All Symmetric Predicates in $NSPACE(n^2)$ are Stably Computable by the Mediated Population Protocol Model*

Ioannis Chatzigiannakis[1,2], Othon Michail[1,2], Stavros Nikolaou[2], Andreas Pavlogiannis[2], and Paul G. Spirakis[1,2]

[1] Research Academic Computer Technology Institute (RACTI), Patras, Greece
[2] Computer Engineering and Informatics Department (CEID), University of Patras
Email: {ichatz, michailo, spirakis}@cti.gr,
{snikolaou, paulogiann}@ceid.upatras.gr

Abstract. This work focuses on the computational power of the *Mediated Population Protocol* model on complete communication graphs and initially identical edges (*SMPP*). In particular, we investigate the class MPS of all predicates that are stably computable by the SMPP model. It is already known that MPS is in the symmetric subclass of $NSPACE(n^2)$. Here we prove that this inclusion holds with equality, thus, providing the following exact characterization for MPS: *A predicate is in MPS iff it is symmetric and is in $NSPACE(n^2)$.*

1 Introduction - Population Protocols

Theoretical models for Wireless Sensor Networks (WSNs) have received great attention recently, mainly because they constitute an abstract but yet formal and precise method for understanding the limitations and capabilities of this widely applicable new technology. The *Population Protocol* model [1] was designed to represent a special category of WSNs which is mainly identified by two distinctive characteristics: each sensor node is an extremely limited computational device and all nodes move according to some mobility pattern over which they have totally no control.

One reason for studying extremely limited computational devices is that in many real WSNs' application scenarios having limited resources is inevitable. For example, power supply limitations may render strong computational devices useless due to short lifetime. In other applications, mote's size is an important constraint that thoroughly determines the computational limitations. The other reason is that the population protocol model constitutes the starting point of a brand new area of research and in order to provide a clear understanding and foundation of the laws and the inherent properties of the studied systems it ought to be minimalistic. In terms of computational characterization each

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

node is simply a finite-state machine additionally equipped with sensing and communication capabilities and is usually called an *agent*. A *population* is the collection of all agents that constitute the distributed computational system.

The prominent characteristic that diversifies population protocols from classical distributed systems is the total inability of the computational devices to control or predict their underlying mobility pattern. Their movement is usually the result of some unstable environment, like water flow or wind, or the natural mobility of their carriers, and is known as *passive mobility*. The agents interact in pairs and are absolutely incapable of knowing the next pair in the interaction sequence. This inherent nondeterminism of the interaction pattern is modeled by an *adversary* whose job is to select interactions. The adversary is a black-box and the only restriction imposed is that it has to be *fair* so that it does not forever partition the population into non-communicating clusters and guaranteeing that interactions cannot follow some inconvenient periodicity.

As expected, due to the minimalistic nature of the population protocol model, the class of computable predicates was proven [1, 2] to be fairly small: it is the class of *semilinear predicates*, or, equivalently, all predicates definable by first-order logical formulas in *Presburger arithmetic* [10], which does not include multiplication of variables, exponentiations, and many other important and natural operations on input variables. Moreover, Delporte-Gallet *et al.* [9] showed that population protocols can tolerate only $\mathcal{O}(1)$ crash failures and not even a single Byzantine agent.

2 Enhancing the Model

The next big step is naturally to strengthen the population protocol model with 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. Several promising attempts have appeared towards this direction. In each case, the model enhancement is accompanied by a logical question: What is exactly the class of predicates computable by the new model?

An interesting extension was the *Community Protocol* model of Guerraoui and Ruppert [11] in which the agents have read-only industrial unique ids picked from an infinite set of ids. Moreover, each agent can store up to a constant number of other agents' ids. In this model, agents are only allowed to compare ids, that is, no other operation on ids is permitted. The community protocol model was proven to be extremely strong: the corresponding class consists of all symmetric predicates in $NSPACE(n \log n)$. It was additionally shown that if faults cannot alter the unique ids and if some necessary preconditions are satisfied, then community protocols can tolerate $\mathcal{O}(1)$ Byzantine agents.

The *Passively mobile Machines* (PM) model [5] made the assumption that each agent is a Turing Machine and defined *PALOMA* protocols as those protocols that use in every agent space that is bounded by a logarithm in the population size. Interestingly, it turned out that the agents are able to assign unique consecutive ids to themselves, get informed of the population size and,

by exploiting these, organize themselves into a distributed Nondeterministic TM (NTM) that makes full use of the agents' memories. The TM draws its nondeterminism by the nondeterminism inherent in the interaction pattern. The main result of that work was an exact characterization for the class PLM, of all predicates that are stably computable by PALOMA protocols: it is again precisely the class of all symmetric predicates in $NSPACE(n \log n)$.

3 Our Results - Roadmap

This work focuses on the computational power of another extension of the population protocol model that was proposed in [7] (see also [8] and [6]) and is called the *Mediated Population Protocol* (*MPP*) model. The main additional feature in comparison to the population protocol model is that each link (u, v) can be thought of as being itself an agent that only participates in the interaction (u, v). The agents u and v can exploit this joint memory to store pairwise information and to have it available during some future interaction. Another way to think of this system is that agents store pairwise information into some global storage, like, e.g., a base station, called the *mediator*, that provides a small fixed slot to each pair of agents. Interacting agents communicate with the mediator to read and update their collective information.

From [7] we know that the MPP model is strictly stronger than the population protocol model since it can compute a non-semilinear predicate. Moreover, we know that any predicate that is stably computable by the MPP model is also in $NSPACE(n^2)$. In this work, we show that, for complete graphs, this inclusion holds with equality, thus, providing the following exact characterization for the computational power of the MPP model in the fully symmetric case: *A predicate is stably computable by the MPP model iff it is symmetric and is in* $NSPACE(n^2)$. We show in this manner that the MPP model is surprisingly strong.

In section 4, we give a formal definition of the MPP model and introduce a special class of graphs (the *correctly labeled line graphs*) that comes up in our proof later on. Section 5 holds the actual proof. In particular, Subsection 5.1 presents some basic ideas that help us establish a first inclusion. In Subsection 5.2, we show how to extend these ideas in order to prove our actual statement.

4 The Mediated Population Protocol Model

4.1 Formal Definition

A *Mediated Population Protocol* (MPP) is a 7-tuple $(X, Y, Q, S, I, O, \delta)$, where X, Y, Q, and S are all finite sets and X is the *input alphabet*, Y is the *output alphabet*, Q is the set of *agent states*, S is the set of *edge states*, $I : X \to Q$ is the *input function*, $O : Q \to Y$ is the *output function*, and $\delta : Q \times Q \times S \to Q \times Q \times S$ is the *transition function*. If $\delta(a, b, c) = (a', b', c')$, we call $(a, b, c) \to (a', b', c')$ a *transition*, and we define $\delta_1(a, b, c) = a'$, $\delta_2(a, b, c) = b'$ and $\delta_3(a, b, c) = c'$.

An MPP $\mathcal{A}$ runs on the nodes of a *communication graph* $G = (V, E)$, which is directed without self-loops and multiple edges. V is the population, consisting of $n \equiv |V|$ agents. E is the set of permissible interactions between the agents.

In the most general setting, each agent initially senses its environment, as a response to a global start signal, and receives an input symbol from X. Then all agents apply the input function to their input symbols and obtain their initial state. Each edge is initially in one state from S as specified by some *edge initialization function* $\iota : E \to S$, which is not part of the protocol but generally models some preprocessing on the network that has taken place before the protocol's execution.

A *network configuration*, or simply a *configuration*, is a mapping $C : V \cup E \to Q \cup S$ specifying the state of each agent in the population and each edge in the set of permissible interactions. Let C and C' be configurations, and let u, v be distinct agents. We say that C goes to C' via encounter $e = (u, v)$, denoted $C \xrightarrow{e} C'$, if $C'(u) = \delta_1(C(u), C(v), C(e))$, $C'(v) = \delta_2(C(u), C(v), C(e))$, $C'(e) = \delta_3(C(u), C(v), C(e))$, and $C'(z) = C(z)$ for all $z \in (V - \{u, v\}) \cup (E - e)$. 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 adversary scheduler may stop after a finite number of steps or continue selecting pairs forever. Moreover, a strong global *fairness condition* is imposed on the adversary to ensure the protocol makes progress. 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 so does C'. A *computation* is an infinite fair execution. An interaction between two agents is called *effective* if at least one of the initiator's, the responder's, and the edge's states is modified (that is, if C, C' are the configurations before and after the interaction, respectively, then $C' \neq C$).

4.2 Stable Computation

Throughout this work we assume that the communication graph is complete and that all edges are initially in a common state s_0, that is, $\iota(e) = s_0$ for all $e \in E$. Call this for sake of simplicity the SMPP model ('S' standing for "Symmetric"). An SMPP may run on any such communication graph $G = (V, E)$, where $n \geq 2$, and its input (also called an *input assignment*) is any $x = \sigma_1 \sigma_2 \ldots \sigma_n \in X^{\geq 2} = \{x \in X^* \mid |x| \geq 2\}$. In particular, by assuming an ordering over V, the input to agent i is the symbol σ_i, for all $i \in \{1, 2, \ldots, n\}$. Let $p : X^{\geq 2} \to \{0, 1\}$ be any predicate over $X^{\geq 2}$. p is called *symmetric* if for every $x \in X^{\geq 2}$ and any x' which is a permutation of x's symbols, it holds that $p(x) = p(x')$ (in words, permuting the input symbols does not affect the predicate's outcome). Any language $L \subseteq X^{\geq 2}$ corresponds to a unique predicate p_L defined as $p_L(x) = 1$ iff $x \in L$. Such a

language is symmetric iff p_L is symmetric. Due to this bijection we use the term *symmetric predicate* for both predicates and languages.

Like population protocols, MPPs do not halt. Instead a protocol is required to *stabilize*, in the sense that it reaches a point after which the output of every agent will remain unchanged. A predicate p over $X^{\geq 2}$ is said to be *stably computable* by the SMPP model, if there exists an SMPP $\mathcal{A}$ such that for any input assignment $x \in X^{\geq 2}$, any computation of $\mathcal{A}$ on the complete communication graph of $|x|$ nodes beginning from the initial configuration corresponding to x reaches a configuration after which all agents forever output $p(x)$.

Let MPS (standing for "Mediated Predicates in the fully Symmetric case") be the class of all stably computable predicates by the SMPP model. Note that all predicates in MPS have to be symmetric because the communication graph is complete and all edges are initially in the same state. Let $SSPACE(f(n))$ and $SNSPACE(f(n))$ be $SPACE(f(n))$'s and $NSPACE(f(n))$'s restrictions to symmetric predicates, respectively.

4.3 Correctly Labeled Line Graphs

Let $G = (V, E)$ be a communication graph. A *line (di)graph* $L = (K, A)$ is either an isolated node (that is, $|K| = 1$ and $A = \emptyset$), which is the *trivial* line graph, or a tree such that, if we ignore the directions of the links, two nodes have degree one and all other nodes have degree two. A *line subgraph* of G is a line graph $L \subseteq G$ and is called *spanning* if $K = V$. Let $d(u)$ denote the degree of $u \in K$ w.r.t. to A. Let $C_l(t)$ denote the *label component* of the state of $t \in V \cup E$ under configuration C (in the beginning, we call it *state* for simplicity).

We say that a line subgraph of G is *correctly labeled* under configuration C, if it is trivial and its state is l with no active edges incident to it or if it is non-trivial and all the following conditions are satisfied:

1. Assume that $u, v \in K$ and $d(u) = d(v) = 1$. These are the only nodes in K with degree 1. Then one of u and v is in state k_t (non-leader or right endpoint) and the other is either in state l_t or in state l_h (leader or left endpoint). The unique $e_u \in A$ incident to u, where u is w.l.o.g. in state k_t, is an outgoing edge and the unique $e_v \in A$ incident to v is outgoing if $C_l(v) = l_t$ and incoming if $C_l(v) = l_h$.
2. For all $w \in K - \{u, v\}$ (internal nodes) it holds that $C_l(w) = k$.
3. For all $a \in A$ it holds that $C_l(a) \in \{p, i\}$ and for all $e \in E - A$ such that e is incident to a node in K it holds that $C_l(e) = 0$.
4. Let $v = u_1, u_2, \ldots, u_r = u$ be the path from the leader to the non-leader endpoint (resulting by ignoring the directions of the arcs in A). Let $P_L = \{(u_i, u_{i+1}) \mid 1 \leq i < r\}$ be the corresponding directed path from v to u. Then for all $a \in A \cap P_L$ it holds that $C_l(a) = p$ (proper edges) and for all $a' \in A - P_L$ that $C_l(a') = i$ (inverse edges).

See Figure 1 for some examples of correctly labeled line subgraphs. The meaning of each state will become clear in the proof of Theorem 1 in the following section.

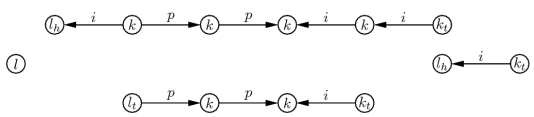


Fig. 1. Some correctly labeled line subgraphs. We assume that all edges not appearing are in state 0 (inactive).

5 The Computational Power of the SMPP Model

In [7], it was shown that MPS is a proper superset of the set of semilinear predicates. Here we are going to establish a much better inclusion. In particular, in Section 5.1 we show that any predicate in $SSPACE(n)$ is also in MPS. In other words, the SMPP model is at least as strong as a linear space TM that computes symmetric predicates. Then in Section 5.2 we extend the ideas used in the proof of this result in order to establish that $SSPACE(n^2)$ is a subset of MPS showing that MPS is a surprisingly wide class. Finally, we improve to $SNSPACE(n^2)$, thus, arriving at an exact characterization for MPS (the inverse inclusion already exists from [7]).

5.1 A First Inclusion: $SSPACE(n) \subseteq MPS$

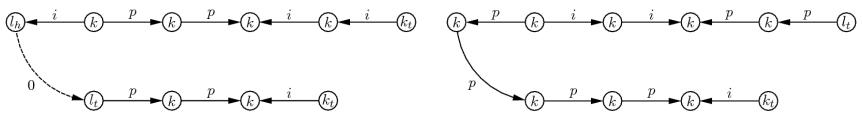
Theorem 1. *There is an SMPP $\mathcal{A}$ that constructs a correctly labeled spanning line subgraph of any complete communication graph G.*

Proof. We provide a high level description of the protocol $\mathcal{A}$ in order to avoid its many low-level details. All agents are initially in state l, thought of as being *simple leaders*. All edges are in state 0 and we think of them as being *inactive*, that is, not part of the line subgraph to be constructed. An edge in state p is interpreted as *proper* while an edge in state i is interpreted as *inverse* and both are additionally interpreted as *active*, that is, part of the line subgraph to be constructed. An agent in state k is a (*simple*) *non-leader*, an agent in state k_t is a non-leader that is additionally the *tail* of some line subgraph (*tail non-leader*), an agent in state l_t is a leader and a tail of some line subgraph (*tail leader*), and an agent in state l_h is a leader and a *head* of some line subgraph (*head leader*). All these will be clarified in the sequel. A *leader* is a simple, tail, or head leader.

The agents become organized in correctly labeled line subgraphs by the following transitions: $(l, l, 0) \to (k_t, l_h, i)$, $(l_h, l, 0) \to (k, l_h, i)$, $(l, l_h, 0) \to (l_t, k, p)$, $(l_t, l, 0) \to (k, l_h, i)$, and $(l, l_t, 0) \to (l_t, k, p)$.

We now describe how two such line graphs L_1 and L_2 are pieced together. Denote by $l(L) \in V$ and by $k_t(L) \in V$ the leader and tail non-leader endpoints of a correctly labeled line graph L, respectively. When $l(L_1) = u$ interacts as the initiator with $l(L_2) = v$, through an inactive edge, v becomes a non-leader with a special mark, e.g. k', the edge becomes proper with a special mark, and u becomes a leader in a special state l' indicating that this state will travel towards $k_t(L_1)$ while making all proper edges that it meets inverse and all inverse edges

proper. In order to know its direction, it marks each edge that it crosses. When it, finally, arrives at the endpoint, it goes to another special state and walks the same path in the inverse direction until it meets v again. This walk can be performed easily, without using the marks, because now all edges have correct labels (states). To diverge from L_1's endpoint it simply follows the proper links as the initiator (moving from their tail to their head) and the inverse links as the responder (moving from their head to their tail) while erasing all marks left from its previous walk. When it reaches v it erases its mark, making its state k, and goes to another special state indicating that it again must walk towards $k_t(L_1)$ for the last time, performing no other operation this time. To do that, it follows the proper links as the responder (from their head to their tail) and the inverse links as the initiator (from their tail to their head). When it, finally, arrives at $k_t(L_1)$ it becomes a normal tail leader and now it is easy to see that L_1 and L_2 have been merged correctly into a common correctly labeled line graph. See Figure 2 for an example. The correctness of this process, called the *merging process*, is captured by Lemma 1.



(a) Before merging. (b) After merging.

Fig. 2. Two line subgraphs just before the execution and after the completion of the merging process.

Lemma 1. *When the leader endpoints of two distinct correctly labeled line subgraphs of G, $L_1 = (K_1, A_1)$ and $L_2 = (K_2, A_2)$, interact via $e \in E$, then, in a finite number of steps, L_1 and L_2 are merged into a new correctly labeled line graph $L_3 = (K_1 \cup K_2, A_1 \cup A_2 \cup \{e\})$.*

Initially, G is partitioned into n correctly labeled trivial line graphs. It is easy to see that correctly labeled line graphs never become smaller and, according to Lemma 1, when their leaders interact they are merged into a new line graph. Moreover, given that there are two correctly labeled line subgraphs in the current configuration there is always the possibility (due to fairness) that these line graphs may get merged and there is no other possible effective interaction between them. In simple words, two line graphs can only get merged and there is always the possibility that merging actually takes place. It is easy to see that this process has to end, due to fairness, in a finite number of steps having constructed a correctly labeled spanning line subgraph of G (for simplicity, we call this process the *spanning process*). □

Theorem 2. *Assume that the communication graph $G = (V, E)$ is a correctly labeled line graph of n agents, where each agent takes its input symbol in a second*

state component *(the first component is used for the labels of the spanning process and is called* label component*). Then there is an MPP $\mathcal{A}$ that when running on such a graph simulates a deterministic TM $\mathcal{M}$ of $\mathcal{O}(n)$ (linear) space that computes symmetric predicates.*

Proof. It is already known from [1, 3] that the theorem holds for population protocols with no inverse edges. It is easy to see that the correct p and i labels can be exploited by the simulation in order to identify the correct directions. □

It must be clear now, that if the agents could detect termination of the spanning process then they would be able to simulate a deterministic TM of $\mathcal{O}(n)$ space that computes symmetric predicates. But, unfortunately, they are unable to detect termination, because if they could, then termination could also be detected in any non-spanning line subgraph constructed in some intermediate step (it can be proven by symmetry arguments together with the fact that the agents cannot count up to the population size). Fortunately, we can overcome this by applying the *reinitialization* technique of [11, 5].

Theorem 3. *$SSPACE(n)$ is a subset of MPS.*

Proof. Take any $p \in SSPACE(n)$. By Theorem 2 we know that there is an MPP $\mathcal{A}$ that stably computes p on a line graph of n nodes. We have to show that there exists an SMPP $\mathcal{B}$ that stably computes p. We construct $\mathcal{B}$ to be the composition of $\mathcal{A}$ and another protocol $\mathcal{I}$ that is responsible for executing the spanning and reinitialization processes. Each agent's state consists of three components: a read-only *input backup*, one used by $\mathcal{I}$, and one used by $\mathcal{A}$. Thus, $\mathcal{A}$ and $\mathcal{I}$ are, in some sense, executed in parallel in different components.

Protocol $\mathcal{I}$ does the following. It always executes the spanning process and when the merging of two line graphs comes to an end it executes the following reinitialization process. The new leader u that resulted from merging becomes marked, e.g. l_t^*. Recall that the new line graph has also correct labels. When u meets its right neighbor, u sets its $\mathcal{A}$ component to its input symbol (by copying it from the input backup), becomes unmarked, and passes the mark to its right neighbor (correct edge labels guarantee that each agent distinguishes its right and left neighbors). When the newly marked agent interacts with its own right neighbor, it does the same, and so on, until the two rightmost agents interact, in which case they are both reinitialized at the same time and the special mark is lost. It is easy to see that this process guarantees that all agents in the line graph become reinitialized and before completion non-reinitialized agents do not have effective interactions with reinitialized ones (the special marked agent acts always as the separator between reinitialized and non-reinitialized agents). Note that if other reinitialization processes are pending from previous reinitialization steps, then the new one may identify and erase them.

From Theorem 1 we know that the spanning process executed by $\mathcal{I}$ results in a correctly labeled spanning line subgraph of G. The spanning process, as already mentioned, terminates when the merging of the last two line subgraphs takes place and merging also correctly terminates in a finite number of steps (Lemma

1). Moreover, from the above discussion we know that, when this happens, the reinitialization process will correctly reinitialize all agents of the spanning line subgraph, thus, all agents in the population. But then, independently of its computation so far, $\mathcal{A}$ will run from the beginning on a correctly labeled line graph of n nodes (this line graph will not be modified again in the future), thus, it will stably compute p. Finally, if we assume that $\mathcal{B}$'s output is $\mathcal{A}$'s output then we conclude that the SMPP $\mathcal{B}$ also stably computes p, thus, $p \in MPS$. □

5.2 An Exact Characterization: $MPS = SNSPACE(n^2)$

We now extend the techniques employed so far to obtain an exact characterization for MPS.

Theorem 4. *Assume that the complete communication graph $G = (V, E)$ contains a correctly labeled spanning line subgraph, where each agent takes its input symbol in a second* state component*. Then there is an MPP $\mathcal{A}$ that when running on such a graph simulates a deterministic TM $\mathcal{M}$ of $\mathcal{O}(n^2)$ space that computes symmetric predicates.*

Proof. For simplicity and w.l.o.g. we assume that $\mathcal{A}$ begins its execution from the leader endpoint, that initially the simulation moves all n input symbols to the leftmost outgoing inactive edges ($n-2$ leaving from the leader and two more leaving from the second agent of the line graph), that the left endpoint is a tail leader, and that the edge states now consist of two components, one used to identify them as active/inactive and the other used by the simulation.

In contrast to Theorem 2 the simulation also makes use of the inactive edges. The agent in control of the simulation is in a special state denoted with a star '$*$'. Since the simulation starts from the left endpoint (tail leader), its state will be l_t^*. When the star-marked leader interacts with its unique right neighbor on the line graph, the neighbor's state is updated to a *r-marked* state (i.e. k^r). The k^r agent then interacts with its own right neighbor which is unmarked and the neighbor updates its state to a special *dot* state (i.e. $\dot{k}$) whereas the other agent (in state k^r) is updated to k. Then the only effective interaction is between the star-marked leader (l_t^*) and the dot non-leader ($\dot{k}$) via the inactive edge joining them. In this way, the inactive edge's state component used for the simulation becomes a part of the TM's tape. In fact $\mathcal{M}$'s tape consists only of the inactive edges and is accessed in a systematic fashion which is described below.

If the simulation has to continue to the right, the interaction $(l_t^*, \dot{k})$ sends the dot agent to state k^r. If it has to proceed left, the dot agent goes to state k^l. An agent in state k^r interacts with its right neighbor sending it to dot state whereas a k^l agent does the same for its left neighbor. In this way, the dot mark is moving left and right between the agents by following the active edges in the appropriate interaction role (initiator or responder) as described in Theorem 1 for the special states traversing through the line graph. The dot mark's (state's) position in the line graph determines which outgoing inactive edge of l_t^* will be used. The sequence in which the dot mark is traversing the graph is the sequence

in which l_t^* visits its outgoing inactive edges. Therefore if it has to visit the next inactive edge it moves the dot mark to the right (via a k^r state) or to the left (via a k^l state) if it has to visit the previous one. It should be noted that the dot marked agent plays the role of the TM's head since it points the edge (which would correspond to a tape's cell in $\mathcal{M}$) that is visited. As stated above only the inactive edges hold the contents of the TM's tape. The active ones are used for allowing the special states (symbols) traverse the line graph.

Consider the case where the dot mark reaches the right non-leader endpoint (k_t) and the simulation after the interaction $(l_t^*, \dot{k}_t)$ demands to proceed right. Since l_t^*'s outgoing edges have all been visited by the simulation, the execution must continue on the next agent (right neighbor of leader endpoint l_t) in the line graph. This is achieved by having another special state traversing from right to left (since we are in the right non-leader endpoint) until it finds l_t^*. Then it removes its star mark (state) and assigns it to its right neighbor which now takes control of the simulation visiting its own inactive edges. A similar process takes place when the simulation, controlled by any non-leader agent, reaches the left leader endpoint and needs to proceed to the left cell.

When the control of the simulation reaches a non-leader agent (e.g. from the left leader endpoint side) in order to visit its first edge it places the dot mark to the left leader endpoint and then to the next (on the right) non-leader and so forth. If the dot mark reaches the star-marked agent (in the previous example from the left endpoint side) then it moves the dot to the closer (in the line graph) agent that can "see" via an inactive edge towards the right non-leader endpoint. In this way, each agent visits its outgoing edges in a specific sequence (from leader to non-leader when the simulation moves right and the reverse when it moves left) providing the $\mathcal{O}(n^2)$ space needed for the simulation. □

Theorem 5. *$SSPACE(n^2)$ is a subset of MPS.*

Proof. The main idea is similar to that in the proof of Theorem 3 (based again on the reinitialization technique). We assume that the edge states consist now of two components, one used to identify them as active/inactive and the other used by the simulation (protocol $\mathcal{A}$ from Theorem 4).

This time, the reinitialization process attempts to reinitialize not only all agents of a line graph but also all of their outgoing edges. We begin by describing the reinitialization process in detail. Whenever the merging process of two line graphs comes to an end, resulting in a new line graph L, the leader endpoint of L goes to a special *blocked* state, let it be l^b, blocking L from getting merged with another line graph while the reinitialization process is being executed. Keep in mind that L will only get ready for merging just after the completion of the reinitialization process. By interacting with its unique right neighbor in state k via an active edge it propagates the blocked state towards that neighbor updating its state to k^b and reinitializing the agent. The block state propagates in the same way towards the tail non-leader reinitializing and updating all intermediate non-leaders to k^b from left to right. Once it reaches this endpoint, a new special state k_0 is generated which traverses L in the inverse direction. Once k_0 reaches the leader endpoint, it disappears and the leader updates its state to l^*.

Now reinitialization of the inactive edges begins. When the leader in l^* interacts with its unique right neighbor (via the active edge joining them) it updates its neighbor's state to a special *bar* state (e.g. $\bar{k}$). When the agent with the bar state interacts with its own right neighbor, which is unmarked, the neighbor updates its state to a special *dot* state (e.g. $\dot{k}$). Now the bars cannot propagate and the only effective interaction is between the star leader and the dot non-leader. This interaction reinitializes the state component of the edge used for the simulation and makes the responder non-leader a bar non-leader. Then the new bar non-leader turns its own right neighbor to a dot non-leader, the second outgoing edge of the leader is reinitialized in this manner, and so on, until the edge joining the star leader (left endpoint) with the dot tail non-leader (right endpoint) is reinitialized. What happens then is that the bars are erased one after the other from right to left and finally the star moves one step to the right. So the first non-leader has now the star and it reinitializes its own inactive outgoing edges from left to right in a similar manner. The process repeats the same steps over and over, until the right endpoint of L reinitializes all of its outgoing edges. When this happens, $\mathcal{A}$ will execute its simulation on the correct reinitialized states. The above process is clearly executed correctly when L is spanning (because all outgoing edges have their heads on the line graph). When it isn't, the correctness of the process is captured by the following lemma.

Lemma 2. *Let L and L' be two distinct line subgraphs of G. If L runs a reinitialization process then it always terminates in a finite number of steps.*

Proof. If L' is not running a reinitialization process then there can be no conflict between L and L'. If L' is also running its own reinitialization process, a conflict occurs when a star agent of one graph interacts with a dot agent of the other, but in both cases the reinitialization process is either not affected or cannot be delayed, thus, it always terminates in a finite number of steps. □

We finally ensure that the simulation does not ever alter the agent labels used by the spanning and reinitialization processes. In the proof of Theorem 4 we made $\mathcal{A}$ put marks on the labels in order to get executed correctly. Now we simply think of these marks as being placed in a separate subcomponent of $\mathcal{A}$ that is ignored by the other processes. □

Theorem 6. *$SNSPACE(n^2)$ is a subset of MPS.*

Proof. We modify the deterministic TM of Theorem 5 by adding another component in each agent's state which stores a non-negative integer of value at most equal to the greatest number of non-deterministic choices that the new NTM $\mathcal{N}$ can face at any time. Note that this number is independent of the population size. In every reinitialization each agent obtains this value from its neighbors according to its position (which depends on the distance from the leader endpoint) in the line graph. Nondeterministic choices are mapped to these values and whenever such a choice has to be made, the agent in control of the simulation uses the value of the agent with whom it has the next arbitrary interaction. The inherent nondeterminism of the interaction pattern ensures that choices are made

nondeterministically. If the accept state is reached all agents accept whereas if the reject state is reached the TM's computation is reinitialized. Fairness guarantees that all paths in the tree representing $\mathcal{N}$'s nondeterministic computation will eventually (although maybe after a long time) be followed. □

We have now arrived at the following exact characterization for MPS.

Theorem 7. $MPS = SNSPACE(n^2)$.

Proof. Follows from Theorem 6 and Theorem 8 of [7]. □

See the corresponding technical report [4] for a formal constructive proof and some graphical examples.

References

1. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, mar 2006.
2. Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *PODC '06: Proceedings of the 25th annual ACM Symposium on Principles of Distributed Computing*, pages 292–299. ACM Press, 2006.
3. James Aspnes and Eric Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science*, 93:98–117, 2007.
4. Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. All symmetric predicates in $NSPACE(n^2)$ are stably computable by the mediated population protocol model. Technical Report FRONTS-TR-2010-17, RACTI, 2010. `http://fronts.cti.gr/aigaion/?TR=155`.
5. Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. Passively mobile communicating logarithmic space machines. Technical Report FRONTS-TR-2010-16, RACTI, 2010. `http://fronts.cti.gr/aigaion/?TR=154`. Also CoRR. `http://arxiv.org/abs/1004.3395`.
6. Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. Brief announcement: Decidable graph languages by mediated population protocols. In *DISC*, pages 239–240, 2009.
7. Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. Mediated population protocols. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2, pages 363–374, 2009.
8. Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. Recent advances in population protocols. In *MFCS '09: Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science 2009*, pages 56–76. Springer-Verlag, 2009.
9. Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When birds die: Making population protocols fault-tolerant. In *DCOSS*, pages 51–66, 2006.
10. S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.
11. Rachid Guerraoui and Eric Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 484–495, 2009.