Presburger. Property 1 can be expressed by the following Presburger formula for every $b \in \{0,1\}$:

$$\forall C \forall C' \colon \widetilde{\text{Reach}}(\mathcal{I}_b)(C) \wedge C \to C' \Rightarrow \widetilde{\text{Reach}}(\mathcal{I}_b)(C').$$

Since subset relations, intersection of predicates, and boolean combinations thereof are expressible in Presburger arithmetic, Property 2 and Property 3 are effectively Presburger, too.

Properties 1-3 entail (5.16) for a given partition $\mathcal{I}_0 \uplus \mathcal{I}_1 = \mathcal{I}$ and sets $\widetilde{\text{Reach}}(\mathcal{I}_b) \supseteq \mathcal{I}_b$. Thus, a semi-decision procedure for WellSpecification can be given by enumeration of all Presburger-definable partitions $\mathcal{I}_0 \uplus \mathcal{I}_1 = \mathcal{I}$ and Presburger sets $\widetilde{\text{Reach}}(\mathcal{I}_b) \supseteq \mathcal{I}_b$, and verification of Properties 1-3.

Other subclasses with elementary verification complexity. In [EGMWK18], Esparza *et al.* considered the well-specification problem for *immediate observation protocols*, a subclass of population protocols that can be characterized syntactically as only containing transitions of the form $q_1, q_2 \to q_1, q_3$ (*i.e.* at most one agent is affected by a transition). They show that WellSpecification is solvable in exponential space for immediate observation protocols. They further improve this result by showing PSPACE-completeness in [ERWK19].

Other verification approaches. To the best of our knowledge, our approach is the first fully automatic and parametric approach to Correctness. Previous approaches can be roughly divided into two camps: Those verifying correctness with the help of an interactive theorem prover [DM09], and those that verify correctness only for *fixed* inputs [PLD08, SLDP09b, CMS10, CDFS11]. The former requires human-computer interaction, and is not fully automatic. The latter is not parametric, and requires exploration of the respective reachability graph. Finite-state exploration of the reachability graph can be implemented automatically with the help of model checking techniques, but with the following caveat: Standard specification languages like linear temporal logic (LTL) or Computation Tree Logic (CTL) are not expressive enough to specify correctness under global fairness, and local fairness or some other notion of fairness has to be assumed instead, if the model checker is restricted to these languages (see e.g. [CDFS11]). A model checker that explicitly supports global fairness was presented in [SLDP09a].

A year after we published our approach, Blondin *et al.* introduced the notion of *stage graphs* in [BEK18]. Stage graphs can be regarded as a generalization of the notion of LayeredTermination. With the help of stage graphs, the authors managed to automatically derive upper bounds on the parallel convergence time of population protocols. As opposed to our approach, stage graphs can also deal with non-silent protocols, and may be used for automatic verification of a larger class of protocols in the future.

5.4 Open Research Questions

Both WellSpecification and Correctness have non-elementary complexity. The complexity of the following promise variant of Correctness is unknown:

Promise variant of Correctness
Input: A well-specified population protocol $\mathcal{P}$ and a QFPA formula φ.
Question: Does $\mathcal{P}$ compute the predicate specified by φ?

The promise variant differs from the original problem in the promise that $\mathcal{P}$ is well-specified. The reduction from WellSpecification to Correctness sketched at the beginning of this chapter does not necessarily map to a well-specified protocol, and thus cannot be used to show that the promise problem is at least as hard as WellSpecification. Moreover, there seems to be no straightforward way to adapt the reduction from the Petri net reachability problem to WellSpecification, as shown by Esparza *et al.* in [EGLM17], to the promise problem above.

6 NL-powerful Extensions

The standard model of population protocols does not include a mechanism to detect whether the computation has stabilized. In general, agents cannot know whether their current output is permanent. For instance, consider the *parity predicate*:

$$\varphi(x) = \begin{cases} 1 & \text{if } x \text{ is even,} \\ 0 & \text{otherwise.} \end{cases}$$

The output of any protocol that computes $\varphi(x)$ must alternate with population size. In particular, every agent needs to interact with some other agent at least *once*, in order to stabilize to the correct output. At no point during the computation can an agent be certain that all agents in the population took part in the computation – fairness only ensures that every agent will eventually participate, but interactions can be delayed arbitrarily long. Thus agents can never be certain that their current output is the final output.

This apparent lack of termination indication complicates the design of population protocols, since structuring of protocols into modular subroutines is hard to achieve; the absence of an equivalent for a return statement makes it difficult to compose population protocols sequentially. In Section 4.2.6, we showed how to overcome this difficulty under the assumption that the population size is known: We showed how to implement an analogue of subroutines and return statements by means of a leader that stores the population size in a counter. The leader essentially emulates the behavior of a *global broadcast signal* through sequential pairwise interactions, keeping track of occurrences of interactions. Another way to look at the leader is to say that it provides a *phase clock* that indicates when a task is completed. On a more fundamental level, one may argue that the crucial functionality provided by the leader is a *cover-time service* [MS15]: some way of knowing when it has covered the entire population by interacting with every agent. The mechanism of a cover-time service, in turn, allows the leader to implement a some restricted form of (possibly nondeterministic) register machine.

It is thus only natural to consider extensions of the standard model of population protocols by one of these primitives: cover-time services, clocks, and global broadcasts. In [MS15], Michail *et al.* considered population protocols augmented by a cover-time service, which is given by a distinct agent that may store in its state when it has met all agents in the current iteration. In [Asp17], J. Aspnes introduced *clocked protocols*, which extend the standard model by a clock whose clock ticks signal stabilization for the current computation phase. Finally, in [BEJ19a], we consider *broadcast protocols*, which augment the standard model with the possibility to emit global broadcast signals simultaneously received by all agents. All these extensions can be implemented via modifying the one-step relation $\rightarrow$ appropriately.

6.1 Problem Statement

We consider extensions of population protocols by modification of the one-step relation $\rightarrow$. We call the superclass of all those extensions *generic consensus protocols.* Formally, an generic consensus protocol can be specified by a tuple $(Q, \text{Step}, \Sigma, L, I, O)$, where Q, L, I, O are defined as in population protocols with leaders, and $\text{Step} \subseteq \mathbb{N}^Q \times \mathbb{N}^Q$ is the step relation. The one-step relation is given by $C \rightarrow C' \Leftrightarrow (C, C') \in \text{Step}$; the other notions, like executions, stable computation, fairness, are defined with respect to the one-step relation.

We investigate the expressive power of generic consensus protocols, when restricted to step relations of reasonable complexity. As a special case, we consider *broadcast consider protocols* (short: broadcast protocols), which extend population protocols by reliable broadcasts. A broadcast protocol can be formally specified by a tuple (Q, T, B, Σ, I, O), where Q, T, I, O are defined as in population protocols, and B is the set of broadcast transitions. A broadcast transition is a triple (q, r, f), where $q, r \in Q$, and $f : Q \rightarrow Q$ is a transfer function. Intuitively, q is the state of the broadcaster, r is its state after emitting the broadcast, and f determines how the other agents change their states upon receiving the broadcast signal. A step $C \rightarrow C'$ is either due to some transition $t \in T$ like in standard population protocols, or due to some broadcast transition. In the case of broadcast transitions, for every $(q, r, f) \in B$ and every configuration C, the step relation satisfies minimally:

$$(\wr q \wr + C) \rightarrow (\wr r \wr + C')$$

where $C'(q') = \sum_{q \in f^{-1}(q')} C(q)$ for every $q' \in Q$.

Example 6.1.1. Assuming every agent is initially in leader state L, a leader election can be implemented by means of the broadcast transition (L, L, f) where $f(L) = N$, and $f(q) = q$ for all $q \neq L$. After firing the transition (L, L, f), exactly one agent is in the leader state L, and all the other agents are in the non-leader state N.

6.2 New Contributions

We relate broadcast protocols to the complexity class $\mathsf{NL} = \mathsf{NSPACE}(\log n)$, where n is the size of the input in *unary* encoding. Unary encoding breaks with the default assumption in complexity theory, where input numbers are usually given in binary. However, unary encoding is the more natural assumption in our case, since protocols receive their input in unary. Hence we say a k-ary predicate φ belongs to NL if there is a nondeterministic Turing machine that accepts in $\mathcal{O}(\log |\boldsymbol{x}|)$-space the tuples $\boldsymbol{x} \in \mathbb{N}^k$, encoded in unary, precisely if $\varphi(\boldsymbol{x})$ holds.

The following theorem is the central result of [BEJ19a]:

Theorem 14. *[BEJ19a] The predicates computable by broadcast protocols are precisely the predicates in* NL.

We establish the NL upper bound for all generic broadcast consensus protocols whose Step relation is in NL (see Subsection 6.2.1 for details). The upper bound for broadcast protocols is a corollary of the more general result. The proof of the lower bound is discussed in Subsection 6.2.2.

Furthermore, we investigated the expressiveness of two natural subclasses of broadcast protocols: 1.) Broadcast protocols where the transfer function is identical for all broadcast transitions, and 2.) broadcast protocols where each broadcast resets the execution to initial. We established that the first class still computes all predicates in NL, while the second class is no more expressive than the basic model of population protocols. These two results are discussed in more detail in Subsection 6.2.3.

6.2.1 Predicates computable by Broadcast Protocols are in NL

For the NL upper bound, we establish the following general result:

Theorem 15. *Let $\mathcal{P} = (Q, Step, \Sigma, L, I, O)$ be a generic consensus protocol computing a predicate φ. If $Step \in \mathsf{NL}$, then $\varphi \in \mathsf{NL}$. In particular, predicates computable by broadcast protocols are in NL.*

Generic consensus protocols whose step relation is not in NL are unreasonable in terms of practicality. Theorem 15 can thus be interpreted as establishing an upper bound on the expressiveness of "reasonable" extensions of population protocols. The proof of 15 uses the fact that $\mathsf{coNL} = \mathsf{NL}$ [Imm88], and that protocols can be simulated by an NL procedure, if Step is in NL.

It can further be shown that all semilinear predicates are in $\mathsf{DSPACE}(\log\log n)$, where n is the size of the input in unary encoding [BEJ19b]. Thus population protocols are very far away from the theoretical limit of NL.

6.2.2 All Predicates in NL are computable by Broadcast Protocols

We establish the NL lower bound for broadcast protocols by simulating a (nondeterministic) counter machine due to Minsky [Min61]. A counter machine can be defined as a vector addition system with states, augmented by the possibility to perform a *zero test*, that is, the possibility to execute a state transition dependent on whether a given component of the counting vector equals 0. Each vector component is said to represent one *counter* of the counter machine. We further assume the counter machine to have a distinct *accepting state* q_a. A counter machine accepts an input $\boldsymbol{v}$, if there is a path from the initial configuration $(\boldsymbol{v}, q_0)$ to the configuration $(\mathbf{0}, q_a)$. It is well-known that any nondeterministic Turing machine that needs $\mathcal{O}(\log n)$ space with unary input encoding can be simulated by a counter machine whose counters are polynomially bounded in n [Min61]. It can further be shown [BEJ19a] that every counter machine whose counters are polynomially bounded can be simulated by a counter machine whose counter values are bounded by the size of the input. Thus, in order to show that every predicate in NL is computable by broadcast protocols, it suffices to simulate a counter machine under the assumption that each counter value is bounded by n, where n is the number of tokens needed to represent the input in unary.

Representing counters and control states. Counter values up to n can be represented by n agents, where the maximal n tokens of each counter are scattered across the population. For example, the counting vector $(1, 3)$ can be represented by the population $\wr(1,1),(0,1),(0,1)\wr$, where the first bit contributes to the first counter, and the second bit contributes to the second counter. The control state of the counter machine can be represented by a leader, and incrementation/decrementation of counter values can be implemented by the leader through pairwise interactions.

Simulating zero tests. Broadcast protocols can only *weakly* simulate zero tests, in the following sense: The leader who represents the control state guesses whether the zero test succeeds. If the guess has been correct, the computation may proceed as usual. If the guess is wrong, then the error will eventually be detected further along the computation, but the configuration may by then no longer correspond to a reachable configuration in the counter machine, and the configuration must be reset to initial in order to start a new, clean simulation attempt.

More specifically, the weak zero test is implemented as follows: Whenever the leader simulates a zero test, it first assumes the tested counter value to be greater than 0, and waits until it interacts with a *witness* for a positive counter value. A witness is an agent having the corresponding counter bit set to 1. If the counter value is indeed positive, then by fairness the leader will eventually interact with a witness, and may then proceed with the computation. The leader may nondeterministically decide it has waited long enough to find a witness, and then switch to the assumption that the counter value equals 0. The leader announces the new guess via a broadcast. If the guess is correct, and the counter value is indeed 0, then the other agents are unaffected by the broadcast, and the leader proceeds with the computation under the assumption that the counter value is 0. If the guess is incorrect, then at least one witness for a positive counter value receives the broadcast signal. The witness then reacts by transitioning to a designated *error state*, which is unaffected by any interaction with the leader. An agent in an error state may emit a broadcast signal that resets the entire population to initial (resets can be implemented by a broadcast, given that every state is annotated with the original input of the agent). By fairness, the simulation will eventually be reset to initial, whenever an erroneous guess has been made.

Simulating nondeterminism. Since the simulated counter machine is nondeterministic, it may be the case that only some, but not all executions of the counter machine accept the input. In order to explore different computational branches, the leader may nondeterministically broadcast a signal that resets the configuration to initial, as long as the accepting state q_a has not been reached. If the machine accepts the input, then by fairness the protocol will eventually simulate an accepting execution faithfully (without erroneous zero tests), and the leader reaches an accepting state. The leader propagates the acceptance of the input to the other agents, and the computation permanently stabilizes to consensus 1.

From semi-computation to computation. From the previous considerations we conclude that for every predicate φ in NL, there exists a broadcast protocol $\mathcal{P}_\varphi$ that stabilizes to 1 for input $\boldsymbol{v}$ if $\varphi(\boldsymbol{v})$ holds. In the case where $\varphi(\boldsymbol{v})$ does *not* hold, the execution of $\mathcal{P}_\varphi$ may not stabilize. In this sense, $\mathcal{P}_\varphi$ can be regarded as "semi-computing" φ. By the Immerman-Stelepcsényi's theorem [Imm88], we have $\mathsf{NL} = \mathsf{coNL}$, and consequently there also exists a broadcast protocol $\mathcal{P}_{\neg\varphi}$ that semi-computes the complement predicate $\neg\varphi(\boldsymbol{v})$. Now, in order to obtain a broadcast protocol that stably computes φ (instead of merely semi-computing), we assemble $\mathcal{P}_\varphi$ and $\mathcal{P}_{\neg\varphi}$ as follows: The new protocol alternates between executing $\mathcal{P}_\varphi$ and $\mathcal{P}_{\neg\varphi}$, starting with, say, $\mathcal{P}_\varphi$. The leader nondeterministically resets the configuration to initial like in $\mathcal{P}_\varphi$, but instead of always resetting to $\mathcal{P}_\varphi$, it nondeterministically picks between $\mathcal{P}_\varphi$ or $\mathcal{P}_{\neg\varphi}$ for the next execution. Eventually, either the execution of $\mathcal{P}_\varphi$ or $\mathcal{P}_{\neg\varphi}$ stabilizes, and no more resets occur (recall that the leader is not allowed to reset the population when an accepting state is reached). If the current protocol is $\mathcal{P}_\varphi$, the protocol shall stabilize to 1, otherwise to

0. By fairness, the execution of the protocol eventually stabilizes to the correct output. This construction computes φ for every predicate φ in NL.

6.2.3 Single-Signal Protocols and Protocols with Resets

We also investigated the expressiveness of two subclasses of broadcast protocols: Single-signal protocols, and protocols with resets.

Single-Signal protocols. Call a broadcast protocol *single-signal*, if for any two broadcast transitions (q, r, f) and (q', r', f') the transfer function is identical: $f = f'$. We establish the following:

Theorem 16. *[BEJ19a] For every predicate $\varphi \in \mathsf{NL}$, there is a single-signal broadcast protocol that computes φ.*

The idea of Theorem 16 is to weakly simulate arbitrary broadcasts by means of rendezvous-transitions with a leader; the single broadcast signal functions as analogue of a phase clock, which signals to regular agents when the broadcast starts/ends. Agents assume an error state upon receiving two consecutive broadcasts, without interacting with the leader agent in between. Error states eventually lead to resets like in the weak simulation of counter machines. For the single-signal protocol to be well-specified, it must be ensured that there are no fair runs with infinitely many broadcast transitions in the simulated protocol (for otherwise there would be infinitely many failed simulation attempts of broadcasts, and consequently infinitely many resets). The NL-powerful construction for the simulation of counter machines from the previous section satisfies this property.

Protocols with resets. One may be tempted to think that the true power of the previous constructions lies in the power to reset configurations to initial. Perhaps surprisingly, the ability to reset an execution to initial alone does not increase expressiveness relative to population protocols. Formally, a *population protocol with reset* is a broadcast protocol, such that for every infinite execution $C_0C_1C_2\dots$ and every k the following holds: If $C_k \xrightarrow{b} C_{k+1}$ for some broadcast transition b, then there exists some $l > k$ such that $C_l = C_0$.

Theorem 17. *[BEJ19a] Every predicate computable by a population protocol with reset is Presburger-definable, and thus computable by a standard protocol.*

The proof is inspired by the proof for decidability of WellSpecification. Like the latter, the proof of Theorem 17 uses deep results about semiliniarity of certain reachability sets established in the literature on Petri nets (see [BEJ19a] for details).

6.3 Related Work

Broadcast protocols were introduced by Emerson and Namjoshi in [EN98] to specify the behavior of bus-based hardware protocols. Broadcast protocols have been used for verification of multithreaded programs [DRB02], and for controlling of gene expression in living systems [UMD$^+$15, BDGG17]. In [EFM99], Esparza *et al.* considered parameterized verification of broadcast protocols, with respect to safety and liveness properties which are given by a regular or omega-regular expression, respectively. They showed that liveness properties are in general undecidable. This entails undecidability of Correctness for broadcast protocols.

In [MS15], Michail and Spirakis prove that protocols with cover-time service can compute all predicates in DSPACE($\log n$) and that they can compute no more than the predicates in NL, where n is the number of agents. It is readily seen that protocols with cover-time service can implement broadcast transitions; from this and our results follows that protocols with cover-time service compute precisely the predicates in NL.

In the clocked protocols by Aspnes ([Asp17]), agents carry an additional *clock bit* which indicates whether the agent has waited long enough for the current phase of computation to stabilize. The clock bit is set by *clock tick* received by all agents simultaneously upon stabilization. Since stabilization cannot be ensured after a finite amount of time, time intervals up to stabilization are modeled as transfinite intervals, and the time points of stabilization are represented by limit ordinals. Aspnes shows that clocked protocols which stabilize after ω^2 steps compute precisely the predicates in NL. He also gave an equivalent characterization of clocked protocols via inductively defined reachability graphs to show that their step relation is in NL.

Both clocked protocols and protocols with cover-time service are capable of implementing a zero test in a straightforward way[1], with total accuracy. By contrast, simulation of zero tests in broadcast protocols is necessarily weak, and subject to (detectable) errors. The novelty of our contribution lies in showing that the weak form of zero test already suffices to achieve the NL lower bound. Moreover, it is readily seen that both clocked protocols, and protocols with cover-time service are capable of simulating broadcast transitions. Thus Theorem 14 can be regarded a strengthening of the NL-completeness results of the other two models.

In [AAE08a], Angluin *et al.* showed that population protocols with one leader can simulate a register machine with high probability, under the assumption that the scheduler picks pairs of agents uniformly at random. The leader uses population epidemics for information propagation, and to simulate an epidemics-based phase clock. In this restricted sense, the random execution model is capable of implementing a probabilistic variant of clocked protocols.

6.4 Open Research Questions

Future research may study the time and space complexity of broadcast consensus protocols in greater depth. Concerning time complexity, it is obvious that broadcasts – besides being more powerful than rendez-vous transitions – may speed up computations: For example, the leader election broadcast protocol from Example 6.1.1 elects a leader in constant time, while leader election in leaderless population protocols requires linear time [DS18]. Conditions for similar speed-ups certainly deserve further investigation, also with respect to protocols with resets, which are no more expressive than population protocols, but may behave differently in terms of time or space characteristics.

In the model of broadcast protocols under consideration, broadcasts are reliable. This is hardly the case in real-world applications, and signals are frequently lost during transmission. Under fair semantics, unreliable broadcasts are clearly of little help (unreliable broadcasts can be simulated by a sequence of rendez-vous transitions); however, assuming a probabilistic scheduler, unreliable broadcasts open another avenue for investigating

[1]In [MS15] zero tests are called *absence detectors*.

broadcast protocols: How failure probabilities of broadcasts influence the failure probability of the computation as a whole, is of immediate practical significance. To the best of our knowledge, this question has not been considered under the stable-consensus computing paradigm assumed in broadcast consensus protocols.

Bibliography

[AAD+06] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, 2006. doi:10.1007/s00446-005-0138-3.

[AAE06] Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the 25th annual ACM symposium on Principles of distributed computing (PODC)*, pages 292–299, 2006. doi:10.1145/1146381.1146425.

[AAE08a] Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, pages 183–199, 2008. doi:10.1007/s00446-008-0067-z.

[AAE08b] Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, pages 87–102, 2008. doi:10.1007/s00446-008-0059-z.

[AAE+17] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L Rivest. Time-space trade-offs in population protocols. In *Proceedings of the 28th annual ACM-SIAM symposium on discrete algorithms (SODA)*, pages 2560–2579. SIAM, 2017. doi:10.1137/1.9781611974782.169.

[AAER07] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, pages 279–304, 2007. doi:10.1007/s00446-007-0040-2.

[AAOW15] S Akshay, Timos Antonopoulos, Joël Ouaknine, and James Worrell. Reachability problems for markov chains. *Information Processing Letters*, pages 155–158, 2015. doi:10.1016/j.ipl.2014.08.013.

[ABBS16] James Aspnes, Joffroy Beauquier, Janna Burman, and Devan Sohier. Time and space optimal counting in population protocols. In *Proceedings of the 20th International Conference on Principles of Distributed Systems (OPODIS)*, pages 13:1–13:17, 2016. doi:10.4230/LIPIcs.OPODIS.2016.13.

[AC02] Andrew Adamatzky and Benjamin De Lacy Costello. Experimental logical gates in a reaction-diffusion medium: The xor gate and beyond. *Physical Review E*, 2002. doi:10.1103/PhysRevE.66.046112.

[AG15] Dan Alistarh and Rati Gelashvili. Polylogarithmic-time leader election in population protocols. In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 479–491, 2015. doi:10.1007/978-3-662-47666-6_38.

[AGV15] Dan Alistarh, Rati Gelashvili, and Milan Vojnović. Fast and exact majority in population protocols. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 47–56, 2015. doi:10.1145/2767386.2767429.

[AR09] James Aspnes and Eric Ruppert. An introduction to population protocols. In *Middleware for Network Eccentric and Mobile Applications*, pages 97–120. Springer, 2009. doi:10.1007/978-3-540-89707-1_5.

[Asp17] James Aspnes. Clocked population protocols. In *Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 431–440, 2017. doi:10.1145/3087801.3087836.

[BBB13] Joffroy Beauquier, Peva Blanchard, and Janna Burman. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS)*, pages 38–52, 2013. doi:10.1007/978-3-319-03850-6_4.

[BBC14] Joffroy Beauquier, Janna Burman, and Simon Claviere. Comptage et nommage simples et efficaces dans les protocoles de populations symétriques. *ALGOTEL 2014*, pages 1–4, 2014.

[BBCS15] Joffroy Beauquier, Janna Burman, Simon Clavière, and Devan Sohier. Space-optimal counting in population protocols. In *Proceedings of the 29th International Symposium on Distributed Computing (DISC)*, pages 631–646, 2015. doi:10.1007/978-3-662-48653-5_42.

[BCM$^+$07] Joffroy Beauquier, Julien Clement, Stephane Messika, Laurent Rosaz, and Brigitte Rozoy. Self-stabilizing counting in mobile sensor networks with a base station. In *Proceedings of the 21st International Symposium on Distributed Computing (DISC)*, pages 63–76. Springer, 2007. doi:10.1007/978-3-540-75142-7_8.

[BDG$^+$19] Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, Hugo Gimbert, and Adwait Amit Godbole. Controlling a population. *Logical Methods in Computer Science*, 2019. doi:10.23638/LMCS-15(3:6)2019.

[BDGG17] Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, and Hugo Gimbert. Controlling a population. In *Proceedings of the 28th International Conference on Concurrency Theory (CONCUR)*, 2017. doi:10.4230/LIPIcs.CONCUR.2017.12.

[BEG$^+$20] Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. Succinct Population Protocols for Presburger Arithmetic. In *Proceedings of the 37th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 40:1–40:15, 2020. doi:10.4230/LIPIcs.STACS.2020.40.

[BEJ18a] Michael Blondin, Javier Esparza, and Stefan Jaax. Large Flocks of Small Birds: on the Minimal Size of Population Protocols. In *Proceedings of the*

35th Symposium on Theoretical Aspects of Computer Science (STACS), pages 16:1–16:14, 2018. doi:10.4230/LIPIcs.STACS.2018.16.

[BEJ18b] Michael Blondin, Javier Esparza, and Stefan Jaax. Peregrine: A tool for the analysis of population protocols. In *Proceedings of the 30th International Conference on Computer Aided Verification (CAV)*, pages 604–611, 2018. doi:10.1007/978-3-319-96145-3_34.

[BEJ19a] Michael Blondin, Javier Esparza, and Stefan Jaax. Expressive Power of Broadcast Consensus Protocols. In *Proceedings of the 30th International Conference on Concurrency Theory (CONCUR)*, pages 31:1–31:16, 2019. doi:10.4230/LIPIcs.CONCUR.2019.31.

[BEJ19b] Michael Blondin, Javier Esparza, and Stefan Jaax. Expressive power of oblivious consensus protocols. *arXiv preprint arXiv:1902.01668v1*, 2019.

[BEJK18] Michael Blondin, Javier Esparza, Stefan Jaax, and Antonín Kučera. Black ninjas in the dark: Formal analysis of population protocols. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10, 2018. doi:10.1145/3209108.3209110.

[BEJM17] Michael Blondin, Javier Esparza, Stefan Jaax, and Philipp J. Meyer. Towards efficient verification of population protocols. In *Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 423–430, 2017. doi:10.1145/3087801.3087816.

[BEK18] Michael Blondin, Javier Esparza, and Antonín Kucera. Automatic analysis of expected termination time for population protocols. In *Proceedings of the 29th International Conference on Concurrency Theory (CONCUR)*, pages 33:1–33:16, 2018. doi:10.4230/LIPIcs.CONCUR.2018.33.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT press, 2008.

[BLM86] Jean-Pierre Banâtre and Daniel Le Métayer. A new computational model and its discipline of programming. *Technical Report RR-0556*, 1986.

[BT05] Ahmed Bouajjani and Tayssir Touili. On computing reachability sets of process rewrite systems. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA)*, pages 484–499, 2005. doi:10.1007/978-3-540-32033-3_35.

[CCDS14] Ho-Lin Chen, Rachel Cummings, David Doty, and David Soloveichik. Speed faults in computation by chemical reaction networks. In *DISC*, Lecture Notes in Computer Science, pages 16–30. Springer, 2014. doi:10.1007/978-3-662-45174-8_2.

[CDFS11] Julien Clément, Carole Delporte-Gallet, Hugues Fauconnier, and Mihaela Sighireanu. Guidelines for the verification of population protocols. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 215–224, 2011. doi:10.1109/ICDCS.2011.36.

[CDS$^+$13] Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from DNA. *Nature nanotechnology*, page 755, 2013. doi:10.1038/nnano.2013.189.

[CFQS12] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, pages 387–408, 2012. doi:10.1080/17445760.2012.668546.

[CLL$^+$19] Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 24–33, 2019. doi:10.1145/3313276.3316369.

[CMS10] Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. Algorithmic verification of population protocols. In *Proceedings of the 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 221–235, 2010. doi:10.1007/978-3-642-16023-3_19.

[DA94] René David and Hassane Alla. Petri nets for modeling of dynamic systems: A survey. *Automatica*, pages 175–202, 1994. doi:10.1016/0005-1098(94)90024-8.

[DDPM19] Marta Dueñas-Díez and Juan Pérez-Mercader. How chemistry computes: language recognition by non-biochemical chemical automata. from finite automata to turing machines. *iScience*, pages 514–526, 2019. doi:10.1016/j.isci.2019.08.007.

[DF01] Zoë Diamadi and Michael J Fischer. A simple game for the study of trust in distributed systems. *Wuhan University Journal of Natural Sciences*, pages 72–82, 2001.

[DLBBC14] Giuseppe Antonio Di Luna, Roberto Baldoni, Silvia Bonomi, and Ioannis Chatzigiannakis. Counting in anonymous dynamic networks under worst-case adversary. In *Proceedings of the 34th International Conference on Distributed Computing Systems (ICDCS)*, pages 338–347, 2014. doi:10.1109/ICDCS.2014.42.

[DM09] Yuxin Deng and Jean-François Monin. Verifying self-stabilizing population protocols with Coq. In *Proceedings of the 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 201–208, 2009. doi:10.1109/TASE.2009.9.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 337–340. Springer, 2008. doi:10.1007/978-3-540-78800-3_24.

[DRB02] Giorgio Delzanno, Jean-François Raskin, and Laurent Van Begin. Towards the automated verification of multithreaded Java programs. In *Proceedings of the 8th International Conference on Tools and Algorithms for the*

Construction and Analysis of Systems (TACAS), pages 173–187, 2002. doi:10.1007/3-540-46002-0_13.

[DS18] David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, pages 257–271, 2018. doi:10.1007/s00446-016-0281-z.

[EFM99] Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 352–359, 1999. doi:10.1109/LICS.1999.782630.

[EGLM17] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, pages 191–215, 2017. doi:10.1007/s00236-016-0272-3.

[EGMWK18] Javier Esparza, Pierre Ganty, Rupak Majumdar, and Chana Weil-Kennedy. Verification of Immediate Observation Population Protocols. In *Proceedings of the 29th International Conference on Concurrency Theory (CONCUR)*, pages 31:1–31:16, 2018. doi:10.4230/LIPIcs.CONCUR.2018.31.

[EN98] E. Allen Emerson and Kedar S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 70–80, 1998. doi:10.1109/LICS.1998.705644.

[ERWK19] Javier Esparza, Mikhail Raskin, and Chana Weil-Kennedy. Parameterized analysis of immediate observation petri nets. In *Proceedings of the 40th International Conference on Applications and Theory of Petri Nets and Concurrency (PETRI NETS)*, pages 365–385. Springer, 2019. doi:10.1007/978-3-030-21571-2_20.

[Gil77] Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, pages 2340–2361, 1977.

[Gil92] Daniel T Gillespie. A rigorous derivation of the chemical master equation. *Physica A: Statistical Mechanics and its Applications*, pages 404–425, 1992.

[GR07] Rachid Guerraoui and Eric Ruppert. Even small birds are unique: Population protocols with identifiers. Technical report, 2007.

[GS64] Seymour Ginsburg and Edwin H Spanier. Bounded algol-like languages. *Transactions of the American Mathematical Society*, pages 333–368, 1964.

[Haa18] Christoph Haase. A survival guide to presburger arithmetic. *SIGLOG News*, pages 67–82, 2018.

[HH14] Christoph Haase and Simon Halfon. Integer vector addition systems with states. In *Proceedings of the International Workshop on Reachability Problems*, pages 112–124, 2014.

[Huy85] Dung T Huynh. Complexity of the word problem for commutative semigroups of fixed dimension. *Acta informatica*, pages 421–432, 1985. doi:10.1007/BF00288776.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, pages 935–938, 1988. doi:10.1137/0217058.

[ISV17] Tomoko Izumi, Nicola Santoro, and Giovanni Viglietta. Population protocols with faulty interactions: The impact of a leader. In *Proceedings of the 10th International Conference on Algorithms and Complexity (CIAC)*, page 454, 2017. doi:10.1007/978-3-319-57586-5_38.

[Ler11] Jérôme Leroux. Vector addition system reachability problem: A short self-contained proof. In *Language and Automata Theory and Applications (LATA)*, pages 41–64, 2011. doi:10.1007/978-3-642-21254-3_3.

[MCdFD05] Naoki Matsumaru, Florian Centler, Pietro Speroni di Fenizio, and Peter Dittrich. Chemical organization theory as a theoretical base for chemical computing. In *Proceedings of the 2005 Workshop on Unconventional Computing: From Cellular Automata to Wetware*, pages 75–88, 2005.

[MCS11] Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. Mediated population protocols. *Theoretical Computer Science*, pages 2434–2450, 2011. doi:10.1016/j.tcs.2011.02.003.

[MCS13] Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. Naming and counting in anonymous unknown dynamic networks. In *Proceedings of the Symposium on Self-Stabilizing Systems*, pages 281–295, 2013.

[MH89] Charles E McDowell and David P Helmbold. Debugging concurrent programs. *ACM Computing Surveys (CSUR)*, pages 593–622, 1989.

[Min61] Marvin L Minsky. Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. *Annals of Mathematics*, pages 437–455, 1961.

[MM15] Alessia Milani and Miguel A Mosteiro. A faster counting protocol for anonymous dynamic networks. *arXiv preprint arXiv:1509.02140*, 2015.

[MNRS14] George B Mertzios, Sotiris E Nikoletseas, Christoforos L Raptopoulos, and Paul G Spirakis. Determining majority in networks with local interactions and very small local memory. In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 871–882, 2014. doi:10.1007/978-3-662-43948-7_72.

[MS15] Othon Michail and Paul G Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *Journal of Parallel and Distributed Computing*, pages 1–10, 2015. doi:10.1016/j.jpdc.2015.02.005.

[PLD08] Jun Pang, Zhengqin Luo, and Yuxin Deng. On automatic verification of self-stabilizing population protocols. In *Proceedings of the 2nd IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 185–192, 2008. doi:10.1109/TASE.2008.8.

[Rac78] Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, pages 223–231, 1978.

[Sch00] Karsten Schmidt. Lola a low level analyser. In *Proceedings of the 21st International Conference on Application and Theory of Petri Nets (ICATPN)*, pages 465–474. Springer, 2000. doi:10.1007/3-540-44988-4_27.

[SLDP09a] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. PAT: towards flexible verification under fairness. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV)*, pages 709–714, 2009. doi:10.1007/978-3-642-02658-4_59.

[SLDP09b] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. PAT: Towards flexible verification under fairness. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV)*, pages 709–714, 2009. doi:10.1007/978-3-642-02658-4_59.

[SVR17] NE Shlyakhov, IV Vatamaniuk, and AL Ronzhin. Survey of methods and algorithms of robot swarm aggregation. In *Journal of Physics: Conference Series*, page 012146. IOP Publishing, 2017.

[UMD$^+$15] Jannis Uhlendorf, Agnès Miermont, Thierry Delaveau, Gilles Charvin, François Fages, Samuel Bottani, Pascal Hersen, and Gregory Batt. In silico control of biomolecular processes. In *Computational Methods in Synthetic Biology*, pages 277–285. Springer, 2015. doi:10.1007/978-1-4939-1878-2_13.

[Val96] Antti Valmari. The state explosion problem. In *Advanced Course on Petri Nets*, pages 429–528. Springer, 1996.

Appendix I: Publications on Space Complexity

A Large Flocks of Small Birds: on the Minimal Size of Population Protocols. STACS, 2018.

This chapter has been published as **peer-reviewed conference paper**.

Michael Blondin, Javier Esparza, and Stefan Jaax. Large Flocks of Small Birds: on the Minimal Size of Population Protocols. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 16:1–16:14, 2018. doi:10.4230/LIPIcs.STACS.2018.16

Synopsis. We investigate the space complexity of population protocols: the minimal amount of memory required to compute a given predicate φ, relative to its description size $|\varphi|$. We consider the family of simple counting predicates $x \geq n$ for $n \in \mathbb{N}$, and more generally, the predicates representing systems of linear inequalities over the integers. We show that simple counting predicates can be computed by protocols with $\mathcal{O}(\log n)$ states, and that any system of linear inequalities A can be computed with poly$(|A|)$ states, where $|A|$ denotes the description size of A, with numbers encoded in binary. We further prove the existence of some infinite sequence of integers $c_1 < c_2 < c_3 < \ldots$ such that $x \geq c_i$ is computable with $\mathcal{O}(\log \log c_i)$ states for every i. It remains open whether this bound can further be improved to $\mathcal{O}(\log \log \log c_i)$ – however, we show that this is not achievable for the class of so-called 1-aware protocols: These are the protocols where the occurrence of a state with output 1 guarantees stabilization to a positive consensus.

Contributions of thesis author. Composition and revision of the manuscript. Joint discussion and development of the results and proofs presented in the paper, with the following notable individual contributions: design of the first correct construction that computes $x \geq c$ with $\mathcal{O}(\log c)$ states, and its generalization to arbitrary threshold predicates; design of the construction for the simulation of k-way protocols by 2-way protocols; writing the elaborate proof for Lemma 3, and developing all ideas and definitions required for its completion.

Large Flocks of Small Birds: on the Minimal Size of Population Protocols

Michael Blondin
Technische Universität München, Munich, Germany
blondin@in.tum.de

Javier Esparza
Technische Universität München, Munich, Germany
esparza@in.tum.de

Stefan Jaax
Technische Universität München, Munich, Germany
jaax@in.tum.de

Abstract

Population protocols are a well established model of distributed computation by mobile finite-state agents with very limited storage. A classical result establishes that population protocols compute exactly predicates definable in Presburger arithmetic. We initiate the study of the minimal amount of memory required to compute a given predicate as a function of its size. We present results on the predicates $x \geq n$ for $n \in \mathbb{N}$, and more generally on the predicates corresponding to systems of linear inequalities. We show that they can be computed by protocols with $O(\log n)$ states (or, more generally, logarithmic in the coefficients of the predicate), and that, surprisingly, some families of predicates can be computed by protocols with $O(\log \log n)$ states. We give essentially matching lower bounds for the class of 1-aware protocols.

2012 ACM Subject Classification Theory of computation → Distributed computing models, Theory of computation → Complexity theory and logic, Theory of computation → Logic and verification

Keywords and phrases Population Protocols, Presburger Arithmetic

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.16

Related Version The full version of this paper can be found at `https://arxiv.org/abs/1801.00742`.

Funding M. Blondin was supported by the Fonds de recherche du Quebec – Nature et technologies (FRQNT).

1 Introduction

Population protocols [4] are a model of distributed computation by anonymous, identical, and mobile finite-state agents. Initially introduced to model networks of passively mobile sensors, they also capture the essence of distributed computation in trust propagation or chemical reactions, the latter under the name of chemical reaction networks (see e.g. [18]). Structurally, population protocols can also be seen as a special class of Petri nets or vector addition systems [11].

Since the agents executing a protocol are anonymous and identical, its global state – called a *configuration* – is completely determined by the number of agents at each local state. In each computation step, a pair of agents, chosen by an adversary subject to a fairness condition stating that any repeatedly reachable configuration is eventually reached, interact and move to new states according to a joint transition function. In a closely related model, the adversary chooses the pair of agents uniformly at random.

A protocol computes a boolean value for a given initial configuration if in all fair executions all agents eventually agree to this value – so, intuitively, population protocols compute by reaching consensus. Given a set of initial configurations, the predicate computed by a protocol is the function that assigns to each configuration C the boolean value computed by the protocol starting from C.

Much research on population protocols has focused on their expressive power, i.e., the class of predicates computable by different classes of protocols (see e.g. [3, 6, 13, 16, 7]). In a famous result [6], Angluin et al. have shown that predicates computable by population protocols are exactly the predicates definable in Presburger arithmetic. There is also much work on complexity metrics for protocols. The main two metrics are the *runtime* of a protocol – defined for the model with a randomized adversary as the expected number of pairwise interactions until all agents have the correct output value – and its *state space size*, e.g. the number of states of each agent. In [5], Angluin et al. show that every Presburger predicate is computed with high probability by a population protocol with a leader – a distinguished auxiliary agent that assumes a specific state in the initial configuration irrespective of the input – in $O(n \log^4 n)$ interactions in expectation, where n is the number of agents of the initial configuration. Several recent papers study time-space trade-offs for specific tasks, like electing a leader [10], or for specific predicates, like majority [2, 1, 9].

In this paper we study the state space size of protocols as a function of the predicate they compute. In particular, we are interested in the minimal number of states needed to evaluate systems of linear constraints (a large subclass of the predicates computed by population protocols) as a function of the number of bits needed to describe the system. To the best of our knowledge, this question has not been considered so far. We study the question for protocols with and without leaders. Our results show that protocols with leaders can be exponentially more compact than leaderless protocols.

In order to introduce our results in the simplest possible setting, in the first part of the paper we focus on the family of predicates $\{x \geq n : n \in \mathbb{N}\}$. These predicates specify the well-known flock-of-birds problem [4], in which tiny sensors placed on birds have to reach consensus on whether the number of sick birds in a flock exceeds a given constant. The minimal number of states for computing $x \geq n$ formalizes a very natural question about emerging behavior: How many states must agents have in order to exhibit a “phase transition” when their number reaches n? The standard protocol for the predicate $x \geq n$ (see Example 1) has $n+1$ states. We are interested in protocols with at most $O(\log n)$ states, either leaderless or with at most $O(\log n)$ leaders. In the second part of the paper, we generalize our results to a much larger class of predicates, namely systems of linear inequalities $\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}$. Since $x \geq n$ is a (very) special case, our lower bounds for flock-of-birds protocols apply, while the upper bounds require new (and involved) constructions.

Protocol size for the flock-of-birds problem. In a first warm-up phase we exhibit a family of leaderless protocols with only $O(\log n)$ states. More precisely, we prove:

(1) There exists a family $\{\mathcal{P}_n : n \in \mathbb{N}\}$ of leaderless population protocols such that $\mathcal{P}_n$ has $O(\log_2 n)$ states and computes the predicate $x \geq n$ for every $n \in \mathbb{N}$.

We also give a lower bound:

(2) For every family $\{\mathcal{P}_n : n \in \mathbb{N}\}$ of leaderless population protocols such that $\mathcal{P}_n$ computes $x \geq n$, there exist infinitely many n such that $\mathcal{P}_n$ has at least $(\log n)^{1/4}$ states.

However, this bound is only *existential* (“there exists infinitely many n” instead of “for all n”). Moreover, it follows from a counting argument that does not provide any information on the values of n realizing the bound. Is there a poly-logarithmic universal bound? We show that, surprisingly, the answer is negative:

(3) There exists a family $\{\mathcal{P}_n : n \in \mathbb{N}\}$ of population protocols with two leaders, and values $c_0 < c_1 < \ldots \in \mathbb{N}$, such that $\mathcal{P}_n$ has $O(\log \log c_n)$ states and computes the predicate $x \geq c_n$ for every $n \in \mathbb{N}$.

Observe that in these protocols the “phase transition” occurs at $x = c_n$, even though no agent has enough memory to index a particular bit of c_n.

Can one go even further, and design $O(\log \log \log c_n)$ protocols? We show that the answer is negative for *1-aware* protocols. Both the standard protocol for $x \geq n$ and the families of (1) and (3) have the following, natural property: If the number of agents is greater than or equal to n, then the agents not only reach consensus 1, they also eventually *know* that they will reach this consensus. We say that these protocols are 1-aware.

We obtain lower bounds for 1-aware protocols that essentially match the upper bounds of (1) and (3):

(4) Every leaderless, 1-aware population protocol computing $x \geq n$ has at least $\log_3 n$ states.

(5) Every 1-aware protocol (leaderless or not) computing $x \geq n$ has at least $(\log \log(n)/151)^{1/9}$ states.

Protocols for systems of linear inequalities. In the second part of the paper we show that our results can be extended to other predicates. First, instead of the simple predicate $x \geq n$, we study the general linear predicate $a_1x_1 + a_2x_2 + \cdots + a_kx_k \geq c$ for arbitrary integer coefficients $a_1, \ldots, a_k, c \in \mathbb{Z}$. By means of a delicate construction we give protocols whose number of states grows only logarithmically in the size of the coefficients:

(6) There is a protocol with at most $O(kn)$ states and $O(n)$ leaders that computes $a_1x_1 + \cdots + a_kx_k \geq c$, where n is the size of the binary encoding of $\max(|a_1|, |a_2|, \ldots, |a_k|, |c|)$.

Finally, in the most involved construction of the paper, we show that the same applies to arbitrary systems of linear inequalities. Note that the standard conjunction construction, which produces a protocol for $\varphi_1 \wedge \varphi_2$ from protocols computing predicates φ_1 and φ_2, cannot be applied because it would lead to exponentially large protocols.

(7) There is a protocol with at most $O((\log m + n)(m + k))$ states and $O(m(\log m + n))$ leaders that computes $A\boldsymbol{x} \geq \boldsymbol{c}$, where $A \in \mathbb{Z}^{m \times k}$ and n is the size of the largest entry in A and $\boldsymbol{c}$.

Structure of the paper. Section 2 introduces basic definitions, protocols with and without leaders, and a simple construction with an involved correctness proof showing how to simulate protocols with k-way interactions by standard protocols. Sections 3 to 5 present our bounds on the flock-of-birds predicates, and Section 6 the bounds on systems of linear inequalities. Due to space constraints, some proofs are deferred to the full version of this paper.

2 Preliminaries

Numbers. Let $n \in \mathbb{N}_{>0}$. The logarithm in base b of n is denoted by $\log_b n$. Whenever $b = 2$, we omit the subscript. We define $\text{bits}(n)$ as the set of indices of the bits occurring in the binary representation of n, e.g. $\text{bits}(13) = \{0, 2, 3\}$ since $13 = 1101_2$. The *size* of n, denoted $\text{size}(n)$, is the number of bits required to represent n in binary. Note that $|\text{bits}(n)| \leq \text{size}(n) = \lfloor \log n \rfloor + 1$.

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, and for every $E' \subseteq E$ we define $M(E') \overset{\text{def}}{=} \sum_{e \in E'} M(e)$. The *support* and *size* of M are defined respectively as $[\![M]\!] \overset{\text{def}}{=} \{e \in E : M(e) > 0\}$ and $|M| \overset{\text{def}}{=} \sum_{e \in E} M(e)$. *Addition* and *comparison* are extended to multisets componentwise, i.e. $(M + M')(e) \overset{\text{def}}{=} M(e) + M'(e)$ for every $e \in E$, and $M \leq M' \overset{\text{def}}{\iff} M(e) \leq M(e)$ for every $e \in E$. We define *multiset difference* as $(M \ominus M')(e) \overset{\text{def}}{=} \max(M(e) - M'(e), 0)$ for every $e \in E$. The empty multiset is denoted $\mathbf{0}$. We sometimes denote multisets using a set-like notation, e.g. $\wr f, 2 \cdot g, h \int$ is the multiset M such that $M(f) = 1$, $M(g) = 2$, $M(h) = 1$ and $M(e) = 0$ for every $e \in E \setminus \{f, g, h\}$.

Population protocols. We introduce a rather general model of population protocols, allowing for interactions between more than two agents and for leaders. A *k-way population protocol* is a tuple $\mathcal{P} = (Q, T, I, L, O)$ such that

- Q is a finite set of *states*,
- $T \subseteq \bigcup_{2 \leq i \leq k} Q^i \times Q^i$ is a set of *transitions*,
- $I \subseteq Q$ is a set of *initial states*,
- $L \in \mathbb{N}^Q$ is a set of *leaders*, and
- $O \colon Q \to \{0, 1\}$ is the *output mapping*.

We assume throughout the paper that agents can always interact, i.e., that for every pair of states (p, q), there exists a pair of states (p', q') such that $((p, q), (p', q')) \in T$.

A *configuration* of $\mathcal{P}$ is a multiset $C \in \mathbb{N}^Q$ such that $|C| > 0$. Intuitively, C describes a non empty collection containing $C(q)$ agents in state q for every $q \in Q$. We denote the set of configurations over $E \subseteq Q$ by $\text{Pop}(E)$. A configuration C is *initial* if $C = D + L$ for some $D \in \text{Pop}(I)$. So, intuitively, leaders are distinguished agents that are present in every initial configuration. The *number of leaders* of $\mathcal{P}$ is $|L|$. We say that $\mathcal{P}$ is *leaderless* if it has no leader, i.e. if $L = \mathbf{0}$. We discuss protocols with and without leaders later in this section.

Let $t = ((p_1, p_2, \ldots, p_i), (q_1, q_2, \ldots, q_i))$ be a transition. To simplify the notation, we denote t as $p_1, p_2, \ldots, p_i \mapsto q_1, q_2, \ldots, q_i$. Intuitively, t describes that i agents at states $p_1, \ldots, p_i$ may interact and move to states $q_1, \ldots, q_i$. The *preset* and *postset* of t are respectively defined as ${}^\bullet t \overset{\text{def}}{=} \{p_1, p_2, \ldots, p_i\}$ and $t^\bullet \overset{\text{def}}{=} \{q_1, q_2, \ldots, q_i\}$. We extend presets and postsets to sets of transitions, e.g. ${}^\bullet T \overset{\text{def}}{=} \bigcup_{t \in T} {}^\bullet t$. The *pre-multiset* and *post-multiset* of t are respectively defined as $\text{pre}(t) \overset{\text{def}}{=} \wr p_1, p_2, \ldots, p_i \int$ and $\text{post}(t) \overset{\text{def}}{=} \wr q_1, q_2, \ldots, q_i \int$.

We say that t is *enabled* at $C \in \text{Pop}(Q)$ if $C \geq \text{pre}(t)$. If t is enabled at C, then it can *occur*, in which case it leads to the configuration $C' = (C \ominus \text{pre}(t)) + \text{post}(t))$. We denote this by $C \xrightarrow{t} C'$. We say that t is *silent* if $\text{pre}(t) = \text{post}(t)$. In particular, if t is silent and $C \xrightarrow{t} C'$, then $C = C'$. We write $C \to C'$ if $C \xrightarrow{t} C'$ for some $t \in T$. We write $C \xrightarrow{t_1 t_2 \cdots t_k} C'$ if there exist $C_0, C_1, \ldots, C_k \in \text{Pop}(Q)$ and $t_1, t_2, \ldots, t_k \in T$ such that $C = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \cdots C_k = C'$. We write $C \xrightarrow{*} C'$ if $C \xrightarrow{\sigma} C'$ for some $\sigma \in T^*$. We say that C' is *reachable* from C if $C \xrightarrow{*} C'$. The *support* of a sequence $\sigma = t_1 t_2 \cdots t_n \in T^*$ is $[\![\sigma]\!] \overset{\text{def}}{=} \{t_i : 1 \leq i \leq n\}$.

► **Example 1.** The flock-of-birds protocol mentioned in the introduction is formally defined as $\mathcal{P}_n = (Q, T, I, L, O)$ where $Q = \{0, 1, \ldots, n\}$, $I = \{1\}$, $L = \mathbf{0}$, $O(a) = 1 \iff a = n$, and where T consists of the following transitions:

$$\begin{aligned} s_{a,b} &: a, b \mapsto 0, \min(a + b, n) && \text{for every } 0 \leq a, b < n, \\ t_a &: a, n \mapsto n, n && \text{for every } 0 \leq a \leq n. \end{aligned}$$

$\mathcal{P}_n$ is 2-way and leaderless. Intuitively, it works as follows. Each agent stores a number. When two agents meet, one agent stores the sum of their values and the other one stores

0. Sums cap at n. Once an agent reaches n, all agents eventually get converted to n. To illustrate the above definitions, observe that: ${}^\bullet s_{2,3} = \{2, 3\}$, $t_2^\bullet = \{n\}$, $\mathrm{pre}(s_{2,3}) = \wr 2, 3 \wr$ and $\mathrm{post}(t_2) = \wr n, n \wr$. Configuration $\wr 1, 1, 1 \wr$ is initial, but $\wr 1, 0, 2 \wr$ is not. We have $\wr 1, 1, 1 \wr \xrightarrow{s_{1,1}} \wr 1, 0, 2 \wr \xrightarrow{t_0} \wr 1, 2, 2 \wr \xrightarrow{t_1} \wr 2, 2, 2 \wr$, or more concisely $\wr 1, 1, 1 \wr \xrightarrow{\sigma} \wr 2, 2, 2 \wr$ where $\sigma = s_{1,1} t_0 t_1$.

Computing with population protocols. An *execution* π is an infinite sequence of configurations $C_0 C_1 \cdots$ such that $C_0 \to C_1 \to \cdots$. We say that π is *fair* if for every configuration D the following holds[1]:

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

In other words, fairness ensures that a configuration cannot be avoided forever if it can be reached infinitely often along π. We say that a configuration C is a *consensus configuration* if $O(p) = O(q)$ for every $p, q \in [\![C]\!]$. If a configuration C is a consensus configuration, then its *output* $O(C)$ is the unique output of its states, otherwise it is $\bot$. An execution $\pi = C_0 C_1 \cdots$ *stabilizes* to $b \in \{0, 1\}$ if $O(C_i) = O(C_{i+1}) = \cdots = b$ for some $i \in \mathbb{N}$. The *output* of π is $O(\pi) \overset{\text{def}}{=} b$ if it stabilizes to b, and $O(\pi) \overset{\text{def}}{=} \bot$ otherwise. A consensus configuration C is *stable* if every configuration C' reachable from C is a consensus configuration such that $O(C') = O(C)$. It can easily be shown that a fair execution stabilizes to $b \in \{0, 1\}$ if and only if it contains a stable configuration whose output is b.

A population protocol $\mathcal{P} = (Q, T, I, L, O)$ is *well-specified* if for every initial configuration C_0, there exists $b \in \{0, 1\}$ such that every fair execution π starting at C_0 has output b. If $\mathcal{P}$ is well-specified, then we say that it *computes* the predicate $\varphi\colon \mathrm{Pop}(I) \to \{0, 1\}$ if for every $D \in \mathrm{Pop}(I)$, every fair execution starting at $D + L$ has output $\varphi(D)$.

▶ **Example 2.** Consider the protocol $\mathcal{P}_2$ defined in Example 1 (i.e, $n = 2$). We have $O(\wr 1, 1, 1 \wr) = 0$, $O(\wr 2, 2, 2 \wr) = 1$ and $O(\wr 1, 0, 2 \wr) = \bot$. The execution $\wr 1, 1, 1 \wr \to \wr 1, 0, 2 \wr \to \wr 1, 2, 2 \wr \to \wr 2, 2, 2 \wr \to \wr 2, 2, 2 \wr \to \cdots$ is fair and its output is 1. However, the execution $\wr 1, 1, 1 \wr \to \wr 1, 0, 2 \wr \to \wr 1, 0, 2 \wr \to \cdots$ is not fair since $\wr 1, 0, 2 \wr$ occurs infinitely often and can lead to $\wr 2, 2, 2 \wr$ which does not occur.

Leaders. Intuitively, leaders are extra agents present in every initial configuration. Allowing a large number of leaders may help to compute predicates with fewer states. To illustrate this, consider the leaderless protocol of Example 1. It computes $x \geq n$ with $n + 1$ states. We describe a 2-way protocol with only 4 states, but n leaders. It is an adaptation of the well-known basic majority protocol (see, e.g., [8]). Let $\mathcal{P}'_n = (Q, T, I, L_n, O)$ be the protocol where $Q \overset{\text{def}}{=} \{x, y, \overline{x}, \overline{y}\}$, $I \overset{\text{def}}{=} \{x\}$, $L_n \overset{\text{def}}{=} \wr n \cdot y \wr$, $O(x) = O(\overline{x}) \overset{\text{def}}{=} 1$, $O(y) = O(\overline{y}) \overset{\text{def}}{=} 0$, and where T consists of the following transitions:

$$x, y \mapsto \overline{x}, \overline{y}, \qquad x, \overline{y} \mapsto x, \overline{x}, \qquad y, \overline{x} \mapsto y, \overline{y}, \qquad \overline{x}, \overline{y} \mapsto \overline{x}, \overline{x}.$$

Informally, "active" agents in states x and y collide and become "passive" agents in states $\overline{x}$ and $\overline{y}$. At some point, some active agents "win" and convert all passive agents to their output. It is known that this protocol is well-specified and computes the predicate $x \geq y$ when there are no leaders (i.e., if we set $L_n = \mathbf{0}$). So, by initially fixing n leaders in state y, $\mathcal{P}'_n$ computes $x \geq n$.

[1] This definition of fairness differs from the original definition of Angluin et al. [4], but is equivalent.

Thus, the predicate $x \geq n$ can be computed either with $O(n)$ states and no leaders, or with 4 states and $O(n)$ leaders. This indicates a trade-off between states and leaders, and one should avoid hiding all of the complexity in one of them. For this reason, we make these two quantities explicit in all of our results.

The reason for considering protocols with leaders is that, as we shall see, even a constant number of leaders demonstrably leads to exponentially more compact protocols for some predicates. Other papers have made similar observations with respect to other resource measures (see e.g. [5, 14]).

From k-way to 2-way protocols. In our constructions it is very convenient to use k-way transitions for $k > 2$. The following lemma shows that k-way protocols can be transformed into 2-way protocols by introducing a few extra states. Intuitively, a k-way transition is simulated by a chain of 2-way transitions. The first part of the chain "collects" k participants one by one. First, two agents agree to participate, and one of them becomes "passive", while the second "searches" for a third participant. This is iterated until k participants are collected. In the second part, the last collected agent "informs" all passive agents, one by one, that k agents have been collected; upon hearing this, the passive agents move to their destination states and become active again. To prevent faulty behavior when there are not enough agents, all transitions of the first part can be "reversed", that is, the agent that is currently searching and the last collected agent can "repent" and "undo" the transition. While the construction is simple and intuitive, its correctness proof is very involved, because agents that reach their destination can engage in other interactions while other participants are still passive. The construction is presented in the full version of this paper.

▶ **Lemma 3.** *Let $\mathcal{P} = (Q, T, I, L, O)$ be a well-specified k-way population protocol. For every $3 \leq i \leq k$, let n_i be the number of i-way transitions of $\mathcal{P}$. There exists a 2-way population protocol $\mathcal{P}'$, with at most $|Q| + \sum_{3 \leq i \leq k} 3i \cdot n_i$ states, which is well-specified and computes the same predicate as $\mathcal{P}$.*

3 Leaderless protocols for $x \geq n$

In this section, we consider *leaderless* protocols for the predicate $x \geq n$. We first show that the number of states required to compute this predicate can be reduced from the known $O(n)$ bound to $O(\log n)$, using a similar binary encoding as in [1]. Then we show an existential lower bound of $O((\log n)^{1/4})$.

A protocol with $O(\log n)$ states. We describe a leaderless size(n)-way protocol $\mathcal{P}_n = (Q_n, T_n, I_n, \mathbf{0}, O_n)$ with size$(n) + 3$ states that computes $x \geq n$. The states are $Q_n \overset{\text{def}}{=} \{\mathbf{0}, \mathbf{2^0}, \ldots, \mathbf{2^{size(n)}}, \mathbf{n}\}$ and the sole initial state is $I_n \overset{\text{def}}{=} \{\mathbf{2^0}\}$. The output mapping is defined as $O_n(\mathbf{n}) \overset{\text{def}}{=} 1$ and $O_n(q) \overset{\text{def}}{=} 0$ for every state $q \neq \mathbf{n}$.

Before defining the set T_n of transitions, we need some preliminaries. For every state $q \in Q_n$, let val(q) denote the number q stands for, i.e. val$(\mathbf{0}) = 0$, val$(\mathbf{n}) = n$ and val$(\mathbf{2^i}) = 2^i$ for every $0 \leq i \leq$ size(n). Moreover, for every configuration C, let val$(C) \overset{\text{def}}{=} \sum_{q \in Q_n}$ val$(q) \cdot C(q)$. A configuration C is a *representation of* m if val$(C) = m$. For example, the configuration $\wr \mathbf{0}, \mathbf{2^1}, 5 \cdot \mathbf{2^3} \wr$ is a representation of $0 + 2^1 + 5 \cdot 2^3 = 42$. Observe that every initial configuration C_0 is a representation of $|C_0|$.

T_n is the union of two sets T_n^1 and T_n^2. Intuitively, T_n^1 allows the protocol to reach from a representation of a number, say m, other representations of m. Formally, the transitions of

T_n^1 are:

$$\begin{array}{rcll} \mathbf{2^i}, \mathbf{2^i} & \mapsto & \mathbf{2^{i+1}}, \mathbf{0} & \text{for every } 0 \leq i < \text{size}(n) \\ \mathbf{2^{i+1}}, \mathbf{0} & \mapsto & \mathbf{2^i}, \mathbf{2^i} & \text{for every } 0 \leq i < \text{size}(n) \\ \wr \mathbf{2^i} : i \in \text{bits}(n) \wr & \mapsto & \mathbf{n}, \underbrace{\mathbf{0}, \cdots, \mathbf{0}}_{|\text{bits}(n)|-1 \text{ copies}} & \end{array}$$

The transitions of T_n^2 allow agents in state $\mathbf{n}$ to "attract" all other agents to $\mathbf{n}$. Formally, they are:

$$\mathbf{n}, q \quad \mapsto \quad \mathbf{n}, \mathbf{n} \quad \text{for every } q \in Q_n.$$

Let us show that $\mathcal{P}_n$ computes $x \geq n$. Let $C_0 = \wr m \cdot \mathbf{2^0} \wr$. If $m < n$, then $C(\mathbf{n}) = 0$ holds for every representation C of m. Therefore, every configuration C reachable from C_0 satisfies $C(\mathbf{n}) = 0$ and, since $\mathbf{n}$ is the only state with output 1, the protocol stabilizes to 0. If $m \geq n$, then it is possible to reach a representation C of m satisfying $C(\mathbf{n}) > 0$, for example $C = \wr \mathbf{n}, (m - n) \cdot \mathbf{2^0} \wr$. Since for every transition $\mathbf{2^i}, \mathbf{2^i} \mapsto \mathbf{2^{i+1}}, \mathbf{0}$ the set T_n also contains the reverse transition $\mathbf{2^{i+1}}, \mathbf{0} \mapsto \mathbf{2^i}, \mathbf{2^i}$, every representation C of m satisfying $C(\mathbf{n}) = 0$ can reach a representation C' of m satisfying $C'(\mathbf{n}) > 0$. Let $\pi = C_0 C_1 C_2 \cdots$ be a fair execution. By fairness, there is some $i \in \mathbb{N}$ such that $C_i(\mathbf{n}) > 0$. Again by fairness, and because of T_n^2, there is also an index j such that $C_k = \wr m \cdot \mathbf{n} \wr$ for every $k \geq j$, and so π stabilizes to 1.

Note that $|Q_n| = \text{size}(n) + 3$. Moreover, $\mathcal{P}_n$ has one $|\text{bits}(n)|$-way transition. Thus, by Lemma 3, we obtain the following theorem:

▶ **Theorem 4.** *There exists a family $\{\mathcal{P}_0, \mathcal{P}_1, \ldots\}$ of leaderless and 2-way population protocols such that $\mathcal{P}_n$ has at most $4\lfloor \log n \rfloor + 7$ states and computes the predicate $x \geq n$.*

An existential $(\log n)^{1/4}$ lower bound. We show that every family $\{\mathcal{P}_n\}_{n \in \mathbb{N}}$ of leaderless and 2-way protocols computing the family of predicates $\{x \geq n\}_{n \in \mathbb{N}}$ must contain infinitely many members of size $\Omega((\log n)^{1/4})$. We call this an existential lower bound, contrary to a universal lower bound, which would state that $\mathcal{P}_n$ has size $\Omega((\log n)^{1/4})$ for every $n \geq 1$.

▶ **Theorem 5.** *Let $\{\mathcal{P}_0, \mathcal{P}_1, \ldots\}$ be an infinite family of leaderless and 2-way population protocols such that $\mathcal{P}_n$ computes the predicate $x \geq n$ for every $n \in \mathbb{N}$. There exist infinitely many indices n such that $\mathcal{P}_n$ has at least $(\log n)^{1/4}$ states.*

Proof sketch. The proof boils down to bounding the number $d(m)$ of unary predicates computed by protocols with m states. The number of distinct sets of transitions, excluding silent ones, is bounded by $2^{m^4 - m^2}$. The number of possible initial states and output mappings are respectively m and 2^m. Altogether, we obtain:

$$d(m) \leq 2^{m^4 - m^2} \cdot m \cdot 2^m = 2^{m^4} \cdot \frac{2^m \cdot m}{2^{m^2}} \leq 2^{m^4}. \qquad ◀$$

4 A $O(\log \log n)$ protocol with leaders for some $x \geq n$

The lower bound of Section 3 is not valid for every n, it only ensures that, for some values of n, protocols computing $x \geq n$ must have a logarithmic number of states. We prove that, surprisingly, there is an infinite sequence $n_1 < n_2 < \cdots$ of values that break through the logarithmic barrier: The predicates $x \geq n_i$ can be computed by protocols with only $O(\log \log n_i)$ states and two leaders. So, loosely speaking, a flock of birds can decide if it contains at least n_i birds, even though no bird has enough memory to index a bit of n_i.

The result is based on a construction of [15]. In this paper, Mayr and Meyer study the word problem for commutative semigroup presentations. Given a finite set $\mathcal{A}$ of generators, a presentation of a commutative semigroup generated by $\mathcal{A}$ is a finite set of productions $\mathcal{S} = \{l_1 \to r_1, \dots, l_m \to r_m\}$, where $l_i, r_i \in \mathcal{A}^*$ for every $1 \leq i \leq m$, satisfying:

- Commutativity: $ab \to ba \in \mathcal{S}$ for every $a, b \in \mathcal{A}$;[2] and
- Reversibility: if $l \to r \in \mathcal{S}$, then $r \to l \in \mathcal{S}$.

Given $\alpha, \beta \in \mathcal{A}^*$, we say that β is *derived* from α *in one step*, denoted by $\alpha \to \beta$, if $\alpha = \gamma\, l\, \delta$ and $\beta = \gamma\, r\, \delta$ for some $\gamma, \delta \in \mathcal{A}^*$ and some $r \to l \in \mathcal{S}$. We say that β is *derived* from α if $\alpha \xrightarrow{*} \beta$, where $\xrightarrow{*}$ is the reflexive transitive closure of the relation induced by $\to$. Observe that, by reversibility, we have $\alpha \xrightarrow{*} \beta$ iff $\beta \xrightarrow{*} \alpha$. Further, by commutativity we have $\alpha \xrightarrow{*} \beta$ iff $\pi(\alpha) \xrightarrow{*} \pi'(\beta)$ for every permutation π of $\mathcal{A}$.

Mayr and Meyer study the following question: given a commutative semigroup presentation $\mathcal{S}$ over $\mathcal{A}$, and initial and final letters $s, f \in \mathcal{A}$, what is the length of the shortest word α such that $s \xrightarrow{*} f\alpha$? They exhibit a family of presentations of size $O(n)$ for which the shortest α has double exponential length 2^{2^n}. More precisely, in [15, Sect. 6], they construct a family $\{\mathcal{S}_n\}_{n \geq 1}$ of presentations over alphabets $\{\mathcal{A}_n\}_{n \geq 1}$ satisfying the following properties:

(1) $|\mathcal{A}_n| = 14n + 10$, $|\mathcal{S}_n| = 20n + 8$, and $\max\{|l|, |r| : l \to r \in \mathcal{S}_n\} = 5$.

(2) $\{s_n, f_n, b_n, c_n\} \subseteq \mathcal{A}_n$ for every $n \geq 1$.

(3) $s_n c_n \xrightarrow{*} f_n \alpha$ iff $\alpha = c_n b_n^{2^{2^n}}$ [15, Lemma 6 and 8].

To apply this result, for each $n \geq 1$ we construct a 5-way population protocol $\mathcal{P}_n = (Q_n, T_n, I_n, L_n, O_n)$ with two leaders as follows:

- $Q_n \overset{\text{def}}{=} \mathcal{A}_n \cup \{x\}$ for some $x \notin \mathcal{A}_n$.
- $T_n \overset{\text{def}}{=} T_n^1 \cup T_n^2$, where:
  - T_n^1 contains a transition pad(p) for every production $p = l \to r$ of $\mathcal{S}_n$, obtained by "padding" p with x so that its left and right sides have the same length. For example, $\text{pad}(aab \to cd) = a, a, b \mapsto c, d, x$, and $\text{pad}(a \to bc) = a, x \mapsto b, c$,
  - $T_n^2 \overset{\text{def}}{=} \{f_n, q \mapsto f_n, f_n \mid q \in Q_n\}$,
- $I_n \overset{\text{def}}{=} \{x\}$,
- $L_n \overset{\text{def}}{=} \wr c_n, s_n \wr$, and
- $O_n(f_n) \overset{\text{def}}{=} 1$ and $O_n(q) \overset{\text{def}}{=} 0$ for every $q \neq f_n$.

Intuitively, T_n^1 allows $\mathcal{P}_n$ to simulate derivations of $\mathcal{S}_n$: a step $C \xrightarrow{\text{pad}(p)} C'$ of $\mathcal{P}_n$ simulates a one-step derivation of $\mathcal{S}_n$. We make this more precise. Given $\alpha \in \mathcal{A}_n^*$ and $m \geq |\alpha|$, let $C_{\alpha,m}$ be the configuration of $\mathcal{P}_n$ defined as follows: $C_{\alpha,m}(x) = m$, and $C_{\alpha,m}(a) = |\alpha|_a$ for every $a \in \mathcal{A}_n$, where $|\alpha|_a$ is the number of occurrences of a in α. Further, given a configuration C of $\mathcal{P}_n$, let α_C be the element of $\mathcal{S}_n$ given by $\alpha_C = a_1^{C(a_1)} \cdots a_m^{C(a_m)}$, where $a_1, \dots, a_m$ is a fixed enumeration of $\mathcal{A}_n$. We have:

► **Lemma 6.** *Let $\alpha, \beta \in \mathcal{A}_n^*$ and let C, C' be configurations of $\mathcal{P}_n$.*

(a) *If $\alpha \xrightarrow{p_1 \cdots p_k} \beta$ in $\mathcal{S}_n$, then for every $m \geq 4k$, $C_{\alpha,m} \xrightarrow{\text{pad}(p_1) \cdots \text{pad}(p_k)} C_{\beta,m'}$ in $\mathcal{P}_n$ for some $m' \geq 0$.*

(b) *If $C \xrightarrow{\text{pad}(p_1) \cdots \text{pad}(p_k)} C'$ in $\mathcal{P}_n$, then $\alpha_C \xrightarrow{p_1 \cdots p_k} \alpha_{C'}$ in $\mathcal{S}_n$.*

From Lemma 6, (1) and (3), the following can be shown:

► **Theorem 7.** *For every $n \in \mathbb{N}$, there is a 5-way protocol $\mathcal{P}_n$ with at most $14n+11$ states and at most $34n + 19$ transitions that computes the predicate $x \geq c_n$ for some number $c_n \geq 2^{2^n}$.*

[2] In [15], the elements of S are written using uppercase letters. We use lowercase for convenience.

Using Theorem 7 and Lemma 3, we obtain:

▶ **Corollary 8.** *There exists a family $\{\mathcal{P}_0, \mathcal{P}_1, \dots\}$ of 2-way protocols with two leaders and a family $\{c_0, c_1, \dots\}$ of natural numbers such that for every $n \in \mathbb{N}$ the following holds: $c_n \geq 2^{2^n}$ and protocol $\mathcal{P}_n$ has at most $314 \log\log c_n + 131$ states and computes the predicate $x \geq c_n$.*

5 Universal lower bounds for 1-aware protocols

To the best of our knowledge, all the protocols in the literature for predicates $x \geq n$, including those of Section 3 and Section 4, share a very natural property: if the number of agents is greater than or equal to n, then the agents not only eventually reach consensus 1, they also eventually *know* that they will reach this consensus. Let us formalize this idea:

▶ **Definition 9.** A well-specified population protocol $\mathcal{P} = (Q, T, I, L, O)$ is *1-aware* if there is a set $Q_1 \subseteq Q \setminus (I \cup [\![L]\!])$ of states such that for every initial configuration C_0 and every fair execution $\pi = C_0 C_1 \cdots$

(1) if π stabilizes to 0, then $C_i(Q_1) = 0$ for every $i \geq 0$, and
(2) if π stabilizes to 1, then there is some $i \geq 0$ such that $C_j(Q \setminus Q_1) = 0$ for every $j \geq i$.

If in the course of an execution π an agent reaches a state of Q_1, then π cannot stabilize to 0 by (1), and so, since $\mathcal{P}$ is well-specified, it stabilizes to 1; intuitively, at this moment the agent "knows" that the consensus will be 1. Further, if an execution stabilizes to 1, then all agents eventually reach and remain in Q_1 by (2), and so eventually all agents "know".[3] Albeit seemingly restrictive, 1-aware protocols compute a significant subclass of predicates: monotonic Presburger predicates (see the full version of the paper for more details).

We say that a state q is *coverable* from a configuration C if $C \xrightarrow{*} C'$ for some configuration C' such that $C'(q) > 0$. The fundamental property of 1-aware protocols is that, loosely speaking, consensus reduces to coverability:

▶ **Lemma 10.** *Let $\mathcal{P} = (Q, T, \{x\}, L, O)$ be a 1-aware protocol computing a unary predicate φ. We have $\varphi(n) = 1$ if and only if some state of Q_1 is coverable from $\wr n \cdot x \wr + L$.*

We show that for 1-aware protocols, the bounds of Sections 3 and 4 are essentially tight.

Leaderless protocols. We prove that a 1-aware, leaderless and 2-way protocol computing $x \geq n$ has at least $\log_3 n$ states. By Lemma 10, it suffices to show that some state of Q_1 is coverable from $\wr 3^k \cdot q \wr$, where q is the initial state. Proposition 11 below is the key to the proof. It states that for every finite execution $C_1 \xrightarrow{\pi} C_2$, there is $C'_1 \xrightarrow{\pi'} C'_2$ such that C'_1 has the same support as C_1 and is not too large, and C'_2 contains a "record" of all states encountered during the execution of π (this is the set $[\![C_1]\!] \cup [\![\pi]\!]^\bullet$).

Let us define the *norm* of a configuration C as $\|C\| \overset{\text{def}}{=} \max\{C(q) : q \in [\![C]\!]\}$. We obtain:

▶ **Proposition 11.** *Let $\mathcal{P} = (Q, T, I, L, O)$ be a k-way population protocol and let $C_1 \xrightarrow{\pi} C_2$ be a finite execution of $\mathcal{P}$. There exists a finite execution $C'_1 \xrightarrow{\pi'} C'_2$ such that (a) $[\![C'_1]\!] = [\![C_1]\!]$, (b) $[\![C'_2]\!] = [\![C_1]\!] \cup [\![\pi']\!]^\bullet$, and (c) $\|C'_1\| \leq (k+1)^{|Q|}$.*

[3] We could also require the seemingly weaker property that eventually at least one agent "knows". However, by adding transitions that "attract" all other agents to Q_1, we can transform a protocol in which some agent "knows" into a protocol computing the same predicate in which all agents "know".

▶ **Theorem 12.** *Every 1-aware, leaderless and 2-way population protocol $\mathcal{P} = (Q, T, \{q_0\}, \mathbf{0}, O)$ computing $x \geq n$ has at least $\log_3 n$ states.*

Proof. Let $Q_1 \subseteq Q$ be the set of states from the definition of 1-awareness. Since $L = \mathbf{0}$, $C_0 = \wr n \cdot q_0 \wr$ is the smallest initial configuration with output 1, and by Lemma 10 the smallest initial configuration from which some state $q_1 \in Q_1$ is coverable. Let $C_0 \xrightarrow{\pi} C \geq \wr q_1 \wr$. Since $q_1 \neq q_0$, we have $q_1 \in [\![\pi]\!]^\bullet$. By Proposition 11, and since $\mathcal{P}$ is 2-way, q_1 is also coverable from C_0' satisfying $[\![C_0']\!] = [\![C_0]\!] = \{q_0\}$ and $\|C_0'\| = 3^{|Q|}$. Thus, $C_0' = \wr 3^{|Q|} \cdot q_0 \wr$. By minimality of n, we get $n \leq 3^{|Q|}$, and thus $|Q| \geq \log_3 n$. ◀

Observe that the proof Theorem 12 uses the fact that $\mathcal{P}$ is leaderless to conclude $C_0' = \wr 3^{|Q|} \cdot q_0 \wr$ from $[\![C_0']\!] = [\![C_0]\!]$ and $\|C_0'\| = 3^{|Q|}$, which is not necessarily true with leaders.

Protocols with leaders. In the case of protocols with leaders we obtain a lower bound from Rackoff's procedure for the coverability problem of vector addition systems [17].

A *vector addition system* of dimension k (k-VAS) is a pair $(A, \boldsymbol{v}_0)$, where $\boldsymbol{v}_0 \in \mathbb{N}^k$ is an initial vector and $A \subseteq \mathbb{Z}^k$ is a set of vectors. An execution of a k-VAS is a sequence $\boldsymbol{v}_0 \boldsymbol{v}_1 \cdots \boldsymbol{v}_n$ of vectors of $\mathbb{N}^k$ such that each $\boldsymbol{v}_{i+1} = \boldsymbol{v}_i + \boldsymbol{a}_i$ for some $\boldsymbol{a}_i \in A$. We write $\boldsymbol{v}_0 \xrightarrow{*} \boldsymbol{v}_n$ and say that the execution has *length* n. A vector $\boldsymbol{v}$ is *coverable* in $(A, \boldsymbol{v}_0)$ if $\boldsymbol{v}_0 \xrightarrow{*} \boldsymbol{v}'$ for some $\boldsymbol{v}' \geq \boldsymbol{v}$. The *size* of a vector $\boldsymbol{v} \in \mathbb{Z}^k$ is $\sum_{1 \leq i \leq k} \text{size}(\max(|\boldsymbol{v}(i)|, 1))$. The *size* of a set of vectors is the sum of the size of its vectors. In [17] Rackoff proves:

▶ **Theorem 13** ([17]). *Let $A \subseteq \mathbb{Z}^k$ be a set of vectors of size at most n and dimension $k \leq n$, and let $\boldsymbol{v}_0 \in \mathbb{N}^k$ be a vector of size n. For every $\boldsymbol{v} \in \mathbb{N}^k$, if $\boldsymbol{v}$ is coverable in $(A, \boldsymbol{v}_0)$, then $\boldsymbol{v}$ is coverable by means of an execution of length at most $2^{(3n)^n}$.*

Using a standard construction from the Petri net literature, it can be shown that every 2-way protocol $\mathcal{P}$ with n states can be simulated by a VAS $\mathcal{V}_\mathcal{P}$ of size at most $12n^8$, where each execution of $\mathcal{P}$ has a corresponding execution twice as long in $\mathcal{V}_\mathcal{P}$. Thus, by Theorem 13:

▶ **Proposition 14.** *Let $\mathcal{P} = (Q, T, I, L, O)$ be a 2-way population protocol and let $q \in Q$. For every configuration C, if q is coverable from C, then it is coverable by means of a finite execution of length at most $2^{(3m)^m - 1}$ where $m = 12|Q|^8$.*

Using the above proposition, we derive:

▶ **Theorem 15.** *Let $\mathcal{P}$ be a 1-aware and 2-way population protocol. For every $n \geq 2$, if $\mathcal{P}$ computes $x \geq n$, then $\mathcal{P}$ has at least $(\log\log(n)/151)^{1/9}$ states.*

6 Protocols for systems of linear inequalities

In Section 3, we have shown that the predicate $x \geq c$ can be computed by a leaderless protocol with $O(\log c)$ states. In this section, we will see that adding a few leaders allows to compute systems of linear inequalities. More formally, we show that there exists a protocol with $O((m+k) \cdot \log(dm))$ states and $O(m \cdot \log(dm))$ leaders computing the predicate $A\boldsymbol{x} \geq \boldsymbol{c}$, where $A \in \mathbb{Z}^{m \times k}$, $\boldsymbol{c} \in \mathbb{Z}^m$ and d is the the largest absolute value occuring in A and $\boldsymbol{c}$.

There are three crucial points that make systems of linear inequalities more complicated than flock-of-birds predicates: (1) variables have *coefficients*, (2) coefficients may be positive or *negative*, and (3) they are the *conjunction* of linear inequalities. We will explain how to address the two first points by considering the special case of linear inequalities. We will then discuss how to handle the third point.

Linear inequalities. Note that the predicate $\sum_{1 \leq i \leq k} a_i x_i \geq c$ is equivalent to $\sum_{1 \leq i \leq k} a_i x_i + (1-c) > 0$. Therefore, it suffices to describe protocols for predicates of the form $\sum_{1 \leq i \leq k} a_i x_i + c > 0$. In order to make the presentation more pleasant, we will first restrain ourselves to the predicate $ax - by + c > 0$ for some fixed $a, b \in \mathbb{N}$ and $c \in \mathbb{Z}$. Such a predicate admits the difficult aspects, i.e. coefficients and negative numbers. Moreover, as we will see, handling more than two variables is not an issue.

Let us now describe a protocol $\mathcal{P}_{\text{lin}}$ for the predicate $ax - by + c > 0$. The idea is to keep a representation of $ax - by + c$ throughout executions of the protocol. Let $n \overset{\text{def}}{=} \text{size}(\max(\log |a|, \log |b|, \log |c|, 1))$. As in Section 3, we construct states to represent powers of two. However, this time, we also need states to represent negative numbers:

$$Q^+ \overset{\text{def}}{=} \{\mathbf{+2^i} : 0 \leq i \leq n\} \quad \text{and} \quad Q^- \overset{\text{def}}{=} \{\mathbf{-2^i} : 0 \leq i \leq n\}.$$

We also need states $X \overset{\text{def}}{=} \{\mathbf{x}, \mathbf{y}\}$ for the variables, and two additional states $R \overset{\text{def}}{=} \{\mathbf{+0}, \mathbf{-0}\}$. The set of all states of $\mathcal{P}_{\text{lin}}$ is $Q \overset{\text{def}}{=} X \cup Q^+ \cup Q^- \cup R$, and the initial states are $I \overset{\text{def}}{=} X$.

Let us explain the purpose of R. Intuitively, we would like to have the transitions:

$$x \mapsto \wr \mathbf{+2^i} : i \in \text{bits}(a) \wr \quad \text{and} \quad y \mapsto \wr \mathbf{-2^i} : i \in \text{bits}(|b|) \wr.$$

This way, every agent in state $\mathbf{x}$ (resp. $\mathbf{y}$) could be converted to the binary representation of a (resp. b). Unfortunately, this is not possible as these transitions produce more states than they consume. This is where leaders become useful. If R initially contains enough leaders, then R can act as a *reservoir* of extra states which allow to "pad" transitions. More formally, let $\text{rep}(z) \colon \mathbb{Z} \to \text{Pop}(Q \setminus X)$ be defined as follows:

$$\text{rep}(z) \overset{\text{def}}{=} \begin{cases} \wr \mathbf{+2^i} : i \in \text{bits}(z) \wr & \text{if } z > 0, \\ \wr \mathbf{-2^i} : i \in \text{bits}(|z|) \wr & \text{if } z < 0, \\ \wr \mathbf{-0} \wr & \text{if } z = 0. \end{cases}$$

For every $r \in R$, we add to $\mathcal{P}_{\text{lin}}$ the following transitions:

$$\text{add}_{\mathbf{x},r} : \ \mathbf{x}, \underbrace{r, r, \ldots, r}_{|\text{rep}(a)|-1 \text{ times}} \mapsto \text{rep}(a) \quad \text{and} \quad \text{add}_{\mathbf{y},r} : \ \mathbf{y}, \underbrace{r, r, \ldots, r}_{|\text{rep}(b)|-1 \text{ times}} \mapsto \text{rep}(b).$$

We set the leaders to $L \overset{\text{def}}{=} \text{rep}(c) + \wr (4n+2) \cdot \mathbf{-0} \wr$. We claim that $4n + 2$ reservoir states are enough, we will explain later why. Now, the key idea of the construction is that it is always possible to put $2n$ agents back into R. Thus, fairness ensures that the number of agents in X eventually decreases to zero, and then that the value represented over $Q^+ \cup Q^-$ is $ax - by + c$. We let the representations over Q^+ and Q^- "cancel out" until one side "wins". If the positive (resp. negative) side wins, i.e. if $ax - by + c > 0$ (resp. $ax - by + c \leq 0$), then it signals all agents in R to move to $\mathbf{+0}$ (resp. $\mathbf{-0}$). To achieve this, for every $0 \leq i \leq n$, we add transition $\text{cancel}_i : \mathbf{+2^i}, \mathbf{-2^i} \mapsto \mathbf{+0}, \mathbf{-0}$ to the protocol. Since bits of the positive and negative numbers may not be "aligned", we follow the idea of Section 3 and add further transitions to change representations to equivalent ones:

$$\begin{aligned} &\text{up}_i^+ : \ \mathbf{+2^i}, \mathbf{+2^i} \mapsto \mathbf{+2^{i+1}}, \mathbf{+0}, \qquad && \text{down}_{i+1,r}^+ : \ \mathbf{+2^{i+1}}, r \mapsto \mathbf{+2^i}, \mathbf{+2^i}, \\ &\text{up}_i^- : \ \mathbf{-2^i}, \mathbf{-2^i} \mapsto \mathbf{-2^{i+1}}, \mathbf{-0}, && \text{down}_{i+1,r}^- : \ \mathbf{-2^{i+1}}, r \mapsto \mathbf{-2^i}, \mathbf{-2^i}, \end{aligned}$$

where $0 \leq i < n$ and $r \in R$. Finally, for every $0 \leq i \leq n$, we add transitions to signal which side wins:

$$\begin{aligned} &\text{signal}_i^+ : \ \mathbf{+2^i}, \mathbf{-0} \mapsto \mathbf{+2^i}, \mathbf{+0}, \qquad && \text{signal} : \ \mathbf{-0}, \mathbf{+0} \mapsto \mathbf{-0}, \mathbf{-0}, \\ &\text{signal}_i^- : \ \mathbf{-2^i}, \mathbf{+0} \mapsto \mathbf{-2^i}, \mathbf{-0}. && \end{aligned}$$

Note that $\mathbf{-0}$ "wins" over $\mathbf{+0}$ because the predicate is false whenever $ax - by + c = 0$. It remains to specify the output mapping of $\mathcal{P}_{\text{lin}}$ which we define as expected, i.e. $O(q) \overset{\text{def}}{=} 1$ if $q \in Q^+ \cup \{\mathbf{+0}\}$, and $O(q) \overset{\text{def}}{=} 0$ otherwise.

Let us briefly explain why $4n + 2$ reservoir states suffice. At any reachable configuration C, transitions of the form up_i^+ and up_i^- can occur until $C(\mathbf{\pm 2^i}) \leq 1$ for every $0 \leq i < n$. Afterwards, at most $2n$ agents remain in these states. There can however be many agents in $S = \{\mathbf{+2^n}, \mathbf{-2^n}\}$. But, these two states represent numbers respectively larger and smaller than any coefficient, hence the number of agents in S can only grow by one each time a state from X is consumed. Overall, this means that $C \xrightarrow{*} C'$ for some C' such that $C'(R) \geq 2n$.

In order to handle more variables $\{x_1, x_2, \ldots, x_k\}$, note that all we need to do is to set $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k}\}$ instead, and add transitions $\text{add}_{\mathbf{x_i},r}$ for every $1 \leq i \leq k$ and $r \in R$.

By applying Lemma 3 on $\mathcal{P}_{\text{lin}}$, we obtain:

▶ **Theorem 16.** *Let $a_1, a_2, \ldots, a_k, c \in \mathbb{Z}$ and let $n = \text{size}(\max(|a_1|, |a_2|, \ldots, |a_k|, |c|, 1))$. There exists a 2-way population protocol, with at most $10kn$ states and at most $5n+2$ leaders, that computes the predicate $\sum_{1 \leq i \leq k} a_i x_i + c > 0$.*

Conjunction of linear inequalities. We briefly explain how to lift the construction for linear inequalities to systems of linear inequalities. The details of the formal construction and proofs are a bit involved, and are thus deferred to the full version of this paper. Let us fix some $A \in \mathbb{Z}^{m \times k}$ and $\boldsymbol{c} \in \mathbb{Z}^m$. We sketch a protocol $\mathcal{P}_{\text{sys}}$ for the predicate $A\boldsymbol{x} + \boldsymbol{c} > \mathbf{0}$. For every $1 \leq i \leq m$, we construct a protocol $\mathcal{P}_i$ for the predicate $\sum_{1 \leq j \leq k} A_{i,j} \cdot \boldsymbol{x}_j + \boldsymbol{c}_i > 0$. Protocol $\mathcal{P}_i$ is obtained as presented earlier, but with some modifications. The largest power of two is picked as $n \overset{\text{def}}{=} \text{size}(d) + \lceil \log 2m^2 \rceil$ where

$$d \overset{\text{def}}{=} \max(1, \{|A_{i,j}| : 1 \leq i \leq m, 1 \leq j \leq k\}, \{|c_i| : 1 \leq i \leq m\}).$$

The reason for this modification is that the number of agents, in a largest power of two, should now increase by at most $1/m$ each time an initial state is consumed, as opposed to 1.

We also replace each *positive state* $q \in Q^+$ of $\mathcal{P}_i$ by two states q_0 and q_1, its *0-copy* and *1-copy*. The reason behind this is that positive states should not necessarily have output 1. Indeed, one linear inequality may be satisfied while the other ones are not. Therefore, $\mathbf{-0}$ and each *negative state* $q \in Q^-$ should be able to signal a 0-consensus to the positive states. The transitions of the form up_j^+, down_j^+ and cancel_j are adapted accordingly.

Protocol $\mathcal{P}_{\text{sys}}$ is obtained as follows. First, subprotocols $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$ are put side by side. Their initial (resp. reservoir) states are merged into a single set X (resp. R). For every $1 \leq j \leq k$, transitions $\text{add}_{\mathbf{x_j},r}$ of the m subprotocols are replaced by a single transition consuming $\mathbf{x_j}$, and enough reservoir states, and producing $\text{rep}(A_{i,j})$ in each subprotocol $\mathcal{P}_i$, where $1 \leq i \leq m$. The signal mechanisms are replaced by these new ones:

- the 0-copy of state $\mathbf{+2^0}$ of *all* subprotocols can meet to convert $\mathbf{-0}$ to $\mathbf{+0}$,
- state $\mathbf{+0}$ can convert any positive state to its 1-copy,
- state $\mathbf{-0}$ or any negative state can convert $\mathbf{+0}$ to $\mathbf{-0}$, and any positive state to its 0-copy.

A careful analysis of the formal construction of $\mathcal{P}_{\text{sys}}$ combined with Lemma 3 yields:

▶ **Theorem 17.** *Let $A \in \mathbb{Z}^{m \times k}$, $\boldsymbol{c} \in \mathbb{Z}^m$ and $n = \text{size}(\max(1, \{|A_{i,j}| : 1 \leq i \leq m, 1 \leq j \leq k\}),$ $\{|c_i| : 1 \leq i \leq m\})$. There exists a 2-way population protocol, with at most $27(\log m + n)(m + k)$ states and at most $14m(\log m + n)$ leaders, that computes the predicate $A\boldsymbol{x} + \boldsymbol{c} > \mathbf{0}$.*

7 Conclusion and further work

We have initiated the study of the state space size of population protocols as a function of the size of the predicate they compute. Previous lower bounds were only for single predicates, like the majority predicate $x \leq y$, or for a variant of the model in which the number of states is a function of the number of agents.

There are many open questions. We conjecture that systems of linear inequalities can be computed by leaderless protocols with a polynomial number of states. A second, very intriguing question is whether the function $f(n)$ giving the minimal number of states of a two-leader protocol computing $x \geq n$ exhibits large gaps, i.e., if there are (families of) numbers c and $c+1$ such that $f(c)$ is exponentially larger than $f(c+1)$. A third question is whether there exist protocols with $O(\log\log\log n)$ states for the flock-of-birds predicates $x \geq n$. Such protocols cannot be 1-aware, but they might exist. Their existence is linked to the long standing question of whether the reachability problem for reversible VAS (a model equivalent to the commutative semigroup representations of [15]) has the same complexity as reachability for arbitrary VAS (see [12] for a brief introduction).

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, 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.

3 Dana Angluin, James Aspnes, Melody Chan, Michael J. Fischer, Hong Jiang, and René Peralta. Stably computable properties of network graphs. In Viktor K. Prasanna, S. Sitharama Iyengar, Paul G. Spirakis, and Matt Welsh, editors, *Distributed Computing in Sensor Systems, First IEEE International Conference, DCOSS 2005, Marina del Rey, CA, USA, June 30 - July 1, 2005, Proceedings*, volume 3560 of *Lecture Notes in Computer Science*, pages 63–74. Springer, 2005. doi:10.1007/11502593_8.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In Soma Chaudhuri and Shay Kutten, editors, *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 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. *Distributed Computing*, 21(3):183–199, 2008. doi:10.1007/s00446-008-0067-z.

6 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007. doi:10.1007/s00446-007-0040-2.

7 James Aspnes. Clocked population protocols. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 431–440. ACM, 2017. doi:10.1145/3087801.3087836.

8 James Aspnes and Eric Ruppert. An introduction to population protocols. In *Middleware for Network Eccentric and Mobile Applications*, pages 97–120. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-540-89707-1_5.

9 Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. Brief announcement: Population protocols for leader election and exact majority with $O(\log^2 n)$ states and $O(\log^2 n)$ convergence time. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 451–453. ACM, 2017. doi:10.1145/3087801.3087858.

10 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. In Yoram Moses, editor, *Distributed Computing - 29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*, volume 9363 of *Lecture Notes in Computer Science*, pages 602–616. Springer, 2015. doi:10.1007/978-3-662-48653-5_40.

11 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Inf.*, 54(2):191–215, 2017. doi:10.1007/s00236-016-0272-3.

12 Alain Finkel and Jérôme Leroux. Recent and simple algorithms for petri nets. *Software and System Modeling*, 14(2):719–725, 2015. doi:10.1007/s10270-014-0426-0.

13 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.

14 Giuseppe Antonio Di Luna, Paola Flocchini, Taisuke Izumi, Tomoko Izumi, Nicola Santoro, and Giovanni Viglietta. Population protocols with faulty interactions: The impact of a leader. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 454–466, 2017. doi:10.1007/978-3-319-57586-5_38.

15 Ernst W. Mayr and Albert R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46(3):305–329, 1982. doi:10.1016/0001-8708(82)90048-2.

16 Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Mediated population protocols. *Theor. Comput. Sci.*, 412(22):2434–2450, 2011. doi:10.1016/j.tcs.2011.02.003.

17 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.

18 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008. doi:10.1007/s11047-008-9067-y.

B Succinct Population Protocols for Presburger Arithmetic, STACS, 2020.

This chapter has been published as **peer-reviewed conference paper**.

Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. Succinct Population Protocols for Presburger Arithmetic. In *Proceedings of the 37th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 40:1–40:15, 2020. doi:10.4230/LIPIcs.STACS.2020.40

Synopsis. In [BEJ18a] we showed that every predicate φ corresponding to a system of linear inequalities can be computed by a population protocol with poly($|\varphi|$) states and with $\mathcal{O}(|\varphi|)$ leaders. We generalize this result to the class of all Presburger-definable predicates φ, without leaders. We assume that φ is given in the quantifier-free fragment of Presburger arithmetic with remainder predicates. We show that every predicate φ can be computed by a leaderless population protocol with poly($|\varphi|$) states. Further, the protocol can be constructed in polynomial time. This is a major improvement to the previously known construction by Angluin *et al.*, which requires $2^{\mathrm{poly}(|\varphi|)}$ states [AAER07].

Contributions of thesis author. Composition and revision of the manuscript. Joint discussion and development of the results and proofs presented in the paper, with the following notable individual contributions: proposal of the notion of *reversible dynamic initialization* for succinct parallel composition; sole contribution of all constructions for small populations in the second half of the paper, and writing their correctness proofs.

Succinct Population Protocols for Presburger Arithmetic

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

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

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

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

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

Abstract

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

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

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

Keywords and phrases Population protocols, Presburger arithmetic, state complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.40

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

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

1 Introduction

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

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

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

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

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

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

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

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

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

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

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

2 Preliminaries

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

Presburger arithmetic. *Presburger arithmetic* (PA) is the first-order theory of $\mathbb{N}$ with addition, i.e. $\mathsf{FO}(\mathbb{N},+)$. For example, the PA formula $\psi(x,y,z) = \exists x' \exists z' (x = x'+x') \land (y = z+z') \land \neg(z'=0)$ states that x is even and that $y > z$. It is well-known that for every formula of PA there is an equivalent formula of quantifier-free Presburger arithmetic (QFPA) [13], the theory with syntax given by the grammar

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

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

3 Population protocols

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

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

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

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

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

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

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

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

An execution is a sequence of configurations $\sigma = C_0 C_1 \cdots$ such that $C_i \rightarrow C_{i+1}$ for every $i \in \mathbb{N}$. We write σ_i to denote configuration C_i. The *output* of an execution σ is defined as follows. If there exist $i \in \mathbb{N}$ and $b \in \{0,1\}$ such that $O(\sigma_i) = O(\sigma_{i+1}) = \cdots = b$, then $O(\sigma) \overset{\text{def}}{=} b$, and otherwise $O(\sigma) \overset{\text{def}}{=} \bot$.

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

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

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

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

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

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

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

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

4 Main result

The main result of this paper is the following theorem:

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

To prove Theorem 2, we first provide a construction that uses $\ell \in \mathcal{O}(|\varphi|^3)$ leaders. If there are at least $|\boldsymbol{v}| \geq \ell$ input agents $\boldsymbol{v}$ (*large inputs*), we will show how the protocol can be made leaderless by having agents encode both their state and the state of some leader. Otherwise, $|\boldsymbol{v}| < \ell$ (*small inputs*), and we will resort to a special construction, with a single leader, that only works for populations of bounded size. We will show how the leader can be simulated collectively by the agents. Hence, we will construct succinct protocols computing φ for large and small inputs, respectively. Formally, we prove:

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

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

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

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

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

5 Succinct protocols for large populations

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

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

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

5.1 From protocols with helpers to leaderless protocols

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

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

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

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

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

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

5.2 From multi-output protocols to protocols with helpers

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

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

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

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

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

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

5.3 From reversible dynamic initialization to multi-output protocols

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.4 Atomic predicates under reversible dynamic initialization

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

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

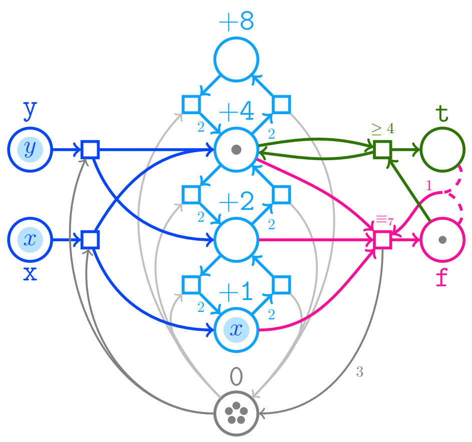


Figure 1 Partial representation of the protocol computing $5x + 6y \geq 4 \pmod 7$ as a Petri net, where places (circles), transitions (squares) and tokens (smaller filled circles) represent respectively states, transitions and agents. Non-helper agents remember their input variable (labeled here within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

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

The protocol has an *input state* `x` for each variable $x \in X$, *output states* `f` and `t`, a *neutral state* `0`, and *numerical states* of the form $+2^{\mathtt{i}}$ for every $0 \leq i \leq n$, where n is the smallest number such that $2^n > \|\varphi\|$. Initially, (at least) one helper is set to `f` and (at least) $2n$ helpers set to `0`. In order to compute $5x + 6y \geq 4 \pmod 7$ for $x := r$ and $y := s$, we initially place r and s agents in the states `x` and `y`, i.e., the agents in state `x` encode the number r in unary, and similarly for `y`. The blue transitions on the left of Figure 1 "convert" each agents in input states to a binary representation of their corresponding coefficient. In our example, agents in state `x` are converted to $\boldsymbol{a}(x) = 5 = 0101_2$ by putting one agent in `4` and another one in `1`. Since two agents are needed to encode 5, the transition "recruits" one helper from state `0`. Observe that, since the inputs can be arbitrarily large, but a protocol can only use a constant number of helpers, the protocol must reuse helpers in order to convert all agents in input states. This happens as follows. If two agents are in the same power of two, say $+2^{\mathtt{i}}$, then one of them can be "promoted" to $+2^{\mathtt{i+1}}$, while the other one moves to state `0`, “liberating” one helper. This allows the agents to represent the overall value of all converted agents in the most efficient representation. That is, from any configuration, one can always reach a configuration where there is at most one agent in each place $2^0, \dots, 2^{n-1}$, there are at most the number of agents converted from input places in place 2^n, and hence there are at least n agents in place 0, thus ready to convert some agent from the input place. Similar to promotions, “demotions” to smaller powers of two can also happen. Thus, the agents effectively shift through all possible binary representations of the overall value of all converted agents. The $\equiv_7$ transition in Figure 1 allows 3 agents in states `4`, `2` and `1` to “cancel out” by moving to state `0`, and it moves the output helper to `f`. Furthermore, there are RDI-transitions that allow to revert the effects of conversion and cancel transitions. These are not shown in Figure 1.

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

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

6 Succinct protocols for small populations

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

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

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

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

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

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

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

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

6.2 Computing boolean combinations of predicates for fixed-size inputs

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

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

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

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

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

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

6.3 Computing atomic predicates for fixed-size inputs

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

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

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

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

Greater-Sum calls procedures $\mathit{Probe}_1(j)$ and $\mathit{Probe}_2(j)$ that return the j-th bits of $\boldsymbol{\alpha x}$ and $\boldsymbol{\beta y}$, respectively, where $j = 1$ is the most significant bit. Since $|\boldsymbol{x}| \leq i$, and the largest constant in $\boldsymbol{\alpha}$ is at most $||\varphi||$, we have $\boldsymbol{\alpha} \cdot \boldsymbol{x} \leq i \cdot ||\varphi||$, and so $\boldsymbol{\alpha} \cdot \boldsymbol{x}$ has at most $m \overset{\text{def}}{=} |\varphi| + \lfloor \log(i) \rfloor + 1$ bits, and the same holds for $\boldsymbol{\beta y}$. So we have $1 \leq j \leq m$. Let us first describe Greater-Sum, and then $\mathit{Probe}_1(j)$; the procedure $\mathit{Probe}_2(j)$ is similar.

Greater-Sum$(\boldsymbol{x}, \boldsymbol{y})$ loops through $j = 1, \dots, m$. For each j, it calls $\mathit{Probe}_1(j)$ and $\mathit{Probe}_2(j)$. If $\mathit{Probe}_1(j) > \mathit{Probe}_2(j)$, then it answers $\varphi(\boldsymbol{x}, \boldsymbol{y}) = 1$, otherwise it moves to $j + 1$. If Greater-Sum reaches the end of the loop, then it answers $\varphi(\boldsymbol{x}, \boldsymbol{y}) = 0$. Observe that Greater-Sum only needs to store the current value of j and the bits returned by $\mathit{Probe}_1(j)$ and $\mathit{Probe}_2(j)$. Since $j \leq m$, Greater-Sum only needs $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory.

$\mathit{Probe}_1(j)$ uses a decreasing counter $k = m, \dots, j$ to successively compute the bits $b_1(k)$ of $\boldsymbol{\alpha} \cdot \boldsymbol{x}$, starting at the least significant bit. To compute $b_1(k)$, the procedure stores the carry $c_k \leq i$ of the computation of $b_1(k+1)$; it then computes the sum $s_k := c_k + \boldsymbol{\alpha}(k) \cdot \boldsymbol{x}$ (where $\boldsymbol{\alpha}(k)$ is the k-th vector of bits of $\boldsymbol{\alpha}$), and sets $b_k := s_k \bmod 2$ and $c_{k-1} := s_k \div 2$. The procedure needs $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory for counter k, $\log(i) + 1$ bits for encoding s_k, and $\mathcal{O}(\log(i))$ bits for encoding c_k. So it only uses $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory.

Let us now simulate Greater-Sum$(\boldsymbol{x}, \boldsymbol{y})$ by a halting protocol with one leader agent. Intuitively, the protocol proceeds in rounds corresponding to the counter k. The leader stores in its state the value j and the current values of the program counter, of counter k, and of variables b_k, s_k, and c_k. The crucial part is the implementation of the instruction $s_k := c_k + \boldsymbol{\alpha}(k) \cdot \boldsymbol{x}$ of $\mathit{Probe}_1(j)$. In each round, the leader adds input agents one by one. As the protocol only needs to work for populations with i agents, it is possible for each agent to know if it already interacted with the leader in this round, and for the leader to count the number of agents it has interacted with this round, until it reaches i to start the next round.

7 Conclusion and further work

We have proved that every predicate φ of quantifier-free Presburger arithmetic (QFPA) is computed by a leaderless protocol with poly$(|\varphi|)$ states. Further, the protocol can be computed in polynomial time. The number of states of previous constructions was exponential both in the bit-length of the coefficients of φ, and in the number of occurrences of boolean connectives. Since QFPA and PA have the same expressive power, every computable predicate has a succinct leaderless protocol. This result completes the work initiated in [8], which also constructed succinct protocols, but only for some predicates, and with the help of leaders.

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

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

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

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

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

References

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

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

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

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

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

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

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

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

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

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

11 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, 54(2):191–215, 2017. doi:10.1007/s00236-016-0272-3.

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

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

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

Appendix II: Publications on Verification

C Towards efficient verification of population protocols. PODC, 2017.

This chapter has been published as **peer-reviewed conference paper**.

Michael Blondin, Javier Esparza, Stefan Jaax, and Philipp J. Meyer. Towards efficient verification of population protocols. In *Proceedings of the 2017 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 423–430, 2017. doi:10.1145/3087801.3087816

Synopsis. We consider the two central parametric verification problems for population protocols: the *well-specification problem* (given a protocol $\mathcal{P}$, does $\mathcal{P}$ stabilize to a unique output for every input?) and the *correctness problem* (given a protocol $\mathcal{P}$ and a predicate φ, does $\mathcal{P}$ compute φ?). Esparza *et al.* have shown that both problems are decidable, but of very high complexity: they are least as hard as the Petri net reachability problem, which is TOWER-hard. We identify a fully expressive subclass of population protocols that is suitable for efficient automatic parametric verification, leading to the first automatic procedure for verifying well-specification and correctness of population protocols. We evaluated our procedure on a set of benchmarks from the literature on population protocols.

Contributions of thesis author. Composition and revision of the manuscript. Collection of protocols for benchmarks. Joint discussion and development of the theoretical results and proofs presented in the paper, with the following notable individual contributions: invention of the notion of LayeredTermination and design of the verification method; writing the proofs for the complexity of deciding WS^3, and writing the proofs in Section 5.

Towards Efficient Verification of Population Protocols

Michael Blondin
Technische Universität München
blondin@in.tum.de

Javier Esparza
Technische Universität München
esparza@in.tum.de

Stefan Jaax
Technische Universität München
jaax@in.tum.de

Philipp J. Meyer
Technische Universität München
meyerphi@in.tum.de

ABSTRACT

Population protocols are a well established model of computation by anonymous, identical finite state agents. A protocol is well-specified if from every initial configuration, all fair executions of the protocol reach a common consensus. The central verification question for population protocols is the *well-specification problem*: deciding if a given protocol is well-specified. Esparza et al. have recently shown that this problem is decidable, but with very high complexity: it is at least as hard as the Petri net reachability problem, which is EXPSPACE-hard, and for which only algorithms of non-primitive recursive complexity are currently known.

In this paper we introduce the class WS^3 of well-specified strongly-silent protocols and we prove that it is suitable for automatic verification. More precisely, we show that WS^3 has the same computational power as general well-specified protocols, and captures standard protocols from the literature. Moreover, we show that the membership problem for WS^3 reduces to solving boolean combinations of linear constraints over $\mathbb{N}$. This allowed us to develop the first software able to automatically prove well-specification for *all* of the infinitely many possible inputs.

CCS CONCEPTS

• **Networks** → **Protocol testing and verification**; • **Theory of computation** → **Logic and verification**;

KEYWORDS

population protocols; automated verification; termination

1 INTRODUCTION

Population protocols [1, 2] are a model of distributed computation by many anonymous finite-state agents. They were initially introduced to model networks of passively mobile sensors [1, 2], but are now also used to describe chemical reaction networks (see e.g. [7, 19]).

PODC '17, July 25-27, 2017, Washington, DC, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-4992-5/17/07…$15.00
https://doi.org/http://dx.doi.org/10.1145/3087801.3087816

In each computation step of a population protocol, a fixed number of agents are chosen nondeterministically, and their states are updated according to a joint transition function. Since agents are anonymous and identical, the global state of a protocol is completely determined by the number of agents at each local state, called a configuration. A protocol computes a boolean value b for a given initial configuration C_0 if in all fair executions starting at C_0, all agents eventually agree to b — so, intuitively, population protocols compute by reaching consensus under a certain fairness condition. A protocol is *well-specified* if it computes a value for each of its infinitely many initial configurations (also called *inputs*). The predicate computed by a protocol is the function that assigns to each input the corresponding consensus value. In a famous series of papers, Angluin *et al.* [1, 2] have shown that well-specified protocols compute exactly the predicates definable in Presburger arithmetic [1–4].

In this paper we search for efficient algorithms for the *well-specification problem*: Given a population protocol, is it well-specified? This is a question about an infinite family of finite-state systems. Indeed, for every input the semantics of a protocol is a finite graph with the reachable configurations as nodes. Deciding if the protocol reaches consensus for a fixed input only requires to inspect one of these graphs, and can be done automatically using a model checker. This approach has been followed in a number of papers [6, 8, 20, 24], but it only shows well-specification for some inputs. There has also been work in formalizing well-specification proofs in interactive theorem provers [10], but this approach is not automatic: a human prover must first come up with a proof for each particular protocol.

Recently, the second author, together with other co-authors, has shown that the well-specification problem is decidable [13]. That is, there is an algorithm that decides if for all inputs the protocol stabilizes to a boolean value. The proof uses deep results of the theory of Petri nets, a model very close to population protocols. However, the same paper shows that the well-specification problem is at least as hard as the reachability problem for Petri nets, a famously difficult problem. More precisely, the problem is known to be EXPSPACE-hard, and all known algorithms for it have non-primitive recursive complexity [22]. In particular, there are no stable implementations of any of these algorithms, and they are considered impractical for nearly all applications.

For this reason, in this paper we search for a class of well-specified protocols satisfying three properties:

(a) *No loss of expressive power*: the class should compute all Presburger-definable predicates.

(b) *Natural*: the class should contain most protocols discussed in the literature.

(c) *Feasible membership problem*: membership for the class should have reasonable complexity.

The class *WS* of all well-specified protocols obviously satisfies (a) and (b), but not (c). So we introduce a new class WS^3, standing for *Well-Specified Strongly Silent* protocols. We show that WS^3 still satisfies (a) and (b), and then prove that the membership problem for WS^3 is in the complexity class DP; the class of languages L such that $L = L_1 \cap L_2$ for some languages $L_1 \in$ NP and $L_2 \in$ coNP. This is a dramatic improvement with respect to the EXPSPACE-hardness of the membership problem for *WS*.

Our proof that the problem is in DP reduces membership for WS^3 to checking (un)satisfiability of two systems of boolean combinations of linear constraints over the natural numbers. This allowed us to implement our decision procedure on top of the constraint solver Z3 [9], yielding the first software able to automatically prove well-specification for *all* inputs. We tested our implementation on the families of protocols studied in [6, 8, 20, 24]. These papers prove well-specification for some inputs of protocols with up to 9 states and 28 transitions. Our approach proves well-specification for all inputs of protocols with up to 20 states in less than one second, and protocols with 70 states and 2500 transitions in less than one hour. In particular, we can automatically prove well-specification for *all* inputs in less time than previous tools needed to check *one single large input*.

The verification problem for population protocols naturally divides into two parts: checking that a given protocol is well specified, and checking that a given well-specified protocol computes the desired predicate. While in this paper we are concerned with well-specification, our implementation is already able to solve the second problem for all the families of protocols described above. This is achieved by adding to the second system of constraints used to check well-specification further linear constraints describing the sets of input configurations for which the protocol should return *true* or *false*. An extension of the software that, given a protocol and an arbitrary Presburger predicate, checks whether the protocol computes the predicate, requires to solve implementation problems related to Presburger arithmetic, and is left for future research.

The paper is organized as follows. Section 2 contains basic definitions. Section 3 introduces an intermediate class WS^2 of silent well-specified protocols, and shows that its membership problem is still as hard as for *WS*. In Section 4, we characterize WS^2 in terms of two properties which are then strengthened to define our new class WS^3. We then show that the properties defining WS^3 can be tested in NP and coNP, and so that membership for WS^3 is in DP. Section 5 proves that WS^3-protocols compute all Presburger predicates. Section 6 reports on our experimental results, and Section 7 presents conclusions.

2 PRELIMINARIES

Multisets. A *multiset* over a finite set E is a mapping $M : E \to \mathbb{N}$. 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. $\llparenthesis f, g, g \rrparenthesis$ is the multiset M such that $M(f) = 1$, $M(g) = 2$ and $M(e) = 0$ for every $e \in E \setminus \{f, g\}$. The *support* of $M \in \mathbb{N}^E$ is $\llbracket M \rrbracket \overset{\text{def}}{=} \{e \in E : M(e) > 0\}$. The *size* of $M \in \mathbb{N}^E$ is $|M| \overset{\text{def}}{=} \sum_{e \in E} M(e)$. Addition and comparison are extended to multisets componentwise, i.e. $(M + M')(e) \overset{\text{def}}{=} M(e) + M'(e)$ for every $e \in E$, and $M \leq M' \overset{\text{def}}{\iff} M(e) \leq M(e)$ for every $e \in E$. We define multiset difference as $(M \ominus M')(e) \overset{\text{def}}{=} \max(M(e) - M'(e), 0)$ for every $e \in E$. The empty multiset is denoted $\mathbf{0}$, and for every $e \in E$ we write $\boldsymbol{e} \overset{\text{def}}{=} \llparenthesis e \rrparenthesis$.

Population protocols. A *population* P over a finite set E is a multiset $P \in \mathbb{N}^E$ such that $|P| \geq 2$. The set of all populations over E is denoted by $\text{Pop}(E)$. A *population protocol* is a tuple $\mathcal{P} = (Q, T, \Sigma, I, O)$ where

- Q is a non-empty finite set of *states*,
- $T \subseteq Q^2 \times Q^2$ is a set of *transitions* such that for every $(p, q) \in Q^2$ there exists at least a pair $(p', q') \in Q^2$ such that $(p, q, p', q') \in T$,
- Σ is a non-empty finite *input alphabet*,
- $I : \Sigma \to Q$ is the *input function* mapping input symbols to states,
- $O : Q \to \{0, 1\}$ is the *output function* mapping states to boolean values.

Following the convention of previous papers, we call the populations of $\text{Pop}(Q)$ *configurations*. 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 $q \in Q$, and at least two agents in total.

Pairs of agents[1] interact using transitions. For every $t = (p, q, p', q') \in T$, we write $(p, q) \mapsto (p', q')$ to denote t, and we define $\text{pre}(t) \overset{\text{def}}{=} \llparenthesis p, q \rrparenthesis$ and $\text{post}(t) \overset{\text{def}}{=} \llparenthesis p', q' \rrparenthesis$. For every configuration C and transition $t \in T$, we say that t is *enabled* at C if $C \geq \text{pre}(t)$. Note that by definition of T, every configuration enables at least one transition. A transition $t \in T$ enabled at C can *occur*, leading to the configuration $C \ominus \text{pre}(t) + \text{post}(t)$. Intuitively, a pair of agents in states $\text{pre}(t)$ move to states $\text{post}(t)$. We write $C \xrightarrow{t} C'$ to denote that t is enabled at C and that its occurrence leads to C'. A transition $t \in T$ is *silent* if $\text{pre}(t) = \text{post}(t)$, i.e., if it cannot change the current configuration.

For every sequence of transitions $w = t_1 t_2 \cdots t_k$, we write $C \xrightarrow{w} C'$ if there exists a sequence of configurations $C_0, C_1, \ldots, C_k$ such that $C = C_0 \xrightarrow{t_1} C_1 \cdots \xrightarrow{t_k} C_k = C'$. We also write $C \to C'$ if $C \xrightarrow{t} C'$ for some transition $t \in T$, and call $C \to C'$ a *step*. We write $C \xrightarrow{*} C'$ if $C \xrightarrow{w} C'$ for some $w \in T^*$. We say that C' is *reachable from* C if $C \xrightarrow{*} C'$. An *execution* is an infinite sequence of configurations $C_0 C_1 \cdots$ such that $C_i \to C_{i+1}$ for every $i \in \mathbb{N}$. An execution $C_0 C_1 \cdots$ is *fair* if for every step $C \to C'$, if $C_i = C$ for infinitely many indices $i \in \mathbb{N}$, then $C_j = C$ and $C_{j+1} = C'$ for infinitely many indices $j \in \mathbb{N}$. We say that a configuration C is

- *terminal* if $C \xrightarrow{*} C'$ implies $C = C'$, i.e., if every transition enabled at C is silent;
- a *consensus configuration* if $O(p) = O(q)$ for every $p, q \in \llbracket C \rrbracket$.

[1]While protocols only model interactions between two agents, k-way interactions for a fixed $k > 2$ can be simulated by adding additional states.

For every consensus configuration C, let $O(C)$ denote the unique output of the states in $[\![C]\!]$. An execution $C_0 C_1 \cdots$ *stabilizes* to $b \in \{0,1\}$ if there exists $n \in \mathbb{N}$ such that C_i is a consensus configuration and $O(C_i) = b$ for every $i \geq n$.

Predicates computable by population protocols. Every *input* $X \in \mathrm{Pop}(\Sigma)$ is mapped to the configuration $I(X) \in \mathrm{Pop}(Q)$ defined by

$$I(X)(q) \overset{\text{def}}{=} \sum_{\substack{\sigma \in \Sigma \\ I(\sigma)=q}} X(\sigma) \text{ for every } q \in Q.$$

A configuration C is said to be *initial* if $C = I(X)$ for some input X. A population protocol is *well-specified* if for every input X, there exists $b \in \{0,1\}$ such that every fair execution of $\mathcal{P}$ starting at $I(X)$ stabilizes to b. We say that $\mathcal{P}$ *computes* a predicate φ if for every input X, every fair execution of $\mathcal{P}$ starting at $I(X)$ stabilizes to $\varphi(X)$. It is readily seen that $\mathcal{P}$ computes a predicate if and only if it is well-specified.

Example 2.1. We consider the majority protocol of [3] as a running example. Initially, agents of the protocol can be in either state A or B. The protocol computes whether there are at least as many agents in state B as there are in state A. The states and the input alphabet are $Q = \{A, B, a, b\}$ and $\Sigma = \{A, B\}$ respectively. The input function is the identity function, and the output function is given by $O(B) = O(b) = 1$ and $O(A) = O(a) = 0$. The set of transitions T consists of:

$$\begin{aligned} t_{AB} &= (A, B) \mapsto (a, b) \\ t_{Ab} &= (A, b) \mapsto (A, a) \\ t_{Ba} &= (B, a) \mapsto (B, b) \\ t_{ba} &= (b, a) \mapsto (b, b) \end{aligned}$$

and of silent transitions for the remaining pairs of states. Transition t_{AB} ensures that every fair execution eventually reaches a configuration C such that $C(A) = 0$ or $C(B) = 0$. If $C(A) = 0 = C(B)$, then there were initially equally many agents in A and B. Transition t_{ba} then acts as tie breaker, resulting in a terminal configuration populated only by b. If, say, $C(A) > 0$ and $C(B) = 0$, then there were initially more As than Bs, and t_{Ab} ensures that every fair execution eventually reaches a terminal configuration populated only by A and a.

3 WELL-SPECIFIED SILENT PROTOCOLS

Silent protocols[2] were introduced in [12]. Loosely speaking, a protocol is silent if communication between agents eventually ceases, i.e. if every fair execution eventually stays in the same configuration forever. Observe that a well-specified protocol need not be silent: fair executions may keep alternating from a configuration to another as long as they are consensus configurations with the same output.

More formally, we say that an execution $C_0 C_1 \cdots$ is *silent* if there exists $n \in \mathbb{N}$ and a configuration C such that $C_i = C$ for every $i \geq n$. A population protocol $\mathcal{P}$ is *silent* if every fair execution of $\mathcal{P}$ is silent, regardless of the starting configuration. We call a protocol that is well-specified and silent a WS^2*-protocol*, and denote by WS^2 the set of all WS^2-protocols.

[2]Silent protocols are also referred to as *protocols with stabilizing states* and silent transitions are called *ineffective* in [17, 18].

Example 3.1. As explained in Example 2.1, every fair execution of the majority protocol is silent. This implies that the protocol is silent. If, for example, we add a new state b' where $O(b') = 1$, and transitions $(b, b) \mapsto (b', b')$, $(b', b') \mapsto (b, b)$, then the protocol is no longer silent since the execution where two agents alternate between states b and b' is fair but not silent.

Being silent is a desirable property. While in arbitrary protocols it is difficult to determine if an execution has already stabilized, in silent protocols it is simple: one just checks if the current configuration only enables silent transitions. Even though it is was not observed explicitely, the protocols introduced in [1] to characterize the expressive power of population protocols belong to WS^2. Therefore, WS^2-protocols can compute the same predicates as general ones.

Unfortunately, a slight adaptation of [14, Theorem 10] shows that the complexity of the membership problem for WS^2-protocols is still as high as for the general case:

Proposition 3.2. *The reachability problem for Petri nets is reducible in polynomial time to the membership problem for* WS^2*. In particular, membership for* WS^2 *is* EXPSPACE*-complete.*

To circumvent this high complexity, we will show in the next section how WS^2 can be refined into a smaller class of well-specified protocols with the same expressive power, and a membership problem of much lower complexity.

4 A FINER CLASS OF SILENT WELL-SPECIFIED PROTOCOLS: WS^3

It can be shown that WS^2-protocols are exactly the protocols satisfying the two following properties:

- Termination: for every configuration C, there exists a terminal configuration C' such that $C \xrightarrow{*} C'$.
- Consensus: for every initial configuration C, there exists $b \in \{0,1\}$ such that every terminal configuration C' reachable from C is a consensus configuration with output b, i.e. $C \xrightarrow{*} C'$ implies $O(C') = b$.

We will introduce the new class WS^3 as a refinement of WS^2 obtained by strengthening Termination and Consensus into two new properties called LayeredTermination and StrongConsensus. We introduce these properties in Section 4.1 and Section 4.2, and show that their decision problems belong to NP and coNP respectively.

Before doing so, let us introduce some useful notions. Let $\mathcal{P} = (Q, T, \Sigma, I, O)$ be a population protocol. For every $S \subseteq T$, $\mathcal{P}[S]$ denotes the *protocol induced by* S, i.e. $\mathcal{P}[S] \overset{\text{def}}{=} (Q, S \cup T', \Sigma, I, O)$ where $T' \overset{\text{def}}{=} \{(p, q, p, q) : p, q \in Q\}$ is added to ensure that any two states can interact. Let $\rightarrow_S$ denote the transition relation of $\mathcal{P}[S]$. An *ordered partition* of T is a tuple $(T_1, T_2, \ldots, T_n)$ of nonempty subsets of T such that $T = \bigcup_{i=1}^{n} T_i$ and $T_i \cap T_j = \emptyset$ for every $1 \leq i < j \leq n$.

4.1 Layered termination

We replace Termination by a stronger property called LayeredTermination, and show that deciding LayeredTermination belongs to NP. The definition of LayeredTermination is inspired by the typical structure of protocols found in the literature. Such protocols are organized in layers such that transitions of higher layers cannot be enabled by executing transitions of lower layers. In particular, if the protocol reaches a configuration of the highest layer that does not enable any transition, then this configuration is terminal. For such protocols, Termination can be proven by showing that every (fair or unfair) execution of a layer is silent.

Definition 4.1. A population protocol $\mathcal{P} = (Q, T, \Sigma, I, O)$ satisfies LayeredTermination if there is an ordered partition

$$(T_1, T_2, \dots, T_n)$$

of T such that the following properties hold for every $i \in [n]$:

(a) For every configuration C, every (fair or unfair) execution of $\mathcal{P}[T_i]$ starting at C is silent.
(b) For every configurations C and C', if $C \xrightarrow{*}_{T_i} C'$ and C is terminal in $\mathcal{P}[T_1 \cup T_2 \cup \dots \cup T_{i-1}]$, then C' is also terminal in $\mathcal{P}[T_1 \cup T_2 \cup \dots \cup T_{i-1}]$.

Example 4.2. The majority protocol satisfies LayeredTermination. Indeed, consider the ordered partition (T_1, T_2), where

$$T_1 = \{(A, B) \mapsto (a, b), (A, b) \mapsto (A, a)\}$$
$$T_2 = \{(B, a) \mapsto (B, b), (b, a) \mapsto (b, b)\}.$$

All executions of $\mathcal{P}[T_1]$ and $\mathcal{P}[T_2]$ are silent. For every terminal configuration C of $\mathcal{P}[T_1]$, we have $\llbracket C \rrbracket \subseteq \{A, a\}$ or $\llbracket C \rrbracket \subseteq \{B, a, b\}$. In the former case, no transition of T_2 is enabled; in the latter case, taking transitons of T_2 cannot enable T_1.

As briefly sketched above, LayeredTermination implies Termination. In the rest of this section, we prove that checking LayeredTermination is in NP. We do this by showing that conditions (a) and (b) of Definition 4.1 can be tested in polynomial time.

We recall a basic notion of Petri net theory recast in the terminology of population protocols. For every step $C \xrightarrow{t} C'$ and every state q of a population protocol, we have $C'(q) = C(q) + \mathrm{post}(t)(q) - \mathrm{pre}(t)(q)$. This observation can be extended to sequences of transitions. Let $|w|_t$ denote the number of occurrences of transition t in a sequence w. We have $C'(q) = C(q) + \sum_{t \in T} |w|_t \cdot (\mathrm{post}(t)(q) - \mathrm{pre}(t)(q))$. Thus, a necessary condition for $C \xrightarrow{w} C'$ is the existence of some $\boldsymbol{x} : T \to \mathbb{N}$ such that

$$C'(q) = C(q) + \sum_{t \in T} \boldsymbol{x}(t) \cdot (\mathrm{post}(t)(q) - \mathrm{pre}(t)(q)). \qquad (1)$$

We call (1) the *flow equation* for state q.

Proposition 4.3. *Let $\mathcal{P} = (Q, T, \Sigma, I, O)$ be a population protocol. Deciding whether an ordered partition $(T_1, T_2, \dots, T_n)$ of T satisfies condition (a) of Definition 4.1 can be done in polynomial time.*

Proof. Let $i \in [n]$ and let U_i be the set of non silent transitions of T_i. It can be shown that $\mathcal{P}[T_i]$ is non silent if and only if there exists $\boldsymbol{x} : U_i \to \mathbb{Q}$ such that $\sum_{t \in U_i} \boldsymbol{x}(t) \cdot (\mathrm{post}(t)(q) - \mathrm{pre}(t)(q)) = 0$ and $\boldsymbol{x}(q) \geq 0$ for every $q \in Q$, and $\boldsymbol{x}(q) > 0$ for some $q \in Q$. Therefore, since linear programming is in P, we can check for the (non) existence of an appropriate rational solution $\boldsymbol{x}_i$ for every $i \in [n]$. □

We show how to check condition (b) of Definition 4.1 in polynomial time. Let $U \subseteq T$ be a set of transitions. A configuration $C \in \mathrm{Pop}(Q)$ is U*-dead* if for every $t \in U$, $C \xrightarrow{t} C'$ implies $C' = C$. We say that $\mathcal{P}$ is U*-dead from* $C_0 \in \mathrm{Pop}(Q)$ if every configuration reachable from C_0 is U-dead, i.e. $C_0 \xrightarrow{*} C$ implies that C is U-dead. Finally, we say that $\mathcal{P}$ is U*-dead* if it is U-dead from every U-dead configuration $C_0 \in \mathrm{Pop}(Q)$.

Proposition 4.4. *Let $\mathcal{P} = (Q, T, \Sigma, I, O)$ be a population protocol. Deciding whether an ordered partition $(T_1, \dots, T_n)$ of T satisfies condition (b) of Definition 4.1 can be done in polynomial time.*

Proof. Let $i \in [n]$ and let $U \overset{\text{def}}{=} T_1 \cup T_2 \cup \dots \cup T_{i-1}$. $\mathcal{P}[T_i]$ satisfies condition (b) if and only if $\mathcal{P}[T_i]$ is U-dead. The latter can be tested in polynomial time through the following characterization: $\mathcal{P}[T_i]$ is *not* U-dead if and only if there exist $t \in T_i$ and non silent $u \in U$ such that for every non silent $u' \in U$:

$$\mathrm{pre}(u') \not\leq \mathrm{pre}(t) + (\mathrm{pre}(u) \ominus \mathrm{post}(t)). \qquad \square$$

Propositions 4.3 and 4.4 yield an NP procedure to decide LayeredTermination. Indeed, it suffices to guess an ordered partition and to check whether it satisfies conditions (a) and (b) of Definition 4.1 in polynomial time.

Corollary 4.5. *Deciding if a protocol satisfies LayeredTermination is in* NP.

4.2 Strong consensus

To overcome the high complexity of reachability in population protocols, we strengthen Consensus by replacing the reachability relation in its definition by an *over-approximation*, i.e., a relation $\dashrightarrow$ over configurations such that $C \xrightarrow{*} C'$ implies $C \dashrightarrow C'$. Observe that the flow equations provide an over-approximation of the reachability relation. Indeed, as mentioned earlier, if $C \xrightarrow{*} C'$, then there exists $\boldsymbol{x} : T \to \mathbb{N}$ such that $(C, C', \boldsymbol{x})$ satisfies all of the flow equations. However, this over-approximation alone is too crude for the verification of protocols.

Example 4.6. For example, let us consider the configurations $C = \llparenthesis A, B \rrparenthesis$ and $C' = \llparenthesis a, a \rrparenthesis$ of the majority protocol. The flow equations are satisfied by the mapping $\boldsymbol{x}$ such that $\boldsymbol{x}(t_{AB}) = \boldsymbol{x}(t_{Ab}) = 1$ and $\boldsymbol{x}(t_{Ba}) = \boldsymbol{x}(t_{ba}) = 0$. Yet, $C \xrightarrow{*} C'$ does not hold.

To obtain a finer reachability over-approximation, we introduce so-called traps and siphons constraints borrowed from the theory of Petri nets [11, 15, 16] and successfully applied to a number of analysis problems (see e.g. [5, 15, 16]). Intuitively, for some subset of transitions $U \subseteq T$, a U-trap is a set of states $P \subseteq Q$ such that every transition of U that removes an agent from P also moves an agent into P. Conversely, a U-siphon is a set $P \subseteq Q$ such that every transition of U that moves an agent into P also removes an agent from P. More formally, let ${}^\bullet R \overset{\text{def}}{=} \{t \in T : \llbracket \mathrm{post}(t) \rrbracket \cap R \neq \emptyset\}$ and $R^\bullet \overset{\text{def}}{=} \{t \in T : \llbracket \mathrm{pre}(t) \rrbracket \cap R \neq \emptyset\}$. U-siphons and U-traps are defined as follows:

Definition 4.7. A subset of states $P \subseteq Q$ is a U*-trap* if $P^\bullet \cap U \subseteq {}^\bullet P$, and a U*-siphon* if ${}^\bullet P \cap U \subseteq P^\bullet$.

For every configuration $C \in \mathrm{Pop}(Q)$ and $P \subseteq Q$, let $C(P) \overset{\text{def}}{=} \sum_{q \in P} C(q)$. Consider a sequence of steps $C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} C_n$

where $t_1, \dots, t_n \in U$. It follows from Definition 4.7 that if some transition t_i moves an agent to a U-trap P, then $C_j(P) > 0$ for every $j \geq i$. Similarly, if some transition t_i removes an agent from a U-siphon, then $C_j(P) > 0$ for every $j < i$. In particular:

Observation 4.8. *Let $U \subseteq T$, and let C and C' be configurations such that $C \xrightarrow{*}_U C'$. For every U-trap P, if $C'(P) = 0$, then ${}^\bullet P \cap U = \emptyset$. For every U-siphon P, if $C(S) = 0$, then $P^\bullet \cap U = \emptyset$.*

We obtain a necessary condition for $C \xrightarrow{*}_U C'$ to hold, which we call potential reachability:

Definition 4.9. Let C, C' be two configurations, let $\boldsymbol{x} : T \to \mathbb{N}$, and let $U = [\![\boldsymbol{x}]\!]$. We say that C' is *potentially reachable from C through $\boldsymbol{x}$*, denoted $C \overset{\boldsymbol{x}}{\dashrightarrow} C'$, if

(a) the flow equation (1) holds for every $q \in Q$,
(b) $C'(P) = 0$ implies ${}^\bullet P \cap U = \emptyset$ for every U-trap P, and
(c) $C(P) = 0$ implies $P^\bullet \cap U = \emptyset$ for every U-siphon P.

Example 4.10. Let us reconsider Example 4.6. Let $U = [\![\boldsymbol{x}]\!] = \{t_{AB}, t_{Ab}\}$ and $P = \{A, b\}$. Recall that $t_{AB} = (A, B) \mapsto (a, b)$ and $t_{Ab} = (A, b) \mapsto (A, a)$. We have $P^\bullet \cap U = U$ which implies that P is a U-trap. This means that Definition 4.9(b) is violated as $C'(P) = 0$ and ${}^\bullet P \cap U = U \neq \emptyset$. Therefore, $\langle A, B \rangle \overset{\boldsymbol{x}}{\dashrightarrow} \langle a, a \rangle$ does not hold.

We write $C \dashrightarrow C'$ if $C \overset{\boldsymbol{x}}{\dashrightarrow} C'$ for some $\boldsymbol{x} : T \to \mathbb{N}$. As an immediate consequence of Observation 4.8, for every configurations C and C', if $C \xrightarrow{*} C'$, then $C \dashrightarrow C'$. This allows us to strengthen Consensus by redefining it in terms of potential reachability instead of reachability:

Definition 4.11. A protocol satisfies StrongConsensus if for every initial configuration C, there exists $b \in \{0, 1\}$ such that every terminal configuration C' potentially reachable from C is a consensus configuration with output b, i.e. $C \dashrightarrow C'$ implies $O(C') = b$.

Since the number of U-traps and U-siphons of a protocol can be exponential in the number of states, checking trap and siphon constraints by enumerating them may take exponential time. Fortunately, this can be avoided. By definition, it follows that the union of two U-traps is again a U-trap, and similarly for siphons. Therefore, given a configuration C, there exists a unique maximal U-siphon $P_{\max}$ such that $C(P_{\max}) = 0$, and a unique maximal U-trap $P'_{\max}$ such that $C(P'_{\max}) = 0$. Moreover, $P_{\max}$ and $P'_{\max}$ can be computed in linear time by means of a simple greedy algorithm (see e.g. [11, Ex. 4.5]). This simplifies the task of checking traps and siphons constraints, and yields a coNP procedure for testing StrongConsensus:

Proposition 4.12. *Deciding if a protocol satisfies StrongConsensus is in* coNP.

Proof. Testing whether a protocol *does not* satisfy StrongConsensus can be done by guessing $C_0, C, C' \in \text{Pop}(Q)$, $b \in \{0, 1\}$, $q, q' \in Q$ and $\boldsymbol{x}, \boldsymbol{x}' : T \to \mathbb{N}$, and testing whether

(a) C_0 is initial, C is terminal, C' is terminal, $q \in [\![C]\!]$, $q' \in [\![C']\!]$, $O(q) \neq O(q')$, and
(b) $C_0 \overset{\boldsymbol{x}}{\dashrightarrow} C$ and $C_0 \overset{\boldsymbol{x}'}{\dashrightarrow} C'$.

Since there is no *a priori* bound on the size of C_0, C, C' and $\boldsymbol{x}, \boldsymbol{x}'$, we guess them carefully. First, we guess whether $D(p) = 0$, $D(p) = 1$ or $D(p) \geq 2$ for every $D \in \{C_0, C, C'\}$ and $p \in Q$. This gives enough information to test (a). Then, we guess $[\![\boldsymbol{x}]\!]$ and $[\![\boldsymbol{x}']\!]$. This allows to test traps/siphons constraints as follows. Let $U \overset{\text{def}}{=} [\![\boldsymbol{x}]\!]$, let $P_{\max}$ be the maximal U-trap such that $C(P_{\max}) = 0$, and let $P'_{\max}$ be the maximal U-siphon such that $C_0(P'_{\max}) = 0$. Conditions (b) and (c) of Definition 4.9 hold if and only if ${}^\bullet(P_{\max}) \cap U = \emptyset$ and $(P'_{\max})^\bullet \cap U = \emptyset$, which can be tested in polynomial time. The same is done for $\boldsymbol{x}'$. If (a) and siphons/traps constraints hold, we build the system $\mathcal{S}$ of linear equations/inequalities obtained from the conjunction of the flow equations together with the constraints already guessed. By standard results on integer linear programming (see e.g. [23, Sect. 17]), if $\mathcal{S}$ has a solution, then it has one of polynomial size, and hence we may guess it. □

4.3 WS^3-protocols

We say that a protocol belongs to WS^3 if it satisfies LayeredTermination and StrongConsensus. Since $WS^3 \subseteq WS^2 \subseteq WS$ holds, every WS^3-protocol is well-specified. Recall that a language L belongs to the class DP [21] if there exist languages $L_1 \in$ NP and $L_2 \in$ coNP such that $L = L_1 \cap L_2$. By taking L_1 and L_2 respectively as the languages of population protocols satisfying LayeredTermination and StrongConsensus, Corollary 4.5 and Proposition 4.12 yield:

Theorem 4.13. *The membership problem for WS^3-protocols is in* DP.

5 WS^3 IS AS EXPRESSIVE AS WS

In a famous result, Angluin *et al.* [3] have shown that a predicate is computable by a population protocol if and only if it is definable in Presburger arithmetic, the first-order theory of addition [1, 3]. In particular, [1] constructs protocols for Presburger-definable predicates by means of a well-known result: *Presburger-definable* predicates are the smallest set of predicates containing all threshold and remainder predicates, and closed under boolean operations. A *threshold predicate* is a predicate of the form

$$P(x_1, \dots, x_k) = \left(\sum_{i=1}^{k} a_i x_i < c \right),$$

where $k \geq 1$ and $a_1, \dots, a_k, c \in \mathbb{Z}$. A *remainder predicate* is a predicate of the form

$$P(x_1, \dots, x_k) = \left(\sum_{i=1}^{k} a_i x_i \equiv c \pmod{m} \right),$$

where $k \geq 1$, $m \geq 2$ and $a_1, \dots, a_k, c, m \in \mathbb{Z}$. Here, we show that these predicates can be computed by WS^3-protocols, and that WS^3 is closed under negation and conjunction. As a consequence, we obtain that WS^3-protocols are as expressive as WS, the class of all well-specified protocols.

Threshold. We describe the protocol given in [1] to compute the threshold predicate $\sum_{i=1}^{k} a_i x_i < c$. Let

$$v_{\max} \overset{\text{def}}{=} \max(|a_1|, |a_2|, \dots, |a_k|, |c| + 1)$$

and define

$$f(m,n) \overset{\text{def}}{=} \max(-v_{\max}, \min(v_{\max}, m+n))$$
$$g(m,n) \overset{\text{def}}{=} (m+n) - f(m,n)$$
$$b(m,n) \overset{\text{def}}{=} (f(m,n) < c)$$

The protocol is $\mathcal{P}_{\text{thr}} \overset{\text{def}}{=} (Q, T, \Sigma, I, O)$, where

$$Q \overset{\text{def}}{=} \{0,1\} \times [-v_{\max}, v_{\max}] \times \{0,1\}$$
$$\Sigma \overset{\text{def}}{=} \{x_1, x_2, \ldots, x_k\}$$
$$I(x_i) \overset{\text{def}}{=} (1, a_i, a_i < c) \text{ for every } i \in [k]$$
$$O(\ell, n, o) \overset{\text{def}}{=} o \text{ for every state } (\ell, n, o),$$

and T contains

$$(1,n,o), (l,n',o') \mapsto (1, f(n,n'), b(n,n')), (0, g(n,n'), b(n,n'))$$

for every $n, n' \in [-v_{\max}, v_{\max}]$, $\ell, o, o' \in \{0,1\}$. Intuitively, a state (ℓ, n, o) indicates that the agent has value n, opinion o, and that it is a leader if and only if $\ell = 1$. When a leader q and a state r interact, r becomes a non leader, and q increases its value as much as possible by substracting from the value of r. Moreover, a leader can change the opinion of any non leader.

Proposition 5.1. *$\mathcal{P}_{thr}$ satisfies StrongConsensus.*

Proof. Let $\text{val}(q) \overset{\text{def}}{=} n$ for every state $q = (\ell, n, o) \in Q$, and let $\text{val}(C) \overset{\text{def}}{=} \sum_{q \in Q} C(q) \cdot \text{val}(q)$ for every configuration $C \in \text{Pop}(Q)$. The following holds for every $C, C' \in \text{Pop}(Q)$:

(a) If $(C, C', \boldsymbol{x})$ is a solution to the flow equations for some $\boldsymbol{x} : T \to \mathbb{N}$, then $\text{val}(C) = \text{val}(C')$.
(b) If C, C' are terminal, C and C' contain a leader, and $\text{val}(C) = \text{val}(C')$, then $O(C) = O(C')$.

Suppose for the sake of contradiction that $\mathcal{P}$ does not satisfy StrongConsensus. There exist $C_0, C, C' \in \text{Pop}(Q)$, $q, q' \in Q$ and $\boldsymbol{x}, \boldsymbol{x}' : T \to \mathbb{N}$ such that $C_0 \xrightarrow{\boldsymbol{x}} C$, $C_0 \xrightarrow{\boldsymbol{x}'} C'$, C_0 is initial, C and C' are terminal consensus configurations, $q \in [\![C]\!]$, $q' \in [\![C']\!]$ and $O(q) \neq O(q')$. Note that $(C_0, C, \boldsymbol{x})$ and $(C_0, C', \boldsymbol{x}')$ both satisfy the flow equations. Thus, by (a), $\text{val}(C) = \text{val}(C_0) = \text{val}(C')$. Since C_0 is initial, it contains a leader. Since the set of leaders forms a U-trap for every $U \subseteq T$, and $(C_0, C, \boldsymbol{x})$ and $(C_0, C', \boldsymbol{x})$ satisfy trap constraints, C and C' contain a leader. By (b), C and C' are consensus configurations with $O(C) = O(C')$, which is a contradiction. □

Proposition 5.2. *$\mathcal{P}_{thr}$ satisfies LayeredTermination.*

Proof. Let $L_0 \overset{\text{def}}{=} \{(1,x,0) : c \leq x \leq v_{\max}\}$, $L_1 \overset{\text{def}}{=} \{(1,x,1) : -v_{\max} \leq x < c\}$, $N_0 \overset{\text{def}}{=} \{(0,0,0)\}$ and $N_1 \overset{\text{def}}{=} \{(0,0,1)\}$. It can be shown that the following ordered partitions satisfy layered termination for $c > 0$ and $c \leq 0$ respectively:

$$T_1 \overset{\text{def}}{=} \{t \in T : \text{pre}(t) \neq \wr q, r \wr \text{ for all } q \in L_0, r \in N_1\},$$
$$T_2 \overset{\text{def}}{=} T \setminus T_1, \text{ and}$$

$$S_1 \overset{\text{def}}{=} \{t \in T : \text{pre}(t) \neq \wr q, r \wr \text{ for all } q \in L_1, r \in N_0\},$$
$$S_2 \overset{\text{def}}{=} T \setminus S_1. \qquad \square$$

Remainder. We give a protocol for the remainder predicate

$$\sum_{i=1}^{k} a_i x_i \equiv c \pmod{m}.$$

The protocol is $\mathcal{P}_{\text{rmd}} = (Q, T, \Sigma, I, O)$, where

$$Q \overset{\text{def}}{=} [0, m) \cup \{\text{true}, \text{false}\}$$
$$\Sigma \overset{\text{def}}{=} \{x_1, x_2, \ldots, x_k\}$$
$$I(x_i) \overset{\text{def}}{=} a_i \bmod m \text{ for every } i \in [k]$$
$$O(q) \overset{\text{def}}{=} \begin{cases} 1 \text{ if } q \in \{c, \text{true}\} \\ 0 \text{ otherwise} \end{cases}$$

and where T contains the following transitions for every $n, n' \in [0, m)$ and $b \in \{\text{false}, \text{true}\}$:

$$(n, n') \mapsto (n + n' \bmod m, n + n' \bmod m = c) \quad \text{and}$$
$$(n, b) \mapsto (n, n = c).$$

In the the full version of this paper[3] we show that $\mathcal{P}_{\text{rmd}}$ belongs to WS^3 by adapting the proof for $\mathcal{P}_{\text{thr}}$.

Negation and conjunction. Let $\mathcal{P}_1 = (Q_1, T_1, \Sigma, I_1, O_1)$ and $\mathcal{P}_2 = (Q_2, T_2, \Sigma, I_2, O_2)$ be WS^3-protocols computing predicates φ_1 and φ_2 respectively. We may assume that $\mathcal{P}_1$ and $\mathcal{P}_2$ are defined over identical Σ, for we can always extend the input domain of threshold/remainder predicates by variables with coefficients of value zero. The predicate $\neg\varphi_i$ can be computed by replacing O_i by the new output function O'_i such that $O'_i(q) \overset{\text{def}}{=} \neg O_i(q)$ for every $q \in Q_i$. To compute $\varphi_1 \wedge \varphi_2$, we build an asynchronous product where steps of $\mathcal{P}_1$ and $\mathcal{P}_2$ can be executed independently.

More formally, the *conjunction* of $\mathcal{P}_1$ and $\mathcal{P}_2$ is defined as the population protocol $\mathcal{P} \overset{\text{def}}{=} (Q, S, I, \Sigma, O)$ such that $Q \overset{\text{def}}{=} Q_1 \times Q_2$, $S \overset{\text{def}}{=} S_1 \cup S_2$, $I(\sigma) \overset{\text{def}}{=} (I_1(\sigma), I_2(\sigma))$ and $O(p,q) \overset{\text{def}}{=} O_1(p) \wedge O_2(q)$ where

$$S_1 \overset{\text{def}}{=} \{(p,r), (p',r') \mapsto (q,r), (q',r') : (p,p',q,q') \in T_1, r, r' \in Q_2\},$$
$$S_2 \overset{\text{def}}{=} \{(r,p), (r',p') \mapsto (r,q), (r',q') : (p,p',q,q') \in T_2, r, r' \in Q_1\}.$$

In the full version of this paper. we show that $\mathcal{P}$ is in WS^3 since terminal/consensus configurations, flow equations, and traps and siphons constraints are preserved by projections from $\mathcal{P}$ onto $\mathcal{P}_1$ and $\mathcal{P}_2$.

[3]https://arxiv.org/abs/1703.04367

Threshold			
$v_{\max}$	$\|Q\|$	$\|T\|$	Time
3	28	288	8.0
4	36	478	26.5
5	44	716	97.6
6	52	1002	243.4
7	60	1336	565.0
8	68	1718	1019.7
9	76	2148	2375.9
10	84	2626	timeout

Remainder			
m	$\|Q\|$	$\|T\|$	Time
10	12	65	0.4
20	22	230	2.8
30	32	495	15.9
40	42	860	79.3
50	52	1325	440.3
60	62	1890	3055.4
70	72	2555	3176.5
80	82	3320	timeout

Flock of birds [6]			
c	$\|Q\|$	$\|T\|$	Time
20	21	210	1.5
25	26	325	3.3
30	31	465	7.7
35	36	630	20.8
40	41	820	106.9
45	46	1035	295.6
50	51	1275	181.6
55	56	1540	timeout

Flock of birds [8]			
c	$\|Q\|$	$\|T\|$	Time
50	51	99	11.8
100	101	199	44.8
150	151	299	369.1
200	201	399	778.8
250	251	499	1554.2
300	301	599	2782.5
325	326	649	3470.8
350	351	699	timeout

Majority		
$\|Q\|$	$\|T\|$	Time
4	4	0.1

Broadcast		
$\|Q\|$	$\|T\|$	Time
2	1	0.1

Table 1: Results of the experimental evaluation where $|Q|$ denotes the number of states, $|T|$ denotes the number of non silent transitions, and the time to prove membership for WS^3 is given in seconds.

6 EXPERIMENTAL RESULTS

We have developed a tool called `Peregrine`[4] to check membership in WS^3. Peregrine is implemented on top of the SMT solver Z3 [9].

`Peregrine` reads in a population protocol $\mathcal{P} = (Q, T, \Sigma, I, O)$ and constructs two sets of constraints. The first set is satisfiable if and only if LayeredTermination holds, and the second is unsatisfiable if and only if StrongConsensus holds.

For LayeredTermination, our tool `Peregrine` iteratively constructs constraints checking the existence of an ordered partition of size $1, 2, \ldots, |T|$ and decides if they are satisfiable. To check that the execution of a layer is silent, the constraints mentioned in the proof of Proposition 4.3 are transformed using Farkas' lemma (see e.g. [23]) into a version that is satisfiable if and only if all the executions of the layer are silent. Also, the constraints for condition (b) of Definition 4.1 are added.

For StrongConsensus, `Peregrine` initially constructs the constraints for the flow equation for three configurations C_0, C_1, C_2 and vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, with additional constraints to guarantee that C_0 is initial, C_1 and C_2 are terminal, and C_1 and C_2 are consensus of different values. If these constraints are unsatisfiable, the protocol satisfies StrongConsensus. Otherwise, `Peregrine` searches for a U-trap or U-siphon to show that either $C_0 \overset{\boldsymbol{x}_1}{\dashrightarrow} C_1$ or $C_0 \overset{\boldsymbol{x}_2}{\dashrightarrow} C_2$ does not hold. If, say, a U-siphon S is found, then `Peregrine` adds the constraint $C_0(S) > 0$ to the set of initial constraints. This process is repeated until either the constraints are unsatisfiable and StrongConsensus is shown, or all possible U-traps and U-siphons are added, in which case StrongConsensus does not hold. We use this refinement-based approach instead of the coNP approach described in Proposition 4.12, as that could require a quadratic number of variables and constraints, and we generally expect to need a small number of refinement steps.

We evaluated `Peregrine` on a set of benchmarks: the threshold and remainder protocols of [2], the majority protocol of [3], the broadcast protocol of [8] and two versions of the flock of birds[5] protocol from [6, 8]. We checked the parametrized protocols for increasing values of their primary parameter until we reached a timeout. For the threshold and remainder protocols, we set the secondary parameter c to 1 since it has no incidence on the size of the protocol, and since the variation in execution time for different values of c was negligible. Moreover, we assumed that all possible values for a_i were present in the inputs, which represents the worst case.

All experiments were performed on the same machine equipped with an Intel Core i7-4810MQ CPU and 16 GB of RAM. The time limit was set to 1 hour. The results are shown in Table 1. In all cases where we terminated within the time limit, we were able to show membership for WS^3. Generally, showing StrongConsensus took much less time than showing LayeredTermination, except for the flock of birds protocols, where we needed linearly many U-traps.

As an extension, we also tried proving correctness after proving membership in WS^3. For this, we constructed constraints for the existence of an input X and configuration C with $I(X) \overset{\boldsymbol{x}}{\dashrightarrow} C$ and $\varphi(X) \neq O(C)$. We were able to prove correctness for all the protocols in our set of benchmarks. The correctness check was faster than the well-specification check for broadcast, majority, threshold and both flock of birds protocols, and slower for the remainder protocol, where we reached a timeout for $m = 70$.

7 CONCLUSION AND FURTHER WORK

We have presented WS^3, the first class of well-specified population protocols with a membership problem of reasonable complexity (i.e. in DP) and with the full expressiveness of well-specified protocols. Previous work had shown that the membership problem for the general class of well-specified protocols is decidable, but at least EXPSPACE-hard with algorithms of non primitive recursive complexity.

[4]`Peregrine` and benchmarks are available from https://gitlab.lrz.de/i7/peregrine/.

[5]The variant from [8] is referred to as *threshold-n* by its authors.

We have shown that WS^3 is a natural class that contains many standard protocols from the literature, like flock-of-birds, majority, threshold and remainder protocols. We implemented the membership procedure for WS^3 on top of the SMT solver Z3, yielding the first software able to automatically prove well-specification of population protocols for *all* (of the infinitely many) inputs. Previous work could only prove partial correctness of protocols with at most 9 states and 28 transitions, by trying exhaustively a *finite* number of inputs [6, 8, 20, 24]. Our algorithm deals with all inputs and can handle larger protocols with up to 70 states and over 2500 transitions.

Future work will concentrate on three problems: improving the performance of our tool; automatically deciding if a WS^3-protocol computes the predicate described by a given Presburger formula; and the diagnosis problem: when a protocol does not belong to WS^3, delivering an explanation, e.g. a non-terminating fair execution. We think that our constraint-based approach provides an excellent basis for attacking these questions.

ACKNOWLEDGMENTS

We wish to thank David de Frutos Escrig and Pierre Ganty for pointing out some mistakes in an earlier version of this paper.

REFERENCES

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

[2] 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. https://doi.org/10.1007/s00446-005-0138-3

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

[4] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. 2007. The computational power of population protocols. *Distributed Computing* 20, 4 (2007), 279–304. https://doi.org/10.1007/s00446-007-0040-2

[5] Konstantinos Athanasiou, Peizun Liu, and Thomas Wahl. 2016. Unbounded-Thread Program Verification using Thread-State Equations. In *IJCAR (Lecture Notes in Computer Science)*, Vol. 9706. Springer, 516–531.

[6] Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. 2010. Algorithmic Verification of Population Protocols. In *Proc. 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. 221–235.

[7] Ho-Lin Chen, Rachel Cummings, David Doty, and David Soloveichik. 2014. Speed Faults in Computation by Chemical Reaction Networks. In *DISC (Lecture Notes in Computer Science)*, Vol. 8784. Springer, 16–30.

[8] Julien Clément, Carole Delporte-Gallet, Hugues Fauconnier, and Mihaela Sighireanu. 2011. Guidelines for the Verification of Population Protocols. In *ICDCS*. IEEE Computer Society, 215–224.

[9] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS (Lecture Notes in Computer Science)*, Vol. 4963. Springer, 337–340. Z3 is available at https://github.com/Z3Prover/z3.

[10] Yuxin Deng and Jean-François Monin. 2009. Verifying Self-stabilizing Population Protocols with Coq. In *Proc. 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE)*. 201–208. https://doi.org/10.1109/TASE.2009.9

[11] Jörg Desel and Javier Esparza. 1995. *Free choice Petri nets.* Cambridge University Press.

[12] Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. 1999. Memory requirements for silent stabilization. *Acta Informatica* 36, 6 (1999), 447–462.

[13] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. 2015. Verification of Population Protocols. In *Proc. 26th International Conference on Concurrency Theory (CONCUR)*. 470–482.

[14] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. 2016. Model Checking Population Protocols. In *FSTTCS (LIPIcs)*, Vol. 65. 27:1–27:14.

[15] Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp J. Meyer, and Filip Niksic. 2014. An SMT-Based Approach to Coverability Analysis. In *CAV (Lecture Notes in Computer Science)*, Vol. 8559. Springer, 603–619.

[16] Javier Esparza and Stephan Melzer. 2000. Verification of Safety Properties Using Integer Programming: Beyond the State Equation. *Formal Methods in System Design* 16, 2 (2000), 159–189.

[17] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. 2011. Mediated population protocols. *Theor. Comput. Sci.* 412, 22 (2011), 2434–2450.

[18] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. 2012. Terminating Population Protocols via Some Minimal Global Knowledge Assumptions. In *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium, SSS 2012, Toronto, Canada, October 1-4, 2012. Proceedings*. 77–89.

[19] Saket Navlakha and Ziv Bar-Joseph. 2015. Distributed information processing in biological and computational systems. *Commun. ACM* 58, 1 (2015), 94–102. https://doi.org/10.1145/2678280

[20] Jun Pang, Zhengqin Luo, and Yuxin Deng. 2008. On Automatic Verification of Self-Stabilizing Population Protocols. In *Proc. 2nd IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering (TASE)*. 185–192. https://doi.org/10.1109/TASE.2008.8

[21] Christos H. Papadimitriou. 2007. *Computational complexity.* Academic Internet Publ.

[22] Sylvain Schmitz. 2016. The complexity of reachability in vector addition systems. *SIGLOG News* 3, 1 (2016), 4–21.

[23] Alexander Schrijver. 1986. *Theory of Linear and Integer Programming.* John Wiley & Sons.

[24] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. 2009. PAT: Towards Flexible Verification under Fairness. In *Proc. 21st International Conference on Computer Aided Verification (CAV)*. 709–714. https://doi.org/10.1007/978-3-642-02658-4_59

D Peregrine: A Tool for the Analysis of Population Protocols. CAV, 2018.

This chapter has been published as **peer-reviewed conference paper**.

> Michael Blondin, Javier Esparza, and Stefan Jaax. Peregrine: A tool for the analysis of population protocols. In *Proceedings of the 30th International Conference on Computer Aided Verification (CAV)*, pages 604–611, 2018. doi:10.1007/978-3-319-96145-3_34

Synopsis. We present PEREGRINE, a tool for the design, simulation, and verification of population protocols. PEREGRINE supports the following features:

- Specification of population protocols in a graphical interface,
- Specification of families of population protocols in a text editor,
- Various modes of visualization for executions of protocols,
- Gathering Statistics of properties such as convergence speed,
- Automatic verification of correctness, with limited support for error diagnostics.

Contributions of thesis author. Composition and revision of the manuscript. Design and implementation of the novel parts of the tool PEREGRINE. Creation of software artifact for conference submission.

Peregrine: A Tool for the Analysis of Population Protocols

Michael Blondin, Javier Esparza, and Stefan Jaax(✉)

Technische Universität München, Munich, Germany
{blondimi,esparza,jaax}@in.tum.de

Abstract. We introduce Peregrine, the first tool for the analysis and parameterized verification of population protocols. Population protocols are a model of computation very much studied by the distributed computing community, in which mobile anonymous agents interact stochastically to achieve a common task. Peregrine allows users to design protocols, to simulate them both manually and automatically, to gather statistics of properties such as convergence speed, and to verify correctness automatically. This paper describes the features of Peregrine and their implementation.

Keywords: Population protocols · Distributed computing
Parameterized verification · Simulation

1 Introduction

Population protocols [1,3,4] are a model of distributed computing in which replicated, mobile agents with limited computational power interact stochastically to achieve a common task. They provide a simple and elegant formalism to model, e.g., networks of passively mobile sensors [1,5], trust propagation [13], evolutionary dynamics [14], and chemical systems, under the name chemical reaction networks [12,16,19].

Population protocols are parameterized: the number of agents does not change during the execution of the protocol, but is *a priori* unbounded. A protocol is correct if it behaves correctly for all of its infinitely many initial configurations. For this reason, it is challenging to design correct and efficient protocols.

In this paper we introduce Peregrine[1], the first tool for the parameterized analysis of population protocols. Peregrine is intended for use by researchers in distributed computing and systems biology. It allows the user to specify protocols either through an editor or as simple scripts, and to analyze them via a

M. Blondin was supported by the Fonds de recherche du Québec – Nature et technologies (FRQNT).

[1] Peregrine can be found at https://peregrine.model.in.tum.de.

graphical interface. The analysis features of PEREGRINE include manual step-by-step simulation; automatic sampling; statistics generation of average convergence speed; detection of incorrect executions through simulation; and formal verification of correctness. The first four features are supported for all protocols, while verification is supported for silent protocols, a large subclass of protocols [6]. Verification is performed automatically over *all* of the infinitely many initial configurations using the recent approach of [6] for solving the so-called well-specification problem.

Related Work. The problem of automatically verifying that a population protocol conforms to its specification for *one fixed initial configuration* has been considered in [10,11,17,20]. In [10], *ad hoc* search algorithms are used. In [11,17], the authors show how to model the problem in the probabilistic model checker PRISM, and under certain conditions in SPIN. In [20], the problem is modeled with the PAT toolkit for model checking under fairness assumptions. All these tools increase our confidence in the correctness of a protocol. However, compared to PEREGRINE, they are not visual tools, they do not offer simulation capabilities, and they can only verify the correctness of a protocol for a finite number of initial configurations, with typically a small number of agents. PEREGRINE proves correctness for all of the infinitely many initial configurations, with an arbitrarily large number of agents.

As mentioned in the introduction, population protocols are isomorphic to chemical reaction networks (CRNs), a popular model in natural computing. Cardelli et al. have recently developed model checking techniques and analysis algorithms for *stochastic* CRNs [7–9]. The problems studied therein are incomparable to the parameterized questions addressed by PEREGRINE.

The verification algorithm of PEREGRINE is based on [6], where a novel approach for the parameterized verification of silent population protocols has been presented. The command-line tool of [6] only offers support for proving correctness, with no functionality for visualization or simulation. Further, contrary to PEREGRINE, the tool cannot produce counterexamples when correctness fails.

2 Population Protocols

We introduce population protocols through a simple example and then briefly formalize the model. We refer the reader to [4] for a more thorough but still intuitive presentation. Suppose anonymous and mobile agents wish to take a majority vote. Intuitively, *anonymous* means that agents have no identity, and *mobile* that agents are "wandering around", and can only interact whenever they bump into each other. In order to vote, all agents conduct the following protocol. Each agent is in one out of four states $\{Y, N, y, n\}$. Initially all agents are in the states Y or N, corresponding to how they want to vote (states y, n are auxiliary states). Agents repeatedly interact pairwise according to the following rules:

$$a\colon YN \mapsto yn \qquad b\colon Yn \mapsto Yy \qquad c\colon Ny \mapsto Nn \qquad d\colon yn \mapsto yy$$

For example, if the population initially has two agents of opinion "yes" and one agent of opinion "no", then a possible execution is:

$$\wr \underline{Y}, Y, \underline{N} \wr \xrightarrow{a} \wr y, \underline{Y}, \underline{n} \wr \xrightarrow{b} \wr y, Y, y \wr, \qquad (1)$$

where e.g. $\wr Y, Y, N \wr$ denotes the multiset with two agents in state Y and one agent in state N.

The goal of every population protocol is to ensure that the agents eventually reach a lasting consensus, i.e., a multiset in which (1) either all agents are in "yes"-states, or all agents are in "no"-states, and (2) further interactions do not destroy the consensus. On top of this universal specification, each protocol has an individual goal, determining which initial configurations should reach the "yes" and the "no" lasting consensus. In the majority protocol above, the agents should reach a "yes"-consensus iff 50% or more agents vote "yes".

Execution (1) above leads to a lasting "yes"-consensus; further, the consensus is the right one, since 2 out of 3 agents voted "yes". In fact, assuming agents interact uniformly and independently at random, the above protocol is correct: executions almost surely reach a correct lasting consensus.

More formally, a population protocol is a tuple (Q, T, I, O) where Q is a finite set of *states*, $T \subseteq Q^2 \times Q^2$ is a set of *transitions*, $I \subseteq Q$ are the *initial states* and $O \colon Q \to \{0, 1\}$ is the *output mapping*. A *configuration* is a non-empty multiset over Q, an *initial configuration* is a non-empty multiset over I, and a configuration is *terminal* if it cannot be altered by any transition. A configuration is in a *consensus* if all of its states map to the same output under O.

An *execution* is a finite or infinite sequence $C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \cdots$ such that C_i is obtained from applying transition t_i to C_{i-1}. A *fair execution* is either a finite execution that reaches a terminal configuration, or an infinite execution such that if $\{i \in \mathbb{N} : C_i \xrightarrow{*} D\}$ is infinite, then $\{i \in \mathbb{N} : C_i = D\}$ is infinite for any configuration D. In other words, fairness ensures that a configuration cannot be avoided forever if it is reachable infinitely often. Fairness is an abstraction of the random interactions occurring within a population. A configuration C is in a *lasting consensus* if every execution from C only leads to configurations of the same consensus.

If for every initial configuration C, all fair executions from C lead to a lasting consensus $\varphi(C) \in \{0, 1\}$, then we say that the protocol *computes* the predicate φ. For example, the above majority protocol with $O(Y) = O(y) = 1$ and $O(N) = O(n) = 0$ computes the predicate $C[Y] \geq C[N]$, where $C[x]$ denotes the number of occurrences of state x in C. A protocol does not necessarily compute a predicate. For example, if we alter the majority protocol by removing transition d, then $\wr Y, N \wr \xrightarrow{a} \wr y, n \wr$ is a fair execution, but $\wr y, n \wr$ is not in a consensus. In other words, transition d acts as a tie-breaker which allows to reach the consensus configuration $\wr y, y \wr$. A protocol that computes a predicate is said to be *well-specified.* It is well-known that well-specified population protocols compute precisely the predicates definable in Presburger arithmetic [3]. On top of different *majority protocols* for the predicate $C[x] \geq C[y]$, the literature contains, e.g., different families of so-called *flock-of-birds protocols* for the predicates $C[x] \geq c$, where c is an integer constant, and families of *threshold protocols* for the predicates $a_1 \cdot C[x_1] + \cdots + a_n \cdot C[x_n] \geq c$, where $a_1, \ldots, a_n, c$ are integer constants and $x_1, \ldots, x_n$ are initial states.

3 Analyzing Population Protocols

PEREGRINE is a web tool with a JavaScript frontend and a Haskell backend. The backend makes use of the SMT solver Z3 [15] to test satisfiability of Presburger arithmetic formulas. The user has access to four main features through the graphical frontend. We present these features in the remainder of the section.

Protocol Description. PEREGRINE offers a description language for both single protocols and families of protocols depending on some parameters. Single protocols are described either through a graphical editor or as simple Python scripts. Families of protocols (called parametric protocols) can only be specified as scripts, but PEREGRINE assists the user by generating a code skeleton.

Simulation. Population protocols can be simulated through a graphical player depicted in Fig. 1. The user can pick an initial configuration and simulate the protocol by either manual selection of interactions, or by letting a scheduler pick interactions uniformly at random. The simulator keeps a history of the execution which can be rewound at any time, making it easy to experiment with the different behaviours of a protocol. Configurations can be displayed in two ways: either as explicit populations, as illustrated in Fig. 1, or as bar charts of the states count, more convenient for large populations.

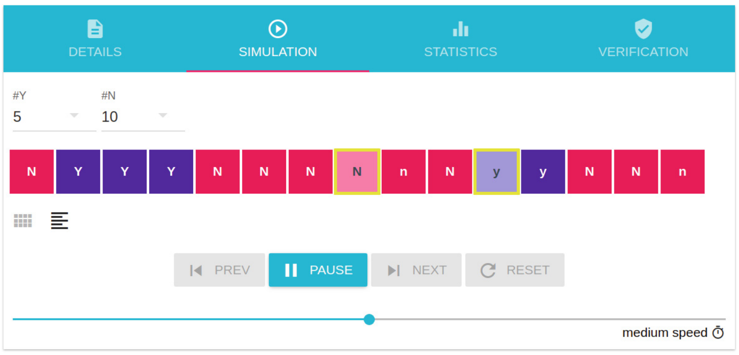

#Y 5
#N 10

Fig. 1. Simulation of the majority protocol from the initial configuration $\wr 5 \cdot Y, 10 \cdot N \wr$.

Statistics. PEREGRINE can generate statistics from batch simulations. The user provides four parameters: $s_{\min}, s_{\max}, m$ and n. PEREGRINE generates n random executions as follows. For each execution, a number s is picked uniformly at random from $[s_{\min}, s_{\max}]$, and an initial configuration of size s is then picked uniformly at random. Each step of an execution is picked uniformly at random among enabled interactions. If no terminal configuration is reached within m steps, then the simulation halts. In the end, n executions of length at most m are gathered. Peregrine classifies the generated executions according to their consensus, and computes statistics on the convergence speed (see the next two paragraphs). The results can be visualized in different ways, and the raw data can be exported as a JSON file.

Consensus. For each random execution, Peregrine checks whether the last configuration of an execution is in a consensus and, if so, whether the consensus corresponds to the expected output of the protocol. Peregrine reports which percentage of the executions reach a consensus, and whether the consensus is correct and/or lasting. In normal mode, Peregrine only classifies an execution as lasting consensus if it ends in a terminal configuration. In the *increased accuracy* mode, if the execution ends in a configuration C of consensus $b \in \{0, 1\}$, then the model checker LoLA [18] is used to determine whether there exists a configuration C' such that $C \xrightarrow{*} C'$ and C' is not of consensus b. If it is not the case, then Peregrine concludes that C is in a lasting consensus. Peregrine plots the percentage of executions in each category as a function of the population size, as illustrated on the left of Fig. 2.

Average Convergence Speed. Peregrine also provides statistics on the convergence speed of a protocol. Let $C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \cdots \xrightarrow{t_\ell} C_\ell$ be an execution such that C_ℓ is in a consensus $b \in \{0, 1\}$. The *number of steps to convergence* of the execution is defined as 0 if all configurations are of consensus b, and otherwise as $i+1$, where i is the largest index such that C_i is not in consensus b. For each population size, Peregrine computes the average number of steps to convergence of all consensus executions of that population size, and plots the information as illustrated on the right of Fig. 2.

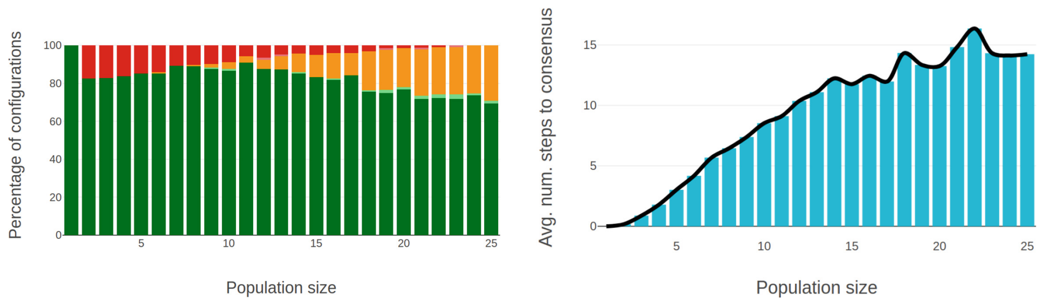
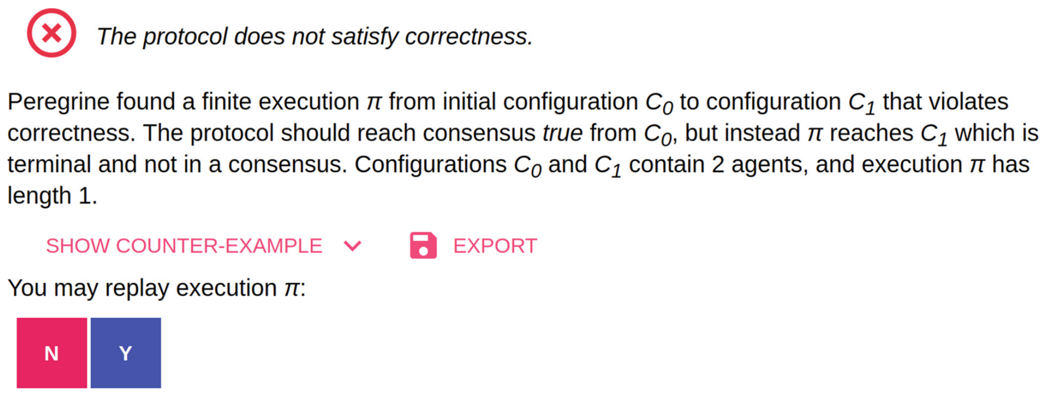


Fig. 2. Statistics for 5000 random executions of the approximate majority protocol of [2], of length at most 40, from initial configurations of size at most 25. The left plot shows the percentage of executions reaching a consensus (dark green: lasting correct, light green: correct, light red: incorrect, dark red: lasting incorrect) and no consensus (orange). In this example the occurrences of light red are negligible. The right plot shows the average number of steps to convergence. (Color figure online)


The protocol does not satisfy correctness.

Peregrine found a finite execution π from initial configuration C_0 to configuration C_1 that violates correctness. The protocol should reach consensus *true* from C_0, but instead π reaches C_1 which is terminal and not in a consensus. Configurations C_0 and C_1 contain 2 agents, and execution π has length 1.

You may replay execution π:

Fig. 3. Verification of the majority protocol of Sect. 2 without transition $d\colon yn \mapsto yy$.

Verification. PEREGRINE can automatically verify that a population protocol computes a given predicate. Predicates can be specified by the user in quantifier-free Presburger arithmetic extended with the family of predicates $\{x \equiv y \pmod{c}\}_{c \geq 2}$, which is equivalent to Presburger arithmetic. For example, for the majority protocol of Sect. 2, the user simply specifies `C[Y] >= C[N]`.

PEREGRINE implements the approach of [6] to verify correctness of protocols which are silent. A protocol is said to be *silent* if from every initial configuration, every fair execution leads to a terminal configuration. The majority protocol of Sect. 2 and most existing protocols from the literature are silent [6]. We briefly describe the approach of [6] and how it is integrated into PEREGRINE.

Suppose we are given a population protocol $\mathcal{P}$ and we wish to determine whether it computes a predicate φ. The procedure first tries to prove that $\mathcal{P}$ is silent. This is done by verifying a more restricted condition called *layered termination*. Verifying the latter property reduces to testing satisfiability of a Presburger arithmetic formula. If this formula holds, then the protocol is silent, otherwise no conclusion is derived. However, essentially all existing silent protocols satisfy layered termination [6].

Once $\mathcal{P}$ is proven to be silent, the procedure attempts to prove that no "bad execution" exists. More precisely, it checks whether there exist configurations C_0 and C_1 such that $C_0 \xrightarrow{*} C_1$, C_0 is initial, C_1 is terminal, and C_1 is not in consensus $\varphi(C_0) \in \{0, 1\}$. Since reachability is not definable in Presburger arithmetic, a Presburger-definable over-approximation $\overset{*}{\rightharpoondown}$ of reachability, borrowed from Petri net theory, is used instead. We obtain the following formula $\Phi_{\text{bad-exec}}$:

$$\exists C_0, C_1 : C_0 \overset{*}{\rightharpoondown} C_1 \wedge \bigwedge_{q \notin I} C_0[q] = 0 \wedge \bigwedge_{t \in T} \text{succ}(C_1, t) \subseteq \{C_1\} \wedge \bigvee_{q \in C_1} (O(q) = \neg\varphi(C_0)).$$

If $\Phi_{\text{bad-exec}}$ is unsatisfiable, then $\mathcal{P}$ is correct. Otherwise, no conclusion is reached, and $\Phi_{\text{bad-exec}}$ is iteratively strengthened by enriching the over-approximation $\overset{*}{\rightharpoondown}$. Whenever $\Phi_{\text{bad-exec}}$ is satisfied by (C_0, C_1), PEREGRINE calls the model-checker LOLA to test whether C_1 is indeed reachable from C_0. If so, then PEREGRINE reports $\mathcal{P}$ to be incorrect, and generates a counter-example execution, which can be replayed or exported as a JSON file (see Fig. 3).

Currently Peregrine can verify protocols with up to a hundred states and a few thousands transitions. The bottleneck is the size of the constraint system. Due to lack of space, we refer the reader to [6] for detailed experimental results.

References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 290–299 (2004). https://doi.org/10.1145/1011767.1011810
2. Angluin, D., Aspnes, J., Eisenstat, D.: A simple population protocol for fast robust approximate majority. Distrib. Comput. **21**(2), 87–102 (2008). https://doi.org/10.1007/s00446-008-0059-z
3. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. **20**(4), 279–304 (2007). https://doi.org/10.1007/s00446-007-0040-2
4. Aspnes, J., Ruppert, E.: An introduction to population protocols. In: Garbinato, B., Miranda, H., Rodrigues, L. (eds.) Middleware for Network Eccentric and Mobile Applications, pp. 97–120. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-89707-1_5
5. Beauquier, J., Blanchard, P., Burman, J., Delaët, S.: Tight complexity analysis of population protocols with cover times - the ZebraNet example. Theor. Comput. Sci. **512**, 15–27 (2013). https://doi.org/10.1016/j.tcs.2012.10.032
6. Blondin, M., Esparza, J., Jaax, S., Meyer, P.J.: Towards efficient verification of population protocols. In: Proceedings of the 36th ACM Symposium on Principles of Distributed Computing (PODC), pp. 423–430 (2017). https://doi.org/10.1145/3087801.3087816
7. Cardelli, L., Češka, M., Fränzle, M., Kwiatkowska, M., Laurenti, L., Paoletti, N., Whitby, M.: Syntax-guided optimal synthesis for chemical reaction networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 375–395. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_20
8. Cardelli, L., Kwiatkowska, M., Laurenti, L.: Stochastic analysis of chemical reaction networks using linear noise approximation. Biosystems **149**, 26–33 (2016). https://doi.org/10.1016/j.biosystems.2016.09.004
9. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Syntactic Markovian bisimulation for chemical reaction networks. In: Aceto, L., et al. (eds.) Models, Algorithms, Logics and Tools. LNCS, vol. 10460, pp. 466–483. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63121-9_23
10. Chatzigiannakis, I., Michail, O., Spirakis, P.G.: Algorithmic verification of population protocols. In: Dolev, S., Cobb, J., Fischer, M., Yung, M. (eds.) SSS 2010. LNCS, vol. 6366, pp. 221–235. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16023-3_19
11. Clément, J., Delporte-Gallet, C., Fauconnier, H., Sighireanu, M.: Guidelines for the verification of population protocols. In: ICDCS, pp. 215–224. IEEE Computer Society (2011). https://doi.org/10.1109/ICDCS.2011.36
12. Cummings, R., Doty, D., Soloveichik, D.: Probability 1 computation with chemical reaction networks. Nat. Comput. **15**(2), 245–261 (2016). https://doi.org/10.1007/s11047-015-9501-x

13. Diamadi, Z., Fischer, M.J.: A simple game for the study of trust in distributed systems. Wuhan Univ. J. Nat. Sci. **6**(1), 72–82 (2001). https://doi.org/10.1007/BF03160228
14. Moran, P.A.P.: Random processes in genetics. Math. Proc. Cambridge Philos. Soc. **54**(1), 60–71 (1958). https://doi.org/10.1017/S0305004100033193
15. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24. z3 is available at https://github.com/Z3Prover/z3
16. Navlakha, S., Bar-Joseph, Z.: Distributed information processing in biological and computational systems. Commun. ACM **58**(1), 94–102 (2014). https://doi.org/10.1145/2678280
17. Pang, J., Luo, Z., Deng, Y.: On automatic verification of self-stabilizing population protocols. In: Proceedings of the 2nd IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering (TASE), pp. 185–192 (2008). https://doi.org/10.1109/TASE.2008.8
18. Schmidt, K.: LoLA a low level analyser. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 465–474. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44988-4_27. LoLA is available at http://service-technology.org/lola/
19. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Nat. Comput. **7**(4), 615–633 (2008). https://doi.org/10.1007/s11047-008-9067-y
20. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: towards flexible verification under fairness. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 709–714. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_59

Appendix III: Publications on Expressiveness of Extensions

E Expressive Power of Broadcast Consensus Protocols. CONCUR, 2019.

This chapter has been published as **peer-reviewed conference paper**.

Michael Blondin, Javier Esparza, and Stefan Jaax. Expressive Power of Broadcast Consensus Protocols. In *Proceedings of the 30th International Conference on Concurrency Theory (CONCUR)*, pages 31:1–31:16, 2019. doi:10.4230/LIPIcs.CONCUR.2019.31

Synopsis. We study the expressive power of *broadcast consensus protocols*, the extension of population protocols by reliable global broadcasts. We show that broadcast consensus protocols compute precisely the predicates in NL, *i.e.* the predicates computable by a nondeterministic Turing machine with logarithmic space, when the input is given in *unary.* Hence broadcast consensus protocols are vastly more expressive than basic population protocols, which are known to compute precisely the semilinear predicates. We further show that broadcast protocols still compute all predicates in NL, when restricted to a single broadcast signal. On the other hand, we prove that adding the power to globally reset configurations to initial does not increase expressiveness relative to population protocols.

Contributions of thesis author. Composition and revision of the manuscript. Joint discussion and development of the results and proofs presented in the paper, with the following notable individual contributions: design of the introductory example of Proposition 3; proposal of the notion of silent semi-computation, and writing the proof of Lemma 7 and Propositions 12-14; vastly simplifying the proof of Lemma 11 compared to a previous version; sole contribution of all proofs in Section 5 for single-signal broadcast protocols and protocols with resets.

Expressive Power of Broadcast Consensus Protocols

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

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

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

Abstract

Population protocols are a formal model of computation by identical, anonymous mobile agents interacting in pairs. Their computational power is rather limited: Angluin et al. have shown that they can only compute the predicates over $\mathbb{N}^k$ expressible in Presburger arithmetic. For this reason, several extensions of the model have been proposed, including the addition of devices called cover-time services, absence detectors, and clocks. All these extensions increase the expressive power to the class of predicates over $\mathbb{N}^k$ lying in the complexity class NL when the input is given in unary. However, these devices are difficult to implement, since they require that an agent atomically receives messages from *all* other agents in a population of unknown size; moreover, the agent must *know* that they have all been received. Inspired by the work of the verification community on Emerson and Namjoshi's broadcast protocols, we show that NL-power is also achieved by extending population protocols with reliable broadcasts, a simpler, standard communication primitive.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Complexity classes; Theory of computation → Automata over infinite objects

Keywords and phrases population protocols, complexity theory, counter machines, distributed computing

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.31

Related Version `https://arxiv.org/abs/1902.01668`

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

Acknowledgements Part of this work was realized while Stefan Jaax was visiting the Université de Sherbrooke. We warmly thank the anonymous reviewers for their helpful comments and suggestions.

1 Introduction

Population protocols are a theoretical model for the study of ad hoc networks of tiny computing devices without any infrastructure [5, 6], intensely investigated in recent years (see e.g. [2, 3, 4, 14]). The model postulates a "soup" of indistinguishable agents that behave identically, and only have a fixed number of bits of memory, i.e., a finite number of local states. Agents repeatedly interact in pairs, changing their states according to a joint transition function. A global fairness condition ensures that every finite sequence of interactions that becomes enabled infinitely often is also executed infinitely often. The purpose of a population protocol is to allow agents to collectively compute some information about their initial configuration, defined as the function that assigns to each local state the number of agents that initially occupy it. For example, assume that initially each agent picks a boolean value by choosing, say, q_0 or q_1 as its initial state. The many *majority protocols* described in the literature allow the agents to eventually reach a stable consensus on the value chosen by a majority of the agents. More formally, let x_0 and x_1 denote the initial numbers of agents in states q_0 and q_1; majority protocols compute the predicate $\varphi\colon \mathbb{N} \times \mathbb{N} \to \{0, 1\}$ given by $\varphi(x_0, x_1) = (x_1 \geq x_0)$. Throughout the paper, we use the term "predicate" as an abbreviation for "function from $\mathbb{N}^k$ to $\{0, 1\}$ for some k".

In a seminal paper, Angluin et al. proved that population protocols compute exactly the predicates expressible in Presburger arithmetic [6, 7]. Thus, for example, agents can decide if they are at least a certain number, if at least $2/3$ of them voted the same way, or, more generally, if the vector $(x_1, x_2, \ldots, x_n)$ representing the number of agents that picked option $1, 2, \ldots, n$ in an election with n choices is a solution of a system of linear inequalities. On the other hand, they cannot decide if they are a square or a prime number, or if the product of the number of votes for options 1 and 2 exceeds the number of votes for option 3. Much work has been devoted to designing more powerful formalisms and analyzing their expressive power. In particular, population protocols have recently been extended with capabilities allowing an agent to obtain global information about the current configuration, which we proceed to describe.

In [22], Michail and Spirakis extend the population protocol model with *absence detectors*, by means of which an agent knows, for every state, whether the state is currently populated or not. Further, they implement absence detectors by a weaker object called a *cover-time service*, which allows an agent to deduce if it has interacted with every other agent in the system. They prove that protocols with cover-time can compute all predicates in $\mathsf{DSPACE}(\log n)$ and can only compute predicates in $\mathsf{NSPACE}(\log n) = \mathsf{NL}$, where n is the number of agents[1].

In [8], Aspnes observes that cover-time services are a kind of internal clock mechanism, and introduces clocked population protocols. Clocked protocols have a clock oracle that signals to one or more agents that the population has reached a bottom strongly connected component of the configuration graph, again an item of global information. Aspnes shows that clocked protocols can compute exactly the predicates in NL.

Absence detectors, cover-time services, and clocked protocols are difficult to implement, since they require that an agent reliably receives information from *all* other agents; moreover, the agent needs to *know* that it has already received messages from all other agents before making a move, which is particularly difficult because agents are assumed to have no identities and to ignore the size of the population. In this paper, we propose a much simpler extension (from an implementation point of view): We allow agents to perform reliable broadcasts, a standard operation in concurrency and distributed computing. We are inspired by the broadcast protocol model introduced by Emerson and Namjoshi in [15] to describe bus-based hardware protocols. The model has been used and further studied in many other contributions, e.g. [16, 18, 12, 24, 9]. In broadcast protocols, agents can perform binary interactions, as in the population protocol model, but, additionally, an agent can also broadcast a signal to all other agents, which are guaranteed to react to it. Broadcast protocols are rather simple to

[1] Observe that, for example, n agents can decide whether n is prime. Indeed, a Turing machine can decide if n is a prime number in $\Theta(\log n)$ space by going through all numbers from 2 to $n-1$, and checking for each of them if they divide n.

implement with current technology on mobile agents moving in a limited area. Broadcasts also appear in biological systems. For example, Uhlendorf et al. describe a system in which a controller adds a sugar or saline solution to a population of yeasts, to which all the yeasts react [27]. An idealized model of the system, which is essentially a broadcast protocol, has been analyzed by Bertrand et al. in [9].

In this paper, we show that population protocols with reliable broadcasts also compute *precisely* the predicates in NL, and are therefore as powerful as absence detectors or clocks. To prove this result, we first define the notion of *silent semi-computation*, a weaker notion than standard computation, and prove that broadcast protocols silently semi-compute all protocols in NL. This result makes crucial use of the ability of broadcast protocols to "restart" the whole population nondeterministically whenever something bad or unexpected is detected. We then prove that silent semi-computability and computability coincide for the class NL.

In a second contribution, we explore in more detail the minimal requirements for achieving NL power. On the one hand, we show that it is enough to allow *a single* agent to broadcast *a single* signal. On the other hand, we prove that the addition of a reset, which causes all agents to return to their initial states, does not increase the power of population protocols.

2 Preliminaries

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

Population protocols. A *population* over a finite set E is a multiset $P \in \mathbb{N}^E$ such that $|P| \geq 2$. The set of all populations over E is denoted by $\mathrm{Pop}(E)$. A *population protocol with leaders* (population protocol for short) is a tuple $\mathcal{P} = (Q, R, \Sigma, L, I, O)$ where:

- Q is a non-empty finite set of *states*,
- $R \subseteq (Q \times Q) \times (Q \times Q)$ is a set of *rendez-vous transitions*,
- Σ is a non-empty finite *input alphabet*,
- $I\colon \Sigma \to Q$ is the *input function* mapping input symbols to states,
- $L \in \mathbb{N}^Q$ is the multiset of *leaders*, and
- $O\colon Q \to \{0, 1\}$ is the *output function* mapping states to boolean values.

Following the standard convention, we call elements of $\mathrm{Pop}(Q)$ *configurations*. 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 $q \in Q$, and at least two agents in total.

We write $(p, q) \mapsto (p', q')$ to denote that $(p, q, p', q') \in R$. The relation $\mathrm{Step}\colon \mathrm{Pop}(Q) \to \mathrm{Pop}(Q)$ is defined by: $(C, C') \in \mathrm{Step}$ iff there exists $(p, q, p', q') \in R$ such that $C \geq \wr p, q \wr$ and $C' = C \ominus \wr p, q \wr + \wr p', q' \wr$. We write $C \to C'$ if $(C, C') \in \mathrm{Step}$, and $C \xrightarrow{*} C'$ if $(C, C') \in \mathrm{Step}^*$, the reflexive and transitive closure of Step. If $C \xrightarrow{*} C'$, then we say that C' is *reachable* from C. An *execution* is an infinite sequence of configurations $C_0 C_1 \cdots$ such that $C_i \to C_{i+1}$ for every $i \in \mathbb{N}$. An execution $C_0 C_1 \cdots$ is *fair* if for every step $C \to C'$ the following holds: if $C_i = C$ for infinitely many indices $i \in \mathbb{N}$, then $C_j = C'$ for infinitely many indices $j \in \mathbb{N}$.

We now explain the roles of the input function I and the multiset L of leaders. The elements of $\mathrm{Pop}(\Sigma)$ are called *inputs*. For every input $X \in \mathrm{Pop}(\Sigma)$, let $I(X) \in \mathrm{Pop}(Q)$ denote the configuration defined by

$$I(X)(q) \overset{\text{def}}{=} \sum_{\{\sigma \in \Sigma : I(\sigma) = q\}} X(\sigma) \qquad \text{for every } q \in Q.$$

A configuration C is *initial* if $C = I(X) + L$ for some input X. Intuitively, the agents of $I(X)$ encode the input, while those of L are a fixed number of agents, traditionally called leaders, that perform the computation together with the agents of $I(X)$.

Predicate computed by a protocol. If $O(p) = O(q)$ for every $p, q \in [\![C]\!]$, then C is a *consensus configuration*, and $O(C)$ denotes the unique output of the states in $[\![C]\!]$. We say that a consensus configuration C is a *b-consensus* if $O(C) = b$. An execution $C_0 C_1 \cdots$ *stabilizes* to $b \in \{0,1\}$ if there exists $n \in \mathbb{N}$ such that C_i is a b-consensus for every $i \geq n$.

A protocol $\mathcal{P}$ over an input alphabet Σ *computes* a predicate $\varphi \colon \mathrm{Pop}(\Sigma) \to \{0,1\}$ if for every input $X \in \mathrm{Pop}(\Sigma)$, every fair execution of $\mathcal{P}$ starting at the initial configuration $I(X) + L$ stabilizes to $\varphi(X)$.

Throughout the paper, we assume $\Sigma = \{A_1, \ldots, A_k\}$ for some $k > 0$. Abusing language, we identify population $M \in \mathrm{Pop}(\Sigma)$ to vector $\boldsymbol{\alpha} = (M(A_1), \ldots, M(A_k))$, and say that $\mathcal{P}$ computes a *predicate* $\varphi \colon \mathbb{N}^k \to \{0,1\}$ *of arity* k. In the rest of the paper, the term "predicate" is used with the meaning "function from $\mathbb{N}^k$ to $\{0,1\}$". It is known that:

▶ **Theorem 1** ([7]). *Population protocols compute exactly the predicates expressible in Presburger arithmetic, i.e. the first-order theory of the natural numbers with addition.*

3 Broadcast consensus protocols

Broadcast protocols were introduced by Emerson and Namjoshi in [15] as a formal model of bus-based hardware protocols, such as those for cache coherency. The model has also been applied to the verification of multithreaded programs [12], and to idealized modeling of control problems for living organisms [27, 9]. Its theory has been further studied in [16, 18, 24].

Agents of broadcast protocols can communicate in pairs, as in population protocols, and, additionally, they can also communicate by means of a reliable broadcast. An agent can broadcast a signal to all other agents, which after receiving the signal move to a new state. Broadcasts are routinely used in wireless ad-hoc and sensor networks (see e.g. [1, 28]), and so they are easy to implement on the same kind of systems targeted by population protocols. They can also model idealized versions of communication in natural computing. For example, in [9] they are used to model "communication" in which an experimenter "broadcasts" a signal to a colony of yeasts by increasing the concentration of a nutrient in a solution.

We introduce broadcast consensus protocols, i.e., broadcast protocols whose goal is to compute a predicate in the computation-by-consensus paradigm.

▶ **Definition 2.** *A broadcast consensus protocol is a tuple $\mathcal{P} = (Q, R, B, \Sigma, L, I, O)$, where all components but B are defined as for population protocols, and B is a set of* broadcast transitions*. A* broadcast transition *is a triple (q, r, f) where $q, r \in Q$ and $f \colon Q \to Q$ is a* transfer function*.*

The relation Step $\subseteq \mathrm{Pop}(Q) \times \mathrm{Pop}(Q)$ *of $\mathcal{P}$ is defined as follows. A pair (C, C') of configurations belongs to* Step *iff*

- *there exists* $(p, q) \mapsto (p', q') \in R$ *such that* $C \geq \wr p, q \wr$ *and* $C' = C \ominus \wr p, q \wr + \wr p', q' \wr$*; or*
- *there exists a transition* $(q, r, f) \in B$ *such that* $C(q) \geq 1$ *and* C' *is the configuration computed from* C *in the following three steps:*

$$C_1 = C \ominus \wr q \wr, \tag{1}$$

$$C_2(q') = \sum_{r' \in f^{-1}(q')} C_1(r') \qquad \textit{for every } q' \in Q, \tag{2}$$

$$C' = C_2 + \wr r \wr. \tag{3}$$

Intuitively, (1)–(3) is interpreted as follows: (1) an agent at state q broadcasts a signal and leaves q, yielding C_1; (2) all other agents receive the signal and move to the states indicated by the function f, yielding C_2; and (3) the broadcasting agent enters state r, yielding C'. Correspondingly, instead of (q, r, f) we use $q \mapsto r;\ f$ as notation for a broadcast transition.

Beyond Presburger arithmetic. As a first illustration of the power of broadcast protocols, we show that their expressive power goes beyond Presburger arithmetic, and so beyond the power of population protocols. We present a broadcast consensus protocol for the predicate φ, defined as $\varphi(x) = 1$ iff $x > 1$ and x is a power of two. For readability, we use the notation $q \mapsto q'; [q_1 \mapsto q'_1, \ldots, q_n \mapsto q'_n]$ for a broadcast transition, where $f(q_i) = q'_i$ and where transfers of the form $q_i \mapsto q_i$ may be omitted.

Let $\mathcal{P} = (Q, R, B, \Sigma, L, I, O)$ be the broadcast consensus protocol where $Q \overset{\text{def}}{=} \{x, \overline{x}, \tilde{x}, 0, 1, \bot\}$, $\Sigma \overset{\text{def}}{=} \{x\}$, $I \overset{\text{def}}{=} x \mapsto x$, $L \overset{\text{def}}{=} \mathbf{0}$, $O(q) = 1 \overset{\text{def}}{\iff} q = 1$, and R and B are defined as follows:

- R contains the rendez-vous transition $s\colon (x, x) \mapsto (\overline{x}, 0)$;
- B contains the broadcast transitions $r\colon \bot \mapsto x; [q \mapsto x : q \in Q]$ and

$$\overline{s}\colon \overline{x} \mapsto x; \begin{bmatrix} x \mapsto \bot \\ \overline{x} \mapsto x \\ 0 \mapsto 1 \end{bmatrix} \quad \overline{t_0}\colon \overline{x} \mapsto \overline{x}; \begin{bmatrix} 1 \mapsto 0 \end{bmatrix} \quad t_0\colon x \mapsto 0; \begin{bmatrix} x \mapsto \bot \\ \overline{x} \mapsto 0 \\ 1 \mapsto \bot \end{bmatrix} \quad t_1\colon x \mapsto 1; \begin{bmatrix} x \mapsto \bot \\ \overline{x} \mapsto \bot \\ 0 \mapsto \bot \end{bmatrix}.$$

Intuitively, $\mathcal{P}$ repeatedly halves the number of agents in state x, and it accepts iff it never obtains an odd remainder. More precisely, the transitions of $\mathcal{P}$ are intended to be fired as follows, where C denotes the current configuration:

```
while $C(x) \neq 1$:
  while $C(x) \geq 2$: fire $s$        /*  split agents equally from $x$ to $\overline{x}$ and $0$ */
  if $C(x) = 0$: fire $\overline{s}$          /* move agents from $\overline{x}$ to $x$ if no remainder */
if $C(\overline{x}) = 0$: fire $t_1$            /*           if no remainder, then accept */
else: fire $\overline{t_0}\,t_0$               /*                     otherwise, reject */
```

It is easy to show that $\mathcal{P}$ produces a (lasting) consensus, and the right one, if transitions are executed as above. However, an arbitrary execution may not follow the above procedure. Firing transition $\overline{t_0}$ when not intended has no incidence on the outcome. Moreover, if another transition is fired when it should not be, then $\overline{s}$, t_0 or t_1 will detect this error by moving an agent to state $\bot$. In this case, by fairness, r eventually resets the agents back to the initial configuration and, again by fairness, transitions are eventually fired as intended.

▶ **Proposition 3.** *The broadcast consensus protocol* $\mathcal{P}$ *described above computes the predicate* φ*, defined as* $\varphi(x) = 1$ *iff* $x > 1$ *and* x *is a power of two.*

Leaderless broadcast protocols. A broadcast protocol $\mathcal{P} = (Q, R, B, \Sigma, L, I, O)$ is *leaderless* if $L = \mathbf{0}$. It can be shown that leaderless broadcast consensus protocols compute the same predicates as the general class. We only sketch the argument. First, a broadcast protocol with leader multiset L can be simulated by a protocol with a single leader. Indeed, the protocol can be designed so that the first task of the leader is to "recruit" the other leaders of L from among the agents. Second, a protocol with one leader can be simulated by a leaderless protocol because, loosely speaking, a broadcast protocol can elect a leader in a single computation step[2]. Indeed, if initially all agents are in a state, say q, then a broadcast $q \mapsto \ell; f$, where $f(q) = q'$, sends exactly one agent to leader state ℓ, and all other agents to state q'. It is simple to construct $\mathcal{P}'$ using this feature, and the details are omitted.

In the rest of the paper, we use protocols with leaders to simplify the constructions, but all results (except Proposition 17) remain valid for leaderless protocols.

4 Broadcast consensus protocols compute exactly NL

In this section, we prove our main theorem: a predicate is computable by a broadcast consensus protocol iff it is in NL. We follow the convention and say that a predicate φ belongs to NL if there is a nondeterministic Turing machine that accepts in $\mathcal{O}(\log n)$-space exactly the tuples $(x_1, x_2, \ldots, x_k) \in \mathbb{N}^k$, encoded in unary, such that $\varphi(x_1, x_2, \ldots, x_k)$ holds.

The proof is divided in two parts. Section 4.1 proves the easier direction: predicates computable by broadcast consensus protocols are in NL. Section 4.2 proves the converse, which is more involved.

4.1 Predicates computable by broadcast consensus protocols are in NL

We prove the result in more generality. We define a generic computational model in which the possible steps between configurations are given by an arbitrary relation preserving the number of agents. Formally, a *generic consensus protocol* is a tuple $\mathcal{P} = (Q, \mathrm{Step}, \Sigma, L, I, O)$ where Q, Σ, L, I, O are defined as for population protocols, and $\mathrm{Step} \subseteq \mathrm{Pop}(Q) \times \mathrm{Pop}(Q)$ is the *step relation* between populations, satisfying $|C| = |C'|$ for every $(C, C') \in \mathrm{Step}$.

Clearly, broadcast consensus protocols are generic consensus protocols. Further, it is easy to see that if Step is the one-step relation of a broadcast protocol, then $\mathrm{Step} \in \mathsf{NL}$. Indeed, $\mathrm{Step} \in \mathsf{NL}$ if there is a nondeterministic Turing machine that given a pair of configurations (C, C') with n agents, uses $\mathcal{O}(\log n)$ space and accepts iff $(C, C') \in \mathrm{Step}$. A quick inspection of the two conditions in the definition of Step (Definition 2) shows that this is the case.

Thus, it suffices to prove that generic consensus protocols satisfying $\mathrm{Step} \in \mathsf{NL}$ can only compute predicates in NL. We sketch the proof, more details can be found in the full version of the paper.

▶ **Proposition 4.** *Let $\mathcal{P} = (Q, \mathrm{Step}, \Sigma, L, I, O)$ be a generic consensus protocol computing a predicate φ. If $\mathrm{Step} \in \mathsf{NL}$, then $\varphi \in \mathsf{NL}$. In particular, predicates computable by broadcast consensus protocols are in NL.*

Proof. We show that there is a nondeterministic Turing machine that decides whether $\varphi(\boldsymbol{x}) = 1$ holds, and uses $\mathcal{O}(\log |\boldsymbol{x}|)$ space. Let $G = (V, E)$ be the graph where V is the set of all configurations of $\mathcal{P}$ of size $|\boldsymbol{x}|$, and $(C, C') \in E$ iff $C \to C'$.

[2] Unlike population protocols, where efficient leader election is non-trivial and much studied; see e.g. [14].

It is easy to see that $\varphi(\boldsymbol{x}) = 1$ iff G contains a configuration C of size $|C| = |I(\boldsymbol{x})| = |\boldsymbol{x}|$ satisfying (1) $C_0 \xrightarrow{*} C$; and (2) every configuration reachable from C, including C itself, is a 1-consensus. Therefore, we can decide $\varphi(\boldsymbol{x}) = 1$ by guessing C, and checking (1) and (2) in $\mathcal{O}(\log |I(\boldsymbol{x})|)$ space. For (1), this follows from the fact that graph reachability is in NL. For (2), we observe that determining whether some configuration reachable from C is not a 1-consensus can be done in NL, and we use the fact that NL = coNL [20]. ◀

▶ Remark 5. Protocols with absence detector [22] are a class of generic consensus protocols, and hence Proposition 4 can be used to give an alternative proof of the fact that these protocols only compute predicates in NL.

4.2 Predicates in NL are computable by broadcast consensus protocols

The proof is involved, and we start by describing its structure. In Section 4.2.1, we show that it suffices to prove that every predicate in NL is *silently semi-computable*. In the rest of the section, we proceed to prove this in three steps. Loosely speaking, we show that:

- predicates computable by nondeterministic Turing machines in $\mathcal{O}(n)$ space can also be computed by counter machines with counters polynomially bounded in n (Section 4.2.2);
- predicates computed by polynomially bounded counter machines can also be computed by n-bounded counter machines, i.e. in which the sum of the values of all counters never exceeds their initial sum (Section 4.2.3);
- predicates computed by n-bounded counter machines can be silently semi-computed by broadcast protocols. (Section 4.2.4).

Finally, Section 4.2.5 puts all parts of the proof together.

4.2.1 Silent semi-computation

Recall that, loosely speaking, a protocol computes φ if it converges to 1 for inputs that satisfy φ, and it converges to 0 for inputs that do not satisfy φ. Additionally, a protocol *silently computes* φ if convergence to $b \in \{0, 1\}$ happens by reaching a *terminal b-consensus*, i.e., a configuration C that is a b-consensus and from which one can only reach C itself. (Intuitively, the protocol eventually becomes "silent" because no agent changes state anymore, and hence communication "stops".) We say that a protocol *silently semi-computes* φ if it reaches a terminal 1-consensus for inputs that satisfy φ, and no terminal configuration for other inputs.

▶ **Definition 6.** *A broadcast consensus protocol $\mathcal{P}$* silently semi-computes *a k-ary predicate φ if for every $\boldsymbol{\alpha} \in \mathbb{N}^k$ the following properties hold:*

1. *if $\varphi(\boldsymbol{\alpha}) = 1$, then every fair execution of $\mathcal{P}$ starting at $I(\boldsymbol{\alpha})$ eventually reaches a terminal 1-consensus configuration;*
2. *if $\varphi(\boldsymbol{\alpha}) = 0$, then no fair execution of $\mathcal{P}$ starting at $I(\boldsymbol{\alpha})$ eventually reaches a terminal configuration.*[3]

We show that if a predicate and its complement are both silently semi-computable by broadcast consensus protocols, say $\mathcal{P}_1$ and $\mathcal{P}_0$, then the predicate is also computable by a broadcast consensus protocol $\mathcal{P}$ which, intuitively, behaves as follows under input $\boldsymbol{\alpha}$. At every moment in time, $\mathcal{P}$ is simulating either $\mathcal{P}_1$ or $\mathcal{P}_0$. Initially, $\mathcal{P}$ simulates $\mathcal{P}_0$. Assume $\mathcal{P}$ is simulating $\mathcal{P}_i$ and the current configuration is C. If C is a terminal configuration of

[3] Since every finite execution can be extended to a fair one, this condition is actually equivalent to "no terminal configuration is reachable from $I(\boldsymbol{\alpha})$".

$\mathcal{P}_i$, then $\mathcal{P}$ terminates too. Otherwise, $\mathcal{P}$ nondeterministically chooses one of three options: continue thesimulation of $\mathcal{P}_i$, "reset" the computation to $I_0(\boldsymbol{\alpha})$, i.e., start simulating $\mathcal{P}_0$, or "reset" the computation to $I_1(\boldsymbol{\alpha})$. Conditions 1 and 2 ensure that exactly one of $\mathcal{P}_0$ and $\mathcal{P}_1$ can reach a terminal configuration, namely $\mathcal{P}_{\varphi(\boldsymbol{\alpha})}$. Fairness ensures that $\mathcal{P}$ will eventually reach a terminal configuration of $\mathcal{P}_{\varphi(\boldsymbol{\alpha})}$, and so, by condition 1, that it will always reach the right consensus. Hence, $\mathcal{P}$ silently computes φ.

The "reset" is implemented by means of a broadcast that sends every agent to its initial state in the configuration $I_j(\boldsymbol{\alpha})$; for this, the states of $\mathcal{P}$ are partitioned into classes, one for each input symbol $x \in X$. Every agent moves only within the states of one of the classes, and so every agent "remembers" its initial state in both $\mathcal{P}_0$ and $\mathcal{P}_1$.

▶ **Lemma 7.** *Let φ be an m-ary predicate, and let $\overline{\varphi}$ be the predicate defined by $\overline{\varphi}(\boldsymbol{\alpha}) \overset{def}{=} 1 - \varphi(\boldsymbol{\alpha})$ for every $\boldsymbol{\alpha} \in \mathbb{N}^m$. Further let $\mathcal{P}_1$ and $\mathcal{P}_0$ be broadcast consensus protocols that silently semi-compute φ and $\overline{\varphi}$, respectively. The following holds: there exists a broadcast consensus protocol $\mathcal{P}$ that silently computes φ.*

Proof. Let $\mathcal{P}_1 = (Q_1, R_1, B_1, \Sigma, I_1, O_1)$ and $\mathcal{P}_0 = (Q_0, R_0, B_0, \Sigma, I_0, O_0)$ be protocols that silently semi-compute φ and $\overline{\varphi}$, respectively. Assume w.l.o.g. that Q_1 and Q_0 are disjoint. We construct a protocol $\mathcal{P} = (Q, R, B, \Sigma, I, O)$ that computes φ.

For the sake of clarity we refrain from giving a fully formal description, but we provide enough details to show that the design idea above can indeed be implemented.

States and mappings. The set of states of $\mathcal{P}$ is defined as:

$$Q \overset{\text{def}}{=} \Sigma \times (Q_1 \cup Q_0 \cup \{\texttt{reset}\})$$

If an agent is in state (x, q), we say that x is its *origin* and that q is its *position.* The initial position of an agent is its initial state in $\mathcal{P}_0$, i.e. $I(x) \overset{\text{def}}{=} (x, I_0(x))$. Transitions will be designed so that agents may update their position, but not their origin. Alternatively, instead of applying a transition, agents can nondeterministically choose to transition from $(x, q) \in X \times (Q_1 \cup Q_0)$ to $(x, \texttt{reset})$. An agent in state $(x, \texttt{reset})$ eventually resets the simulation to either $\mathcal{P}_0$ or $\mathcal{P}_1$.

Simulation transitions. We define transitions that proceed with the simulation of $\mathcal{P}_0$ and $\mathcal{P}_1$ as follows. For every $i \in \{1, 0\}$, every $x, y \in \Sigma$, and every non-silent rendez-vous transition $(q, r) \mapsto (q', r')$ of R_i, we add the following rendez-vous transitions to R:

$$(x, q), (y, r) \mapsto (x, q'), (y, r') \qquad \text{and} \qquad (x, q), (y, r) \mapsto (x, \texttt{reset}), (y, \texttt{reset}).$$

The first transition implements the simulation, while the second transition enables resets when the simulation has not reached a terminal configuration. For every broadcast transition $q \mapsto q'; f$ of B_i and every $x \in \Sigma$, we add the following broadcast transitions to B:

$$(x, q) \mapsto (x, q'); f'$$
$$(x, q) \mapsto (x, \texttt{reset}); f'$$

where f' only acts on Q_i by $f'(y, r) \overset{\text{def}}{=} (y, f(r))$ for every $(y, r) \in \Sigma \times Q_i$. The first transition implements the simulation of a broadcast in the original protocols, while the second transition enables a reset.

Reset transitions. We define transitions that trigger a new simulation of either $\mathcal{P}_0$ or $\mathcal{P}_1$. For every $i \in \{1, 0\}$, let $f_i : Q \to Q$ be the function defined as $f_i(x, q) \overset{\text{def}}{=} (x, I_i(x))$ for every $(x, q) \in Q$. For every $i \in \{1, 0\}$ and every $x \in \Sigma$, we add the following broadcast transition to B: $(x, \texttt{reset}) \mapsto (x, I_i(x)); f_i$. ◀

Using Lemma 7, we may now prove the following:

▶ **Proposition 8.** *If every predicate in* NL *is silently semi-computable by broadcast consensus protocols, then every predicate in* NL *is silently computable (and so computable) by broadcast consensus protocols.*

Proof. Assume every predicate in NL is silently semi-computable by broadcast consensus protocols, and let φ be a predicate in NL. We resort to the powerful result stating that predicates in coNL and NL coincide. This is an immediate corollary of the coNL = NL theorem for languages [20, 26, 23], and the fact that one can check in constant space whether a given word encodes a vector of natural numbers of fixed arity. Thus, both φ and $\overline{\varphi}$ are predicates in NL, and so, by assumption, silently semi-computable by broadcast consensus protocols. By Lemma 7, they are silently computable by broadcast consensus protocols. ◀

4.2.2 Simulation of Turing machines by counter machines

We recall that nondeterministic Turing machines working in $\mathcal{O}(n)$ space can be simulated by counter machines whose counters are polynomially bounded in n, and so that both models compute the same predicates.

Let $X = \{x_1, x_2, \ldots, x_k\}$ and $\mathit{Ins} = \{\texttt{inc}(x), \texttt{dec}(x), \texttt{zro}(x), \texttt{nzr}(x), \texttt{nop} \mid x \in X\}$. A *$k$-counter machine* $\mathcal{M}$ over *counters* X is a tuple $(Q, X, \Delta, m, q_0, q_a, q_r)$, where Q is a finite set of *control states*; $\Delta \subseteq Q \times \mathit{Ins} \times Q$ is the *transition relation*; $m \leq k$ is the *number of input counters*; and q_0, q_a, q_r are the *initial*, *accepting*, and *rejecting states*, respectively.

A configuration of $\mathcal{M}$ is a pair $C = (q, \boldsymbol{v}) \in Q \times \mathbb{N}^k$ consisting of a control state q and counter values $\boldsymbol{v}$. For every $i \in [k]$, we denote the value of counter x_i in C by $C(x_i) \overset{\text{def}}{=} \boldsymbol{v}_i$. The *size* of C is $|C| \overset{\text{def}}{=} \sum_{i=1}^{k} C(x_i)$.

Let $\boldsymbol{e}_i$ be the i-th row of the $k \times k$ identity matrix. Given $\mathit{ins} \in \mathit{Ins}$, we define the relation $\xrightarrow{\mathit{ins}}$ over configurations as follows: $(q, \boldsymbol{v}) \xrightarrow{\mathit{ins}} (q', \boldsymbol{v}')$ iff $(q, \mathit{ins}, q') \in \Delta$ and one of the following holds: $\mathit{ins} = \texttt{inc}(x_i)$ and $\boldsymbol{v}' = \boldsymbol{v} + \boldsymbol{e}_i$; $\mathit{ins} = \texttt{dec}(x_i)$, $\boldsymbol{v}_i > 0$, and $\boldsymbol{v}' = \boldsymbol{v} - \boldsymbol{e}_i$; $\mathit{ins} = \texttt{zro}(x_i)$, $\boldsymbol{v}_i = 0$, and $\boldsymbol{v}' = \boldsymbol{v}$; $\mathit{ins} = \texttt{nzr}(x_i)$, $\boldsymbol{v}_i > 0$, and $\boldsymbol{v}' = \boldsymbol{v}$; $\mathit{ins} = \texttt{nop}$ and $\boldsymbol{v}' = \boldsymbol{v}$.

For every $\boldsymbol{\alpha} \in \mathbb{N}^m$, the initial configuration of $\mathcal{M}$ with input $\boldsymbol{\alpha}$ is defined as:

$$C_{\boldsymbol{\alpha}} \overset{\text{def}}{=} (q_0, (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_m, \underbrace{0, \ldots, 0}_{k-m \text{ times}})).$$

We say $\mathcal{M}$ *accepts* $\boldsymbol{\alpha}$ if there exist counter values $\boldsymbol{v} \in \mathbb{N}^k$ satisfying $C_{\boldsymbol{\alpha}} \xrightarrow{*} (q_a, \boldsymbol{v})$. We say $\mathcal{M}$ *rejects* $\boldsymbol{\alpha}$ if M does not accept $\boldsymbol{\alpha}$ and for all configurations C' with $C_{\boldsymbol{\alpha}} \xrightarrow{*} C'$, there exists $\boldsymbol{v} \in \mathbb{N}^k$ satisfying $C' \xrightarrow{*} (q_r, \boldsymbol{v})$. We say $\mathcal{M}$ *computes* a predicate $\varphi : \mathbb{N}^m \to \{0, 1\}$ if $\mathcal{M}$ accepts all inputs $\boldsymbol{\alpha}$ such that $\varphi(\boldsymbol{\alpha}) = 1$, and rejects all $\boldsymbol{\alpha}$ such that $\varphi(\boldsymbol{\alpha}) = 0$.

A counter machine $\mathcal{M}$ is *$f(n)$-bounded* if $|C| \leq f(|C_{\boldsymbol{\alpha}}|)$ holds for every initial configuration $C_{\boldsymbol{\alpha}}$ and every configuration C reachable from $C_{\boldsymbol{\alpha}}$. It is well-known that counter machines can simulate Turing machines:

▶ **Theorem 9** ([19, Theorem 3.1])**.** *A predicate is computable by an $s(n)$-space-bounded Turing machine iff it is computable by a $2^{s(n)}$-bounded counter machine.*

In [19], a weaker version of Theorem 9 is proven that applies to deterministic Turing and counter machines only. However, the proof can be easily adapted to the nondeterministic setting we consider here.

▶ **Corollary 10.** *A predicate is in NL iff it is computable by a polynomially bounded counter machine.*

4.2.3 Simulation of polynomially bounded counter machines by n-bounded counter machines

▶ **Lemma 11.** *For every polynomially bounded counter machine that computes some predicate φ, there exists an n-bounded counter machine that computes φ.*

Proof. We sketch the main idea of the proof; details can be found in the full version of the paper. Let $c \in \mathbb{N}_{>0}$ and let $\mathcal{M}$ be an n^c-bounded counter machine with k counters. To simulate $\mathcal{M}$ by an n-bounded counter machine $\overline{\mathcal{M}}$, we need some way to represent any value $\ell \in [0, n^c]$ by means of counters with values in $[0, n]$. We encode such a value ℓ by its base $n+1$ representation over c counters. Zero-tests are performed by zero-testing all c counters sequentially. Nonzero-tests are implemented similarly with parallel tests. Incrementation and decrementation are implemented with gadgets to (a) assign 0 to a counter; (b) assign n to a counter; (c) test whether a counter value equals n.

This construction is only *weakly* n-bounded, in the sense that all counters are indeed bounded by n, but the overall sum can reach $k \cdot n$. To circumvent this issue, we simulate $\overline{\mathcal{M}}$ by another counter machine $\mathcal{M}'$ whose counters symbolically hold values from multiple counters of $\overline{\mathcal{M}}$. In more details, the counters are defined as $\{y_S : S \subseteq \overline{X}\}$. Intuitively, if counter y_S has value a, then it contributes by a to the value of each counter of S. For example, if $\overline{X} = \{x_1, x_2, x_3\}$ and the input size is $n = 6$, then counter values $(x_1, x_2, x_3) = (6, 1, 4)$ of $\overline{\mathcal{M}}$ can be represented in $\mathcal{M}'$ as $y_{\{x_1,x_2,x_3\}} = 1$, $y_{\{x_1,x_3\}} = 3$, $y_{\{x_1\}} = 2$, and $y_S = 0$ for every other S. Under such a representation, the sum of all counters equals n. Moreover, all instructions can be implemented quite easily. ◀

4.2.4 Simulation of n-bounded counter machines by broadcast consensus protocols

Let $\mathcal{M} = (Q, X, \Delta, m, q_0, q_a, q_r)$ be an n-bounded counter machine that computes some predicate $\varphi \colon \mathbb{N}^m \to \{0, 1\}$. We construct a broadcast protocol $\mathcal{P} = (Q', R, B, \Sigma, L, I, O)$ that silently semi-computes φ.

States and mappings. Let $X' \overset{\text{def}}{=} X \cup \{\mathit{idle}, \mathit{err}\}$. The states of $\mathcal{P}$ are defined as

$$Q' \overset{\text{def}}{=} \underbrace{Q \times \{0,1\}}_{\text{leader states}} \cup \underbrace{X' \times X \times \{0,1\}}_{\text{nonleader states}}.$$

The protocol will be designed in such a way that there is always exactly one agent, called the *leader*, in states $Q \times \{0, 1\}$. Whenever the leader is in state (q, b), we say that its *position* is q, and its *opinion* is b. Every other agent will remain in a state from $X' \times X \times \{0, 1\}$. Whenever a nonleader agent is in state (x, y, b), we say that its *position* is x, its *origin* is y, and its *opinion* is b. Intuitively, the leader is in charge of storing the control state of $\mathcal{M}$, and the nonleaders are in charge of storing the counter values of $\mathcal{M}$.

The protocol has a single leader whose initial position is the initial control state of $\mathcal{M}$, i.e. $L \overset{\text{def}}{=} ⦃(q_0, 0)⦄$. Moreover, every nonleader agent initially has its origin set to its initial position, which will remain unchanged by definition of the forthcoming transition relation: $I(x) \overset{\text{def}}{=} (x, x, 0)$ for every $x \in X$. The output of each agent is its opinion:

$$\begin{aligned} O(q, b) &\overset{\text{def}}{=} b \\ O(x, y, b) &\overset{\text{def}}{=} b \end{aligned} \quad \text{for every } q \in Q, x \in X', y \in X, b \in \{0, 1\}.$$

We now describe how $\mathcal{P}$ simulates the instructions of $\mathcal{M}$.

Decrementation/incrementation. For every transition $q \xrightarrow{\texttt{dec}(x)} r \in \Delta$, every $y \in X$ and every $b, b' \in \{0, 1\}$, we add to R the rendez-vous transition:

$$(q, b), (x, y, b') \mapsto (r, b), (\mathit{idle}, y, b').$$

These transitions change the position of one agent from x to *idle*, and thus decrement the number of agents in position x.

Similarly, for every transition $q \xrightarrow{\texttt{inc}(x)} r$, every $y \in X$ and every $b, b' \in \{0, 1\}$, we add to R the rendez-vous transition:

$$(q, b), (\mathit{idle}, y, b') \mapsto (r, b), (x, y, b').$$

These transitions change the position of an idle agent to x, and thus increment the number of agents in position x. If no agent is in position *err*, then at least one idle agent is available when a counter needs to be incremented, since $\mathcal{M}$ is n-bounded.

Nonzero-tests. For every $q \xrightarrow{\texttt{nzr}(x)} r \in \Delta$, every $y \in X$ and every $b, b' \in \{0, 1\}$, we add to R the rendez-vous transition:

$$(q, b), (x, y, b') \mapsto (r, b), (x, y, b').$$

These transitions can only be executed if there is at least one agent in position x, and thus only if the value of x is nonzero.

Zero-tests. For a given $x \in X$, let $f^x_{err} \colon Q' \to Q'$ be the function that maps every nonleader in position x to the error position, i.e. $f^x_{err}(x, y, b) \overset{\text{def}}{=} (\mathit{err}, y, b)$ for every $y \in X, b \in \{0, 1\}$, and f^x is the identity for all other states.

For every transition $q \xrightarrow{\texttt{zro}(x)} r \in \Delta$ and every $b \in \{0, 1\}$, we add to B the broadcast transition $(q, b) \mapsto (r, b); f^x_{err}$. If such a transition occurs, then nonleaders in position x move to *err*. Thus, an error is detected iff the value of x is nonzero.

To recover from errors, $\mathcal{P}$ can be reset to its initial configuration as follows. Let $f_{rst} \colon Q' \to Q'$ be the function that sends every state back to its origin, i.e.

$$\begin{aligned} f_{rst}(q, b) &\overset{\text{def}}{=} (q_0, 0) && \text{for every } q \in Q, b \in \{0, 1\}, \\ f_{rst}(x, y, b) &\overset{\text{def}}{=} (y, y, 0) && \text{for every } x \in X', y \in X, b \in \{0, 1\}. \end{aligned}$$

For every $y \in X$ and every $b \in \{0, 1\}$, we add the following broadcast transition to B to reset $\mathcal{P}$ to its initial configuration:

$$(\mathit{err}, y, b) \mapsto (y, y, 0); f_{rst}.$$

Acceptance. For every $q \in Q \setminus \{q_a\}$ and $b \in \{0,1\}$, we add to B the broadcast transition $(q, b) \mapsto (q_0, 0); f_{rst}$. Intuitively, as long as the leader's position differs from the accepting control state q_a, it can reset $\mathcal{P}$ to its initial configuration. This ensures that $\mathcal{P}$ can try *all* computations.

Let $f_{err}\colon Q' \to Q'$ be the function that changes the opinion of each state to 1, i.e.

$$f_{err}(q, b) \overset{\text{def}}{=} (q, 1) \qquad \text{for every } q \in Q, b \in \{0,1\},$$
$$f_{err}(x, y, b) \overset{\text{def}}{=} (x, y, 1) \qquad \text{for every } x \in X', y \in X, b \in \{0,1\}.$$

For every $b \in \{0,1\}$, we add the following transition to B:

$$t_{one,b}\colon (q_a, b) \to (q_a, 1); f_{one}.$$

Intuitively, these transitions change the opinion of every agent to 1. If such a transition occurs in a configuration with no agent in *err*, then no agent can change its state anymore, and the stable consensus 1 has been reached.

Correctness. Let us fix some some input $\boldsymbol{\alpha} \in \mathbb{N}^m$. Let C_0 and D_0 be respectively the initial configurations of $\mathcal{M}$ and $\mathcal{P}$ on input $\boldsymbol{\alpha}$. Abusing notation, for every $D \in \mathrm{Pop}(Q')$, let

$$D(x) \overset{\text{def}}{=} \sum_{(x,y,b) \in Q'} D(x, y, b).$$

The two following propositions state that every execution of $\mathcal{M}$ has a corresponding execution in $\mathcal{P}$ and vice versa. The proofs are routine.

▶ **Proposition 12.** *Let C be a configuration of $\mathcal{M}$ such that C is in control state q and $C_0 \xrightarrow{*} C$. There exists a configuration $D \in \mathrm{Pop}(Q')$ such that (i) $D_0 \xrightarrow{*} D$; (ii) $D(x) = C(x)$ for every $x \in X$; (iii) $D(err) = 0$; and (iv) $D(q, b) = 1$ for some $b \in \{0,1\}$.*

▶ **Proposition 13.** *Let $D \in \mathrm{Pop}(Q')$ be such that $D_0 \xrightarrow{*} D$. If $D(err) = 0$, then there is a configuration C of $\mathcal{M}$ such that (i) $C_0 \xrightarrow{*} C$; (ii) $C(x) = D(x)$ for every $x \in X$; and (iii) if $D(q, b) = 1$ for some $(q, b) \in Q'$, then C is in control state q.*

We may now prove that $\mathcal{P}$ silently semi-computes φ.

▶ **Proposition 14.** *For every n-bounded counter machine $\mathcal{M}$ that computes some predicate φ, there exists a broadcast consensus protocol that silently semi-computes φ.*

Proof. We show that $\mathcal{P}$ silently semi-computes φ by proving the two properties of Definition 6. Let $\boldsymbol{\alpha}$ be an input.

1. Assume $\varphi(\boldsymbol{\alpha}) = 1$. Then $\mathcal{M}$ accepts $\boldsymbol{\alpha}$, and so there is a configuration C such that $C_0 \xrightarrow{*} C$ and C is in control state q_a. By Proposition 12, there exists some configuration $D \in \mathrm{Pop}(Q')$ satisfying $D_0 \xrightarrow{*} D$, $D(err) = 0$ and $D(q_a, b) = 1$. Since $\mathcal{M}$ halts when reaching q_a, the only transition enabled at D is $t_{one,b}$, and its application yields a terminal configuration D' of consensus 1. Further, every configuration reachable from D_0, where the leader is not in position q_a or where some nonleader is in position *err*, can be set back to D_0 via some reset transition. Therefore, every fair execution of $\mathcal{P}$ starting at $I(\boldsymbol{\alpha}) = C_0$ will eventually reach D'.
2. Assume $\varphi(\boldsymbol{\alpha}) = 0$. We prove by contradiction that no configuration D reachable from D_0 is terminal. Assume the contrary. We must have $D(q_a, 1) = 1$, $D(err) = 0$ and $O(D) = 1$, for otherwise some broadcast transition with f_{rst} or f_{one} would be enabled. From this and by Proposition 13, there exists some configuration C of $\mathcal{M}$ in control state q_a and satisfying $C_0 \xrightarrow{*} C$. Thus, $\mathcal{M}$ accepts $\boldsymbol{\alpha}$, contradicting $\varphi(\alpha) = 0$. ◀

4.2.5 Main theorem

We prove our main result, namely that broadcast consensus protocols precisely compute the predicates in NL.

▶ **Theorem 15.** *Broadcast consensus protocols compute exactly the predicates in* NL*.*

Proof. Proposition 4 shows that every predicate computable by broadcast consensus protocols is in NL. For the other direction, let φ be a predicate in NL. Since $\mathsf{NL} = \mathsf{coNL}$ by Immerman-Stelepcsényi's theorem, the complement predicate $\overline{\varphi}$ is also in NL. Thus, φ and $\overline{\varphi}$ are computable by $\mathcal{O}(\log n)$-space-bounded nondeterministic Turing machines. By Theorem 9 and Proposition 11, φ and $\overline{\varphi}$ are computable by polynomially bounded counter machines, and thus by n-bounded counter machines. Therefore, by Proposition 14, φ and $\overline{\varphi}$ are silently semi-computable by broadcast consensus protocols. By Proposition 8, this implies that φ is silently computable by a broadcast consensus protocol. ◀

Actually, the proof shows this slightly stronger result:

▶ **Corollary 16.** *A predicate is computable by a broadcast consensus protocol iff it is silently computable by a broadcast consensus protocol. In particular, broadcast consensus protocols silently compute all predicates in* NL*.*

5 Subclasses of broadcast consensus protocols

While broadcasting is a natural, well understood, and much used communication mechanism, it also consumes far more energy than rendez-vous communication. In particular, agents able to broadcast are more expensive to implement. In this section, we briefly analyze which restrictions can be imposed on the broadcast model without reducing its computational power. We show that all predicates in NL can be computed by protocols satisfying two properties:

1. only one agent broadcasts; all other agents only use rendez-vous communication.
2. the broadcasting agent only needs to send one signal, meaning that the receivers' response is independent of the broadcast signal.

Finally, we show that a third restriction *does* decrease the computational power. In simulations of the previous section, broadcasts are often used to "reset" the system. Since computational models with resets have been devoted quite some attention [21, 25, 13, 11], we investigate the computational power of protocols with resets.

Protocols with only one broadcasting agent. Loosely speaking, a broadcast protocol with one broadcasting agent is a broadcast protocol $\mathcal{P} = (Q, R, B, \Sigma, L, I, O)$ with a set Q_ℓ of leader states such that $L = \wr q \wr$ for some $q \in Q_\ell$ (i.e., there is exactly one leader), and whose transitions ensure that the leader always remains within Q_ℓ, that no other agent enters Q_ℓ, and that only agents in Q_ℓ can trigger broadcast transitions. Protocols with multiple broadcasting agents can be simulated by protocols with one broadcasting agent, say b. Instead of directly broadcasting, an agent communicates with b by rendez-vous, and delegates to b the task of executing the broadcast. More precisely, a broadcast transition $q \mapsto q'; f$ is simulated by a rendez-vous transition $(q, q_\ell) \mapsto (q_{aux}, q_{\ell,f})$, followed by a broadcast transition $q_{\ell,f} \mapsto q_\ell; (f \cup \{q_{aux} \mapsto q'\})$.

Single-signal broadcast protocols. In single-signal protocols the receivers' response is independent of the broadcast signal. Formally, a broadcast protocol (Q, R, B, Σ, I, O) is a *single-signal protocol* if there exists a function $f\colon Q \to Q$ such that $B \subseteq Q^2 \times \{f\}$.

▶ **Proposition 17.** *Predicates computable by broadcast consensus protocols are also computable by single-signal broadcast protocols.*

Proof. We give a proof sketch; details can be found in the full version of the paper. We simulate a broadcast protocol $\mathcal{P}$ by a single-signal protocol $\mathcal{P}'$. The main point is to simulate a broadcast step $C_1 \xrightarrow{q_1 \mapsto q_2; g} C_2$ of $\mathcal{P}$ by a sequence of steps of $\mathcal{P}'$.

In $\mathcal{P}$, an agent at state q_1, say a, moves to q_2, and broadcasts the signal with meaning "react according to g". Intuitively, in $\mathcal{P}'$, agent a broadcasts the unique signal of $\mathcal{P}'$, which has the meaning "freeze". An agent that receives the signal, say b, becomes "frozen". Frozen agents can only be "awoken" by a rendez-vous with a. When the rendez-vous happens, a tells b which state it has to move to according to g.

The problem with this procedure is that a has no way to know if it has already performed a rendez-vous with all frozen agents. Thus, frozen agents can spontaneously move to a state *err* indicating "I am tired of waiting". If an agent is in this state, then eventually all agents go back to their initial states, reinitializing the computation. This is achieved by letting agents in state *err* move to their initial states while broadcasting the "freeze" signal. ◀

Protocols with reset. In protocols with reset, all broadcasts transitions reset the protocol to its initial configuration. Formally, a *population protocol with reset* is a broadcast protocol $\mathcal{P} = (Q, R, B, \Sigma, I, O)$ such that for every finite execution $C_0 C_1 \cdots C_k$ from an initial configuration C_0, the following holds: $C_k \xrightarrow{b} C'$ implies $C' = C_0$ for every $b \in B$ and every $C' \in \mathrm{Pop}(Q)$.

▶ **Proposition 18.** *Every predicate computable by a population protocol with reset is Presburger-definable, and thus computable by a standard population protocol.*

Proof. We give a proof sketch; details can be found in the full version of the paper. Let $\mathcal{P} = (Q, R, B, \Sigma, I, O)$ be a population protocol with reset that computes some predicate. We show that the set of accepting initial configurations of $\mathcal{P}$, denoted I_1, is Presburger-definable as follows. Let:

- $\mathcal{P}'$ be the population protocol obtained from $\mathcal{P}$ by eliminating the resets;
- $\mathcal{N}$ be the set of configurations C of $\mathcal{P}'$ from which no reset can occur, i.e., no configuration reachable from C enables a reset of $\mathcal{P}$;
- S_1 be the set of configurations C of $\mathcal{P}'$ that are stable 1-consensuses, i.e., $O(C') = 1$ for every C' reachable from C;
- $\mathcal{B}$ be the set of configurations C of $\mathcal{P}'$ that belong to a bottom strongly connected component of the configuration graph, i.e., C can reach C' iff C' can reach C.

We show that an initial configuration C belongs to I_1 iff it belongs to S_1 or it can reach a configuration from $S_1 \cap \mathcal{B} \cap \mathcal{N}$. Using results from [17], showing in particular that $\mathcal{B}$ is Presburger-definable, we show that I_1 is Presburger-definable. ◀

6 Conclusion

We have studied the expressive power of broadcast consensus protocols: an extension of population protocols with reliable broadcasts, a standard communication primitive in concurrency and distributed computing. We have shown that, despite their simplicity, they precisely compute predicates from the complexity class NL, and are thus as expressive as several other proposals from the literature which require a primitive more difficult to implement: *receiving* messages from all agents, instead of sending messages to all agents.

As future work, we wish to study properties beyond expressiveness, such as state complexity and space vs. speed trade-offs. It would also be interesting to tackle the formal verification of broadcast consensus protocols. Although this is challenging as it goes beyond Presburger arithmetic and the decidability frontier, it has recently been shown that models with broadcasts admit more tractable approximations [10].

References

1. Mehran Abolhasan, Tadeusz A. Wysocki, and Eryk Dutkiewicz. A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, 2(1):1–22, 2004. `doi:10.1016/S1570-8705(03)00043-X`.
2. Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-Space Trade-offs in Population Protocols. In *Proc. Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2560–2579, 2017. `doi:10.1137/1.9781611974782.169`.
3. Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-Optimal Majority in Population Protocols. In *Proc. Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2221–2239, 2018. `doi:10.1137/1.9781611975031.144`.
4. Dan Alistarh and Rati Gelashvili. Recent Algorithmic Advances in Population Protocols. *SIGACT News*, 49(3):63–73, 2018. `doi:10.1145/3289137.3289150`.
5. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 290–299, 2004. `doi:10.1145/1011767.1011810`.
6. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.
7. Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007. `doi:10.1007/s00446-007-0040-2`.
8. James Aspnes. Clocked Population Protocols. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*, pages 431–440, 2017.
9. Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, and Hugo Gimbert. Controlling a Population. In *Proc. 28th International Conference on Concurrency Theory (CONCUR)*, volume 85, pages 12:1–12:16, 2017. `doi:10.4230/LIPIcs.CONCUR.2017.12`.
10. Michael Blondin, Christoph Haase, and Filip Mazowiecki. Affine Extensions of Integer Vector Addition Systems with States. In *Proc. 29th International Conference on Concurrency Theory (CONCUR)*, pages 14:1–14:17, 2018. `doi:10.4230/LIPIcs.CONCUR.2018.14`.
11. Dmitry Chistikov, Christoph Haase, and Simon Halfon. Context-free commutative grammars with integer counters and resets. *Theoretical Computer Science*, 735:147–161, 2018. `doi:10.1016/j.tcs.2016.06.017`.
12. Giorgio Delzanno, Jean-François Raskin, and Laurent Van Begin. Towards the Automated Verification of Multithreaded Java Programs. In *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 173–187, 2002. `doi:10.1007/3-540-46002-0_13`.
13. Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset Nets Between Decidability and Undecidability. In *Proc. 25th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 103–115, 1998. `doi:10.1007/BFb0055044`.

14 Robert Elsässer and Tomasz Radzik. Recent Results in Population Protocols for Exact Majority and Leader Election. *Bulletin of the EATCS*, 126, 2018.

15 E. Allen Emerson and Kedar S. Namjoshi. On Model Checking for Non-Deterministic Infinite-State Systems. In *Proc. Thirteenth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 70–80, 1998. doi:10.1109/LICS.1998.705644.

16 Javier Esparza, Alain Finkel, and Richard Mayr. On the Verification of Broadcast Protocols. In *Proc. 14th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 352–359, 1999. doi:10.1109/LICS.1999.782630.

17 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, 54(2):191–215, 2017. doi:10.1007/s00236-016-0272-3.

18 Alain Finkel and Jérôme Leroux. How to Compose Presburger-Accelerations: Applications to Broadcast Protocols. In *Proc. 22nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 145–156, 2002. doi:10.1007/3-540-36206-1_14.

19 Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter Machines and Counter Languages. *Mathematical Systems Theory*, 2(3):265–283, 1968.

20 Neil Immerman. Nondeterministic Space is Closed Under Complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988. doi:10.1137/0217058.

21 David Lee and Mihalis Yannakakis. Testing Finite-State Machines: State Identification and Verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994. doi:10.1109/12.272431.

22 Othon Michail and Paul G. Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *Journal of Parallel and Distributed Computing*, 81-82:1–10, 2015.

23 Christos H. Papadimitriou. *Computational complexity.* Academic Internet Publ., 2007.

24 Sylvain Schmitz and Philippe Schnoebelen. The Power of Well-Structured Systems. In *Proc. 24th International Conference on Concurrency Theory (CONCUR)*, pages 5–24, 2013. doi:10.1007/978-3-642-40184-8_2.

25 Philippe Schnoebelen. Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets. In *Proc. 35th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 616–628, 2010. doi:10.1007/978-3-642-15155-2_54.

26 Róbert Szelepcsényi. The Method of Forced Enumeration for Nondeterministic Automata. *Acta Informatica*, 26(3):279–284, 1988. doi:10.1007/BF00299636.

27 Jannis Uhlendorf, Agnès Miermont, Thierry Delaveau, Gilles Charvin, François Fages, Samuel Bottani, Pascal Hersen, and Gregory Batt. In silico control of biomolecular processes. In *Computational Methods in Synthetic Biology*, pages 277–285. Marchisio, Mario Andrea, 2015. doi:10.1007/978-1-4939-1878-2_13.

28 Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008. doi:10.1016/j.comnet.2008.04.002.