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

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 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

[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.

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. $(M + M')(e) \stackrel{\text{def}}{=} M(e) + M'(e)$ for every $e \in E$, and $M \leq M' \stackrel{\text{def}}{\iff} M(e) \leq M'(e)$ for every $e \in E$. We define multiset difference as $(M \ominus M')(e) \stackrel{\text{def}}{=} \max(M(e) - M'(e), 0)$ for every $e \in E$. The empty multiset is denoted $\mathbf{0}$ and, for every $e \in E$, we write $\boldsymbol{e} \stackrel{\text{def}}{=} \wr e \wr$. Finally, we define the *support* and *size* of $M \in \mathbb{N}^E$ respectively as $[\![M]\!] \stackrel{\text{def}}{=} \{e \in E : M(e) > 0\}$ and $|M| \stackrel{\text{def}}{=} \sum_{e \in E} M(e)$.

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

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

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 $Q \stackrel{\text{def}}{=} \{x, \overline{x}, \tilde{x}, 0, 1, \bot\}$, $\Sigma \stackrel{\text{def}}{=} \{x\}$, $I \stackrel{\text{def}}{=} x \mapsto x$, $L \stackrel{\text{def}}{=} \mathbf{0}$, $O(q) = 1 \stackrel{\text{def}}{\iff} q = 1$, 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) ≠ 1:
  while C(x) ≥ 2: fire s      /*  split agents equally from x to x̄ and 0 */
  if C(x) = 0: fire s̄         /* move agents from x̄ to x if no remainder */
if C(x̄) = 0: fire t1          /*            if no remainder, then accept */
else: fire t̄0 t0              /*                       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 $C \to C'$.

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.

[2] Unlike population protocols, where efficient leader election is non-trivial and much studied; see e.g. [14].

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 $\mathcal{P}_i$, 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

[3] Since every finite execution can be extended to a fair one, this condition is actually equivalent to "no terminal configuration is reachable from $I(\boldsymbol{\alpha})$".

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 $\overline{\varphi}(\boldsymbol{\alpha}) \stackrel{def}{=} 1 - \varphi(\boldsymbol{\alpha})$ 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:

$$Q \stackrel{\text{def}}{=} \Sigma \times (Q_1 \cup Q_0 \cup \{\texttt{reset}\})$$

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. $I(x) \stackrel{\text{def}}{=} (x, I_0(x))$. 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 $f'(y, r) \stackrel{\text{def}}{=} (y, f(r))$ 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 $f_i \colon Q \to Q$ be the function defined as $f_i(x, q) \stackrel{\text{def}}{=} (x, I_i(x))$ 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 $C(x_i) \stackrel{\text{def}}{=} \boldsymbol{v}_i$. The *size* of C is $|C| \stackrel{\text{def}}{=} \sum_{i=1}^{k} C(x_i)$.

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:

$$C_{\boldsymbol{\alpha}} \stackrel{\text{def}}{=} (q_0, (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_m, \underbrace{0, \ldots, 0}_{k-m \text{ times}})).$$

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 $\varphi \colon \mathbb{N}^m \to \{0, 1\}$ 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 $X' \stackrel{\text{def}}{=} X \cup \{\mathit{idle}, \mathit{err}\}$. The states of $\mathcal{P}$ are defined as

$$Q' \stackrel{\text{def}}{=} \underbrace{Q \times \{0,1\}}_{\text{leader states}} \cup \underbrace{X' \times X \times \{0,1\}}_{\text{nonleader states}}.$$

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. $L \stackrel{\text{def}}{=} \{\!\!\{(q_0, 0)\}\!\!\}$. Moreover, every nonleader agent initially has its origin set to its initial position, which will remain unchanged by definition of the forthcoming transition relation: $I(x) \stackrel{\text{def}}{=} (x, x, 0)$ for every $x \in X$. The output of each agent is its opinion:

$$\begin{aligned} O(q,b) &\stackrel{\text{def}}{=} b \\ O(x,y,b) &\stackrel{\text{def}}{=} b \end{aligned} \quad \text{for every } q \in Q, x \in X', y \in X, b \in \{0,1\}.$$

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:

$$(q, b), (x, y, b') \mapsto (r, b), (idle, y, b').$$

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:

$$(q, b), (idle, y, b') \mapsto (r, b), (x, y, b').$$

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. $f^x_{err}(x, y, b) \stackrel{\text{def}}{=} (err, y, b)$ 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.

$$\begin{aligned} f_{rst}(q, b) &\stackrel{\text{def}}{=} (q_0, 0) && \text{for every } q \in Q, b \in \{0, 1\}, \\ f_{rst}(x, y, b) &\stackrel{\text{def}}{=} (y, y, 0) && \text{for every } x \in X', y \in X, b \in \{0, 1\}. \end{aligned}$$

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:

$$(err, y, b) \mapsto (y, y, 0); f_{rst}.$$

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.

$$\begin{aligned} f_{err}(q, b) &\stackrel{\text{def}}{=} (q, 1) && \text{for every } q \in Q, b \in \{0, 1\}, \\ f_{err}(x, y, b) &\stackrel{\text{def}}{=} (x, y, 1) && \text{for every } x \in X', y \in X, b \in \{0, 1\}. \end{aligned}$$

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 $D \in \text{Pop}(Q')$, let

$$D(x) \stackrel{\text{def}}{=} \sum_{(x,y,b)\in Q'} D(x, y, b).$$

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 $D \in \text{Pop}(Q')$ 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 $D \in \text{Pop}(Q')$ 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 $D \in \text{Pop}(Q')$ 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 NL = 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 $f : Q \to Q$ 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.