Population Protocols and Related Models*

Paul G. Spirakis

Abstract This is a joint work with Ioannis Chatzigiannakis and Othon Michail. We discuss here the population protocol model and most of its well-known extensions. The population protocol model aims to represent sensor networks consisting of tiny computational devices with sensing capabilities that follow some unpredictable and uncontrollable mobility pattern. It adopts a minimalistic approach and, thus, naturally computes a quite restricted class of predicates and exhibits almost no fault-tolerance. Most recent approaches make extra realistic and implementable assumptions, in order to gain more computational power and/or speed-up the time to convergence and/or improve fault-tolerance. In particular, the mediated population protocol model, the community protocol model, and the PALOMA model, which are all extensions of the population protocol model, are thoroughly discussed. Finally, the inherent difficulty of verifying the correctness of population protocols that run on complete communication graphs is revealed, but a promising algorithmic solution is presented.

1 Introduction

Wireless Sensor Networks (WSNs) will play an increasingly important role in critical systems' infrastructure and should be correct, reliable and robust. Formal specification helps to obtain not only a better (more modular) description, but also a clear understanding and an abstract view of a system [8]. Given the increasing sophistication of WSN algorithms and the difficulty of modifying an algorithm once the network is deployed, there is a clear need to use formal methods to validate system

Research Academic Computer Technology Institute (RACTI), +302610960200, Patras, Greece, e-mail: spirakis@cti.gr

* This work has been partially supported by the ICT Programme of the European Union under contract number ICT-2008-215270 (FRONTS).

performance and functionality prior to implementing such algorithms [34]. Formal analysis requires the use of models, trusted to behave like a real system. It is therefore critical to find the correct abstraction layer for the models and to verify the models.

Towards providing a concrete and realistic model for future sensor networks, Angluin *et al*. [2] introduced the notion of a computation by a population protocol. Due to the minimalistic nature of their model, individual agents are extremely limited and can be represented as finite-state machines. The computation is carried out by a collection of agents, each of which receives a piece of the input. Information can be exchanged between two agents whenever they come into contact with (or sufficiently close to) each other. The goal is to ensure that every agent can eventually output the value that is to be computed. The critical assumptions that diversify the population protocol model from traditional distributed systems is that the interaction pattern is inherently nondeterministic and that the protocols' description is independent of the population size (that is, need $\mathcal{O}(1)$ total memory capacity in each agent). The latter is known as the *uniformity property* of population protocols. Moreover, population protocols are *anonymous* since there is no room in the state of an agent to store a unique identifier.

The population protocol model was designed to represent sensor networks consisting of very limited mobile agents with no control over their own movement. It also bears a strong resemblance to models of interacting molecules in theoretical chemistry [27, 26]. The defining features of the population protocol model are:

1. Anonymous, finite-state agents. The system consists of a large population of indistinguishable finite-state agents.
2. Computation by direct interaction. In the original model, agents do not send messages or share memory; instead, an interaction between two agents updates both of their states according to a global transition table. The actual mechanism of such interactions is abstracted away.
3. Unpredictable interaction patterns. The choice of which agents interact is made by an adversary. Agents have little control over which other agents they interact with, although the adversary may be limited to pairing only agents that are adjacent in an interaction graph, typically representing distance constraints or obstacle presence. A strong global fairness condition is imposed on the adversary to ensure that the protocol makes progress (e.g. the adversary cannot keep the agents forever disconnected).
4. Distributed inputs and outputs. The input to a population protocol is distributed across the agents of the entire population. In what concerns predicates, all agents are expected to give the correct output value (which is known as the *predicate output convention* [2]), thus, the output is collected from any agent in the population (after, of course, the computation has stabilized).
5. Convergence rather than termination. Population protocols generally cannot detect when they have finished; instead, the agents' outputs are required to converge after some finite time to a common correct value.

The population protocol model was inspired in part by work by Diamadi and Fischer [23] on trust propagation in a social network. The *urn automata* of [3] can be seen as a first draft of the model that retained in vestigial form several features of classical automata: instead of interacting with each other, agents could only interact with a finite-state controller, complete with input tape. The motivation given for the current model in [2] was the study of sensor networks in which passive agents were carried along by other entities; the canonical example was sensors attached to a flock of birds. The name of the model was chosen by analogy to population processes [33] in probability theory.

The initial goal of the model was to study the *computational limitations* of cooperative systems consisting of many limited devices (agents), imposed to passive (but *fair*) communication by some *scheduler*. Much work showed that there exists an exact characterization of the computable predicates: they are precisely the *semilinear predicates* or equivalently the predicates definable by first-order logical formulas in *Presburger arithmetic* [2, 5, 6, 7]. More recent work has concentrated on performance, supported by a random scheduling assumption. [16] proposed a collection of *fair* schedulers and examined the performance of various protocols. [12] went one step further by proposing a generic definition of probabilistic schedulers and proving that the schedulers of [16] are all fair with probability 1, and revealed the need for the protocols to adapt when natural modifications of the mobility pattern occur. [11, 19] considered a huge population hypothesis (population going to infinity) and studied the dynamics, stability, and computational power of probabilistic population protocols by exploiting the tools of continuous nonlinear dynamics. In [11] it was also proven that there is a strong relation between classical finite population protocols and models given by ordinary differential equations.

There exist a few extensions of the population protocol model in the relevant literature to more accurately reflect the requirements of practical systems. In [1] they studied what properties of restricted communication graphs are stably computable, gave protocols for some of them, and proposed the model extension with *stabilizing inputs*. The results of [5] show that again the semilinear predicates are all that can be computed by this model. Finally, some works incorporated agent failures [22] and gave to some agents slightly increased computational power [9] (heterogeneous systems). For an excellent introduction to most of the preceding subjects see [7].

In this chapter we start by presenting in detail the basic population protocol model. Unfortunately, the class of solvable problems by this theoretical model is fairly small. For instance, it does not include multiplication. Moreover, even for this restricted class, algorithms tolerate no failures or, at worst, a fixed number of benign failures [22]. Therefore, we present four interesting extensions of the population protocol model that investigate the computational benefits of cooperative systems when adding new features (e.g. to the hardware of the devices). The extended models are summarized as follows:

- First, based on [17] (see also [18]), the population protocol model is extended to include a *Mediator*, i.e., a global storage capable of storing very limited information for each communication arc (the state of the arc). When pairs of agents interact, they can read and update the state of the link (arc). The extended model is called the *Mediated Population Protocol* (*MPP*) model. Interestingly, although anonymity and uniformity are preserved in this model, *the presence of a mediator provides us with significantly more computational power* and gives birth to a new collection of interesting problems in the area of tiny networked and possibly moving artefacts; based on this model we can build systems with the ability of computing subgraphs and solve optimization problems concerning the communication graph. Moreover, as we shall see, MPPs are capable of computing non-semilinear predicates and here any stably computable predicate belongs to $NSPACE(m)$, where m denotes the number of edges of the communication graph.

- One of the most interesting and applicable capabilities of the mediated population protocol model is its ability to decide graph properties. To understand properties of the communication graph is an important step in almost any distributed system. In particular, if we temporarily disregard the input notion of the population and assume that all agents simply start from a unique initial state (and the same holds for the edges), then we obtain another interesting model that is called the *GDM* (standing for Graph Decision Mediated) model [15]. When GDM protocols are executed fairly on any communication graph G, after a finite number of steps stabilize to a configuration where all agents give 1 as output if G belongs to a graph language L, and 0 otherwise. This is motivated by the idea of having protocols that eventually accept all communication graphs (on which they run) that satisfy a specific property, and eventually reject all remaining communication graphs. The motivation for studying a simplified version of the mediated population protocol model is that it enables us to study what graph properties are stably computable by the mediated model without the need to keep in mind its remaining parameters (which, as a matter of fact, are a lot).
- Another direction for extending the population protocol model is to assume the existence of a unique identifier for each agent. This is a natural extension since, although a tiny device's memory is often very constrained, it is usually sufficient to store a unique identity. In fact, in most modern tiny devices the communication module is often equipped with a unique identifier. For example, they might contain Maxim's DS2411 chip, which stores just 64 bits of ROM and is set by the factory to store a unique serial number. This idea gave birth to the *Community Protocol* model [29]. In this model, all n agents have unique identifiers (ids) and can store $\mathcal{O}(1)$ other agents' ids. The ids are stored in ROM (as in the DS2411 chip), so that Byzantine agents cannot alter their ids. The usage of ids is restricted to their fundamental purpose, *identification*, by assuming that algorithms can only compare ids (an algorithm cannot, for example, perform arithmetic on ids). In addition to *having* ids, the ability of agents to *remember* other ids is crucial as, otherwise, the model would be as weak as population protocols. The computational power of this extension is greatly increased; a community protocol of n agents can simulate a nondeterministic Turing Machine of $\mathcal{O}(n\log n)$ space. In particular, it can compute any symmetric predicate in $NSPACE(n\log n)$. Moreover, as in the population protocol model, a single algorithm must work for all values of n. Furthermore, the simulation is resilient to a constant number of Byzantine failures. So, although community protocols only make a rational addi-

tional assumption (that is, the ids equipment), they are much more powerful than population protocols: they solve a much wider class of problems and tolerate Byzantine failures.

- Finally, we present another extension called the *PALOMA* model [14]. In this model, the system consists of PAssively mobile LOgarithmic space MAchines. The idea is to provide each agent with a memory whose size is logarithmic in the population size, which seems a very natural assumption: only 266 bits are required for 2^{266} agents (which is an astronomical population size)! Moreover, we can think of an agent as a small Turing Machine, which also seems natural: mobile phones, PDAs and many other common mobile devices are in fact sophisticated Turing Machines. The PALOMA model is also extremely strong, since it can stably compute any symmetric predicate in $NSPACE(n \log n)$.

A very important aspect of WSNs is to provide solutions that are verifiably correct, in the sense of giving a "proof" that the solution will work, given the application goals and network set-up. Population protocol models can detect errors in the design that are not so easily found using emulation or testing. Formal analysis techniques are also supported by (semi-)automated tools. Such tools can also detect errors in the design and they can be used to establish correctness. Model checking is an exhaustive state space exploration technique that is used to validate formally specified system requirements with respect to a formal system description [21]. Such a system is verified for a fixed configuration; so in most cases, no general system correctness can be obtained. Using some high-level formal modeling language, automatically an underlying state space can be derived, be it implicitly or symbolically. The system requirements are specified using some logical language, like LTL, CTL or extensions thereof [32]. Well-known and widely applied model checking tools are SPIN [31], Uppaal [10] (for timed systems), and PRISM [30] (for probabilistic systems). The system specification language can, e.g., be based on process algebra, automata or Petri nets. However, model checking suffers from the so-called state explosion problem, meaning that the state space of a specified system grows exponentially with respect to its number of components. The main challenge for model checking lies in modeling large-scale dynamic systems.

The important feature that diversifies the population protocol model from traditional distributed systems is that the protocol specifications are independent of the population size which makes them suitable for the verification of protocols that target systems spanning thousands of objects. Evaluating if a property is valid or not in the system can be done with a number of components that is independent to the size of the population. The most important factor to decide the reachability of a certain configuration is the size of the protocol. Towards further minimizing the configuration space of the protocols we can apply the protocol composition methodology. This approach states that one may reduce a protocol into two (or more) protocols of reduced state space that maintain the same correctness and efficiency properties. The combination of the above help overcome the state explosion problem and speed up the verification process. We expect that population protocol models will be used to model such networks and the interactions, as dictated by the MAC protocol or the overall protocol stack, providing the ability, in a formal and modern way, to define the system in a minimalist way (in contrast to other approaches).

Section 2 discusses the population protocol model of Angluin *et al* [2]. Section 3 deals with a first extension of the population protocol model, the mediated population protocol model [17]. Section 4 goes one step further in the investigation of the mediated population protocol model by focusing on its ability to decide interesting graph properties. The simplified version of the mediated population protocol model discussed there is the GDM model [15]. In Section 5, the community protocol model of Guerraoui and Ruppert [29] is discussed and in Section 6 the PALOMA model [14]. Both models have the same computational power and are particularly powerful. Section 7 deals with correctness of population protocols that run on complete communication graphs. In particular, it focuses on the problem of algorithmically verifying whether a given population protocol is correct w.r.t. its specifications and is based on [13]. The problem is shown to be hard, but a promising algorithmic solution is presented. Finally, Section 8 discusses some interesting open problems in the area of small passively mobile communicating devices.

2 Population Protocols

We begin with a formal definition of the population protocol model proposed in a seminal work of Angluin *et al.* in [2]. The model represents sensor networks consisting of extremely limited agents that may move and interact in pairs.

2.1 The Model

Definition 1. A *population protocol* (PP) is a 6-tuple (X,Y,Q,I,O,δ), where X, Y, and Q are all finite sets and

1. X is the *input alphabet*,
2. Y is the *output alphabet*,
3. Q is the set of *states*,
4. $I : X \to Q$ is the *input function*,
5. $O : Q \to Y$ is the *output function*, and
6. $\delta : Q \times Q \to Q \times Q$ is the *transition function*.

If $\delta(a,b) = (a',b')$, we call $(a,b) \to (a',b')$ a *transition* and we define $\delta_1(a,b) = a'$ and $\delta_2(a,b) = b'$. We call δ_1 the *initiator's acquisition* and δ_2 the *responder's acquisition*.

A population protocol $\mathscr{A} = (X,Y,Q,I,O,\delta)$ runs on a communication graph $G = (V,E)$ with no self-loops and no multiple edges. From now on, we will denote by n the number of nodes of the communication graph and by m the number of its edges. Initially, all agents (i.e. the elements of V) receive a global start signal, sense their environment and each one receives an input symbol from X. All agents are initially in a special empty state $\sqcup \notin Q$. When an agent receives an input symbol σ, applies the input function to it and goes to its initial state $I(\sigma) \in Q$. An adversary scheduler selects in each step a directed pair of distinct agents $(u, \upsilon) \in E$ (that is, $u, \upsilon \in V$ and $u \neq \upsilon$) to interact. The interaction is established only if both agents are not in the empty state (they must both have been initialized). Assume that the scheduler selects the pair (u, υ), that the current states of u and υ are $a, b \in Q$, respectively, and that $\delta(a,b) = (a', b')$. Agent u plays the role of the *initiator* in the interaction (u, υ) and υ that of the *responder*. During their interaction u and υ apply the transition function to their directed pair of states (to be more precise, the initiator applies δ_1 while the responder δ_2) and, as a result, u goes to a' and υ to b' (both update their states according to δ).

A *configuration* is a snapshot of the population states. Formally, a configuration is a mapping $C : V \to Q$ specifying the state of each agent in the population. C_0 is the initial configuration (for simplicity, we assume that all agents apply the input function at the same time, which is one step before C_0, so in C_0 all empty states have been already replaced, and that' s the reason why we have chosen not to include $\sqcup$ in the model definition) and, for all $u \in V$, $C_0(u) = I(x(u))$, where $x(u)$ is the input symbol sensed by agent u. Let C and C' be configurations, and let u, υ be distinct agents. We say that C goes to C' via encounter $e = (u, \upsilon)$, denoted $C \xrightarrow{e} C'$, if

$$\begin{aligned}
&C'(u) = \delta_1(C(u), C(\upsilon)),\\
&C'(\upsilon) = \delta_2(C(u), C(\upsilon)), \text{ and}\\
&C'(w) = C(w) \text{ for all } w \in V - \{u, \upsilon\},
\end{aligned}$$

that is, C' is the result of the interaction of the pair (u, υ) under configuration C and is the same as C except for the fact that the states of u, υ have been updated according to δ_1 and δ_2, respectively. We say that C can *go to C' in one step*, denoted $C \to C'$, if $C \xrightarrow{e} C'$ for some encounter $e \in E$. We write $C \xrightarrow{*} C'$ if there is a sequence of configurations $C = C_0, C_1, \ldots, C_t = C'$, such that $C_i \to C_{i+1}$ for all i, $0 \leq i < t$, in which case we say that C' is *reachable* from C.

An *execution* is a finite or infinite sequence of configurations C_0, C_1, $C_2, \ldots$, where C_0 is an initial configuration and $C_i \to C_{i+1}$, for all $i \geq 0$. We have both finite and infinite kinds of executions since the scheduler may stop in a finite number of steps or continue selecting pairs for ever. Moreover, note that, according to the preceding definitions, a scheduler may partition the agents into non-communicating clusters. If that's the case, then it is easy to see that no meaningful computation is possible. To avoid this unpleasant scenario, a strong global *fairness condition* is imposed on the scheduler to ensure the protocol makes progress. Formally, an infinite execution is *fair* if for every pair of configurations C and C' such that $C \to C'$, if C occurs infinitely often in the execution, then C' also occurs infinitely often in the execution. A scheduler is fair if it always leads to fair executions. A *computation* is an infinite fair execution.

The above fairness condition, although at first sight may seem too strong, is in fact absolutely natural. The reason is that in most natural systems, between those under consideration, the passive mobility pattern that the agents follow will be the result of some natural phenomenon, like, for example, birds flying, river flow, and so on, that usually follows some probability distribution or, possibly, a collection of such probability distributions. Most of these schedulers, as indicated by [12], satisfy the above fairness condition; they only have to also satisfy some natural properties.

The following are two critical properties of population protocols:

1. *Uniformity*: Population protocols are uniform. This means that any protocol's description is independent of the population size. Since we assume that the agents have finite storage capacity, and independent of the population size, uniformity enables us to store the protocol code in each agent of the population.
2. *Anonymity*: Population protocols are anonymous. The set of states is finite and does not depend on the size of the population. This implies that there is no room in the state of an agent to store a unique identifier, and, thus, all agents are treated in the same way by the transition function.

Example 1. A very celebrated population protocol is the "*flock of birds*" (or "*count to five*") protocol. Every bird in a particular flock is equipped with a sensor node that can determine whether the bird's temperature is elevated or not, and we wish to know whether at least 5 birds in the flock have elevated temperatures. Moreover, we assume that all ordered pairs of sensor nodes are permitted interaction. This was the motivating scenario of population protocols [2].

We think as follows. The sensor senses the temperature of its corresponding bird (its carrier) and if it is found elevated it outputs 1, otherwise 0. As soon as the agent receives a global start signal (e.g. from a base station) it reads its sensor's output $\sigma \in \{0,1\}$ and applies to it the input function I. We can assume here that I maps 0 to initial state q_0 and 1 to q_1. This means that the number of agents that are in state q_1 under the initial configuration is equal to the number of sick birds, while all remaining agents are in state q_0. Now, when two agents interact, the initiator sets its state index to the sum of the state indices and the responder goes to q_0, except for the case in which the sum of the indices is at least 5. In the latter case both agents set their indices to 5. The idea is to try aggregating the initial number of 1 indices to one agent's state index. Note that the sum of nonzero indices is always equal to the number of sick birds; obviously, this holds until index 5 first appears. But what about the output of the protocol? If an agent gets q_5 then it knows that initially at least 5 birds were sick, and it outputs the value 1 in order to indicate this fact, and eventually q_5 is propagated to all agents. Otherwise, it outputs 0 because it may still have partial information.

Let us now formalize the above description. The "*flock of birds*" protocol is $\mathscr{F} = (X,Y,Q,I,O,\delta)$. The input and output alphabets are $X = Y = \{0,1\}$, the set of states is $Q = \{q_0,q_1,\dots,q_5\}$, the input function I maps 0 to q_0 and 1 to q_1, the output function O maps q_5 to 1 and all states in $\{q_0,\dots,q_4\}$ to 0, and the transition function $\delta(q_i,q_j)$ is defined as follows:

1. if $i+j<5$, then the result is (q_{i+j},q_0), and
2. if $i+j \geq 5$, then the result is (q_5,q_5).

$\square$

Exercise 1. Assume that all agents may err in two different ways. One possibility is that they do not apply the input function correctly and another is that they do not apply the transition function correctly. Fortunately, all are equipped with a special mechanism that automatically overwrites the faulty state with state $r \in \{r_1, r_2\}$, where r_1 and r_2 are the error reports/identifiers for the input function and the transition function, respectively. Try to adapt the "*flock of birds*" protocol to this scenario by keeping in mind that we require the protocol to give the correct output or report all the errors that have occurred.

Exercise 2. All birds in the flock are now additionally equipped with a sensor that determines their color, which is either black or white. Try to modify the "*flock of birds*" protocol in order to determine whether at least 3 black birds in the flock have elevated temperatures. Also exploit the white birds in order to (possibly[2]) improve performance.
Hint: assume that the input symbols are of the form (i, j) where i corresponds to the temperature and j to the color.

2.2 Stable Computation

Assume a fair scheduler that keeps working forever and a protocol $\mathscr{A}$ that runs on a communication graph $G = (V, E)$. As already said, initially, each agent receives an input symbol from X. An *input assignment* $x : V \to X$ is a mapping specifying the input symbol of each agent in the population. Let $\mathscr{X} = X^V$ be the set of all possible input assignments, given the population V and the input alphabet X of $\mathscr{A}$. Population protocols, when controlled by infinitely working schedulers, do not halt. Instead of halting we require any computation of a protocol to *stabilize*. An output assignment $y : V \to Y$ is a mapping specifying the output symbol of each agent in the population. Any configuration $C \in \mathscr{C} = Q^V$ is associated with an output assignment $y_C = O \circ C$. A configuration C is said to be *output-stable* if for any configuration C' such that $C \xrightarrow{*} C'$ (any configuration reachable from C) $y_{C'} = y_C$. In words, a configuration C is output-stable if all agents maintain the output symbol that have under C in all subsequent steps, no matter how the scheduler proceeds thereafter. A computation $C_0, C_1, C_2, \ldots$ is *stable* if it contains an output-stable configuration C_i, where i is finite.

Definition 2. A population protocol $\mathscr{A}$ running on a communication graph $G = (V, E)$ *stably computes a predicate* $p : \mathscr{X} \to \{0, 1\}$, if, for any $x \in \mathscr{X}$, every computation of $\mathscr{A}$ on G beginning from $C_0 = I \circ x$ reaches in a finite number of steps an output-stable configuration C_{stable} such that $y_{C_{stable}}(u) = p(x)$ for all $u \in V$. A predicate is *stably computable* if some population protocol stably computes it.

Assume, for example, that a computation of $\mathscr{A}$ on G begins from the initial configuration corresponding to an input assignment x. Assume, also, that $p(x) = 1$.

[2] We say *possibly*, because performance mainly depends on the scheduler. But if the scheduler is assumed to be probabilistic, then exploiting all agents should improve expected performance.

If $\mathscr{A}$ stably computes p, then we know that after a finite number of steps (if, of course, the scheduler is fair) all agents will give 1 as output, and will continue doing so for ever. This means, that if we wait for a sufficient, but finite, number of steps we can obtain the correct answer of p with input x by querying any agent in the population.

Definition 3. The *basic population protocol model* (or *standard*) assumes that the communication graph G is always directed and complete.

In the case of the basic model, a configuration simplifies to a vector of nonnegative integers that sum up to n indexed by states, and similarly for input assignments. Intuitively, we are allowed to do so because agents are anonymous and fairness guarantees that it does not matter in which agent each symbol or state lies. Moreover, here, stably computable predicates have to be *symmetric*. A predicate on input assignments p is called *symmetric* if for every $x = (\sigma_1, \sigma_2, \ldots, \sigma_n) \in \mathscr{X}$ and any x' which is a permutation of x's components, it holds that $p(x) = p(x')$ (in words, permuting the input symbols does not affect the predicate's outcome).

Thus, in the basic model, we can ignore the agents' underlying names to obtain a, seemingly, less descriptive, but sufficient for the basic model, definition of a configuration c as a $|Q|$-vector of nonnegative integers $(c_i)_{i=0,\ldots,|Q|-1}$, where $c_i = |c^{-1}(q_i)|$ and $|c^{-1}(q_i)|$ is equal to the number of agents to which state q_i is assigned by configuration c (the cardinality of the *preimage* of q_i), for all $i \in \{0, \ldots, |Q|-1\}$. It is not hard to see that the above definition implies that $\sum_{i=0}^{|Q|-1} c_i = n$ for any configuration c.

Exercise 3. Do the same for the input assignments; that is, define formally their vector description.

Example 2. Now, that the most important notions have been defined, we are ready to prove that the "*flock of birds*" protocol stably computes the predicate

$$p(x) = \begin{cases} 1, \text{ if } x_1 >= 5 \\ 0, \text{ if } x_1 < 5, \end{cases}$$

where x_1 denotes the number of agents that get input symbol 1. Another way to write the predicate is $(x_1 >= 5)$, which specifies that the value "true" is expected as output by all agents for every input assignment that provides at least 5 agents with the input symbol 1.

Proof. There is no transition in δ that decreases the sum of the indices. In particular, if $i + j < 5$ then transitions are of the form $(q_i, q_j) \to (q_{i+j}, q_0)$ and leave the sum unaffected, while if $i + j \geq 5$ then transitions are of the form $(q_i, q_j) \to (q_5, q_5)$ and all strictly increase it except for $(q_5, q_5) \to (q_5, q_5)$ that leaves it unaffected. So the initial sum is always preserved except for the case where state q_5 appears. If $x_1 < 5$ then it suffices to prove that state q_5 does not appear, because then all agents will forever remain in states $\{q_0, \ldots, q_4\}$ that give output 0. Assume that it appears. When this happened for the first time it was because the sum of the states of two interacting agents was at least 5. But this is a contradiction, because the initial sum should have been preserved until q_5 appeared. We now prove that if q_5 ever appears then all agents will eventually get it and remain to it forever. Obviously, if all get q_5 then they cannot escape from it, because no transition does this, thus, they forever remain to it. Now assume that q_5 has appeared in agent u and that agent $\upsilon \neq u$ never gets it. From the time that u got q_5 it could not change its state, thus any interaction of u and υ would make υ's state be q_5. This implies that u and υ did not interact for infinitely many steps, but this clearly violates the fairness condition (a configuration in which υ is in q_5 was always reachable in one step but was never reached). Now, if $x_1 \geq 5$ then it suffices to prove that q_5 appears. To see this, notice that all reachable configurations c for which $c_{q_5} = 0$ can reach in one step themselves and some configurations that preserve the sum but decrease the number of agents not in state q_0. Due to fairness, this will lead to a decrease by one in the number of non-q_0 agents in a finite number of steps, implying an increase in one agent's state index. This process ends either when all indices have been aggregated to one agent or when two agents, having a sum of indices at least 5, interact, and it must end, otherwise the number of q_0 agents would increase an unbounded number of times, being impossible for a fixed n. $\square$

Note that such proofs are simplified a lot when we use arguments of the form "if q_5 appears then due to fairness all agents will eventually obtain it" and "due to fairness the sum will eventually be aggregated to one agent unless q_5 appears first" without getting into the details of the fairness assumption. Of course, we have to be very careful when using abstractions of this kind. $\square$

Exercise 4. Consider the following protocol, known as "*parity protocol*":

The input and output alphabets are $X = Y = \{0,1\}$. The state of each agent consists of a data bit and a live bit. Initially, the data bit is equal to the input bit and the live bit is 1. For each state, the output bit is equal to the data bit. When two agents meet whose live bits are both 1, one sets its live bit to 0, and the other sets its data bit to the mod 2 sum of their data bits. When an agent with live bit 0 (a *sleeping* agent) meets an agent with live bit 1 (an *awake* agent), the former copies the data bit of the latter.

Prove that the "*parity protocol*" stably computes the predicate $(x_1 \bmod 2 = 1)$, which is true iff there is an odd number of 1s in the input.

Exercise 5. Given a population protocol $\mathscr{A}$, if Q is the set of $\mathscr{A}$'s states and if $\mathscr{A}$ runs on the complete communication graph of n nodes (basic model), show that there are $(1 + \frac{n}{|Q|-1})^{|Q|-1}$ different configurations.

Semilinear predicates are predicates whose support is a semilinear set. A *semilinear set* is the finite union of linear sets. A set of vectors in $\mathbb{N}^k$ is *linear* if it is of the form

$$\{\mathbf{b} + l_1\mathbf{a}_1 + l_2\mathbf{a}_2 + \cdots + l_m\mathbf{a}_m \mid l_i \in \mathbb{N}\},$$

where $\mathbf{b}$ is a *base* vector, $\mathbf{a}_i$ are *basis* vectors, and l_i are nonnegative integer coefficients. Moreover, semilinear predicates are precisely those predicates that can be defined by first-order logical formulas in *Presburger arithmetic*, as was proven by Ginsburg and Spanier [28].

Angluin *et al*. proved in [2] that any semilinear predicate is stably computable by the basic population protocol model and in [5] that any stably computable predicate, by the same model, is semilinear, thus together providing an exact characterization of the class of stably computable predicates:

Theorem 1 ([2, 5]). *A predicate is stably computable by the basic population protocol model iff it is semilinear.*

An immediate observation is that predicates like “the number of c’s is the product of the number of a’s and the number of b’s (in the input assignment)” and “the number of 1’s is a power of 2” are not stably computable by the basic model.

A *graph family*, or *graph universe*, is any set of communication graphs. Let $\mathscr{G}$ be a graph family. For any $G \in \mathscr{G}$, and given that X is the input alphabet of some protocol $\mathscr{A}$, there exists a set $\mathscr{X}_G$ of all input assignments *appropriate* for G, defined as $\mathscr{X}_G = X^{V(G)}$. Let now $\mathscr{X}_{\mathscr{G}} = \bigcup_{G \in \mathscr{G}} (\mathscr{X}_G \times \{G\})$ or, equivalently, $\mathscr{X}_{\mathscr{G}} = \{(x, G) \mid G \in \mathscr{G}$ and x is an input assignment appropriate for $G\}$. Then we have the following definition:

Definition 4. A population protocol $\mathscr{A}$ *stably computes a predicate* $p : \mathscr{X}_{\mathscr{G}} \to \{0,1\}$ in a family of communication graphs $\mathscr{G}$, if, for any $G \in \mathscr{G}$ and any $x \in \mathscr{X}_G$, every computation of $\mathscr{A}$ on G beginning from $C_0 = I \circ x$ reaches in a finite number of steps an output-stable configuration C_{stable} such that $y_{C_{stable}}(u) = p(x, G)$ for all $u \in V(G)$.

Moreover, if p is a mapping from $\mathscr{G}$ to $\{0,1\}$, that is, a *graph property* (obviously, independent of the input assignment), then we say that $\mathscr{A}$ stably computes property p.

Note that we can also consider undirected communication graphs. In the case of an undirected graph we only require that E is symmetric, but we keep the initiator-responder assumption. The latter is important to ensure deterministic transitions, since otherwise we would not be able to know which agent applies δ_1 and which δ_2.

3 Mediated Population Protocols

Consider now the following question: “Is there a way to extend the population protocol model and obtain a stronger model, without violating the uniformity and anonymity properties”? As we shall, in this section, see, the answer is in the affirmative. Although the idea is simple, it provides us with a model with significantly more computational power and extra capabilities in comparison to the population protocol model. The main modification is to allow the edges of the communication graph to store states from a finite set, whose cardinality is independent of the population size. Two interacting agents read the corresponding edge’s state and update it, according to a global transition function, by also taking into account their own states.

3.1 Formal Definition

Definition 5. A *mediated population protocol* (MPP) is a 12-tuple $(X,Y,Q,I,O,S,\iota,\omega,r,K,c,\delta)$, where X, Y, Q, S, and K are all finite sets and

1. X is the *input alphabet*,
2. Y is the *output alphabet*,
3. Q is the set of *agent states*,
4. $I : X \rightarrow Q$ is the *agent input function*,
5. $O : Q \rightarrow Y$ is the *agent output function*,
6. S is the set of *edge states*,
7. $\iota : X \rightarrow S$ is the *edge input function*,
8. $\omega : S \rightarrow Y$ is the *edge output function*,
9. r is the *output instruction* (informing the output-viewer how to interpret the output of the protocol),
10. K is the totally ordered *cost set*,
11. $c : E \rightarrow K$ is the *cost function*
12. $\delta : Q \times Q \times K \times S \rightarrow Q \times Q \times K \times S$ is the *transition function*.

We assume that the cost remains the same after applying δ and so we omit specifying an output cost. If $\delta(q_i,q_j,x,s) = (q_i',q_j',s')$ (which, according to our assumption, is equivalent to $\delta(q_i,q_j,x,s) = (q_i',q_j',x,s')$), we call $(q_i,q_j,x,s) \rightarrow (q_i',q_j',s')$ a *transition*, and we define $\delta_1(q_i,q_j,x,s) = q_i'$, $\delta_2(q_i,q_j,x,s) = q_j'$ and $\delta_3(q_i,q_j,x,s) = s'$. Here, we, additionally, call δ_3 the *edge acquisition* (after the corresponding interaction).

In most cases we assume that $K \subset \mathbb{Z}^+$ and that $c_{max} = \max_{w \in K}\{w\} = \mathcal{O}(1)$. Generally, if $c_{max} = \max_{w \in K}\{|w|\} = \mathcal{O}(1)$ then any agent is capable of storing at most k cumulative costs (at most the value kc_{max}), for some $k = \mathcal{O}(1)$, and we say that the cost function is *useful* (note that a cost range that depends on the population size could make the agents incapable for even a single cost storage and any kind of optimization would be impossible).

A *network configuration* is a mapping $C : V \cup E \rightarrow Q \cup S$ specifying the agent state of each agent in the population and the edge state of each edge in the communication graph. Let C and C' be network configurations, and let u, υ be distinct agents. We say that C goes to C' via encounter $e = (u,\upsilon)$, denoted $C \xrightarrow{e} C'$, if

$$\begin{aligned}
C'(u) &= \delta_1(C(u),C(\upsilon),x,C(e)) \\
C'(\upsilon) &= \delta_2(C(u),C(\upsilon),x,C(e)) \\
C'(e) &= \delta_3(C(u),C(\upsilon),x,C(e)) \\
C'(z) &= C(z), \text{ for all } z \in (V - \{u,\upsilon\}) \cup (E - e).
\end{aligned}$$

The definitions of *execution* and *computation* are the same as in the population protocol model but concern network configurations. Note that the mediated population protocol model *preserves both uniformity and anonymity* properties. As a result, any

MPP's *code* is of *constant size* and, thus, can be stored in each agent (device) of the population.

A configuration C is called *r-stable* if one of the following conditions holds:

- If the problem concerns a subgraph to be found, then C should fix a subgraph that will not change in any C' reachable from C.
- If the problem concerns a function to be computed by the agents, then an r-stable configuration drops down to an output-stable configuration.

We say that a protocol $\mathscr{A}$ *stably solves* a problem Π, if for every instance I of Π and every computation of $\mathscr{A}$ on I, the network reaches an r-stable configuration C that gives the correct solution for I if interpreted according to the output instruction r. If instead of a problem Π we have a function f to be computed, we say that $\mathscr{A}$ *stably computes* f.

In the special case where Π is an optimization problem, a protocol that stably solves Π is called an *optimizing population protocol* for problem Π.

Example 3. We will present now a MPP with a leader that stably solves the following problem:

Problem 1 (Transitive Closure). We are given a complete directed communication graph $G = (V, E)$. Let E' be a subset of E. For all $e \in E'$) it holds that initially the state of e is 1. We are asked to find the transitive closure of G', that is, find a new edge set E^* that will contain a directed edge (u, υ) joining any nodes u, υ for which there is a non-null path from u to υ in G' (note that always $E' \subseteq E^*$).

We assume a *controlled input assignment* $W : E \to X$ that allows us to give input 1 to any edge belonging to E' and input 0 to any other edge. Moreover, we assume that initially all agents are in state q_0, except for a unique leader that is in state l.

Protocol 1 *TranClos*

```
1: $X = Y = \{0,1\}$
2: $Q = \{l, q_0, q_1, q_1', q_2, q_2', q_3\}$
3: $S = \{0,1\}$
4: controlled input assignment: "$W(e') = 1$, for all $e' \in E'$, and $W(e) = 0$, for all $e \in E - E'$"
5: $\iota(x) = x$, for all $x \in X$
6: $\omega(s) = s$, for all $s \in S$
7: $r$: "Collect the subgraph induced by all $e \in E$ for which $\omega(s_e) = 1$ (where $s_e$ is the state of $e$)"
8: $\delta$:
       $(l, q_0, 0) \to (q_0, l, 0)$        $(q_2, q_0, 1) \to (q_2', q_3, 1)$
       $(l, q_0, 1) \to (q_1, q_2, 1)$      $(q_1, q_3, x) \to (q_1', q_0, 1)$, for $x \in \{0,1\}$
       $(q_1, q_2, 1) \to (q_0, l, 1)$      $(q_1', q_2', 1) \to (q_0, l, 1)$
```

□

The MPP *TranClos* (Protocol 1) stably solves the transitive closure problem (Problem 1). You are asked to prove this in Exercise 6. Let us, first, explain one after the other the protocol's components before explaining its functionality (that is, what is done by the transition function δ). The input alphabet X and the output alphabet Y are both binary, that is, both consists of the symbols 0 and 1. The set of agent states Q consists of the states l, q_0, q_1, q_1', q_2, q_2', and q_3, which are the states that agents may obtain during the protocol's computations. The set of edge states S is binary, which means that the edges joining the agents will at any time be in one of the states 0 and 1. The controlled input assignment simply specifies that all edges belonging to E' are initially in state 1 (by taking into account that $\iota(x) = x$, for all $x \in X$) and all remaining edges of the communication graph are initially in state 0. This is done in order to help the protocol distinguish E'. $\omega(s) = s$, for all $s \in S$, simply says that the output of any edge is its current state, thus, either 0 or 1. Finally, the output instruction r informs the user that the protocol's output will consist of all edges that will eventually output the value 1. In this case, these edges will form the transitive closure of the communication graph G. We next discuss the protocol's functionality, which is described by the transition function δ.

The protocol *TranClos* (Protocol 1) repeats the following procedure. Initially, by assumption, there is a unique leader u in state l and all the other agents are in q_0. When the leader u interacts with an agent υ in q_0 through (u,υ) in state 0, the agents swap states, that is, now υ is the unique leader. If, instead, (u,υ) is in state 1, then the leader gets q_1 and υ gets q_2. After the latter has happened, all agents are in q_0 except for u and υ which are in q_1 and q_2, respectively, while (u,υ) is in state 1, and only the rules $(q_1,q_2,1) \to (q_0,l,1)$ and $(q_2,q_0,1) \to (q_2',q_3,1)$ can apply (all the other rules have no effect). If the former applies first, then the population goes to a configuration similar to the initial one, with a unique leader and all the other agents in q_0. This rule is important (although maybe not obvious why) since it guarantees that, if υ, which is in q_2, has no outgoing neighbor w, where $q_w = q_0$ and $s_{(\upsilon,w)} = 1$, then the protocol won't get stuck. If, instead, the latter applies first, then υ has interacted with an agent w in q_0 where (υ,w) is in state 1. Now υ gets q_2' and w gets q_3. After this step, the protocol has formed the directed path $u\upsilon w$, with agent states q_1,q_2',q_3, respectively, and (u,υ), (υ,w) (i.e. the edges of the path) in state 1. From now on, only $(q_1,q_3,x) \to (q_1',q_0,1)$ can apply, which simply assigns state 1 to the edge (u,w). Finally, the population remains again with a unique leader, υ, and all the other agents in q_0, simply proceeding with the same general operation that we have just described.

Exercise 6. Give a formal proof that the MPP *TranClos* (Protocol 1) stably solves the transitive closure problem (Problem 1).

Exercise 7. Assume that the input symbols are only 0 and 1 and that the communication graph $G = (V,E)$ is any directed graph. Let G' be the subgraph of G induced by $V' = \{u \in V \mid u \text{ gets the input symbol } 1\}$. Devise a MPP that will construct a (not necessarily connected) subgraph $G'' = (V'',E'')$ of G', in which all nodes have in-degree at most one and out-degree at most one.

3.2 Computational Power

The population protocol model is a special case of the mediated population protocol model (try to prove it). Moreover, as we shall see, there exists a MPP protocol that stably computes the non-semilinear predicate $(N_c = N_a \cdot N_b)$. In words, it eventually decides whether the number of c's in the input assignment is equal to the product of the number of a's and the number of b's. The following definitions will prove useful for our purpose.

Definition 6. A MPP $\mathscr{A}$ has *stabilizing states* if in any computation of $\mathscr{A}$, after a finite number of interactions, the states of all agents stop changing.

Definition 7. We say that a predicate is *strongly stably computable* by the MPP model, if it is stably computable with the predicate output convention, that is, all agents eventually agree on the correct output value.

Protocol 2 *VarProduct*

```
1: $X = \{a,b,c,0\}$
2: $Y = \{0,1\}$
3: $Q = \{a,\dot{a},b,c,\bar{c},0\}$
4: $S = \{0,1\}$
5: $I(x) = x$, for all $x \in X$
6: $O(a) = O(b) = O(\bar{c}) = O(0) = 1$ and $O(c) = O(\dot{a}) = 0$
7: $\iota(x) = 0$, for all $x \in X$
8: $r$: "If there is at least one agent with output 0, reject; otherwise, accept."
9: $\delta$: $(a,b,0) \to (\dot{a},b,1)$, $(c,\dot{a},0) \to (\bar{c},a,0)$, $(\dot{a},c,0) \to (a,\bar{c},0)$
```

Theorem 2. *The MPP VarProduct (Protocol 2) stably computes (according to the output instruction r) the predicate $(N_c = N_a \cdot N_b)$ in the family of complete directed communication graphs.*

Proof. Notice that the number of links leading from agents in state a to agents in state b equals $N_a \cdot N_b$. For each a the protocol tries to erase b c's. Each a is able to remember the b's that it has already counted (for every such b a c has been erased) by marking the corresponding links. If the c's are less than the product then at least one $\dot{a}$ remains and if the c's are more at least one c remains. In both cases at least one agent that outputs 0 remains. If $N_c = N_a \cdot N_b$ then every agent eventually outputs 1. $\square$

Theorem 3. *Let p be any predicate on input assignments. Let $\mathscr{A}$ be a MPP that stably computes p with stabilizing states in some family of directed and connected communication graphs $\mathscr{G}$ and also assume that $\mathscr{A}$ contains an instruction r that defines a semilinear predicate t on multisets of $\mathscr{A}$'s agent states. Since t is semilinear, it is stably computable with stabilizing inputs by the PP model [1, 5], and, thus, by* *the MPP model. Let $\mathscr{B}$ be a MPP that strongly stably computes t with stabilizing inputs in $\mathscr{G}$.*

If all the above hold then $\mathscr{A}$ can be composed with $\mathscr{B}$ to give a new MPP $\mathscr{C}$ satisfying the following properties:

- *$\mathscr{C}$ is formed by the composition of $\mathscr{A}$ and $\mathscr{B}$,*
- *its input is $\mathscr{A}$'s input,*
- *its output is $\mathscr{B}$'s output, and*
- *$\mathscr{C}$ strongly stably computes p (i.e. all agents agree on the correct output) in $\mathscr{G}$.*

Exercise 8. Prove Theorem 3.
Hint: $\mathscr{B}$ will make use of the stabilizing inputs idea from [1]; its inputs that eventually stabilize are $\mathscr{A}$'s states.

Theorems 2 and 3 together imply that the MPP model strongly stably computes *VarProduct* which is non-semilinear. Since the MPP model strongly stably computes a non-semilinear predicate and the PP model is a special case of MPP, it follows that the class of computable predicates by MPP is a proper superset of the class of computable predicates by PP. In other words, *the MPP model is computationally stronger than the PP model*.

In what concerns the class of stably computable predicates by MPP, recent (unpublished) research shows that is a superset of $SSPACE(n)$ (symmetric predicates in $LINSPACE$). We also know the following upper bound: "Any predicate that is stably computable by the MPP model in any family of communication graphs belongs to the space complexity class $NSPACE(m)$" (recall that $m = |E|$). The idea is simple: By using the MPP that stably computes the predicate we construct a nondeterministic Turing machine that guesses in each step the next selection of the scheduler (thus the next configuration). The machine always replaces the current configuration with a new legal one, and, since any configuration can be represented explicitly with $\mathscr{O}(m)$ space, any branch uses $\mathscr{O}(m)$ space. The machine accepts if some branch reaches a configuration C that satisfies instruction r of the protocol, and if, moreover, no configuration reachable from C violates r (i.e. C must also be r-stable).

4 The GDM model

Here we deal with MPP's ability to decide graph languages. To do so, we consider a special case of the mediated population protocol model, the *Graph Decision Mediated population protocol model*, or simply *GDM* model.

4.1 Formal Definition

Definition 8. A *GDM* protocol is an 8-tuple $(Y, Q, O, S, r, \delta, q_0, s_0)$, where Y, Q, and S are all finite sets and

1. $Y = \{0,1\}$ is the *binary output alphabet*,
2. Q is the set of *agent states*,
3. $O : Q \to Y$ is the *agent output function*,
4. S is the set of *edge states*,
5. r is the *output instruction*,
6. $\delta : Q \times Q \times S \to Q \times Q \times S$ is the *transition function*,
7. $q_0 \in Q$ is the *initial agent state*, and
8. $s_0 \in S$ is the *initial edge state*.

If $\delta(a,b,s) = (a',b',s')$, we call $(a,b,s) \to (a',b',s')$ a *transition* and we define $\delta_1(a,b,s) = a'$, $\delta_2(a,b,s) = b'$, and $\delta_3(a,b,s) = s'$.

Let $\mathscr{U}$ be a graph universe. A *graph language* L is a subset of $\mathscr{U}$ containing communication graphs that possibly share some common property. For example, a common graph universe is the set of all possible directed and weakly connected communication graphs, denoted by $\mathscr{G}$, and $L = \{G \in \mathscr{G} \mid G$ has an even number of edges$\}$ is a possible graph language w.r.t. $\mathscr{G}$.

A GDM protocol may run on any graph from a specified graph universe. The graph on which the protocol runs is considered as the *input graph* of the protocol. Note that GDM protocols have no sensed input. Instead, we require each agent in the population to be initially in the initial agent state q_0 and each edge of the communication graph to be initially in the initial edge state s_0. In other words, the initial network configuration, C_0, of any GDM protocol is defined as $C_0(u) = q_0$, for all $u \in V$, and $C_0(e) = s_0$, for all $e \in E$, and any input graph $G = (V,E)$.

We say that a GDM protocol $\mathscr{A}$ *accepts* an input graph G if in any computation of $\mathscr{A}$ on G after finitely many interactions all agents output the value 1 and continue doing so in all subsequent (infinite) computational steps. By replacing 1 with 0 we get the definition of the *reject* case.

Definition 9. We say that a GDM protocol $\mathscr{A}$ *decides* a graph language $L \subseteq \mathscr{U}$ if it accepts any $G \in L$ and rejects any $G \notin L$.

Definition 10. A graph language is said to be *GDM-decidable*, or simply *decidable*, if some GDM protocol decides it.

4.2 Weakly Connected Graphs

4.2.1 Decidability

The most meaningful graph universe is $\mathscr{G}$ containing all possible directed and weakly connected communication graphs, without self-loops or multiple edges, of any finite number of nodes greater than or equal to 2 (we do not allow the empty graph, the graph with a unique node and we neither allow infinite graphs). Here the graph universe is $\mathscr{G}$ and, thus, a graph language can only be a subset of $\mathscr{G}$ (moreover, its elements must share some common property).

We begin with some easy to prove, but often useful, closure results.

Theorem 4. *The class of decidable graph languages is closed under complement, union and intersection operations.*

Proof. First we show that for any decidable graph language L its complement $\overline{L}$ is also decidable. From definition of decidability there exists a GDM protocol $\mathscr{A}_L$ that decides L. Thus, for any $G \in \mathscr{G}$ and any computation of $\mathscr{A}_L$ on G all agents eventually output 1 if $G \in L$ and 0 otherwise. By complementing the output map $O_{\mathscr{A}}$ of $\mathscr{A}$ we obtain a new protocol $\overline{\mathscr{A}}$, with output map defined as $O_{\overline{\mathscr{A}}}(q) = 1$ iff $O_{\mathscr{A}}(q) = 0$, for all $q \in Q_{\mathscr{A}} = Q_{\overline{\mathscr{A}}}$, whose agents eventually output 1 if $G \notin L$ and 0 otherwise, thus deciding $\overline{L}$.

Now we show that for any decidable graph languages L_1 and L_2, $L_3 = L_1 \cup L_2$ is also decidable. Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be GDM protocols that decide L_1 and L_2, respectively (we know their existence). We let the two protocols operate in parallel, i.e. we devise a new protocol $\mathscr{A}_3$ whose agent and edge states consist of two components, one for protocol $\mathscr{A}_1$ and one for $\mathscr{A}_2$. Let O_1 and O_2 be the output maps of the two protocols. We define the output map O_3 of $\mathscr{A}_3$ as $O_3(q,q') = 1$ iff at least one of $O_1(q)$ and $O_2(q')$ equals to 1, for all $q \in Q_{\mathscr{A}_1}$ and $q' \in Q_{\mathscr{A}_2}$. If $G \in L_3$ then at least one of the two protocols has eventually all its agent components giving output 1, thus $\mathscr{A}_3$ correctly answers “accept”, while if $G \notin L_3$ then both protocols have eventually all their agent components giving output 0, thus $\mathscr{A}_3$ correctly answers “reject”. We conclude that $\mathscr{A}_3$ decides L_3 which proves that L_3 is decidable.

By defining the output map O_3 of $\mathscr{A}_3$ as $O_3(q,q') = 1$ iff $O_1(q) = O_2(q') = 1$, for all $q \in Q_{\mathscr{A}_1}$ and $q' \in Q_{\mathscr{A}_2}$, and making the same composition as before, it is easy to see that in this case $\mathscr{A}_3$ decides the intersection of L_1 and L_2.

Note, however, that in each union and intersection operation the resulting protocol's size is the product of the sizes of the composed protocols. It follows that the closure under these two operations can only hold for a constant number of subsequent applications. □

Example 4. Let us now illustrate what we have seen so far by presenting a parametric GDM protocol that decides the graph language $N_k^{out} = \{G \in \mathscr{G} \mid G$ has some node with at least k outgoing neighbors$\}$ for any $k = \mathscr{O}(1)$.

We provide a high-level description of the protocol. Initially all agents are in q_0 and all edges in 0. The set of agent states is $Q = \{q_0, \ldots, q_k\}$ the set of edge states is binary and the output function is defined as $O(q_k) = 1$ and $O(q_i) = 0$ for all $i \in \{0, \ldots, k-1\}$. We now describe the transition function. In any interaction through an edge in state 0, the initiator visits an unvisited outgoing edge, so it marks it by updating the edge's state to 1 and increases its own state index by one, e.g. initially $(q_0, q_0, 0)$ yields $(q_1, q_0, 1)$, and, generally, $(q_i, q_j, 0) \to (q_{i+1}, q_j, 1)$, if $i+1 < k$ and $j < k$, and $(q_i, q_j, 0) \to (q_k, q_k, 1)$, otherwise. Whenever two agents meet through a marked edge they do nothing, except for the case where only one of them is in the special alert state q_k. If the latter holds, then both go to the alert state, since in this case the protocol has located an agent with at least k outgoing neighbors. To conclude, all agents count their outgoing edges and initially output 0. Iff one of them marks its k-th outgoing edge, both end points of that edge go to an alert state q_k that is eventually propagated to the whole population and whose output is 1, indicating that G belongs to N_k^{out}. Clearly, the described protocol decides N_k^{out}, which means that N_k^{out} is a decidable graph language. Moreover, the same must hold for $\overline{N}_k^{out}$ because, according to Theorem 4, the class of decidable graph languages is closed under complement. Note that $\overline{N}_k^{out}$ contains all graphs that have no node with at least $k = \mathcal{O}(1)$ outgoing neighbors. In other words, the GDM model can decide if all nodes have less than k outgoing edges, which is simply the well-known bounded by k out-degree predicate. □

Example 5. We show now that the graph language $P_k = \{G \in \mathcal{G} \mid G$ has at least one directed path of at least k edges$\}$ is decidable for any $k = \mathcal{O}(1)$ (the same holds for $\overline{P}_k$).

If $k = 1$ the protocol that decides P_1 is trivial, since it accepts iff at least one interaction happens (in fact it can always accept since all graphs have at least two nodes and they are weakly connected, and thus $P_1 = \mathcal{G}$). The protocol *DirPath* (Protocol 3) that we have constructed decides P_k for any constant $k > 1$.

Protocol 3 *DirPath*

```
1: $Q = \{q_0, q_1, 1, \dots, k\}$
2: $S = \{0, 1\}$
3: $O(k) = 1$, $O(q) = 0$, for all $q \in Q - \{k\}$
4: $r$: “Get any $u \in V$ and read its output”
5: $\delta$:
```

$$\begin{aligned}
(q_0, q_0, 0) &\to (q_1, 1, 1)\\
(q_1, x, 1) &\to (x-1, q_0, 0),\ \text{if } x \geq 2\\
&\to (q_0, q_0, 0),\ \text{if } x = 1\\
(x, q_0, 0) &\to (q_1, x+1, 1),\ \text{if } x+1 < k\\
&\to (k, k, 0),\ \text{if } x+1 = k\\
(k, \cdot, \cdot) &\to (k, k, \cdot)\\
(\cdot, k, \cdot) &\to (k, k, \cdot)
\end{aligned}$$

Initially all nodes are in q_0 and all edges in 0. The protocol tries to expand disjoint paths. When rule $(q_0, q_0, 0) \to (q_1, 1, 1)$ applies, the initiator goes to q_1 indicating that it is a node of an active path, the responder goes to 1 indicating that it is the head of an active path of length 1 and the edge goes to 1 indicating that it is part of an active path. By inspecting the transition function it is easy to see that the nodes of two disjoint active paths have no way of interacting with each other (in fact, the interactions happening between them leave their interacting components unaffected). This holds because all nodes in q_1 do nothing when communicating through an edge in state 0 and disjoint active paths can only communicate through edges in state 0.

Moreover, the heads of the paths only expand by communicating with nodes in q_0 which, of course, cannot be nodes of active paths (all nodes of active paths are in q_1 except for the heads which are in states from $\{1,\ldots,k-1\}$). There are two main possibilities for an active path: either the protocol expands it, thus obtaining a node and an edge and increasing the head counter by one, or shrinks it, thus releasing a node and an edge and decreasing the head counter by one. Eventually, a path will either be totally released (all its nodes and edges will return to the initial states) or it will become of length k. In the first case the protocol simply keeps working but in the second, a path of length at least k was found and state k that outputs 1 is correctly propagated. The crucial point is that state k is the only state that outputs 1 and can only be reached and propagated by the agents iff there exists some path of length at least k. Moreover, if such a path exists, due to fairness assumption, the protocol will eventually manage to find it. □

The following graph languages are also decidable by the GDM model:

1. $N_{even} = \{G \in \mathscr{G} \mid |V(G)| \text{ is even}\}$.
2. $E_{even} = \{G \in \mathscr{G} \mid |E(G)| \text{ is even}\}$.
3. $K_k^{out} = \{G \in \mathscr{G} \mid \text{Any node in } G \text{ has at least } k \text{ outgoing neighbors}\}$ for any $k = \mathscr{O}(1)$.
4. $M_{out} = \{G \in \mathscr{G} \mid G \text{ has some node with more outgoing than incoming neighbors}\}$.

Of course, by closure under complement, the same holds for their complements.

Exercise 9. Do you think it is possible to construct a GDM protocol that decides the graph language consisting of all directed and weakly connected communication graphs in which all nodes have at most $k = \mathscr{O}(1)$ incoming edges and in which the number of nodes is at least 5% of the number of edges? If yes, construct the protocol and prove its correctness; if no, explain why.

4.2.2 Undecidability

If we allow only GDM protocols with stabilizing states, i.e. GDM protocols that in any computation after finitely many interactions stop changing their states, then we can prove that a specific graph language w.r.t. $\mathscr{G}$ is *undecidable*. In particular, we can prove that there exists no GDM protocol with stabilizing states to decide the graph language

$$2C = \{G \in \mathscr{G} \mid G \text{ has at least two nodes } u, \upsilon \text{ s.t. both } (u,\upsilon),(\upsilon,u) \in E(G) \text{ (in other words, } G \text{ has at least one 2-cycle)}\}.$$

The proof is based on the following lemma.

Lemma 1. *For any GDM protocol $\mathscr{A}$ and any computation $C_0, C_1, C_2, \ldots$ of $\mathscr{A}$ on G (Figure 1(a)) there exists a computation $C'_0, C'_1, C'_2, \ldots, C'_i, \ldots$ of $\mathscr{A}$ on G' (Figure 1(b)) s.t.*

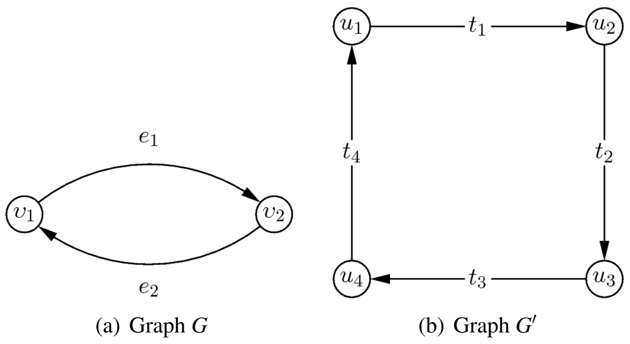

Fig. 1 $G \in 2C$ and $G' \notin 2C$.

$$\begin{aligned} C_i(\upsilon_1) &= C'_{2i}(u_1) = C'_{2i}(u_3) \\ C_i(\upsilon_2) &= C'_{2i}(u_2) = C'_{2i}(u_4) \\ C_i(e_1) &= C'_{2i}(t_1) = C'_{2i}(t_3) \\ C_i(e_2) &= C'_{2i}(t_2) = C'_{2i}(t_4) \end{aligned}$$

for any finite $i \geq 0$.

Exercise 10. Prove Lemma 1 by using induction on i.

Lemma 1 shows that if a GDM protocol $\mathscr{A}$ with stabilizing states could decide $2C$ then there would exist a computation of $\mathscr{A}$ on G' forcing all agents to output incorrectly the value 1 in finitely many steps. But G' does not belong to $2C$, and, since $\mathscr{A}$ decides $2C$, all agents must correct their states to eventually output 0. By taking into account the fact that $\mathscr{A}$ has stabilizing states it is easy to reach a contradiction and prove that no GDM protocol with stabilizing states can decide $2C$. Whether the graph language $2C$ is undecidable by the GDM model in the general case (not only by GDM protocols with stabilizing states) remains an interesting open problem.

4.3 All Possible Directed Graphs

It is not hard to show that if the graph universe, $\mathscr{H}$, is allowed to contain also disconnected communication graphs, then in this case the GDM model is incapable of deciding even a single nontrivial graph language (we call a graph language L *nontrivial* if $L \neq \emptyset$ and $L \neq \mathscr{H}$). Here we assume the graph universe $\mathscr{H}$ consisting of all possible directed communication graphs, without self-loops or multiple edges of any finite number of nodes greater or equal to 2 (we now also allow graphs that are not even weakly connected). So, now, a graph language can only be a subset of $\mathscr{H}$.

The crucial part is to show that for any nontrivial graph language L, there exists some disconnected graph G in L where at least one component of G does not belong to L or there exists some disconnected graph G' in $\overline{L}$ where at least one component of G' does not belong to $\overline{L}$ (or both). If the statement does not hold then any disconnected graph in L has all its components in L and any disconnected graph in $\overline{L}$ has all its components in $\overline{L}$.

1. All connected graphs belong to L. Then $\overline{L}$ contains at least one disconnected graph (since it is nontrivial) that has all its components in L, which contradicts the fact that the components of any disconnected graph in $\overline{L}$ also belong to $\overline{L}$.
2. All connected graphs belong to $\overline{L}$. The contradiction is symmetric to the previous case.
3. L and $\overline{L}$ contain connected graphs G and G', respectively. Their disjoint union $U = (V \cup V', E \cup E')$ is disconnected, belongs to L or $\overline{L}$ but one of its components belongs to L and the other to $\overline{L}$. The latter contradicts the fact that both components should belong to the same language.

Now it won't be hard to prove the impossibility result as an exercise.

Exercise 11. Prove that any nontrivial graph language $L \subset \mathscr{H}$ is undecidable by the GDM model.
Hint: notice that agents of different components cannot communicate with each other.

Exercise 12. Do you think that Connectivity property is GDM-decidable?

5 Community Protocols

In this section, we present the *Community Protocol* model, which was proposed by Guerraoui and Ruppert in [29] and is another extension of the population protocol model. In fact, this, recently proposed, model makes the assumption that the agents are equipped with unique ids and are also allowed to store a fixed number of other agents' ids. The term "community" in the model's name is used to emphasize the fact that the agents here form a collection of unique individuals similar to the notion of a human community, in contrast to a population which is merely an agglomeration of nameless multitude.

5.1 The Model

As usual, we start with a formal definition of the model, and then a somewhat informal description of its functionality follows.

Definition 11. Let U be an infinite ordered set containing all possible ids. A *Community Protocol Algorithm* is an 8-tuple (X,Y,B,d,I,O,Q,δ), where X, Y, and B are all finite sets and

1. X is the *input alphabet*,
2. Y is the *output alphabet*,
3. B is the set of *basic states*,
4. d is a nonnegative integer representing the number of ids that can be remembered by an agent,
5. $I : X \to B$ is the *input function* mapping input symbols to basic states,
6. $O : B \to Y$ is the *output function* mapping basic states to outputs,
7. $Q = B \times (U \cup \{\perp\})^d$ is the set of *agent states*, and
8. $\delta : Q \times Q \to Q \times Q$ is the *transition function*.

If $\delta(a,b) = (a',b')$, we call $(a,b) \to (a',b')$ a *transition* and we define $\delta_1(a,b) = a'$ and $\delta_2(a,b) = b'$.

The first obvious difference between this and the population protocol model is that the agent states are allowed to contain up to d ids. Additionally, each agent is assumed to have its own unique id from the industry (which is an element of U). As in the population protocol model, initially each agent $i \in \{1,\ldots,n\}$ receives an input symbol from X. Note that the ith agent is the agent whose id is in position i in the ascending ordering of agent ids. An input assignment $x \in \mathscr{X} = X^V$ is again any n-vector of input symbols, where x_i is the input to agent i. Moreover, let id_i denote the actual id of agent i and $b_i = I(x_i)$ (that is, the initial basic state of agent i). Then the *initial state* of each agent i is of the form $(b_i, id_i, \perp, \perp, \ldots, \perp)$. Thus initially, each agent i is in basic state b_i, contains its own unique id id_i in the first position of its list of ids, and the remaining list is filled with $d-1$ repetitions of the symbol $\perp$.

A *configuration* C is a vector in Q^n of the form $C = (q_1, q_2, \ldots, q_n)$, where q_i is simply the state of agent i for all $i \in \{1,\ldots,n\}$. Thus, the *initial configuration* corresponding to input assignment x is $((b_i, id_i, \perp, \perp, \ldots, \perp))_{i=1}^{|x|}$, where again $b_i = I(x_i)$ and id_i is the actual id of agent i. The notions of *execution*, *computation*, and *fairness* are defined in the same way as in the population protocol model. Moreover, we will call the community protocol model, in which the communication graph is directed and complete, *basic community protocol* model (like we did with the population protocol model). The scheduler choosing the interactions is again assumed to be *fair*.

The output of an agent at any step of the computation is the output of its basic state. For example, the output of an agent in state $(b_i, id_i, 1, 5, \ldots, \perp)$ is $O(b_i) \in Y$. A community protocol algorithm for the basic model *stably computes* a function $f : X^{\geq 2} \to Y$, where $X^{\geq 2}$ denotes the set of all finite strings over X of length at least 2, if for any $x \in X^{\geq 2}$ and any assignment of the symbols in x to the nodes of the complete communication graph of $|x|$ nodes, all agents, independently of the fair computation followed, eventually stabilize to the output $f(x)$; that is, a configuration is reached under which all agents output $f(x)$ and continue doing so forever, no matter how the computation proceeds thereafter (such a configuration is, as usual, called an *output stable* configuration).

As in population protocols, algorithms are *uniform* (but, clearly, not *anonymous*). The reason is that their description makes no assumption of the *community size n*, thus their functionality remains identical for all complete communication graphs. That's why the set of ids U is infinite. The suspicious reader would notice that if we do not impose further restrictions on the model then the agents can use their d slots to store arbitrary amounts of information (by exploiting the fact that U is defined to be infinite), which is artificial. To avoid this, we impose a *local knowledge* constraint, according to which agents can only store ids that they have learned from other agents via interactions with them. To formalize this, let $l(q)$ denote the set of different ids appearing in the list of ids of state q. If $\delta(q_1,q_2) = (q'_1,q'_2)$ and $id \in l(q'_1) \cup l(q'_2)$ then $id \in l(q_1) \cup l(q_2)$ (in words, no new ids appear in the outcome of an interaction).

Additionally, an *operational* constraint is imposed that allows no other operations except for comparisons to be performed on ids by the agents. This constraint is only imposed to keep the model minimal, because it turns out that, even in the presence of this constraint, the model is surprisingly strong (computationally). Intuitively, if $((b_1,\ldots),(b_2,\ldots)) \rightarrow ((b'_1,\ldots),(b'_2,\ldots))$ is a transition in δ, then any transition with precisely the same basic states in which the ids of the lhs are replaced by ids that preserve the order (which, according to the local knowledge constraint, implies that also the ids in the rhs will preserve the order) also belongs to δ. Since this may be a little subtle, another way to think of it is the following. All interactions that do not differ w.r.t. the basic states of the agents and whose lists of ids contain ids that preserve the order, provide the agents with the same new pair of basic states and with new lists of ids that do not different w.r.t. the order of ids.

To make this precise, let $\delta(q_1,q_2) = (q'_1,q'_2)$. Moreover, let $id_1 < id_2 < \ldots < id_k$ be all ids in $l(q_1) \cup l(q_2) \cup l(q'_1) \cup l(q'_2)$ and let $id'_1 < id'_2 < \ldots < id'_k$ be ids. If $\rho(q)$ is the state obtained from q by replacing all occurences of each id id_i by id'_i, then we require that $\delta(\rho(q_1),\rho(q_2)) = (\rho(q'_1),\rho(q'_2))$ also holds.

Example 6. Assume that $\delta((b_1,1,2,\perp),(b_2,7,\perp,\perp)) = ((b'_1,1,7,\perp),(b'_2,2,2,1))$. Then it holds that $\delta((b_1,2,5,\perp),(b_2,8,\perp,\perp)) = ((b'_1,2,8,\perp),(b'_2,5,5,2))$. The reason is that $1 < 2 < 7$ and $2 < 5 < 8$ and we have replaced 1 by 2, 2 by 5, and 7 by 8, thus, preserving the order of ids. Generally, $\delta((b_1,id_1,id_2,\perp),(b_2,id_3,\perp,\perp))$ $= ((b'_1,id_1,id_3,\perp),(b'_2,id_2,id_2,id_1))$ must hold for all $id_1,id_2,id_3 \in U$, where $id_1 < id_2 < id_3$, for the same reason. $\square$

Exercise 13. Consider a transition function δ and let $\delta(q_1,q_2) = (q'_1,q'_2)$ be any transition. Let b_q denote the basic state of state q, and $id_{q,j}$ the jth id in the id list of q. δ is defined as follows. If $b_{q_1} = b_{q_2}$ then nothing happens. If $b_{q_1} \neq b_{q_2}$ then

- If $id_{q_1,j} > id_{q_2,j}$ and $id_{q_1,j}, id_{q_2,j} \neq \perp$ for some $j \in \{2,\ldots,d\}$, then $id_{q'_2,i} = \perp$ and $id_{q'_1,i} = id_{q_2,i}$ for all $i \in \{2,\ldots,d\}$, and $b_{q'_1} = b_{q'_2} = b_{q_1}$.
- Else $id_{q'_1,i} = \perp$ and $id_{q'_2,i} = id_{q_1,i}$ for all $i \in \{2,\ldots,d\}$, and $b_{q'_1} = b_{q'_2} = b_{q_2}$.

Does δ satisfy the local knowledge and operational constraints? Support your answer with a formal proof.

5.2 Computational Power

The community protocol model turns out to be extremely strong in terms of its computational power. In fact it turns out, that any symmetric predicate $p : X^{\geq 2} \to \{0,1\}$ is (stably) computable by the basic community protocol model if and only if it belongs to $NSPACE(n \log n)$, where, as usual, n denotes the community size. The reason that we consider symmetric predicates is that the identifiers of the model cannot be used to order the input symbols, thus an algorithm's functionality in the basic model has to be identical for any permutation of the inputs w.r.t. to the agents' ordering.

Definition 12. Let CP denote the class of all symmetric predicates p that are stably computable by the basic community protocol model.

First of all, we prove that any stably computable symmetric predicate p is in $NSPACE(n \log n)$.

Theorem 5 ([29]). *CP is a subset of* $NSPACE(n \log n)$.

Proof. We will construct a nondeterministic TM N that decides the language $L_p = \{x \in X^{\geq 2} \mid p(x) = 1\}$ (the *support* of p) using at most $NPSACE(n \log n) = NPSACE(|<x>| \log |<x>|)$ cells on any branch of its computation. The reason that the latter equality holds is that the input of p consists of n input symbols, picked from the set X whose cardinality is independent of n. This means that for any input x to the machine N (any element of L_p) it holds that $|<x>| = \mathcal{O}(n)$, where n is the community size.

First of all, we make the following natural assumption: n agents have w.l.o.g. the unique ids $1, 2, \ldots, n$. This implies that each id occupies $\mathcal{O}(\log n)$ cells in a TM. Moreover, there are d id slots in an agent's state, and since d is independent of n again $\mathcal{O}(\log n)$ cells suffice to store the list of ids of any state. The cardinality of B is also independent of n, thus we conclude that $\mathcal{O}(\log n)$ cells suffice to store any state of Q. A configuration is simply a vector consisting of n states, thus a configuration will occupy $\mathcal{O}(n \log n)$ cells of memory storage.

To accept input x, N must verify two conditions: That there exists a configuration C reachable from $I(x)$ (that here denotes the initial configuration corresponding to x), in which all basic states output $p(x)$, and that there is no configuration C' reachable from C, in which some basic state does not output $p(x)$.

The first condition is verified by guessing and checking a sequence of configurations, starting from $I(x)$ and reaching such a C. N guesses a C_{i+1} each time, verifies that $C_i \to C_{i+1}$ (begins from $C_0 = I(x)$, i.e. $i = 0$) and, if so, replaces C_i by C_{i+1}, otherwise drops this C_{i+1}. The second condition is the complement of a similar reachability problem. But $NSPACE$ is closed under complement for all space functions $\geq \log n$ (Immerman-Szelepcsényi theorem). Thus, by taking into account that only one configuration is kept at any step of any branch and that the size of any configuration is $\mathcal{O}(n \log n)$, we conclude that N decides L_p in $\mathcal{O}(n \log n)$ space. □

A Schönhage's *Storage Modification Machine* (SMM) is a kind of pointer machine (not a distributed system). Its memory stores a finite directed graph of constant out-degree with a distinguished node called the *center*. The edges of the graph are called *pointers*. The edges out of each node are labeled by distinct *directions* drawn from a finite set Δ. For example, a reasonable implementation of Δ could use all nonnegative integers up to the maximum out-degree in the graph minus one. Any string $x \in \Delta^*$ can be used as a reference to the node that is reached if we begin from the center and follow the pointers whose labels are indicated by the sequence of symbols in x. We denote the node indicated by $x \in \Delta^*$ by $p^*(x)$. The basic operations of an *SMM* allow the machine to create nodes, modify pointers and follow paths of pointers. We now formalize the above description.

Definition 13. A *Nondeterministic Storage Modification Machine* (NSMM) is a 3-tuple (Σ, Δ, P), where Σ, and Δ are both finite sets and

1. Σ is the *input alphabet*,
2. Δ is the set of *distinct directions*, and
3. P is the *program*, which is a finite list of instructions.

Inputs to the SMM are finite strings from Σ^*. Programs may use instructions of the following types:

- *new*: creates a node, makes it the center, and sets all its outgoing pointers to the old center.
- *recenter* x, where $x \in \Delta^+$: changes the center of the graph to $p^*(x)$.
- *set* $x\delta$ *to* y, where $x, y \in \Delta^*$ and $\delta \in \Delta$: changes the pointer of node $p^*(x)$ that is labeled by δ to point to node $p^*(y)$.
- *if* $x = y$ *then goto* l, where $x, y \in \Delta^*$: jumps to (program) line l if $p^*(x) = p^*(y)$.
- *input* $l_1, \dots, l_r$, where $l_1, \dots, l_r$ are (program) line numbers: consumes the next input symbol (if there is one) and jumps to line l_i if that symbol is σ_i.
- *output* o, where $o \in \{0, 1\}$: causes the machine to halt and output o.
- *choose* l_0, l_1, where l_0 and l_1 are line numbers: causes the machine to transfer control either to line l_0 or to line l_1 nondeterministically.

When a node becomes unreachable from the center, it can be dropped from the graph, since it plays no further role in the computation. Space complexity is measured by the maximum number of (reachable) nodes present at any step of any branch of the machine's nondeterministic computation.

It can be proved that any language decided by a nondeterministic Turing Machine using $\mathcal{O}(S \log S)$ space can be decided by an NSMM using S nodes. Thus, to prove that all symmetric predicates in $NSPACE(n \log n)$ also belong to CP it suffices to show that there exists a community protocol that simulates an NSMM that uses $\mathcal{O}(n)$ nodes. The latter can be shown but, unfortunately, the construction is quite involved, so we skip it. Now by taking into account Theorem 5 we get the following exact characterization.

Theorem 6 ([29]). *CP is equal to the class of all symmetric predicates in* $NSPACE($ $n \log n)$.

6 Logarithmic-Space Machines

In this section, we study another recently proposed model, called the *PALOMA* model [14]. In fact, it is a model of PAssively mobile MAchines (that we keep calling agents) equipped with two-way communication and each having a memory whose size is LOgarithmic in the population size n.

The reason for studying such an extension is that having *logarithmic communicating machines* seems to be more natural than communicating automata of constant memory. First of all, the *communicating machines* assumption is perfectly consistent with current technology (cellphones, iPods, PDAs, and so on). Moreover, *logarithmic* is, in fact, *extremely small*. For a convincing example, it suffices to mention that for a population consisting of 2^{266} agents, which is a number greater than the number of atoms in the observable universe, we only require each agent to have 266 cells of memory (while small-sized flash memory cards nowadays exceed 16GB of storage capacity)! Interestingly, it turns out that the agents, by assigning unique ids to themselves, are able to get organized into a distributed nondeterministic TM that makes full use of the agents' memories! The TM draws its nondeterminism by the nondeterminism inherent in the interaction pattern. It is here like the nameless multitude can turn itself into a well organized community.

Definition 14. A *PALOMA* protocol $\mathscr{A}$ is a 7-tuple $(\Sigma, X, \Gamma, Q, \delta, \gamma, q_0)$ where Σ, X, Γ and Q are all finite sets and

1. Σ is the *input alphabet*, where $\#, \sqcup \notin \Sigma$,
2. $X \subseteq \Sigma^*$ is the *set of input strings*,
3. Γ is the *tape alphabet*, where $\#, \sqcup \in \Gamma$ and $\Sigma \subset \Gamma$,
4. Q is the set of *states*,
5. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\} \times \{0, 1\}$ is the *internal transition function*,
6. $\gamma : Q \times Q \to Q \times Q$ is the *external transition function* (or *interaction transition function*), and
7. $q_0 \in Q$ is the *initial state*.

Each agent is equipped with the following:

- A *sensor* in order to sense its environment and receive a piece of the input (which is an input string from X).
- A *tape* (memory) consisting of $\mathscr{O}(\log n)$ cells. The tape is partitioned into three parts each consisting of $\mathscr{O}(\log n)$ cells: the leftmost part is the *working tape*, the middle part is the *output tape*, and the rightmost part is the *message tape* (we call the parts "tapes" because such a partition is equivalent to a 3-tape machine). The last cell of each part contains permanently the symbol # (we assume that the machine never alters it); it is the symbol used to separate the three tapes and to mark the end of the overall tape.
- A *control unit* that contains the state of the agent and applies the transition functions.
- A *head* that reads from and writes to the cells and can move one step at a time, either to the left or to the right.

- A binary *working flag* either set to 1 meaning that the agent is *working* internally or to 0 meaning that the agent is *ready* for interaction.

Initially, all agents are in state q_0 and all their cells contain the *blank symbol* $\sqcup$ except for the last cell of the working, output, and message tapes that contain the *separator* #. We assume that all agents concurrently receive their sensed input (different agents may sense different data) as a response to a global start signal. The input is a string from X and after reception (or, alternatively, during reception, in an online fashion) it is written symbol by symbol on their working tape beginning from the leftmost cell. During this process the working flag is set to 1 and remains to 1 when this process ends (the agent may set it to 0 in future steps).

When its working flag is set to 1 we can think of the agent working as a usual Turing Machine (but it additionally writes the working flag). In particular, whenever the working flag is set to 1 the internal transition function δ is applied, the control unit reads the symbol under the head and its own state and updates its state and the symbol under the head, moves the head one step to the left or to the right and sets the working flag to 0 or 1, according to the internal transition function.

We assume that the set of states Q and the tape alphabet Γ, are both sets whose size is fixed and independent of the population size (i.e. $|Q| = |\Gamma| = \mathscr{O}(1)$), thus, there is, clearly, enough room in the memory of an agent to store both the internal and the external transition functions.

Again here, a *fair adversary scheduler* selects ordered pairs of agents to interact. Assume now that two agents u and υ are about to interact with u being the *initiator* of the interaction and υ being the *responder*. Let $f : V \to \{0,1\}$ be a function returning the current value of each agent's working flag. If at least one of $f(u)$ and $f(\upsilon)$ is equal to 1, then nothing happens, because at least one agent is still working internally. Otherwise ($f(u) = f(\upsilon) = 0$), both agents are ready and an *interaction* is established. In the latter case, the external transition function γ is applied, the states of the agents are updated accordingly, the message of the initiator is copied to the message tape of the responder (replacing its contents) and vice versa (the real mechanism would require that each one receives the other's message and then copies it to its memory, because instant replacement would make them lose their own message, but this can be easily implemented with $\mathscr{O}(\log n)$ extra cells of memory, so it is not an issue), and finally the working flags of both agents are again set to 1.

Since each agent is a TM (of logarithmic memory), we use the notion of a configuration to capture its "state". An *agent configuration* is a quadruple (q,l,r,f), where $q \in Q$, $l,r \in \Gamma^{\mathscr{O}(\log n)} = \{s \in \Gamma^* \mid |s| = \mathscr{O}(\log n)\}$, and $f \in \{0,1\}$. q is the state of the control unit, l is the string to the left of the head (including the symbol scanned), r is the string to the right of the head, and f is the working flag indicating whether the agent is ready to interact ($f = 0$) or carrying out some internal computation ($f = 1$). Let $\mathscr{B}$ be the set of all agent configurations. Given two agent configurations $A,A' \in \mathscr{B}$, we say that A *yields* A' if A' follows A by a single application of δ.

A *population configuration* is a mapping $C : V \to \mathscr{B}$, specifying the agent configuration of each agent in the population. Let C, C' be population configurations and let $u \in V$. We say that C *yields* C' via *agent transition* u, denoted $C \xrightarrow{u} C'$, if $C(u)$ yields $C'(u)$ and $C'(w) = C(w)$, $\forall w \in V - \{u\}$.

Let $q(A)$ denote the state of an agent configuration A, $l(A)$ its string to the left of the head including the symbol under the head, $r(A)$ its string to the right of the head, and $f(A)$ its working flag. Given two population configurations C and C', we say that C *yields* C' via *encounter* $e = (u, \upsilon) \in E$, denoted $C \xrightarrow{e} C'$, if one of the following two cases holds:

Case 1:

- $f(C(u)) = f(C(\upsilon)) = 0$ which guarantees that both agents u and υ are ready for interaction under the population configuration C.
- $r(C(u))$ and $r(C(\upsilon))$ are precisely the message strings of u and υ, respectively (this is a simplifying assumption stating that when an agent is ready to interact its head is over the last # symbol, just before the message tape),
- $C'(u) = (\gamma_1(q(C(u)), q(C(\upsilon))), l(C(u)), r(C(\upsilon)), 1)$,
- $C'(\upsilon) = (\gamma_2(q(C(u)), q(C(\upsilon))), l(C(\upsilon)), r(C(u)), 1)$, and
- $C'(w) = C(w)$, $\forall w \in V - \{u, \upsilon\}$.

Case 2:

- $f(C(u)) = 1$ or $f(C(\upsilon)) = 1$, which means that at least one agent between u and υ is working internally under the population configuration C, and
- $C'(w) = C(w)$, $\forall w \in V$. In this case no effective interaction takes place, thus the population configuration remains the same.

Generally, we say that C *yields* (or *can go in one step to*) C', and write $C \rightarrow C'$, if $C \xrightarrow{e} C'$ for some $e \in E$ (via encounter) or $C \xrightarrow{u} C'$ for some $u \in V$ (via agent transition), or both. We say that C' is *reachable* from C, and write $C \xrightarrow{*} C'$ if there is a sequence of population configurations $C = C_0, C_1, \ldots, C_t = C'$ such that $C_i \rightarrow C_{i+1}$ holds for all $i \in \{0, 1, \ldots, t-1\}$. An *execution* is a finite or infinite sequence of population configurations $C_0, C_1 \ldots$, so that $C_i \rightarrow C_{i+1}$. An infinite execution is *fair* if for all population configurations C, C' such that $C \rightarrow C'$, if C appears infinitely often then so does C'. A *computation* is an infinite fair execution.

Note that the PALOMA model *preserves uniformity*, because X, Γ and Q are all finite sets whose cardinality is independent of the population size. Thus, protocol descriptions have also no dependence on the population size. Moreover, PALOMA protocols are *anonymous*, since initially all agents are identical and have no unique identifiers.

Example 7. We present now a PALOMA protocol that stably computes the predicate $(N_c = N_a \cdot N_b)$ (on the complete communication graph of n nodes) that is, all agents eventually decide whether the number of cs in the input assignment is the product of the number of as and the number of bs. We give a high-level description of the protocol.

Initially, all agents have one of a, b and c written on the first cell of their working memory (according to their sensed value). That is, the set of input strings is $X =$

$\Sigma = \{a, b, c\}$. Each agent that receives input a goes to state a and becomes ready for interaction (sets its working flag to 0). Agents in state a and b both do nothing when interacting with agents in state a and agents in state b. An agent in c initially creates in its working memory three binary counters, the a-counter that counts the number of as, the b-counter, and the c-counter, initializes the a and b counters to 0, the c-counter to 1, and becomes ready. When an agent in state a interacts with an agent in state c, a becomes $\bar{a}$ to indicate that the agent is now sleeping, and c does the following (in fact, we assume that c goes to a special state c_a in which it knows that it has seen an a, and that all the following are done internally, after the interaction; finally the agent restores its state to c and becomes again ready for interaction): it increases its a-counter by one (in binary), multiplies its a and b counters, which can be done in binary in logarithmic space (binary multiplication is in *LOGSPACE*), compares the result with the c-counter, copies the result of the comparison to its output tape, that is, 1 if they are equal and 0 otherwise, and finally it copies the comparison result and its three counters to the message tape and becomes ready for interaction. Similar things happen when a b meets a c (interchange the roles of a and b in the above discussion). When a c meets a c, the responder becomes $\bar{c}$ and copies to its output tape the output bit contained in the initiator's message. The initiator remains to c, adds the a-counter contained in the responder's message to its a-counter, the b and c counters of the message to its b and c counters, respectively, multiplies again the updated a and b counters, compares the result to its updated c counter, stores the comparison result to its output and message tapes, copies its counters to its message tape and becomes ready again. When a $\bar{a}$, $\bar{b}$ or $\bar{c}$ meets a c they only copy to their output tape the output bit contained in c's message and become ready again (eg $\bar{a}$ remains $\bar{a}$), while c does nothing.

Note that the number of cs is at most n which means that the c-counter will become at most $\lceil \log n \rceil$ bits long, and the same holds for the a and b counters, so there is enough room in the tape of an agent to store them.

Given a fair execution, eventually only one agent in state c will remain, its a-counter will contain the total number of as, its b-counter the total number of bs, and its c-counter the total number of cs. By executing the multiplication of the a and b counters and comparing the result to its c-counter it will correctly determine whether $(N_c = N_a \cdot N_b)$ holds and it will store the correct result (0 or 1) to its output and message tapes. At that point all other agents will be in one of the states $\bar{a}$, $\bar{b}$, and $\bar{c}$. All these, again due to fairness, will eventually meet the unique agent in state c and copy its correct output bit (which they will find in the message they get from c) to their output tapes. Thus, eventually all agents will output the correct value of the predicate. □

Exercise 14. Prove that the basic PALOMA model is strictly stronger than the basic population protocol model, without exploiting the predicate $(N_c = N_a \cdot N_b)$.
Hint: Find another non-semilinear predicate that is (stably) computable by the basic PALOMA model. Do not forget to show first that the basic PALOMA model is at least as strong as the basic population protocol model.

Definition 15. Let *PLM* denote the class of all symmetric predicates p that are stably computable by the basic PALOMA model.

Then, one can prove the following exact characterization for PLM [14]. Unfortunately, this proof is also quite involved and due to space restrictions we skip it.

Theorem 7. *PLM is equal to the class of all symmetric predicates in* $NSPACE(n \log n)$.

7 Algorithmic Verification of Population Protocols

In order to apply our protocols to real-critical application scenarios, some form of computer-aided *verification* is necessary. Even if a protocol is followed by a formal proof of correctness it would be safer to verify its code before loading it to the real sensor nodes.

It seems that the easiest (but not easy) place to start the investigation of verification is the basic population protocol model. In this model we can exploit the fact that symmetry allows a configuration to be safely represented as a $|Q|$-vector of nonnegative integers. This section, based on [13], will reveal the inherent hardness of algorithmic verification of basic population protocols but will also present a promising algorithmic solution.

Subsection 7.1 provides all necessary definitions. Subsection 7.2 deals with the hardness of algorithmic verification of basic population protocols; the general problem and many of its special cases are proved to be hard. Subsection 7.3 studies an efficiently solvable, but though almost trivial, special case. Finally, Subsection 7.4 presents some non-complete and one complete algorithmic solution.

7.1 Necessary Definitions

7.1.1 Population Protocols

We begin by revising all relevant definitions concerning the basic population protocol model, most of which are now presented in an alternative manner, because throughout this section we will exploit the fact that symmetry allows a configuration to be represented as a $|Q|$-vector of nonnegative integers, and there is no need now to use a function for this purpose.

In this section, the transition function δ is also treated as a relation $\Delta \subseteq Q^4$, defined as $(q_i, q_j, q_l, q_t) \in \Delta$ iff $\delta(q_i, q_j) = (q_l, q_t)$. We assume that the communication graph is a complete digraph, without self-loops and multiple edges (that is, we deal with the basic model). We denote by G^k the complete communication graph of k nodes.

Let now $k \equiv |V|$ denote the *population size*. An *input assignment* x is a mapping from $V = [k]$ to X (where $[l]$, for $l \in \mathbb{Z}_{\geq 1}$, denotes the set $\{1, \ldots, l\}$), assigning an input symbol to each agent of the population. As already mentioned in Subsection 2.2, since the communication graph is complete, due to symmetry, we can, equivalently, think of an input assignment as a $|X|$-vector of integers $x = (x_i)_{i\in[|X|]}$, where, for all i, x_i is nonnegative and equal to the number of agents that receive the symbol $\sigma_i \in X$, assuming an ordering on the input symbols. We denote by $\mathscr{X}$ the set of all possible input assignments. Note that for all $x \in \mathscr{X}$ it holds that $\sum_{i=1}^{|X|} x_i = k$.

A state $q \in Q$ is called *initial* if $I(\sigma) = q$ for some $\sigma \in X$. A *configuration* c is a mapping from $[k]$ to Q, so, again, it is a $|Q|$-vector of nonnegative integers $c = (c_i)_{i\in[|Q|]}$ such that $\sum_{i=1}^{|Q|} c_i = k$ holds. Each input assignment corresponds to an *initial configuration* which is indicated by the input function I. In particular, input assignment x corresponds to the initial configuration $c(x) = (c_i(x))_{i\in[|Q|]}$, where $c_i(x)$ is equal to the number of agents that get some input symbols σ_j for which $I(\sigma_j) = q_i$ (q_i is the ith state in Q if we assume the existence of an ordering on the set of states Q). More formally, $c_i(x) = \sum_{j:I(\sigma_j)=q_i} x_j$ for all $i \in [|Q|]$. By extending I to a mapping from input assignments to configurations we can write $I(x) = c$ to denote that c is the initial configuration corresponding to input assignment x. Let $\mathscr{C} = \{(c_i)_{i\in[|Q|]} \mid c_i \in \mathbb{Z}^+ \text{ and } \sum_{i=1}^{|Q|} c_i = k\}$ denote the set of all possible configurations given the population protocol $\mathscr{A}$ and G^k. Moreover, let $C_I = \{c \in \mathscr{C} \mid I(x) = c \text{ for some } x \in \mathscr{X}\}$ denote the set of all possible initial configurations. Any $r \in \Delta$ has four components which are elements from Q and we denote by r_i, where $i \in [4]$, the i-th component (i.e. state) of r. $r \in Q^4$ belongs to Δ iff $\delta(r_1, r_2) = (r_3, r_4)$. We say that a configuration c can go in one step to c' via transition $r \in \Delta$, and write $c \overset{r}{\rightarrow} c'$, if

- $c_i \geq r_{1,2}(i)$, for all $i \in [|Q|]$ for which $q_i \in \{r_1, r_2\}$,
- $c'_i = c_i - r_{1,2}(i) + r_{3,4}(i)$, for all $i \in [|Q|]$ for which $q_i \in \{r_1, r_2, r_3, r_4\}$, and
- $c'_j = c_j$, for all $j \in [|Q|]$ for which $q_j \in Q - \{r_1, r_2, r_3, r_4\}$,

where $r_{l,t}(i)$ denotes the number of times state q_i appears in (r_l, r_t). Moreover, we say that a configuration c can go in one step to a configuration c', and write $c \rightarrow c'$ if $c \overset{r}{\rightarrow} c'$ for some $r \in \Delta$. We say that a configuration c' is reachable from a configuration c, denoted $c \overset{*}{\rightarrow} c'$ if there is a sequence of configurations $c = c^0, c^1, \ldots, c^t = c'$, such that $c^i \rightarrow c^{i+1}$ for all i, $0 \leq i < t$, where c^i denotes here the $(i+1)$th configuration of an execution (and not the ith component of configuration c which is denoted c_i). An *execution* is a finite or infinite sequence of configurations $c^0, c^1, \ldots$, so that $c^i \rightarrow c^{i+1}$. An execution is *fair* if for all configurations c, c' such that $c \rightarrow c'$, if c appears infinitely often then so does c'. A *computation* is an infinite fair execution. A predicate p is said to be *stably computable* by a PP $\mathscr{A}$ if, for any input assignment x, any computation of $\mathscr{A}$ contains an *output stable configuration* in which all agents output $p(x)$. a configuration c is called *output stable* if $O(c) = O(c')$, for all c' reachable from c (where O, here, is an extended version of the output function from configurations to output assignments in Y^k). We denote by $C_F = \{c \in \mathscr{C} \mid c \rightarrow c' \Rightarrow c' = c\}$ the set of all *final configurations*. We can further extend the output function O to a mapping from configurations to $\{-1, 0, 1\}$, defined as

$$O(c) = \begin{cases} 0, \text{ if } O(c(u)) = 0, \text{ for all } u \in V \\ 1, \text{ if } O(c(u)) = 1, \text{ for all } u \in V \\ -1, \text{ if } \exists u, \upsilon \in V \text{ s.t. } O(c(u)) \neq O(c(\upsilon)). \end{cases}$$

It is known [2, 6] that a predicate is stably computable by the PP model iff it can be defined as a first-order logical formula in *Presburger arithmetic*. Let ϕ be such a formula. There exists some PP that stably computes ϕ, thus ϕ constitutes, in fact, the specifications of that protocol. For example, consider the formula $\phi = (N_a \geq 2N_b)$. ϕ partitions the set of all input assignments, $\mathscr{X}$, to those input assignments that satisfy the predicate (that is, the number of as assigned is at least two times the number of bs assigned) and to those that do not. Moreover, ϕ can be further extended to a mapping from C_I to $\{-1,0,1\}$. In this case, ϕ is defined as

$$\phi(c) = \begin{cases} 0, \text{ if } \phi(x) = 0, \text{ for all } x \in I^{-1}(c) \\ 1, \text{ if } \phi(x) = 1, \text{ for all } x \in I^{-1}(c) \\ -1, \text{ if } \exists x, x' \in I^{-1}(c) \text{ s.t. } \phi(x) \neq \phi(x'), \end{cases}$$

where $I^{-1}(c)$ denotes the set of all $x \in \mathscr{X}$ for which $I(x) = c$ holds (the *preimage* of c).

We now define the *transition graph*, which is similar to that defined in [2], except for the fact that it here contains only those configurations that are reachable from some initial configuration in C_I. Specifically, given a population protocol $\mathscr{A}$ and an integer $k \geq 2$ we can define the transition graph of the pair $(\mathscr{A}, k)$ as $G_{\mathscr{A},k} = (C_r, E_r)$, where the node set $C_r = C_I \cup \{c \in \mathscr{C} \mid c' \xrightarrow{*} c \text{ for some } c' \in C_I\}$ of G_r (we use G_r as a shorthand of $G_{\mathscr{A},k}$) is the subset of $\mathscr{C}$ containing all initial configurations and all configurations that are reachable from some initial one, and the edge (or arc) set $E_r = \{(c,c') \mid c,c' \in C_r \text{ and } c \to c'\}$ of G_r contains a directed edge (c,c') for any two (not necessarily distinct) configurations c and c' of C_r for which it holds that c can go in one step to c'. Note that G_r is a directed (weakly) connected graph with possible self-loops. It was shown in [2] that, given a computation Ξ, the configurations that appear infinitely often in Ξ form a *final strongly connected component* of G_r. We denote by S the collection of all strongly connected components of G_r. Note that each $B \in S$ is simply a set of configurations. Moreover, given $B, B' \in S$ we say the B can go in one step to B', and write $B \to B'$, if $c \to c'$ for $c \in B$ and $c' \in B'$. $B \xrightarrow{*} B'$ is defined as in the case of configurations. We denote by $I_S = \{B \in S \mid \text{such that } B \cap C_I \neq \emptyset\}$ those components that contain at least one initial configuration, and by $F_S = \{B \in S \mid \text{such that } B \to B' \Rightarrow B' = B\}$ the final ones. We can now extend ϕ to a mapping from I_S to $\{-1,0,1\}$ defined as

$$\phi(B) = \begin{cases} 0, \text{ if } \phi(c) = 0, \text{ for all } c \in B \cap C_I \\ 1, \text{ if } \phi(c) = 1, \text{ for all } c \in B \cap C_I \\ -1, \text{ if } \exists c, c' \in B \cap C_I \text{ s.t. } \phi(c) \neq \phi(c'), \end{cases}$$

and O to a mapping from F_S to $\{-1,0,1\}$ defined as

$$O(B) = \begin{cases} 0, \text{ if } O(c) = 0, \text{ for all } c \in B \\ 1, \text{ if } O(c) = 1, \text{ for all } c \in B \\ -1, \text{ otherwise.} \end{cases}$$

7.1.2 Problems' Definitions

We begin by defining the most interesting and natural version of the problem of algorithmically verifying basic population protocols. We call it $GBPVER$ ('G' standing for "General', 'B' for "Basic", and 'P' for "Predicate") and its complement $\overline{GBPVER}$ is defined as follows:

Problem 2 (**$\overline{GBPVER}$**)**.** Given a population protocol $\mathscr{A}$ for the basic model whose output alphabet $Y_{\mathscr{A}}$ is binary (i.e. $Y_{\mathscr{A}} = \{0,1\}$) and a first-order logical formula ϕ in Presburger arithmetic representing the specifications of $\mathscr{A}$, determine whether there exists some integer $k \geq 2$ and some legal input assignment x for the complete communication graph of k nodes, G^k, for which not all computations of $\mathscr{A}$ on G^k beginning from the initial configuration corresponding to x stabilize to the correct output w.r.t. ϕ.

A special case of $GBPVER$ is $BPVER$ (its non-general version as revealed by the missing 'G'), and is defined as follows.

Problem 3 (*BPVER*)**.** Given a population protocol $\mathscr{A}$ for the basic model whose output alphabet $Y_{\mathscr{A}}$ is binary (i.e. $Y_{\mathscr{A}} = \{0,1\}$), a first-order logical formula ϕ in Presburger arithmetic representing the specifications of $\mathscr{A}$, and an integer $k \geq 2$ (in binary) determine whether $\mathscr{A}$ conforms to its specifications on G^k.

"Conforms to ϕ" here means that for any legal input assignment x, which is a $|X_{\mathscr{A}}|$-vector with nonnegative integer entries that sum up to k, and any computation beginning from the initial configuration corresponding to x on G^k, the population stabilizes to a configuration in which all agents output the value $\phi(x) \in \{0,1\}$ (that is, it is equivalent to "stably computes", but we now view it from the verification perspective). On the other hand, "does not conform" means that there is at least one computation of $\mathscr{A}$ on G^k which is unstable or the stable output does not agree with $\phi(x)$ - i.e. not all agents output the value $\phi(x)$.

Problem 4 (*BBPVER*)**.** $BBPVER$ (the additional 'B' is from "Binary input alphabet") is $BPVER$ with $\mathscr{A}$'s input alphabet restricted to $\{0,1\}$.

7.2 NP-hardness Results

7.2.1 *BP* Verification

We first show that $BPVER$ is a hard computational problem.

Theorem 8. *BPVER is coNP-hard.*

Proof. We shall present a polynomial-time reduction from $HAMPATH = \{< D,s,t > \mid D$ is a directed graph with a Hamiltonian path from s to t $\}$ to $\overline{BPVER}$. In other words, we will present a procedure that given an instance $< D,s,t >$ of $HAMPATH$ returns in polynomial time an instance $< \mathscr{A},\phi,k >$ of $\overline{BPVER}$, such that $< D,s,t > \in HAMPATH$ iff $< \mathscr{A},\phi,k > \in \overline{BPVER}$. If there is a hamiltonian path from s to t in D we will return a population protocol $\mathscr{A}$ that for some computation on the complete graph of k nodes fails to conform to its specification ϕ, and if there is no such path all computations will conform to ϕ.

We assume that all nodes in $V(D) - \{s,t\}$ are named $q_1,\dots,q_{n-2}$, where n denotes the number of nodes of D (be careful: n does not denote the size of the population, but the number of nodes of the graph D in $HAMPATH$'s instance). We now construct the protocol $\mathscr{A} = (X,Y,Q,I,O,\delta)$. The output alphabet Y is $\{0,1\}$ by definition. The input alphabet X is $E(D) - (\{(\cdot,s)\} \cup \{t,\cdot\})$, that is, consists of all edges of D except for those leading into s and those going out of t. The set of states Q is equal to $X \cup T \cup \{r\}$, where $T = \{(s,q_i,q_j,l) \mid 1 \leq i,j \leq n-2$ and $1 \leq l \leq n-1\}$ and its usefulness will be explained later. r can be thought of as being the "reject" state, since we will define it to be the only state giving the output value 0. Notice that $|Q| = \mathscr{O}(n^3)$. The input function $I : X \to Q$ is defined as $I(x) = x$, for all $x \in X$, and for the output function $O : Q \to \{0,1\}$ we have $O(r) = 0$ and $O(q) = 1$ for all $q \in Q - \{r\}$. That is, all input symbols are mapped to themselves, while all states are mapped to the output value 1, except for r which is the only state giving 0 as output. Thinking of the transition function δ as a transition matrix Δ it is easy to see that Δ is a $|Q| \times |Q|$ matrix whose entries are elements from $Q \times Q$. Each entry $\Delta_{q,q'}$ corresponds to the rhs of a rule $(q,q') \to (z,z')$ in δ. Clearly, Δ consists of $\mathscr{O}(n^6)$ entries, which is again polynomial in n.

We shall postpone for a while the definition of Δ to first define the remaining parameters ϕ and k of $\overline{BPVER}$'s instance. We define formula ϕ to be a trivial first-order Presburger arithmetic logical formula that is always false. For example, in the natural nontrivial case where $X \neq \emptyset$ (that is, D has at least one edge that is not leading into s and not going out of t) we can pick any $x \in X$ and set $\phi = (N_x < 0)$ which, for N_x denoting the number of xs appearing in the input assignment, is obviously always false. It is useful to notice that the only configuration that gives the correct output w.r.t. ϕ is the one in which all agents are in state r. ϕ being always false means that in a correct protocol all computations must stabilize to the all-zero output, and r is the only state giving output 0. On the other hand for $\mathscr{A}$ not to be correct w.r.t. ϕ it suffices to show that there exists some computation in which r cannot appear. Moreover, we set k equal to $n-1$, that is, the communication graph on which $\mathscr{A}$'s correctness has to be checked by the verifier is the complete digraph of $n-1$ nodes (or, equivalently, agents).

To complete the reduction, it remains to construct the transition function δ:

- $(r,\cdot) \to (r,r)$ and $(\cdot,r) \to (r,r)$ (so r is a propagating state, meaning that once it appears it eventually becomes the state of every agent in the population)

- $((q_i,q_j),(q_i,q_j)) \rightarrow (r,r)$ (if two agents get the same edge of D then the protocol rejects)
- $((q_i,q_j),(q_i,q_l)) \rightarrow (r,r)$ (if two agents get edges of D with adjacent tails then the protocol rejects)
- $((q_j,q_i),(q_l,q_i)) \rightarrow (r,r)$ (if two agents get edges of D with adjacent heads then the protocol rejects - it also holds if one of q_j and q_l is s)
- $((q_i,t),(q_j,t) \rightarrow (r,r)$ (the latter also holds for the sink t)
- $((s,\cdots),(s,\cdots)) \rightarrow (r,r)$ (if two agents have both s as the first component of their states then the protocol rejects)
- $((s,q_i),(q_i,q_j)) \rightarrow ((s,q_i,q_j,2),(q_i,q_j))$ (when s meets an agent υ that contains a successor edge it keeps q_j to remember the head of υ's successor edge and releases a counter set to 2 - it counts the number of edges encountered so far on the path trying to reach t from s)
- $((s,q_i,q_j,i),(q_j,q_l)) \rightarrow ((s,q_i,q_l,i+1),(q_j,q_l))$, for $i < n-2$
- $((s,q_i,q_j,i),(q_j,t)) \rightarrow (r,r)$, for $i < n-2$ (the protocol rejects if s is connected to t through a directed path with less than $n-1$ edges)
- All the transitions not appearing above are identity rules (i.e. they do nothing)

Now we prove that the above, obviously polynomial-time, construction is in fact the desired reduction. If D contains some hamiltonian path from s to t, then the $n-1$ edges of that path form a possible input assignment to protocol $\mathscr{A}$ (since its input symbols are the edges and the population consists of $n-1$ agents). When $\mathscr{A}$ gets that input it cannot reject (r cannot appear) for the following reasons:

- no two agents get the same edge of D
- no two agents get edges of D with adjacent tails
- no two agents get edges of D with adjacent heads
- only one $(s,\cdots)$ exists
- s cannot count less than $n-1$ edges from itself to t

So, when $\mathscr{A}$ gets the input alluded to above, it cannot reach state r, thus, it cannot reject, which implies that $\mathscr{A}$ for that input always stabilizes to the wrong output w.r.t. ϕ (which always requires the "reject" output) when runs on the G^{n-1}. So, in this case $< \mathscr{A},\phi,k >$ consists of a protocol $\mathscr{A}$ that, when runs on G^k, where $k = n-1$, for a specific input it does not conform to its specifications as described by ϕ, so clearly it belongs to $\overline{BPVER}$.

For the other direction, if $< \mathscr{A},\phi,k > \in \overline{BPVER}$ then obviously there exists some computation of $\mathscr{A}$ on the complete graph of $k = n-1$ nodes in which r does not appear at all (if it had appeared once then, due to fairness, the population would have stabilized to the all-r configuration, resulting to a computation conforming to ϕ). It is helpful to keep in mind that most arguments here hold because of the fairness condition. Since r cannot appear, every agent (of the $n-1$ in total) must have been assigned a different edge of D. Moreover, no two of them contain edges with common tails or common heads in D. Note that there is only one agent with state $(s,\cdots)$ because if there were two of them they would have rejected when interacted with each other, and if no $(s,\cdots)$ appeared then two agents would have edges with common tails because there are $n-1$ edges for $n-2$ candidate initiating points (we have not allowed t to be an initiating point) and the pigeonhole principle applies (and by symmetric arguments only one with state $(\cdots,t)$). So, in the induced graph formed by the edges that have been assigned to the agents, s has outdegree 1 and indegree 0, t has indegree 1 and outdegree 0 and all remaining nodes have indegree at most 1 and outdegree at most 1. This implies that all nodes except for s and t must have indegree equal to 1 and outdegree equal to 1. If, for example, some node had indegree 0, then the total indegree could not have been $n-1$ because $n-3$ other nodes have indegree at most 1, t has indegree 1, and s has 0 (the same holds for outdegrees). Additionally, there is some path initiating from s and ending to t. This holds because the path initiating from s (s has outdegree 1) cannot fold upon itself (this would result in a node with indegree greater than 1) and cannot end to any other node different from t because this would result to some node other than t with outdegree equal to 0. Finally, that path has at least $n-1$ edges (in fact, precisely $n-1$ edges), since if it had less the protocol would have rejected. Thus, it must be clear after the above discussion that in this case there must have been a hamiltonian path from s to t in D, implying that $< D,s,t > \in HAMPATH$. $\square$

Note that in the above reduction the communication graph has only $\mathscr{O}(n)$ nodes while the protocol has size $\mathscr{O}(n^6)$. Although this is not the usual case, it is not forbidden because this concerns only the correctness of the protocol on this specific complete graph. The protocol remains independent of the population size; it will still count up to $n-1$ while the population can have arbitrarily large size (another way to think of this is that in the protocol description the population size is not a parameter). The protocol may be wrong or correct for other combinations of specifications and communication graphs but we do not care here. However, it is worth considering the following question: “Can we also prove that the special case of *BPVER* in which the protocol has always size less than the size of the communication graph (which is the natural scenario) is coNP-hard?” Unfortunately, the answer to this question is that we do not know yet.

7.2.2 *BBP* Verification

We now deal with the hardness of *BBPVER* (here, additionally, we have a binary input alphabet).

Theorem 9. *BBPVER is coNP-hard.*

Proof. The reduction is again from $HAMPATH$ to $\overline{BBPVER}$. Let again $< D,s,t >$ be the instance of $HAMPATH$. X is equal to $\{0,1\}$ as is required by definition and so is Y. Q is again equal to $(E(D)-(\{(\cdot,s)\}\cup\{t,\cdot\}))\cup T\cup\{q_0,t',r\}$, where $T=\{(s,q_i,q_j,l)\mid 1\le i,j\le n-2 \text{ and } 1\le l\le n-1\}$. The input function I is defined as $I(0)=(s,f^+(s))$, where $f^+(s)$ is the first (smallest) out-neighbor of s according to the lexicographic order of $V(D)$ (if some node u has no neighbors we assume that $f^+(u)=u$), and $I(1)=q_0$ (recall that the names that we use for nodes are $s,t,q_1,\ldots,q_{n-2}$ so q_0 is just a special initial state). The output function O again maps all states in $Q - \{r\}$ to 1 and r to 0. ϕ is an always false predicate and k is set to $n-1$, where $n = |V(G)|$.

We now define the transition function.

- $(r,\cdot) \rightarrow (r,r)$ and $(\cdot,r) \rightarrow (r,r)$ (so r is a propagating state, meaning that once it appears it eventually becomes the state of every agent in the population)
- $((q_i,q_j),(q_i,q_j)) \rightarrow (r,r)$ (if two agents have obtained the same edge of D then the protocol rejects)
- $((q_i,q_j),(q_i,q_l)) \rightarrow (r,r)$ (if two agents have obtained edges of D with adjacent tails then the protocol rejects)
- $((q_j,q_i),(q_l,q_i)) \rightarrow (r,r)$ (if two agents have obtained edges of D with adjacent heads then the protocol rejects - it also holds if one of q_j and q_l is s)
- $(q_0,q_0) \rightarrow (r,r)$
- $((s,\ldots),(s,\ldots)) \rightarrow (r,r)$
- $(\cdot,t),(\cdot,t) \rightarrow (r,r)$
- $((s,q_i),q_0) \rightarrow ((s,q_i),(f^-(t),t))$ (where $f^-(t)$ denotes the first (smallest) in-neighbor of t according to the lexicographic order; we can w.l.o.g. assume that t has at least one incoming edge)
- $((s,q_i),(q_j,t)) \rightarrow ((s,h_s^+(q_i)),(q_j,t))$ (where $h_s^+(q_i)$ denotes the lexicographically smallest out-neighbor of s that is lexicographically greater than q_i (that is, the next one); note that the lexicographically greatest is matched to the lexicographically smallest in a cyclic fashion)
- $((q_j,t),(s,q_i)) \rightarrow ((h_t^-(q_j),t),(s,q_i))$ (where $h_t^-(q_j)$ denotes the lexicographically smallest in-neighbor of t that is lexicographically greater than q_j)
- $((q_i,q_j),(q_l,t)) \rightarrow ((),())$
- $(q_0,(s,q_i)) \rightarrow ((q_i,f^+(q_i)),(s,q_i))$, if $f^+(q_i) \neq q_i$, and (r,r), otherwise (if $f^+(q_i) = q_i$ then q_i has no outgoing neighbors and the protocol rejects; f^+ does not take into account the edges leading into s and t)
- $((q_i,q_j),(q_l,t)) \rightarrow ((q_i,h_{q_i}^+(q_j)),(q_l,t))$
- $((q_l,t),(q_i,q_j)) \rightarrow ((q_l,t),(q_j,f^+(q_j)))$, if $f^+(q_j) \neq q_j$, and (r,r), otherwise
- $((s,q_i),(q_i,q_j)) \rightarrow ((s,q_i,q_j,2),(q_i,q_j))$ (when s meets an agent υ that contains a successor edge it keeps q_j to remember the head of υ's successor edge and releases a counter set to 2 - it counts the number of edges encountered so far on the path trying to reach t from s)
- $((s,q_i,q_j,i),(q_j,q_l)) \rightarrow ((s,q_i,q_l,i+1),(q_j,q_l))$, for $i < n-2$
- $((s,q_i,q_j,i),(q_j,t)) \rightarrow (r,r)$, for $i < n-2$ (the protocol rejects if s is connected to t through a directed path with less than $n-1$ edges)
- $((s,q_i,q_j,n-2),(q_j,t)) \rightarrow ((s,q_i,q_j,n-1),t')$
- t' and q_0 reject any t' and $(\cdot,t)$ that they encounter
- All the transitions not appearing above are identity rules (i.e. they do nothing)

Given a hamiltonian path $s,u_1,\ldots,u_{n-2},t$ of D we present an erroneous computation of $\mathscr{A}$ on the complete digraph of $k = n-1$ nodes w.r.t. ϕ (that is, a computation in which state r does not appear). A possible input assignment is the 2-vector $(1,n-2)$ in which one agent gets input 0 and $(n-2)$ agents get input 1. So, the initial configuration corresponding to this input has one agent in $(s,f^+(s))$ and all the other agents in q_0. The agent in $(s, f^+(s))$ now interacts as the initiator with some agent and $(f^-(t), t)$ appears. Now we have one agent in $(s, f^+(s))$, one in $(f^-(t), t)$, and all the remaining in q_0. If $f^+(s)$ is not equal to u_1 (the second node in the hamiltonian path) we assume that $(s, f^+(s))$ is the initiator of as many interactions with $(f^-(t), t)$ as needed to make $(s, f^+(s))$ go to (s, u_1). Similarly, with $(f^-(t), t)$ being the initiator we make it interact a sufficient number of times with (s, u_1) so that it becomes (u_{n-2}, t). Now one agent contains (s, u_1), which is the first edge of the hamiltonian path, one agent contains (u_{n-2}, t) which is the last edge of the hamiltonian path, and all remaining agents are in q_0. Now interaction $(q_0, (s, u_1))$ takes place and the result is $((u_1, f^+(u_1)), (s, u_1))$, where $f^+(u_1)$ is the lexicographically first out-neighbor of u_1, which is possibly not u_2. If it is not, then we let the agent which is in $(u_1, f^+(u_1))$ repeatedly interact as the initiator with (u_{n-2}, t), until its state becomes (u_1, u_2) (e.g. during the first interaction $(u_1, f^+(u_1))$ becomes $(u_1, h^+_{u_1}(f^+(u_1)))$, where $h^+_{u_1}(u)$ denotes the out-neighbor of u_1 lexicographically following u). As soon as this happens, (s, u_1) interacts with another agent in q_0 which again updates its state to $(u_1, f^+(u_1))$. Again $(u_1, f^+(u_1))$ interacts as the initiator with (u_{n-2}, t) as many times as needed to make its state (u_1, u_2) and then it interacts once as the responder with (u_{n-2}, t) to change its state to $(u_2, f^+(u_2))$. Even if $f^+(u_2)$ does not happen to be u_3 we can force it to be by subsequent interactions with (u_{n-2}, t) (with the latter now being the responder). In this manner we can easily make each agent in the population contain a different edge of the hamiltonian path. Moreover, notice that we have not allowed any interaction that leads to failure (i.e. that makes state r appear) happen. Now (s, u_1) meets (u_1, u_2) and the former becomes $(s, u_1, u_2, 2)$. Then it meets (u_2, u_3) and becomes $(s, u_1, u_3, 3)$, and so on, and, finally, when it has become $(s, u_1, u_{n-2}, n-2)$ it meets (u_{n-2}, t) and after that interaction the former becomes $(s, u_1, u_{n-2}, n-1)$ and the latter t'. It is easy now to observe that from this point on there is no possible interaction that could make r appear and thus we have just presented an erroneous computation (all agents forever output the value 1, but ϕ requires that any computation stabilizes to the all-zero output). The convincing argument that it is a computation (i.e. a fair execution) is that we keep the execution unfair only for a finite number of steps, which does not violate the fairness condition.

For the inverse, let us assume that there exists some computation of $\mathscr{A}$ on the complete digraph of $k = n-1$ nodes in which r never appears. Clearly, only one $(s, \ldots)$ ever appears (if there were two of them they would eventually meet and reject, because once s appears as the first component of some state it cannot be eliminated, and if there was none the population would solely consist of q_0, which would eventually meet and reject). Similarly, only one t ever appears (since, once they appear, even if they ever become t', they cannot be eliminated and will eventually meet each other and reject). Note also that after a finite number of steps all agents must have obtained some edge (if some agent remains forever in q_0 then it eventually meets $(\cdot, t)$ or t' and rejects). Moreover, t' must have appeared for the following reason: if not, then $(\cdot, t)$ would forever change the agents' edges, so due to fairness two agents would, in a finite number of steps, obtain the same edge, interact with each other, and reject. But since the protocol cannot reject (in the computation under consideration), s must have counted to $n-2$ before meeting t, and by repeating some of the arguments used in the proof of Theorem 8 one can again show that any node in the induced graph (constructed by the edges contained in the agents) has indegree equal to 1 and outdegree equal to 1, which implies that there must exist some hamiltonian path from s to t in D. Clearly, this completes the proof. $\square$

Notice now that Theorem 9 constitutes an immediate alternative proof for Theorem 8. To see this, observe that any protocol with binary input is also a protocol with general input. Thus, in the case where $\mathscr{A}$ has a binary input alphabet, $<\mathscr{A},\phi,k>\in \overline{BBPVER}$ is a sufficient and necessary condition for $<\mathscr{A},\phi,k>\in \overline{BPVER}$, which establishes $\overline{BBPVER} \leq_p \overline{BPVER}$.

7.2.3 *BPVER′* and *BBPVER′*

Let us denote by $BPVER'$ the special case of $BPVER$ in which the protocol size is at least the size k of the communication graph, and similarly for $BBPVER'$. Clearly, the proofs of Theorems 8 and 9 establish that both problems are coNP-hard.

7.2.4 *GBP* Verification

We now study the hardness of $\overline{GBPVER}$.

Theorem 10. *GBPVER is coNP-hard.*

Proof. We will prove the statement by presenting a polynomial-time reduction from $\overline{BPVER'}$ to $\overline{GBPVER}$. Every time that we get an instance $<\mathscr{A},\phi,k>$ of $\overline{BPVER'}$ (where $\mathscr{A}$ is a population protocol for which $|<A>|\geq k$ holds), if $\mathscr{A}$ has a computation on G^k that does not stabilize to the correct output w.r.t. ϕ then we will return a population protocol $\mathscr{A}'$ and a formula ϕ' such that there exists some k' for which $\mathscr{A}'$ has a computation on $G^{k'}$ that does not stabilize to the the correct output w.r.t. ϕ'. On the other hand, if $\mathscr{A}$ has no such erroneous computation on G^k, $\mathscr{A}'$ will also have no erroneous computation (w.r.t. ϕ') for any complete communication graph (of any size greater than or equal to 2). Moreover, we will achieve that in polynomial time.

Keep in mind that the input to the machine computing the reduction is $<\mathscr{A},\phi,k>$. Let $X_{\mathscr{A}}$ be the input alphabet of $\mathscr{A}$. Clearly, $\phi''=\neg(\sum_{x\in X_{\mathscr{A}}} N_x = k)$ is a semilinear predicate if k is treated as a constant (N_x denotes the number of agents with input x). Thus, there exists a population protocol $\mathscr{A}''$ for the basic model that stably computes ϕ''. The population protocol $\mathscr{A}''$ can be constructed efficiently. Its input alphabet $X_{\mathscr{A}''}$ is equal to $X_{\mathscr{A}}$. The construction of the protocol can be found in [2] (in fact they present there a more general protocol for any linear combination of variables corresponding to a semilinear predicate). When the number of nodes of the communication graph is equal to k, $\mathscr{A}''$ always stabilizes to the all-zero output (all agents output the value 0) and when it is not equal to k, then $\mathscr{A}''$ always stabilizes to the all-one output.

We want to construct an instance $<\mathscr{A}',\phi'>$ of $\overline{GBPVER}$. We set $\phi' = \phi \vee \phi''$. Moreover, $\mathscr{A}'$ is constructed to be the composition of $\mathscr{A}$ and $\mathscr{A}''$. Obviously, $Q_{\mathscr{A}'} = Q_{\mathscr{A}} \times Q_{\mathscr{A}''}$. We define its output to be the union of its components' outputs, that is, $O(q_{\mathscr{A}}, q_{\mathscr{A}''}) = 1$ iff at least one of $O(q_{\mathscr{A}})$ and $O(q_{\mathscr{A}''})$ is equal to 1. It is easy to see that the above reduction can be computed in polynomial time.

We first prove that if $<\mathscr{A},\phi,k>\in \overline{BPVER'}$ then $<\mathscr{A}',\phi'>\in \overline{GBPVER}$. When $\mathscr{A}'$ runs on the complete graph of k nodes, the components of its states corresponding to $\mathscr{A}''$ stabilize to the all-zero output, independently of the initial configuration. Clearly, $\mathscr{A}'$ in this case outputs whatever $\mathscr{A}$ outputs. Moreover, for this communication graph, ϕ' is true iff ϕ is true (because $\phi'' = \neg(\sum_{x\in X_{\mathscr{A}}} N_x = k)$ is false, and $\phi' = \phi \vee \phi''$). But there exists some input for which $\mathscr{A}$ does not give the correct output with respect to ϕ (e.g. ϕ is true for some input but $\mathscr{A}$ for some computation does not stabilize to the all-one output). Since ϕ' expects the same output as ϕ and $\mathscr{A}'$ gives the same output as $\mathscr{A}$ we conclude that there exists some erroneous computation of $\mathscr{A}'$ w.r.t. ϕ', and the first direction has been proven.

Now, for the other direction, assume that $<\mathscr{A}',\phi'>\in \overline{GBPVER}$. For any communication graph having a number of nodes not equal to k, ϕ' is true and $\mathscr{A}'$ always stabilizes to the all-one output because of the $\mathscr{A}''$ component. This means that the erroneous computation of $\mathscr{A}'$ happens on the G^k. But for that graph, ϕ'' is always false and $\mathscr{A}''$ always stabilizes its corresponding component to the all-zero output. Now ϕ' is true iff ϕ is true and $\mathscr{A}'$ outputs whatever $\mathscr{A}$ outputs. But there exists some input and a computation for which $\mathscr{A}'$ does not stabilize to a configuration in which all agents give the output value that ϕ' requires which implies that $\mathscr{A}$ does not stabilize to a configuration in which all agents give the output value required by ϕ. Since the latter holds for G^k, the theorem follows. □

7.2.5 *BBPI* Verification

To show the inherent difficulty of the population protocol verification problem we consider an even simpler special case, namely, the *BBPIVER* problem ('I' standing for "Input", because an input assignment is additionally provided as part of the algorithm's input) that is defined as follows:

Problem 5 (*BBPIVER*). Given a population protocol $\mathscr{A}$ for the basic model whose input and output alphabets are binary (i.e. $X_{\mathscr{A}} = Y_{\mathscr{A}} = \{0,1\}$), a two-variable first-order logical formula ϕ in Presburger arithmetic representing the specifications of $\mathscr{A}$, and an input (assignment) $x = (x_0, x_1)$, where x_0 and x_1 are nonnegative integers, determine whether $\mathscr{A}$ conforms to its specifications for the complete digraph of $k = x_0 + x_1$ nodes whenever its computation begins from the initial configuration corresponding to x.

Let $BBPIVER'$ denote the special case of $BBPIVER$ in which $| <\mathscr{A}> | \geq k$.

Theorem 11. *$BBPIVER'$ and $BBPIVER$ are coNP-hard.*

Proof. The reduction is from $HAMPATH$ to $\overline{BBPIVER'}$, which proves that both $BBPIVER$ and $BBPIVER'$ are coNP-hard. In fact, the reduction is the same as in Theorem 9, but here, together with the protocol (as described in the proof of Theorem 9) and the always false specifications, we also return the input assignment $x = (1, n-2)$ and do not return the integer $k = n-1$. By looking carefully at the reduction of Theorem 9 it won't be difficult to see that if G has the desired hamiltonian path, then the protocol returned has an erroneous computation when beginning from input x, and if G does not have the desired hamiltonian path, then, for any input (x inclusive), the protocol is correct w.r.t. its specifications. □

7.2.6 Alternative proof of Theorem 9

We have now arrived to an alternative proof that $\overline{BBPVER}$ is NP-hard. The reduction is from $\overline{BBPIVER'}$ to $\overline{BBPVER}$. Given an instance $< \mathscr{A}, \phi, x = (x_0, x_1) >$ of the former we do as follows (keep in mind that we return an instance of the latter of the form $< \mathscr{A}', \phi', k >$). We set $k = x_0 + x_1$, $\phi' = \phi \vee \neg((N_0 = x_0) \wedge (N_1 = x_1))$, and $\mathscr{A}'$ is the union (w.r.t. the output functions) composition of $\mathscr{A}$ and $\mathscr{A}''$ (as in Theorem 10), where $\mathscr{A}''$ is a population protocol for the basic model that stably computes the predicate $\neg((N_0 = x_0) \wedge (N_1 = x_1))$. It is now easy to see (similarly to Theorem 10) that the reduction is correct and can be performed in polynomial time.

7.3 An Efficiently Solvable Special Case

We are now seeking for efficiently solvable special cases of the general $GBPVER$ problem. A population protocol $\mathscr{A}$ is called *binary* if its input alphabet, its output alphabet, and its set of states are all equal to $\{0, 1\}$. We consider now one of the most trivial cases, which is the $ALLBVER$ problem: We are given a binary population protocol $\mathscr{A}$ for the basic model and a formula ϕ representing its specifications. We want again to determine whether $\mathscr{A}$ is always correct w.r.t. ϕ.

The first question that arises is what can ϕ be in this case. So we have to find out first what is stably computable in this simplified model. If the output function of $\mathscr{A}$ is defined as $O(0) = O(1) = y$, where $y \in \{0, 1\}$, then any configuration of $\mathscr{A}$ on any communication graph gives the all-y output. For example, if $y = 0$ then all configurations correspond to the all-zero output, and if $y = 1$ then all configurations correspond to the all-one output. So we have just shown that the trivial predicates (those that are always true or always false) are stably computable. To seek for non-trivial stably computable predicates we have to agree that $O(0) = 0$ and $O(1) = 1$ (this is w.l.o.g. since the case $O(0) = 1$ and $O(1) = 0$ is symmetric). Moreover, we agree that $(0, 0) \to (0, 0)$ and $(1, 1) \to (1, 1)$ in the transition function δ. To see this, notice that a nontrivial predicate is true for some inputs and false for others. This means that a protocol for the predicate must be able to stabilize to both the all-zero and the all-one output, and this cannot hold in the absence of the above rules.

Now what about the input function I? Clearly, if $I(0) = I(1) = q$ then the initial configuration is always the all-q configuration (all agents are in state q). For example, if $q = 0$ then the initial configuration is for any input the all-zero configuration. But because of the rules $(0,0) \to (0,0)$ and $(1,1) \to (1,1)$ the population can never escape from its initial configuration, and this case again corresponds to trivial predicates. So, we again agree w.l.o.g. that $I(0) = 0$ and $I(1) = 1$.

It suffices to check the predicates that are stably computable by different combinations of right hand sides for the left hand sides $(0,1)$ and $(1,0)$ in δ. There are only 4^2 such combinations so our job is easy. We have the following cases:

- Both $\delta(0,1)$ and $\delta(1,0)$ do not belong to $\{(0,0),(1,1)\}$. Assume that an input assignment contains one 1 and all other agents get 0. In this case no interaction can increase or decrease the number of 1s so the population forever remains to an unstable configuration (not all agents agree on their output value). So there is no additional stably computable predicate from this case.
- Only one of $\delta(0,1)$ and $\delta(1,0)$ belongs to $\{(0,0),(1,1)\}$. If one of them is $(0,0)$ then (since the other offers nothing) if there is at least one 0 in the initial configuration (which is identical to the input assignment, because $I(0) = 0$ and $I(1) = 1$) the protocol rejects, whereas if all inputs are 1 the protocol accepts. We can call this the AND protocol corresponding to the stably computable predicate $\neg(N_0 \geq 1)$. Similarly, if one of them is $(0,0)$ then we have one form of the OR protocol (see e.g. [16]) and the stably computable predicate corresponding to it is $(N_1 \geq 1)$.
- Both $\delta(0,1)$ and $\delta(1,0)$ belong to $\{(0,0),(1,1)\}$ and $\delta(0,1) \neq \delta(1,0)$. In this case the protocol is unstable. Imagine an initial configuration with exactly one 1 (all other agents get 0) and say that $\delta(0,1) = (0,0)$ and $\delta(1,0) = (1,1)$. If the unique 1 interacts as the initiator with all other agents in state 0 (one after the other), the protocol in each step replaces a 0 with a 1 and in N_0 steps the population stabilizes to the all-one output. One the other hand if 1 had interacted as the responder with all other agents in the same way as before, then the population would have stabilized to that all-zero output and the protocol is obviously unstable.
- Both $\delta(0,1)$ and $\delta(1,0)$ belong to $\{(0,0),(1,1)\}$ and $\delta(0,1) = \delta(1,0)$. It is easy to see that again we get alternative versions of the OR protocol and the AND protocol, thus the stably computable predicates resulting from this (last) case are again $\neg(N_0 \geq 1)$ and $(N_1 \geq 1)$.

So we have arrived to a complete characterization of the class of stably computable predicates for the binary basic population protocol model. They are the predicates: always-true, always-false, $\neg(N_0 \geq 1)$, and $(N_1 \geq 1)$.

So we require the specifications ϕ, in the *ALLBVER* problem, to be a stably computable predicate of the binary basic population protocol model, i.e. one of always-true, always-false, $\neg(N_0 \geq 1)$, and $(N_1 \geq 1)$. Obviously, if a binary protocol $\mathscr{A}$ errs on G_2 (the complete graph of 2 nodes) w.r.t. ϕ then it errs in general. But we can also prove that if it errs on some G_k where $k > 2$ then it must err also on G_2 (an easy way to get convinced is to check the statement for all possible classes of protocols as outlined above). This indicates an obvious constant-time algorithm: The transition graph consists of 3 configurations. For every possible initial configuration find all the final strongly connected components that are reachable from it. If all configurations of those components give the correct output w.r.t. ϕ and this holds for all possible initial configurations, then $< \mathscr{A}, \phi >$ belongs to $ALLBVER$; otherwise $< \mathscr{A}, \phi > \notin ALLBVER$.

7.4 Algorithmic Solutions for *BPVER*

Since Theorem 8 established the coNP-hardness of *BPVER* (Problem 3), our only hope is to devise always-correct algorithms whose worst-case running-time will not be bounded by a polynomial in the size of the input, or algorithms that are not always correct, but are, in fact, correct most of the time (the notion of "approximation" seems to be irrelevant here). Before proceeding, we strongly suggest that the reader carefully revises the definitions from Subsection 7.1.

Our algorithms are search algorithms on the transition graph G_r. The general idea is that a protocol $\mathscr{A}$ does not conform to its specifications ϕ on k agents, if one of the following criteria is satisfied:

1. $\phi(c) = -1$ for some $c \in C_I$.
2. $\exists c, c' \in C_I$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq \phi(c')$.
3. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $O(c') = -1$.
4. $\exists c \in C_I$ and $c' \in C_F$ such that $c \xrightarrow{*} c'$ and $\phi(c) \neq O(c')$.
5. $\exists B' \in F_S$ such that $O(B') = -1$.
6. $\exists B \in I_S$ and $B' \in F_S$ such that $B \xrightarrow{*} B'$ and $\phi(B) \neq O(B')$ (possibly $B = B'$).

Note that any algorithm that correctly checks some of the above criteria is a possibly *non-complete verifier*. Such a verifier guarantees that it can discover an error of a specific kind, thus, we can always trust its "reject" answer (the protocol has some error of this kind). On the other hand, an "accept" answer is a weaker guarantee, in the sense that it only informs that the protocol does not have some error of this specific kind. Of course, it is possible that the protocol has other errors, violating criteria that are indetectable by this verifier. However, this is a first sign of *BPVER*'s *parallelizability*.

Theorem 12. *Any algorithm that checks criteria* 1, 5, *and* 6 *decides BPVER.*

Exercise 15. Prove Theorem 12.

7.4.1 Constructing the Transition Graph

Let $FindC_I(\mathscr{A}, k)$ be a function that, given a PP $\mathscr{A}$ and an integer $k \geq 2$, returns the set C_I of all initial configurations. This is not so hard to be implemented. $FindC_I$

simply iterates over the set of all input assignments $\mathscr{X}$ and for each $x \in \mathscr{X}$ computes $I(x)$ and puts it in C_I. Alternatively, computing C_I is equivalent to finding all distributions of indistinguishable objects (agents) into distinguishable slots (initial states), and, thus, Fenichel's algorithm [24] can be used for this purpose.

Algorithm 4 *$ConG_r$*

Input: PP $\mathscr{A}$ and integer $k \geq 2$.
Output: The transition graph G_r.

```
1: $C_I \leftarrow FindC_I(\mathscr{A}, k)$
2: $C_r \leftarrow \emptyset$
3: $E_r \leftarrow \emptyset$
4: while $C_I \neq \emptyset$ do
5:     Pick a $c \in C_I$, $C_I \leftarrow C_I - \{c\}$
6:     $C_r \leftarrow C_r \cup \{c\}$
7:     for all $r \in \Delta$ for which $c_i \geq r_{1,2}(i)$ and all $i \in [|Q|]$ for which $q_i \in \{r_1, r_2\}$ do
8:         Compute the unique configuration $c'$ for which $c \xrightarrow{r} c'$.
9:         if $c' \notin C_r$ then
10:            $C_I \leftarrow C_I \cup \{c'\}$
11:        end if
12:        $E_r \leftarrow E_r \cup (c, c')$
13:    end for
14: end while
15: return $(C_r, E_r)$
```

The transition graph G_r can be constructed by the procedure *$ConG_r$* (Algorithm 4), that takes as input a population protocol $\mathscr{A}$ and the population size k, and returns the transition graph G_r. The order in which configurations are put in and picked out of C_I determines whether BFS or DFS is used.

7.4.2 Non-complete Verifiers

Now, that we know how to construct the transition graph, we can begin constructing some non-complete verifiers (which are the easiest). In particular, we present two verifiers, *SinkBFS* and *SinkDFS*, that check all criteria but the last two. Both are presented via procedure *SinkVER* (Algorithm 5) and the order in which configurations of G_r are visited determines again whether BFS or DFS is used.

7.4.3 *SolveBPVER*: A Complete Verifier

We now construct the procedure *SolveBPVER* (Algorithm 6) that checks criteria 1, 5, and 6 (and also 2 for some speedup) presented in the beginning of Subsection 7.4, and, thus, according to Theorem 12, it correctly solves *BPVER* (i.e. it is a complete verifier for basic population protocols, when the population size is provided as part of the input). In particular, *SolveBPVER* takes as input a PP $\mathscr{A}$, its specifications ϕ and an integer $k \geq 2$, as outlined in the *BPVER* problem description, and returns "accept" if the protocol is correct w.r.t. its specifications on G^k and "reject" otherwise.

Algorithm 5 *SinkVER*

Input: A population protocol $\mathscr{A}$, a Presburger arithmetic formula ϕ, and an integer $k \geq 2$.
Output: ACCEPT if $\mathscr{A}$ is correct w.r.t. its specifications and the criteria 1,2,3, and 4 on G^k and REJECT otherwise.

```
1: $C_I \leftarrow FindC_I(\mathscr{A},k)$
2: if there exists $c \in C_I$ such that $\phi(c) = -1$ then
3:     return REJECT // Criterion 1 satisfied
4: end if
5: $G_r \leftarrow ConG_r(\mathscr{A}, k)$
6: for all $c \in C_I$ do
7:     Collect all $c'$ reachable from $c$ in $G_r$ by BFS or DFS.
8:     while searching do
9:         if one $c'$ is found such that $c' \in C_F$ and $(O(c') = -1$ or $\phi(c) \neq O(c'))$ then
10:            return REJECT // Criterion 3 or 4 satisfied
11:        end if
12:        if one $c'$ is found such that $c' \in C_I$ and $\phi(c) \neq \phi(c')$ then
13:            return REJECT // Criterion 2 satisfied
14:        end if
15:    end while
16: end for
17: return ACCEPT // Tests for criteria 1,2,3, and 4 passed
```

The algorithmic idea is based on the use of Tarjan's [36] or Cheriyan-Mehlhorn's and Gabow's [20, 25] (or any other) algorithm for finding the strongly connected components of G_r. In this manner, we obtain a collection S, where each $B \in S$ is a strongly connected component of G_r, that is, $B \subseteq C_r$. Given S we can easily compress G_r w.r.t. its strongly connected components as follows. The compression of G_r is a dag $D = (S, A)$, where $(B, B') \in A$ if and only if there exist $c \in B$ and $c' \in B'$ such that $c \rightarrow c'$ (that is, iff $B \rightarrow B'$). In words, the node set of D consists of the strongly connected components of G_r and there is a directed edge between two components of D if a configuration of the second component is reachable in one step from a configuration in the first one.

Algorithm 6 *SolveBPVER*

Input: A population protocol $\mathscr{A}$, a Presburger arithmetic formula ϕ, and an integer $k \geq 2$.
Output: ACCEPT if the protocol is correct w.r.t. its specifications on G^k and REJECT otherwise.

```
1: $C_I \leftarrow FindC_I(\mathscr{A}, k)$
2: if there exists $c \in C_I$ such that $\phi(c) = -1$ then
3:     return REJECT
4: end if
5: $G_r \leftarrow ConG_r(\mathscr{A}, k)$
6: Run one of Tarjan's or Gabow's algorithms to compute the collection $S$ of all strongly connected components of the transition graph $G_r$.
7: Compute the dag $D = (S, A)$, where $(B, B') \in A$ (where $B \neq B'$) if and only if $B \rightarrow B'$.
8: Compute the collection $I_S \subseteq S$ of all connected components $B \in S$ that contain some initial configuration $c \in C_I$ and the collection $F_S \subseteq S$ of all connected components $B \in S$ that have no outgoing edges in $A$, that is, all final strongly connected components of $G_r$.
9: for all $B \in F_S$ do
10:     if $O(B) = -1$ then
11:         return REJECT
12:     end if
13:     // Otherwise, all configurations $c \in B$ output the same value $O(B) \in \{0, 1\}$.
14: end for
15: for all $B \in I_S$ do
16:     if there exist initial configurations $c, c' \in B$ such that $\phi(c) \neq \phi(c')$ then
17:         return REJECT
18:     else
19:         // all initial configurations $c \in B$ expect the same output $\phi(B) \in \{0, 1\}$.
20:         Run BFS or DFS from $B$ in $D$ and collect all $B' \in F_S$ s.t. $B \xrightarrow{*} B'$ (possibly including $B$ itself).
21:         if there exists some reachable $B' \in F_S$ for which $O(B') \neq \phi(B)$ then
22:             return REJECT
23:         end if
24:     end if
25: end for
26: return ACCEPT
```

8 Open Problems

The following are some open problems for the interested reader:

- What is the computational power of the variation of the population protocol model in which the agents interact in groups of $k > 2$ agents and not in pairs?
- Recent (unpublished for the time being) research shows that $SPACE(n)$ (that is, $LINSPACE$) is a lower bound for the class of symmetric predicates that are stably computable by the basic MPP model, which may be possibly improved to $NSPACE(n)$ by exploiting the nondeterminism inherent in the interaction pattern. On the other hand, as mentioned in Section 3, the best known upper bound is $NSPACE(m)$, and, since we are dealing with complete communication graphs, it holds that $m = \mathcal{O}(n^2)$, which, clearly, leaves a huge gap between the two bounds. It is possible that $NSPACE(n \log n)$ is a better upper bound. But we do not expect

this to be easy, because it would require to prove that we can *encode* the $\mathscr{O}(n^2)$ sized configurations of MPP by new configurations of $\mathscr{O}(n \log n)$ size whose transition graph is, in some sense, *isomorphic* to the old one (e.g. the new configurations reach the same stable outputs). Thus, an exact characterization of this class is still open.

- Is the mediated population protocol model fault-tolerant? What are the necessary preconditions to obtain satisfactory fault tolerance?
- Is there an exact characterization of the class of decidable graph languages by MPP in the weakly-connected case?
- Is the PALOMA model fault-tolerant? What are the necessary preconditions to obtain satisfactory fault tolerance?
- Are there hierarchy theorems concerning all possible models of passively mobile communicating devices? For example, what is the relationship between MPP's class of computable predicates and *PLM*?
- [12] revealed the need for population protocols to have adaptation capabilities in order to keep working correctly and/or fast when natural modifications of the mobility pattern occur. However, we do not know yet how to achieve *adaptivity*.
- Are there more efficient, possibly logic-based, verification solutions for population protocols? Verifying methods for MPPs, Community Protocols, and PALOMA protocols are still totally unknown, although the ideas of Section 7 may also be applicable to these models.

References

1. D. Angluin, J. Aspnes, M. Chan, M. J. Fischer, H. Jiang, and R. Peralta. Stably computable properties of network graphs. In Proc. Distributed Computing in Sensor Systems: 1st IEEE International Conference, pages 63-74, 2005.
2. D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4): 235-253, 2006. Also in *23rd Annual ACM Symposium on Principles of Distributed Computing* (*PODC*), pages 290-299, New York, NY, USA, 2004. ACM.
3. D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Urn automata. Technical Report YALEU/DCS/TR-1280, Yale University Department of Computer Science, Nov. 2003.
4. D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3): 183-199, Sept. 2008.
5. D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing*, pages 292-299, 2006.
6. D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4): 279-304, November 2007.
7. J. Aspnes and E. Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science*, 93:98-117, October 2007. Columns: *Distributed Computing*, Editor: M. Mavronicolas.
8. R. Bakhshi, F. Bonnet, W. Fokkink, and B. Haverkort. Formal analysis techniques for gossiping protocols. In *ACM SIGOPS Operating Systems Review*, 41(5):28-36, Special Issue on Gossip-Based Networking, October, 2007.
9. J. Beauquier, J. Clement, S. Messika, L. Rosaz, and B. Rozoy. Self-stabilizing counting in mobile sensor networks. Technical Report 1470, LRI, Université Paris-Sud 11, 2007.

10. G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems: Proc. 4th Int. School on Formal Methods for the Design of Comput., Commun. and Software Syst. (SFM-RT 2004)*, number 3185 in LNCS, pages 200-236, Springer, 2004.
11. O. Bournez, P. Chassaing, J. Cohen, L. Gerin, and X. Koegler. On the convergence of population protocols when population goes to infinity. In *Applied Mathematics and Computation*, 215(4):1340-1350, 2009.
12. I. Chatzigiannakis, S. Dolev, S. P. Fekete, O. Michail, and P. G. Spirakis. Not all fair probabilistic schedulers are equivalent. In *13th International Conference On Principles Of DIstributed Systems (OPODIS)*, pages 33-47, Nimes, France, December 15-18, 2009.
13. I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. G. Spirakis. Algorithmic verification of population protocols. FRONTS Technical Report FRONTS-TR-2010-12, http://fronts.cti.gr/aigaion/?TR=148, Jan. 2010.
14. I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. G. Spirakis. Passively mobile communicating logarithmic space machines. FRONTS Technical Report FRONTS-TR-2010-16, http://fronts.cti.gr/aigaion/?TR=154, Feb. 2010.
15. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Decidable graph languages by mediated population protocols. In *23rd International Symposium on Distributed Computing (DISC)*, Elche, Spain, Sept. 2009. (Also FRONTS Technical Report FRONTS-TR-2009-16, http://fronts.cti.gr/aigaion/?TR=80)
16. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Experimental verification and performance study of extremely large sized population protocols. FRONTS Technical Report FRONTS-TR-2009-3, http://fronts.cti.gr/aigaion/?TR=61, Jan. 2009.
17. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Mediated population protocols. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 363-374, Rhodes, Greece, 2009.
18. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Recent advances in population protocols. In *34th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, August 24-28, 2009, Novy Smokovec, High Tatras, Slovakia.
19. I. Chatzigiannakis and P. G. Spirakis. The dynamics of probabilistic population protocols. In *Distributed Computing*, *22nd International Symposium*, *DISC*, volume 5218 of *Lecture Notes in Computer Science*, pages 498-499, 2008.
20. J. Cheriyan and K. Mehlhorn. Algorithms for dense graphs and networks on the random access computer. *Algorithmica*, 15: 521-549, 1996.
21. E. M. Clarke, O. Grumberg, and D. A. Peled. Model checking. MIT Press, 2000.
22. C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. When birds die: Making population protocols fault-tolerant. In *Proc. 2nd IEEE International Conference on Distributed Computing in Sensor Systems*, pages 51-66, 2006.
23. Z. Diamadi and M. J. Fischer. A simple game for the study of trust in distributed systems. *Wuhan University Journal of Natural Sciences*, 6(1-2):72-82, Mar. 2001. Also appears as Yale Technical Report TR-1207, Jan. 2001.
24. R. Fenichel. Distribution of Indistinguishable Objects into Distinguishable Slots. *Communications of the ACM*, 11(6), page 430, June 1968.
25. H. N. Gabow. Path-based depth-first search for strong and biconnected components. *Information Processing Letters*, 74: 107-114, 2000.
26. D. T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, 188:404-425, 1992.
27. D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340-2361, 1977.
28. S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285-296, 1966.
29. R. Guerraoui and E. Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 484-495, Rhodes, Greece, 2009.

30. A. Hinton, M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. In *Proc. 2nd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '06)*, volume 3920 of LNCS, pages 441-444. Springer, 2006.
31. G. Holzmann. The Spin model checker, primer and reference manual. Addison-Wesley, 2003.
32. M. Huth and M. Ryan. Logic in Computer Science: Modelling and reasoning about systems. Cambridge University Press, Cambridge, UK, 2004.
33. T. G. Kurtz. Approximation of population processes. Number 36 in *CBMS-NSF Regional Conference Series in Applied Mathematics*, *Society for Industrial and Applied Mathematics*, Philadelphia, 1981.
34. P.C. Olveczky and S. Thorvaldsen. Formal modeling and analysis of wireless sensor network algorithms in Real-Time Maude. *Parallel and Distributed Processing Symposium, International*, pp. 157, *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, 2006.
35. C. H. Papadimitriou. Computational complexity. Addison-Wesley, 1994.
36. R. Tarjan. Depth-first search and linear graph algorithms. In *SIAM Journal on Computing*, Vol. 1, No. 2, pages. 146-160, 1972.