Mediated Population Protocols[☆,☆☆]

Othon Michail[a,b,*], Ioannis Chatzigiannakis[a,b], Paul G. Spirakis[a,b]

[a] *Research Academic Computer Technology Institute (RACTI), Patras, Greece*
[b] *Computer Engineering and Informatics Department (CEID), University of Patras*

Abstract

We extend here the Population Protocol (PP) model of Angluin *et al.* [2004,2006] in order to model more powerful networks of resource-limited agents that are possibly mobile. The main feature of our extended model, called the *Mediated Population Protocol* (*MPP*) model, is to allow the edges of the interaction graph to have states that belong to a constant-size set. We then allow the protocol rules for pairwise interactions to modify the corresponding edge state. The descriptions of our protocols preserve both the *uniformity* and *anonymity* properties of PPs, that is, they do not depend on the size of the population and do not use unique identifiers. We focus on the computational power of the MPP model on complete interaction graphs and initially identical edges. We provide the following exact characterization of the class **MPS** of stably computable predicates: *A predicate is in* **MPS** *iff it is symmetric and is in* **NSPACE**(n^2).

Keywords:
population protocol, diffuse computation, finite-state agent, intermittent communication, stable computation, passive mobility

1. Introduction - Population Protocols

Theoretical models for Wireless Sensor Networks (WSNs) have received great attention over the past few years, 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 [AAD$^+$04, AAD$^+$06] was designed to represent a special category of WSNs which is mainly identified by two distinctive characteristics: each sensor node

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

[☆☆]Some preliminary versions of the results in this paper have also appeared in [CMS09a, CMN$^+$10a, CMS09b].

[*]Corresponding author (Telephone number: +30 2610 960200, Fax number: +30 2610 960490, Postal Address: Research Academic Computer Technology Institute (RACTI), N. Kazantzaki Str., Patras University Campus, Rio, P.O. Box 1382, 26500, Greece).

Email addresses: michailo@cti.gr (Othon Michail), ichatz@cti.gr (Ioannis Chatzigiannakis), spirakis@cti.gr (Paul G. Spirakis)

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 node is simply a finite-state machine additionaly 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. Two outstanding properties of population protocols is that they are *uniform* and *anonymous*. The so called *uniformity property* requires that the descriptions of the protocols are independent of the population size and the *anonymity property* that there is no room in the state of an agent to store a unique identifier [1].

As already mentioned, another prominent characteristic of population protocols 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, like in the now canonical example in which each bird in a flock is equipped with such an agent and the birds naturally move, 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 noncommunicating clusters and guaranteeing that interactions cannot follow some inconvenient periodicity. The above characteristics render the study of population protocols a non-trivial task.

As expected, due to the minimalistic nature of the population protocol model, the class of computable predicates was proven [AAD$^+$06, AAER07] to be fairly small: it is the class of *semilinear predicates* [GS66] (or, equivalently, all predicates definable by first-order logical formulas in *Presburger arithmetic* [Pre29]), which does not include multiplication of variables, exponentiations, and many other important and natural operations on input variables. Moreover, we only know how to transform any protocol that computes a function in the failure-free model into a protocol that can tolerate $\mathcal{O}(1)$ crash failures. [2] However, this requires some inevitable weakening of the problem specifica-

[1] Throughout the text we abbreviate the word "identifier" by "id" and we use "uid" when we want to emphasize the fact that the identifier is "unique".

[2] Although the letter 'O' is usually used in the Complexity Theory literature for the Big-Oh

tion. This result is due to Delporte-Gallet *et al.* [DGFGR06]. Additionally, Guerraoui and Ruppert [GR09] showed that any function computable by a population protocol tolerating one Byzantine agent is trivial. On the other hand, Angluin, Aspnes, and Eisenstat [AAE08b] described a population protocol that computes majority tolerating $\mathcal{O}(\sqrt{n})$ Byzantine failures. However, that protocol was designed for a much more restricted setting, where the scheduler chooses the next interaction randomly and uniformly (see the probabilistic population protocols discussion in Section 2.1).

2. Enhancing the Model

The work of Angluin *et al.* shed light and opened the way towards a brand new and very promising direction. The lack of control over the interaction pattern, as well as its inherent nondeterminism, gave rise to a variety of new theoretical models for WSNs. Those models draw most of their beauty precisely from their inability to organize interactions in a convenient and predetermined way. In fact, the population protocol model was the minimalistic starting-point of this area of research. Most efforts are now towards strengthening the model of Angluin *et al.* 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*?

One idea is to allow some heterogeneity in the model, so that some agents have more computational power than others. For example, a base station can be an additional part of the network with which the agents are allowed to communicate [BCM$^+$07].

Another extension was the *Community Protocol* model of Guerraoui and Ruppert [GR09] in which the agents are equipped with read-only uids 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 $\mathbf{NSPACE}(n \log n)$, where n is the community size. The proof was based on a simulation of a modified version of Schönhage's (Nondeterministic) *Storage Modification Machine*. It was additionally shown that if faults cannot alter the uids and if some necessary preconditions are satisfied, then community protocols can tolerate $\mathcal{O}(1)$ Byzantine agents.

The *Passively mobile Machines* (*PM*) model [CMN$^+$10c, CMN$^+$10d] made the assumption that each agent instead of being an automaton is a Turing Ma-

notation, we have chosen here to use its calligraphic version '$\mathcal{O}$' in order to avoid confusion with the output function of protocols.

chine [3] with unbounded memory. Then the authors studied computations upper-bounded by plausible space limitations. They focused on complete interaction graphs and defined the complexity classes $\mathbf{PMSPACE}(f(n))$ parametrically, consisting of all predicates that are stably computable by some PM protocol that uses $\mathcal{O}(f(n))$ memory on each agent. That work arrived at an exact characterization of the classes $\mathbf{PMSPACE}(f(n))$ when $f(n) = \Omega(\log n)$: they are precisely the classes of all symmetric predicates in $\mathbf{NSPACE}(nf(n))$. Also the computability of the PM model when the protocols use $o(\log\log n)$ space per machine was explored and was proved that $\mathbf{SEM} = \mathbf{PMSPACE}(f(n))$ when $f(n) = o(\log\log n)$, where **SEM** denotes the class of the semilinear predicates. In fact, it was proved that this bound acts as a threshold, so that $\mathbf{SEM} \subsetneq \mathbf{PMSPACE}(f(n))$ when $f(n) = \mathcal{O}(\log\log n)$.

This work proposes another extension of the population protocol model which seems to be of its own theoretical interest. The main additional feature of the new model is that the communication links are capable of storing limited information. We are mainly interested in the model's computational capabilities and study it on a purely theoretical ground. We call our model the *Mediated Population Protocol* (*MPP*) model.

2.1. Other Previous Work

Much work concerning the population protocol model has been devoted to establishing that the class of computable predicates is precisely the class of *semilinear predicates* [AAD+04, AAD+06, AAE06, AAER07]. Moreover, in [AAD+04, AAD+06], the *probabilistic population protocol* model was proposed, in which the scheduler selects randomly and uniformly the next interaction pair. Some work has concentrated on performance, supported by this random scheduling assumption (see e.g. [AAE08a]). [CDF+09] proposed a generic definition of probabilistic schedulers and a collection of new fair schedulers, and revealed the need for the protocols to adapt when natural modifications of the mobility pattern occur. [BCK+09, CS08] 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. Moreover, several extensions of the basic model have been proposed in order to more accurately reflect the requirements of practical systems. In [AAC+05], Angluin *et al.* studied what properties of restricted interaction graphs are stably computable by the population protocol model, gave protocols for some of them, and proposed an extension of the model with *stabilizing inputs* in order to resolve the resistance of population protocols to composability. Some other works incorporated agent failures [DGFGR06]. Recently, Bournez *et al.* [BCCK08] investigated the possibility of studying population protocols via game-theoretic approaches. For some introductory texts to the subject of population protocols see [AR07, Spi10, MCS10] and for a survey

[3] As common in the CS literature, we abbreviate a Turing Machine by TM and by NTM when we want to emphasize that the TM is Nondeterministic.

mostly based on preliminary results of this work see [CMS09b]. Finally, the *Static Synchronous Sensor Field* (SSSF) [ADGS09, ASS10] is a very promising recently proposed model that addresses networks of tiny heterogeneous computational devices and additionally allows processing over constant flows (*streams*) of data originating from the environment. The latter feature is totally absent from the models discussed so far and is required by various sensing problems. See [ACD+11] for a joint survey on population-protocol-like models and static synchronous sensor fields.

3. Our Results - Roadmap

Section 4 provides a formal definition of the MPP model. Section 5 focuses on the computational power of the model by studying what predicates on input assignments are stably computable in the *fully symmetric case*, in which the interaction graph is complete and all edges are initially in a common state. First Section 5.1 proves that the MPP model is strictly stronger than the population protocol model by showing that the former can stably compute a non-semilinear predicate. Then in Section 5.2 it is shown that the MPP model can turn itself into a deterministic TM of linear space. Section 5.3 first extends the techniques developed in Section 5.2 to show that the MPP model can simulate a NTM of $\mathcal{O}(n^2)$ space and then, by showing that the inverse inclusion also holds, it establishes the following exact characterization of the class of computable predicates by the MPP model: *it is precisely the class of symmetric predicates in* $\mathbf{NSPACE}(n^2)$. Thus, unexpectedly, while preserving both *uniformity* and *anonymity*, the MPP model turns out to be an extremely powerful enhancement: it dramatically extends the class of computable predicates, from semilinear predicates to all symmetric predicates computable by a NTM in $\mathcal{O}(n^2)$ space. Section 6 concludes and discusses some promising future research directions.

4. The Mediated Population Protocols: A Formal Model

4.1. Formal Definition

Definition 1. *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*

1. X *is the* input alphabet*,*
2. Y *is the* output alphabet*,*
3. Q *is the set of* agent states*,*
4. S *is the set of* edge states*,*
5. $I : X \to Q$ *is the* input function*,*
6. $O : Q \to Y$ *is the* output function*,*
7. $\delta : Q \times Q \times S \to Q \times Q \times S$ *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 *interaction graph* is a (usually directed) graph $G = (V, E)$, where V specifies the set of agents (also called a *population*) and E the permissible interactions between them; that is, $(u, v) \in E$ indicates the possibility of an interaction between agents u and v, in which u is the *initiator* and v the *responder*. Throughout this article we use the letters n and m to denote $|V|$ and $|E|$, respectively. A *graph universe* (or *graph family*) $\mathcal{U}$ is any set of interaction graphs. Unless otherwise stated, we assume that the graph universes under consideration consist of directed interaction graphs without self-loops and multiple edges. We denote by $\mathcal{G}_{con}$ the graph universe consisting of all weakly connected interaction graphs of any finite number of nodes greater or equal to 2. Given a fixed graph universe $\mathcal{U}$, a MPP $\mathcal{A}$ runs on the nodes of an interaction graph $G = (V, E) \in \mathcal{U}$.

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 concurrently apply the input function to their input symbols and obtain their initial state (in this way the *initial configuration* of the system is formed). 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

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

that is, C' is the result of the interaction of the pair (u, v) under configuration C and is the same as C except for the fact that the states of u, v, and (u, v) have been updated according to δ_1, δ_2, and δ_3, 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 after a finite number of steps or continue selecting pairs forever. Moreover, note that, according to the preceding definitions, the adversary scheduler may, for example, 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 such unpleasant scenarios, 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'. An adversary scheduler is fair if it always leads to fair executions. 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 $C' \neq C$). Similarly, a transition $(a, b, c) \to (a', b', c')$ is called effective if $a' \neq a$, or $b' \neq b$, or $c' \neq c$.

Note that the mediated population protocol model *preserves both uniformity and anonymity* properties of population protocols. As a result, any MPP's *code* is of *constant size*, thus, can be stored in each agent (device) of the population and, additionally, there is not enough room in the states of the agents and the edges to store uids. Nevertheless, as we shall see, the MPP model can handle far more complicated computations than the population protocol model.

4.2. Stable Computation

The input (also called an *input assignment*) to a MPP is any $x = \sigma_1\sigma_2 \ldots \sigma_n \in X^*$ such that $n = |V|$.[4] In particular, by assuming an ordering over V, the input to agent i is the symbol σ_i, for all i, $1 \leq i \leq n$. Let $p : X^* \to \{0, 1\}$ be some predicate over X^*. p is called *symmetric* if for every $x = \sigma_1\sigma_2 \ldots \sigma_n \in X^*$ and any permutation function $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$, it holds that $p(x) = p(\sigma_{\pi(1)}\sigma_{\pi(2)} \ldots \sigma_{\pi(n)})$ (in words, permuting the input symbols does not affect the predicate's outcome). Similarly, a language $L \subseteq X^*$ is called symmetric if $x = \sigma_1\sigma_2 \ldots \sigma_n \in L$ implies $\sigma_{\pi(1)}\sigma_{\pi(2)} \ldots \sigma_{\pi(n)} \in L$ for all π. Any language $L \subseteq X^*$ corresponds to a unique predicate p_L defined as $p_L(x) = 1$ iff $x \in L$. It is easy to see that L 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 configuration C is called *output stable* if for every configuration C' that is reachable from C it holds that $O(C'(u)) = O(C(u))$ for all $u \in V$, where $O(C(u))$ is the output of agent u under configuration C. In simple words, if an output stable configuration is ever reached, no agent will change its output in any subsequent step and no matter how the computation proceeds thereafter.

A predicate p over X^* is said to be *stably computable* by the MPP model in a graph universe $\mathcal{U}$, if there exists a MPP $\mathcal{A}$ such that for any input assignment $x \in X^*$ and any $G = (V, E) \in \mathcal{U}$ s.t. $|V| = |x|$, any computation of $\mathcal{A}$ on G beginning from the initial configuration corresponding to x eventually (i.e. in a finite number of steps) reaches an output stable configuration in which all agents output $p(x)$.

[4] The truth is that we consider only graphs with at least 2 nodes, since smaller graphs do not even permit a single interaction (so, only inputs in $X^{\geq 2}$ are permitted).

A configuration C is called *state stable* if for every configuration C' s.t. $C \xrightarrow{*} C'$ it holds that $C' = C$. We say that a protocol $\mathcal{A}$ has *stabilizing states* if every computation of $\mathcal{A}$ eventually reaches a state stable configuration; that is, the states of all agents eventually stop changing. Note that any protocol that *state-stabilizes* also *output-stabilizes*, but the inverse is not generally true (stabilizing states is a stronger requirement).

In some cases, a protocol, instead of stably computing a predicate p, may provide some different sort of guarantee. For example, whenever runs on some $x \in X^*$ such that $p(x) = 1$, it may forever remain to configurations where at least one agent is in state a, and whenever $p(x) = 0$ it may forever remain to configurations where no agent is in state a. To formalize this, we say that a MPP $\mathcal{A}$ *guarantees* $t : Q^* \to \{0, 1\}$ *w.r.t.* $p : X^* \to \{0, 1\}$ in a graph universe $\mathcal{U}$ if, for any input assignment $x \in X^*$ and any $G = (V, E) \in \mathcal{U}$ s.t. $|V| = |x|$, any computation of $\mathcal{A}$ on G beginning from the initial configuration corresponding to x eventually reaches a configuration C, s.t. for all C', where $C \xrightarrow{*} C'$, it holds that $t(C') = t(C) = p(x)$.[5]

5. Predicates on Input Assignments

We assume here that the interaction graph is complete and that all edges are initially in a common state s_0, that is, the universe is $\{G \mid G \text{ is complete}\}$ and $\iota(e) = s_0$ for all $e \in E$. Call this for the sake of simplicity the SMPP model ('S' standing for "Symmetric"). We are interested in the computational power of the SMPP model. In particular, we provide an exact characterization of the predicates on input assignments that are stably computable.

Definition 2. *Let* **MPS** *(standing for "Mediated Predicates in the fully Symmetric case"* [6]*) be the class of all stably computable predicates by the SMPP model.*

Lemma 1. *All predicates in* **MPS** *are symmetric.*

Proof. Take any $p \in$ **MPS** and let $\mathcal{A}$ be the SMPP that stably computes it. Take also any input assignment $x = \sigma_1\sigma_2 \ldots \sigma_n$ and let $\pi : V \to V$ be any permutation of $V = \{1, 2, \ldots, n\}$. Now consider the input assignment $x' = \sigma_{\pi(1)}\sigma_{\pi(2)} \ldots \sigma_{\pi(n)}$, which is a permutation of x. Take any fair, w.r.t. $\mathcal{A}$, infinite interaction sequence [7] $e_1, e_2, \ldots$, where $e_i \in E$, and replace each $e_i = (j, k)$ with $(\pi(j), \pi(k))$ to obtain a new infinite interaction sequence, which is well defined due to the fact that the interaction graph is complete. Now consider the two infinite executions of $\mathcal{A}$ that correspond to the two interaction sequences on inputs x and x', respectively. Obviously, $x'_w = x_{\pi(w)}$, so that for the initial configurations C'_0 and C_0 we have that $C'_0(w) = C_0(\pi(w))$ for all agents $w \in V$.

[5] Note that by assuming an ordering on V we can define configurations as strings from Q^*, as we did for the input assignments at the beginning of this subsection.

[6] See the beginning of the Roadmap in Section 3 to have a brief intuition of this notion.

[7] By a *fair interaction sequence* we mean one that leads to a computation of $\mathcal{A}$.

Moreover, we have initially that $C'_0(j,k) = C_0(\pi(j),\pi(k))$ for all $(j,k) \in E$, which holds trivially since all edges are initially in s_0. Assume that the above holds for some interaction step i, that is, $C'_i(w) = C_i(\pi(w))$ for all $w \in V$ and $C'_i(j,k) = C_i(\pi(j),\pi(k))$ for all $(j,k) \in E$. It is not hard to see that the same must hold for step $i+1$, consequently both infinite executions pass in each step through the same multiset of states. This together with the fact that one execution is fair implies that the other must also be fair. So, we have obtained two computations of $\mathcal{A}$ on inputs x and x', respectively, that forever provide the same multisets of output symbols. Now, the fact that p is stably computable implies that $p(x) = p(x')$, which in turn implies that p has to be symmetric. □

Throughout the text, we use $\mathbf{SSPACE}(f(n))$ and $\mathbf{SNSPACE}(f(n))$ to denote $\mathbf{SPACE}(f(n))$'s and $\mathbf{NSPACE}(f(n))$'s restrictions to symmetric predicates, respectively and **SEM** to denote the class of semilinear predicates.

In the rest of this article, we build the machinery required to arrive at the exact characterization of **MPS** that is captured by Theorem 1.

Theorem 1. $\mathbf{MPS} = \mathbf{SNSPACE}(n^2)$.

Proof. One direction follows from Theorem 10 and the inverse direction from Corollary 2. □

We begin by providing an abstract proof idea of the above Theorem that briefly discusses most techniques that we develop throughout the rest of the article. This is done for the sake of clarity and readability of the results that follow.

Proof Idea. The "only if" part is easy. Any predicate in **MPS** is obviously symmetric and additionally we can perform in $\mathcal{O}(n^2)$ space a nondeterministic search on the transition graph of the SMPP that stably computes the predicate.

The sufficiency of the conditions is somewhat more complicated. We have to show that for all symmetric languages $L \in \mathbf{NSPACE}(n^2)$ there exists a SMPP that stably computes p_L, defined as $p_L(x) = 1$ iff $x \in L$. The idea is to organize the agents into a spanning pseudo-path subgraph of the interaction graph (pseudo-path graphs are defined in the beginning Section 5.2). To do that, the agents begin to form small pseudo-path graphs that in the sequel are merged together and are expanded to isolated nodes. When this process ends, the edges of the spanning pseudo-path graph will be active and all other $\mathcal{O}(n^2)$ edges will be inactive. Now the network can operate as a Turing machine of $\mathcal{O}(n^2)$ space by using the agents as the control units and the inactive edges as the cells. Whenever the inactive edges of some agent are exhausted it passes control (via some active edge) to its neighbor on the spanning pseudo-path graph. By also exploiting the nondeterminism inherent in the interaction pattern the agents can simulate the nondeterministic TM that decides L. Note that, since the agents cannot detect termination of the spanning pseudo-path graph construction process, any time that the structure changes they reinitialize their computation in a systematic manner, so that reinitialized agents do not communicate with non-reinitialized ones, and by exploiting a backup of their

input that is maintained throughout the computation. The final reinitialization happens when the spanning pseudo-path graph is formed an then the simulation is executed correctly. □

5.1. **MPS** *is a Proper Superset of* **SEM**

In this section, we provide a first inclusion (in fact, a lower bound) for MPS. By combining Theorems 2 and 3, we get in Corollary 1 that the non-semilinear predicate ($N_c = N_a \cdot N_b$) belongs to **MPS**, where N_σ denotes the number of agents that initially obtain the input symbol σ and '$\cdot$' denotes standard multiplication. This (due to the fact that population protocols cannot handle multiplication of variables [AAE06]) establishes the following separation: **MPS** *is a proper superset of* **SEM** (which is captured by Theorem 4).

Protocol 1 *VarProduct*

1: $X = \{a, b, c\}$
2: $Y = \{0, 1\}$
3: $Q = \{a, \dot{a}, b, c, \bar{c}\}$
4: $S = \{0, 1\}$
5: $I(\sigma) = \sigma$, for all $\sigma \in X$
6: $O(a) = O(b) = O(\bar{c}) = 1$ and $O(c) = O(\dot{a}) = 0$
7: δ: $(a, b, 0) \to (\dot{a}, b, 1)$, $(c, \dot{a}, 0) \to (\bar{c}, a, 0)$, $(\dot{a}, c, 0) \to (a, \bar{c}, 0)$
8: // All transitions that do not appear have no effect, e.g. $\delta(a, b, 1) = (a, b, 1)$

Theorem 2. *Protocol* VarProduct *(see Protocol 1) provides w.r.t. predicate* ($N_c = N_a \cdot N_b$) *the following semilinear guarantee:*

- *If* $N_c \neq N_a \cdot N_b$ *then at least one agent remains in one of the states* $\dot{a}$ *and* c.
- *If* $N_c = N_a \cdot N_b$ *then no agent remains in these states.*

Proof. First of all, we notice that in a complete directed interaction graph, $N_a \cdot N_b$ equals to the number of links leading from agents in state a to agents in state b. The main idea is that we should erase a number of c's equal to the number of a's times the number of b's. That is, for each a we should erase b c's. In the population protocol model, the impossibility for computing such a predicate comes from the fact that there is no way for an agent being w.l.o.g. in state a to be able to say that it has already counted a specific agent in state b. If e.g. $N_a = N_b = N_c = \mathcal{O}(n)$, then it is impossible in the population protocol model for each b to be able to remember all a's that have already counted it.

On the other hand, in the SMPP model this is resolved easily. It is easy to see that when at least one of N_a, N_b, and N_c is equal to zero, then in all such cases, except for the case where $N_c = 0$, $N_a \neq 0$ and $N_b \neq 0$, no computation takes place and the protocol trivially provides the required guarantee. In the case that we referred to explicitly, the first rule of δ is applied at least once,

while the second and third rules cannot be applied (since N_c always remains zero) and, thus, at least one agent goes to state $\dot{a}$ without being able to leave from it. Noticing that $N_a \cdot N_b \neq 0$ it is obvious that in this case a $\dot{a}$ correctly remains in $VarProduct$'s computation.

The interesting case is when all N_a, N_b and N_c are not equal to zero (in fact they are greater than zero since the N_q's are always non-negative). Recall that all edges are initially in their initial common state 0. The protocol proceeds as follows. When an agent in state a interacts as the initiator with an agent in state b, then the initiator goes to $\dot{a}$ and the corresponding edge goes to state 1. The modification in the state of the edge defining this specific ordered pair (a, b) is all the protocol needs to "remember" in order not to count the same pair again. When an agent in state c interacts with an agent in state $\dot{a}$, then c is erased by becoming $\bar{c}$ and $\dot{a}$ returns to its natural operation by becoming a again. The crucial point is to notice that the protocol tries to erase $N_a \cdot N_b$ agents in state c. If $N_c = N_a \cdot N_b$, then it will eventually manage to do it and at that point no agent will be in one of the states $\dot{a}$ and c and, moreover, no agent will be able to go again to one of these states. Thus, in this case, the protocol guarantees that eventually no agent remains in one of the states $\dot{a}$ and c. If $N_c < N_a \cdot N_b$ then eventually at least one agent will remain to state $\dot{a}$, while there will be no unerased agent in state c to be able to turn it again to state a. Finally, when $N_c > N_a \cdot N_b$ some agents in state c will keep their state, since there will be no unvisited (a, b) pair to erase any of those c's. Thus, when $N_c \neq N_a \cdot N_b$ the protocol guarantees that at least one agent remains in one of the states $\dot{a}$ and c and this completes the proof. □

Remark 1. *It is easy to see that Protocol* VarProduct *has stabilizing states.*

Note that Theorem 2 alone does not complete the separation of **SEM** and **MPS**. The reason is that it does not show that the SMPP model stably computes $(N_c = N_a \cdot N_b)$; what it truly shows is that whenever the predicate is true a state stable configuration is reached for which another predicate t on configurations becomes true, and whenever it is false a state stable configuration is reached for which t is also false. In fact, there is a way to exploit the guarantee and the stabilizing states in order to achieve the separation. This is captured by the following general composition theorem holding also for non-complete interaction graphs.

Theorem 3. *Let $\mathcal{G}$ be some family of directed and connected interaction graphs. If there exists a MPP $\mathcal{A}$ with stabilizing states that, in $\mathcal{G}$, guarantees w.r.t. a predicate p a semilinear predicate t, then p is stably computable by the MPP model in $\mathcal{G}$.*

Proof. We show that $\mathcal{A}$ can be composed with a provably existing protocol $\mathcal{B}$ that stably computes t to give a new MPP $\mathcal{C}$ satisfying the following properties:

- $\mathcal{C}$ is formed by the composition of $\mathcal{A}$ and $\mathcal{B}$,
- its input is $\mathcal{A}$'s input,

- its output is $\mathcal{B}$'s output, and
- $\mathcal{C}$ stably computes p (i.e. all agents agree on the correct output) in $\mathcal{G}$.

Protocol $\mathcal{A}$ has stabilizing states and provides a guarantee t which is a semilinear predicate on $\mathcal{A}$'s configurations. Let $X_{\mathcal{A}} = X$ be the input alphabet of $\mathcal{A}$, $Q_{\mathcal{A}}$ the set of $\mathcal{A}$'s states, $\delta_{\mathcal{A}}$ the transition function of $\mathcal{A}$, and similarly for any other component of $\mathcal{A}$. We will use the indexes $\mathcal{B}$ and $\mathcal{C}$, for the corresponding components of the other two protocols.

Since predicate t is semilinear, according to a result in [AAC+05], there is a population protocol $\mathcal{B}'$ that stably computes t with stabilizing inputs in $\mathcal{G}_{con}$. Note that $\mathcal{G} \subseteq \mathcal{G}_{con}$, so any predicate stably computable (both with or without stabilizing inputs) in $\mathcal{G}_{con}$ is also stably computable in $\mathcal{G}$. In fact, the same protocol $\mathcal{B}'$ stably computes t with stabilizing inputs in $\mathcal{G}$. Moreover, there also exists a mediated population protocol $\mathcal{B}$ (the one that is the same as $\mathcal{B}'$ but simply ignores the additional components of the new model) that stably computes t with stabilizing inputs in $\mathcal{G}$. Note that the input alphabet of $\mathcal{B}$ is $X_{\mathcal{B}} = Q_{\mathcal{A}}$, and its transition function is of the form $\delta_{\mathcal{B}} : (Q_{\mathcal{A}} \times Q_{\mathcal{B}}) \times (Q_{\mathcal{A}} \times Q_{\mathcal{B}}) \to Q_{\mathcal{B}} \times Q_{\mathcal{B}}$, since there is no need to specify edge states (formally we should, but the protocol ignores them). In fact, $Q_{\mathcal{A}}$ also plays the role of $\mathcal{B}$'s inputs that eventually stabilize.

We define a mediated population protocol $\mathcal{C}$ as follows: $X_{\mathcal{C}} = X_{\mathcal{A}}$, $Y_{\mathcal{C}} = Y_{\mathcal{B}} = \{0, 1\}$, $Q_{\mathcal{C}} = Q_{\mathcal{A}} \times Q_{\mathcal{B}}$, $I_{\mathcal{C}} : X_{\mathcal{A}} \to Q_{\mathcal{C}}$ defined as $I_{\mathcal{C}}(x) = (I_{\mathcal{A}}(x), i_{\mathcal{B}})$ for all $x \in Q_{\mathcal{C}}$, where $i_{\mathcal{B}} \in Q_{\mathcal{B}}$ is the initial state of protocol $\mathcal{B}$, $S_{\mathcal{C}} = S_{\mathcal{A}}$, $O_{\mathcal{C}}(a, b) = O_{\mathcal{B}}(b)$, for all $q = (a, b) \in Q_{\mathcal{C}}$, and finally its transition function $\delta_{\mathcal{C}} : Q_{\mathcal{C}} \times Q_{\mathcal{C}} \times S_{\mathcal{C}} \to Q_{\mathcal{C}} \times Q_{\mathcal{C}} \times S_{\mathcal{C}}$ is defined as

$$\begin{aligned}\delta_{\mathcal{C}}((a, b), (a', b'), s) = ((&\delta_{\mathcal{A}_1}(a, a', s), \delta_{\mathcal{B}_1}((a, b), (a', b'))),\\ &(\delta_{\mathcal{A}_2}(a, a', s), \delta_{\mathcal{B}_2}((a, b), (a', b'))),\\ &\delta_{\mathcal{A}_3}(a, a', s)),\end{aligned}$$

where for $\delta_{\mathcal{A}}(x, y, z) = (x', y', z')$ (in $\mathcal{A}$'s transition function), we have that $\delta_{\mathcal{A}_1}(x, y, z) = x'$, $\delta_{\mathcal{A}_2}(x, y, z) = y'$, $\delta_{\mathcal{A}_3}(x, y, z) = z'$, and similarly for $\delta_{\mathcal{B}}$.

Intuitively, $\mathcal{C}$ consists of $\mathcal{A}$ and $\mathcal{B}$ running in parallel. The state of each agent is a pair $c = (a, b)$, where $a \in Q_{\mathcal{A}}$, $b \in Q_{\mathcal{B}}$, and the state of each edge is a member of $S_{\mathcal{A}}$. Initially each agent senses an input x from $X_{\mathcal{A}}$ and this is transformed according to $I_{\mathcal{C}}$ to such a pair, where $a = I_{\mathcal{A}}(x)$ and b is always a special $\mathcal{B}$'s initial state $i_{\mathcal{B}} \in Q_{\mathcal{B}}$. When two agents in states (a, b) and (a', b') interact through an edge in state s, then protocol $\mathcal{A}$ updates the first components of the agent states, i.e. a and a', and the edge state s, as if $\mathcal{B}$ didn't exist. On the other hand, protocol $\mathcal{B}$ updates the second components by taking into account the first components that represent its separate input ports at which the current input symbol of each agent is available at every interaction ($\mathcal{B}$ takes $\mathcal{A}$'s states for agent input symbols that may change arbitrarily between any two computation steps, but the truth is that they change due to $\mathcal{A}$'s computation). Since the first

components of $\mathcal{C}$'s agent states eventually stabilize as a result of the fact that $\mathcal{A}$ has stabilizing states, protocol $\mathcal{B}$ will eventually obtain stabilizing inputs, consequently will operate correctly, and will stably compute t as if it had began computing on $\mathcal{A}$'s state stable configuration. But, since t provides the correct answer for p if applied on $\mathcal{A}$'s state stable configuration, it is obvious that $\mathcal{C}$ must stably compute p in $\mathcal{G}$, and the theorem follows. □

Corollary 1. *The non-semilinear predicate* $(N_c = N_a \cdot N_b)$ *belongs to* **MPS**.

Proof. The SMPP *VarProduct* has stabilizing states (Remark 1) and in the family of all complete directed interaction graphs guarantees w.r.t. $(N_c = N_a \cdot N_b)$ a semilinear predicate (Theorem 2). Consequently, the requirements of Theorem 3 are satisfied and $(N_c = N_a \cdot N_b)$ is stably computable by the SMPP model in the family of all complete directed interaction graphs. □

Theorem 4. $\mathbf{SEM} \subsetneq \mathbf{MPS}$.

Proof. Clearly, the population protocol model is a special case of the mediated population protocol model, therefore $\mathbf{SEM} \subseteq \mathbf{MPS}$ and, by Corollary 1 together with the fact that $(N_c = N_a \cdot N_b)$ is non-semilinear, $(N_c = N_a \cdot N_b) \in \mathbf{MPS} - \mathbf{SEM}$. □

5.2. A Better Inclusion: $\mathbf{SSPACE}(n) \subseteq \mathbf{MPS}$

We are now going to establish a much better inclusion. In particular, we will show that any predicate in $\mathbf{SSPACE}(n)$ is also in **MPS**. In other words, the SMPP model is at least as strong as a linear space TM that computes symmetric predicates. We begin with some necessary definitions.

Let $G = (V, E)$ be an interaction graph and let $d_G(u) \equiv |\{w \in V \mid (u, w) \in E \text{ or } (w, u) \in E\}|$ denote the degree of u w.r.t. G. A *pseudo-path graph* $L = (K, A)$ is a directed graph either satisfying $|K| = 1$ and $A = \emptyset$ or $|K| > 1$, $d_L(u) = d_L(v) = 1$ for some $u, v \in K$, and $d_L(w) = 2$ for all $w \in K - \{u, v\}$. In words, it is either an isolated node, which we call the *trivial* pseudo-path graph, or a directed graph that becomes a path graph when the directions of the links are ignored. A *pseudo-path subgraph* of G is a pseudo-path graph $L \subseteq G$ and is called *spanning* if $K = V$. Let $C_l(t)$ denote the *label component* of the state of $t \in V \cup E$ under configuration C.

We say that a pseudo-path subgraph of G is *correctly labeled* under configuration C, if it is trivial and its label 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 $d_L(u) = d_L(v) = 1$. These are the only nodes in K with degree 1. Then one of u and v has label k_t (non-leader endpoint) and the other has either l_t or l_h (leader endpoint). The unique $e_u \in A$ incident to u, where u has w.l.o.g. label 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 $C_l(a) \in \{p, i\}$ (active edges) and for all $e \in E - A$ such that e is incident to a node in K it holds that $C_l(e) = 0$ (inactive edges).
4. Let $\upsilon = u_1, u_2, \ldots, u_r = u$ 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 υ 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 pseudo-path subgraphs. The meaning and service of each label will become clear in the following discussion.

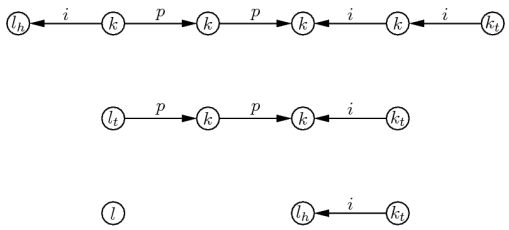


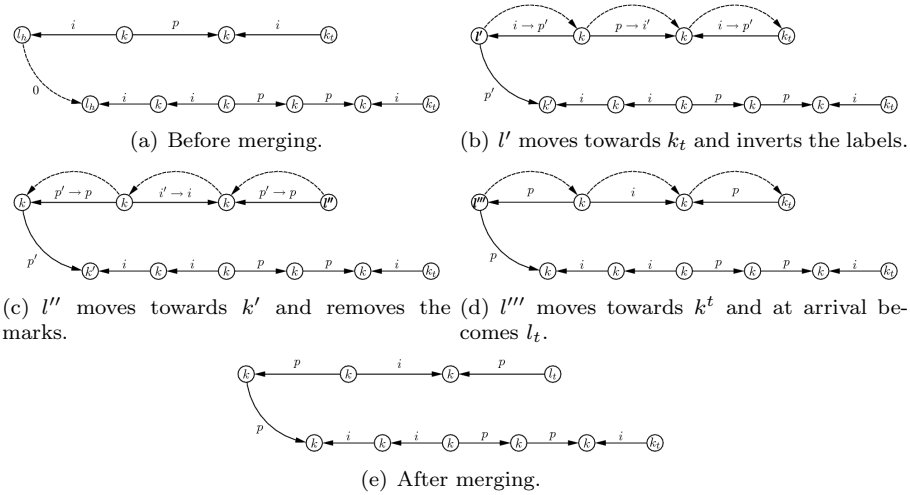
Figure 1: We assume that the above depicted graph, call it G, is complete. We have chosen not to draw the inactive edges for the sake of visibility. Therefore, all edges not appearing have label 0, that is, they are inactive. The top six nodes form a correctly labeled pseudo-path subgraph of G. The reason is that the left endpoint has label l_h, that is, it is a head leader, the right endpoint has label k_t, that is, it is a tail non-leader (condition 1 satisfied), all intermediate nodes are (simple) non-leaders (condition 2 satisfied), the edges that follow the direction from left to right have label p, that is, they are proper, those that follow the direction from right to left have label i, that is, they are inverse, and all other edges incident to these nodes (those that do not appear) are inactive (conditions 3 and 4 satisfied). Similarly, all other appearing graphs are pseudo-path subgraphs of the complete graph G. Note that the left node at the bottom that seems to be isolated, is in fact a node of G whose incident edges are all inactive. Moreover, it has label l, consequently it constitutes a trivial pseudo-path subgraph of G.

We describe now a SMPP, called *Spanning Process*, that constructs a correctly labeled spanning pseudo-path subgraph of any complete interaction graph G. The correctness of the protocol is captured by Theorem 5. We provide a high level description of the protocol in order to avoid its many low-level details. All agents have initially label l, thought of as being *simple leaders*. All edges have label 0 and we think of them as being *inactive*, that is, not part of the pseudo-path subgraph to be constructed. An edge having label p is interpreted as *proper* while an edge having label i is interpreted as *inverse* and both are additionally interpreted as *active*, that is, part of the pseudo-path subgraph to be constructed. An agent with label k is a (*simple*) *non-leader*, an agent with k_t is a non-leader that is additionally the *tail* of some pseudo-path subgraph (*tail non-leader*), an agent having label l_t is a leader and a tail of some pseudo-path subgraph (*tail leader*), and an agent having l_h is a leader and a *head* of some pseudo-path subgraph (*head leader*). A *leader* is a simple, tail, or head leader. All these will be further clarified in the sequel.

When two simple leaders interact through an inactive edge, the initiator becomes a tail non-leader, the responder becomes a head leader, and the edge becomes inverse. When a head leader interacts as the initiator with a simple leader via some inactive edge the initiator becomes a non-leader, the responder becomes a head leader, and the edge becomes inverse. When the simple leader is the initiator, the head leader is the responder, and the edge is again inactive, the initiator becomes a tail leader, the responder becomes a non-leader, and the edge becomes proper. When a tail leader interacts as the initiator with a simple leader via an inactive edge, the initiator becomes a non-leader, the responder becomes a head leader, and the edge becomes inverse. When the simple leader is the initiator, the tail leader is the responder, and the edge is again inactive, the initiator becomes a tail leader, the responder becomes a non-leader, and the edge becomes proper. These transitions can be formally summarized as follows: $(l, l, 0) \to (k_t, l_h, i)$, $(l_h, l, 0) \to (k, l_h, i)$, $(l, l_h, 0) \to (l_t, k, p)$, $(l_t, l, 0) \to (k, l_h, i)$, and $(l, l_t, 0) \to (l_t, k, p)$. In this manner, the agents become organized in correctly labeled pseudo-path subgraphs (see again their definition and Figure 1).

We now describe how two such pseudo-path 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 pseudo-path 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 having a special label l' indicating that this label 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 takes to another special label 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. 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 label k, and obtains another special label 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 correctly merged into a common correctly labeled pseudo-path graph. See Figure 2 for a graphical step by step example. The correctness of this process, called the *merging process*, is captured by Lemma 2.

Lemma 2. *When the leader endpoints of two distinct correctly labeled pseudo-path 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 pseudo-path graph $L_3 = (K_1 \cup K_2, A_1 \cup A_2 \cup \{e\})$.*



(a) Before merging.

(b) l' moves towards k_t and inverts the labels.

(c) l'' moves towards k' and removes the marks.

(d) l''' moves towards k^t and at arrival becomes l_t.

(e) After merging.

Figure 2: Two pseudo-path subgraphs are merged together.

Proof. We study all possible cases:

L_1 and L_2 are both trivial (they are isolated simple leaders, where "isolated" means that all the edges incident to them are inactive): Then the initiator becomes a tail non-leader, the responder becomes a head leader, and the edge becomes inverse.

L_1 is non-trivial and L_2 is trivial: First assume that the leader of L_1 is a tail leader. If the tail leader is the initiator, then it becomes a non-leader, the responder becomes a head leader (the leader end-point of the new pseudo-path graph L_3), and the edge becomes inverse. Clearly, the added edge points towards the new leader of the path and is correctly inverse, all other edges correctly retain their direction labels, the old leader becomes internal, thus, correctly becomes a non-leader, and the other endpoint remains unaffected, thus, correctly remains a tail non-leader. The cases in which the leader of L_1 is a head leader and those where L_1's leader is the responder are handled similarly.

L_2 is non-trivial and L_1 is trivial: This case is symmetric to the previous one.

L_1 and L_2 are both non-trivial: Assume w.l.o.g. that L_1's leader is the initiator. Then L_2's leader will become a non-leader, which is correct since it will constitute an internal node of the new pseudo-path graph L_3 which will be the result of the merging process. But first it becomes a marked non-leader in order to inform L_1's leader where to stop its movement. L_1's leader goes to a special state that only has effective interactions through active edges. This ensures that it only has effective interactions with its neighbors in the new pseudo-path graph L_3. Additionally, the edge via which the pseudo-path graphs L_1 and L_2 where merged goes to a marked proper state. The goal of the merging process is to change all direction labels of L_1, that is, make the proper inverse and the inverse proper. The reason is that L_1's tail non-leader endpoint will now

become L_3's leader endpoint and, if remain unchanged, L_1's direction labels will be totally wrong for the new pseudo-path graph. L_2's direction labels must remain unchanged since their new leader will be in the same side as before, thus, they will still be correct w.r.t. the direction of the path from L_3's new leader endpoint and its non-leader endpoint. When L_1's leader interacts via a non-marked edge it knows that it interacts with a neighbor that it has not visited yet and which lies on the direction towards L_1's tail non-leader endpoint. Thus, it changes the edge's label, if it is proper it makes it inverse and contrariwise, marks it in order to know its direction towards that endpoint, and jumps to its neighboring node, that is, the neighbor becomes the special leader and the node itself becomes a non-leader. In this manner, the leader keeps moving step by step towards L_1's non-leader endpoint while at the same time fixing the direction labels. Eventually, due to fairness, it will reach the endpoint. At this point it goes to another special leader state whose purpose is to walk the same path in the inverse direction until it meets again the old leader of L_2 which is marked, and, thus, can be identified. It simply follows the marked links while erasing the marks of the links that it crosses. When it finally meets the unique marked agent of L_3 it unmarks it, thus, makes it a normal non-leader, unmarks the only edge that still remains marked, which is the edge that joined L_1 and L_2 and goes to another special leader state whose purpose is to walk again back to L_1's endpoint and then become a normal tail leader, that is, L_3's tail leader. This can be done easily, because now all links have correct direction labels. In fact, it knows that if it interacts as the responder via a proper link or as the initiator via an inverse link, then it must cross that link, because in both cases it will move on step closer to L_1's endpoint. All other interactions are ignored by this special leader. It is easy to see that due to fairness and due to the fact that it can only move towards L_1's endpoint it will eventually reach it. When this happens it becomes a normal tail leader. It must be clear that all internal nodes of L_3 are non-leaders, one endpoint has remained a tail non-leader while the other has become a tail leader, all direction labels are correct, and all other edges that are not part of L_3 but are incident to it have remained inactive. Thus, L_3 is correctly labeled. □

Theorem 5. *The SMPP* Spanning Process *constructs a correctly labeled spanning pseudo-path subgraph of any complete interaction graph G.*

Proof. By definition, we consider isolated simple leaders as trivial pseudo-path graphs. Thus, initially, G is partitioned into n correctly labeled trivial pseudo-path graphs. It is easy to see that correctly labeled pseudo-path graphs never become smaller and, according to Lemma 2, when their leaders interact they are merged into a new pseudo-path graph containing all nodes of the interacting pseudo-path graphs plus an additional edge joining them. Moreover, given that there are two correctly labeled pseudo-path subgraphs in the current configuration there is always the possibility (due to fairness) that these pseudo-path graphs may get merged, because they are correctly labeled which implies that there are always inactive edges joining their leader endpoints, and there is no

other possible effective interaction between two pseudo-path graphs. In simple words, two pseudo-path 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 pseudo-path subgraph of G. □

Theorem 6. *Assume that the interaction graph $G = (V, E)$ is a correctly labeled pseudo-path graph of n agents, where each agent takes its input symbol in a second* state component [8]*. Then there is a 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 [AAD+06, AR07] 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. To make this a little more clear, assume that an agent u has $\mathcal{M}$'s head over the last symbol of its state component (each agent can use up to a constant number of such symbols due to the uniformity property). Now, assume that $\mathcal{M}$ moves its head to the right. Then u must pass control to its right neighbor. To do so, it simply follows a proper edge as the initiator of an interaction or an inverse edge as the responder of an interaction. Similarly, when control must be passed to the left neighbor, the agent follows an inverse edge as the initiator of an interaction or a proper edge as the responder of an interaction. □

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 $\mathcal{O}(n)$ (linear) 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 pseudo-path 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 the impossibility of detecting termination by applying the *reinitialization* technique of [GR09, CMN+10c].

Let us first outline the approach that will be followed in Theorem 7. Whenever two correctly labeled pseudo-path subgraphs get merged, we know that a new correctly labeled pseudo-path graph will be constructed in a finite number of steps. Moreover, termination of the merging process can be detected. When the merging process comes to an end, the unique leader of the new pseudo-path graph does the following. It makes the assumption that the spanning process has come to an end (an assumption that is possibly wrong since the pseudo-path subgraph may not be spanning yet), restores its state component to its input symbol (thus, restarting the TM simulation) and informs its right neighbor to

[8] The first component is used for the labels of the spanning process and, as already mentioned, is called *label component*.

do the same. Restoring the input symbol can be trivially achieved, because the agents can forever keep their input symbols in a read-only *input backup* component. The correctness of this idea is based on the fact that the reinitialization process also takes place when the last two pseudo-path subgraphs get merged into the final spanning pseudo-path subgraph. What happens then is that the TM simulation starts again from the beginning like it had never been executed during the spanning process and Theorem 6 guarantees that the simulation will run correctly if not restarted in future steps. Clearly, it will never be restarted again, because no other merging process will ever take place (a unique spanning pseudo-path subgraph is active and all other edges are inactive).

Theorem 7. $\mathbf{SSPACE}(n) \subseteq \mathbf{MPS}$.

Proof. Take any $p \in \mathbf{SSPACE}(n)$. By Theorem 6 we know that there is a MPP $\mathcal{A}$ that stably computes p on a pseudo-path graph of n nodes. We have to show that there exists a 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 two pseudo-path graphs get merged and the merging process comes to an end it executes the following reinitialization process. The new leader u that resulted from merging becomes marked, e.g. l_i^*. Recall that the new pseudo-path 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 pseudo-path 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 erases them. This can be done easily because the new reinitialization signal will always be traveling from left to right and all old signals will be found to its right; in this manner we know which of them has to be erased. Another possible approach is to block the leader from participating in another merging process before completion of the current pending reinitialization process. This approach is also correct: fairness guarantees that the reinitialization process will terminate in a finite number of steps, thus, the merging process will not be blocked forever.

From Theorem 5 we know that the spanning process executed by $\mathcal{I}$ results in a correctly labeled spanning pseudo-path subgraph of G. The spanning process,

as already mentioned, terminates when the merging of the last two pseudo-path subgraphs takes place and merging also correctly terminates in a finite number of steps (Lemma 2). Moreover, from the above discussion we know that, when this happens, the reinitialization process will correctly reinitialize all agents of the spanning pseudo-path subgraph, thus, all agents in the population. But then, independently of its computation so far (in its own component), $\mathcal{A}$ will run from the beginning on a correctly labeled pseudo-path graph of n nodes (this pseudo-path 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, $p \in \mathbf{MPS}$. See Figure 3 for a graphical step by step example. □

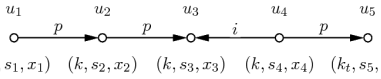
5.3. An Exact Characterization: $\mathbf{MPS} = \mathbf{SNSPACE}(n^2)$

We now extend the ideas used in Section 5.2 in order to establish that $\mathbf{SSPACE}(n^2)$ is a subset of **MPS** showing that **MPS** is a surprisingly wide class. Finally, we improve to $\mathbf{SNSPACE}(n^2)$ and show that the latter inclusion holds with equality, thus, arriving at the following exact characterization of **MPS**: *A predicate is in* **MPS** *iff it is symmetric and is in* $\mathbf{NSPACE}(n^2)$.

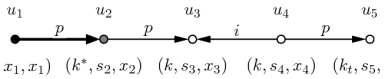
Theorem 8. *Assume that the complete interaction graph $G = (V, E)$ contains a correctly labeled spanning pseudo-path subgraph, where each agent takes its input symbol in a second* state component. *Then there is a MPP $\mathcal{A}$ that when running on such a graph simulates a deterministic TM $\mathcal{M}$ 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 and 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 pseudo-path graph). Consider w.l.o.g. that the left endpoint is a tail leader and the right one the tail non-leader. Each agent can distinguish its neighbors in the pseudo-path graph (in particular, it knows which is the left and which is the right one) from its remaining neighbors, since the latter are via inactive edges. Moreover, the endpoints of the pseudo-path graph can be identified because the pseudo-path graph is correctly labeled (one endpoint is a leader, the other is a tail non-leader, and all intermediate agents are non-leaders). Finally, we assume that the edge states now consist of two components, one used to identify them as active/inactive and the other used by the simulation.

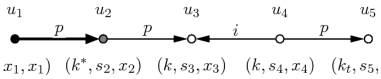
In contrast to Theorem 6 the simulation also makes use of the inactive edges. The agent in control of the simulation is in a special state denoted with a star '$*$'. Since the simulation starts from the left endpoint (tail leader), its state will be l_t^*. When the star-marked leader interacts with its unique right neighbor on the pseudo-path 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



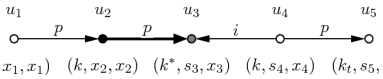
(a) Just after merging. The leader endpoint has the special star mark. The reinitialization process begins.



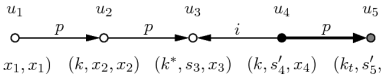
(b) u_1 becomes reinitialized and u_2 obtains the star mark. Note that u_1's first component obtains a bar mark to block u_1 from participating in another merging process.



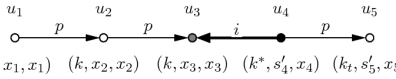
(c) Nothing happens, since only u_1 has been reinitialized (u_2 still has the star mark).



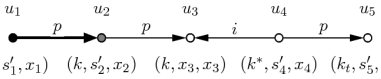
(d) u_2 becomes reinitialized and passes the mark to u_3.



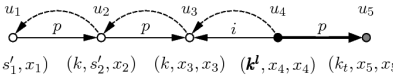
(e) Both u_4 and u_5 have not been reinitialized yet. A simulation step is executed but this is unimportant since both u_4 and u_5 will be reinitialized in future steps and cannot communicate with reinitialized agents (the star mark acts as a separator).



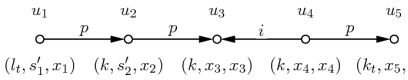
(f) u_3 becomes reinitialized and passes the mark to u_4.



(g) A simulation step is executed normally because both have been reinitialized.



(h) Both become reinitialized and u_4 goes to k^l. The "left" mark will travel to u_1 to remove its mark and indicate the end of the reinitialization process.



(i) After the completion of the reinitialization process. The leader is again ready for merging.

Figure 3: An example of the reinitialization process just after two pseudo-path graphs have been merged together. The agents are named $(u_1, u_2, \ldots, u_5)$. Each agent's state is a 3-vector (c_1, c_2, c_3) where component c_1 contains the label of the agent, c_2 the state of the TM simulation, and c_3 the input backup. The bold edge indicates the pair that has just interacted. The black agent is the initiator and the grey the responder. The states of the corresponding agents are updated in each figure according to their previous states and the state of the edge joining them.

is between the star-marked leader (l_t^*) and the dot non-leader ($\dot{k}$) which can only happen 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.

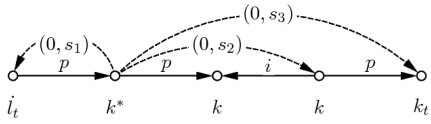
If the simulation has to continue to the right, the interaction $(l_t^*, \dot{k})$ sends the dot agent to state k^r. If it has to proceed left, the dot agent goes to state k^l. An agent in state k^r interacts with its *right* neighbor sending it to dot state whereas a k^l agent does the same for its *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 5 for the special states traversing through the pseudo-path graph. The dot mark's (state's) position in the pseudo-path 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 pseudo-path 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 pseudo-path 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.

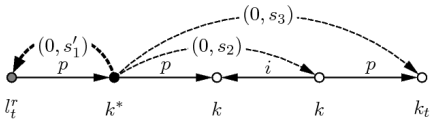
When the control of the simulation reaches a non-leader agent (e.g. from the left leader endpoint side) in order to visit its first edge it places the dot mark to the left leader endpoint and then to the next (on the right) non-leader and so forth. If the dot mark reaches the star-marked agent (in the previous example from the left endpoint side) then it moves the dot to the closer (in the pseudo-path 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. See Figure 4 for a graphical example.

Note that the assumption that only inactive edges are used by the simulation to hold $\mathcal{M}$'s tape is not restrictive. The previously described mechanism can be extended (using a few more special states and little more complicated interaction sequences) to also use the active edges, as well as the agents, for the simulation.

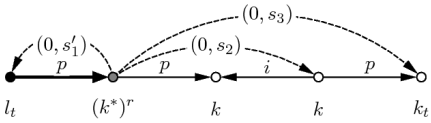
However the inactive edges of each agent towards the rest of the population are asymptotically sufficient for the simulation discussed so far. □



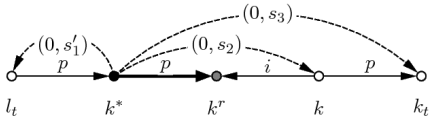
(a) The agent in k^* controls now the simulation.



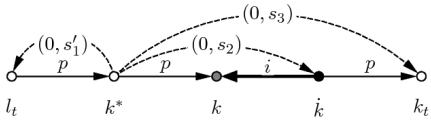
(b) A step of the simulation is executed on the inactive edge. The TM says "right" so k^* must next run the simulation on the first inactive edge to the right.



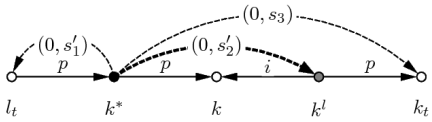
(c) Mark r travels to the right until it meets the first agent that has an incoming inactive edge from k^*.



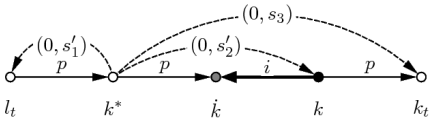
(d) The mark still travels.



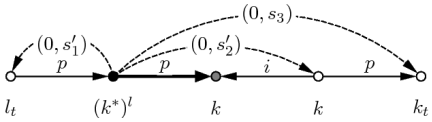
(e) The correct agent was found. The special dot mark will make the simulation run on the next inactive edge.



(f) A step of the simulation is executed. The TM says now "left" so the simulation must next use again the previous inactive edge.



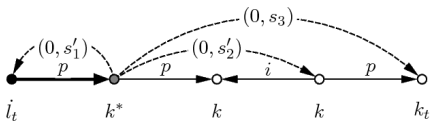
(g) The dot mark is placed on the wrong agent (because we only use for the simulation the inactive outgoing edges of k^*).



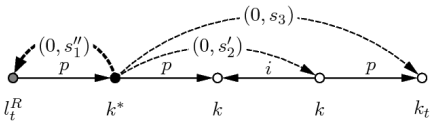
(h) The error is detected because the interaction happened through an active edge. The l mark continues its travel to the left.

Figure 4: An example of simulating a $\mathcal{O}(n^2)$-space deterministic TM. The simulation is performed on the second (state) component of the inactive edges (those whose first component is 0). The bold edge indicates the pair that has just interacted. The black agent is the initiator and the grey the responder. The states of the corresponding agents are updated in each figure according to their previous states and the state of the edge joining them. We only present the effective interactions that take place; it is possible that between two subsequent figures a finite number of ineffective interactions have taken place. Fairness guarantees that an effective interaction that is always possible to occur will eventually occur (continued...).

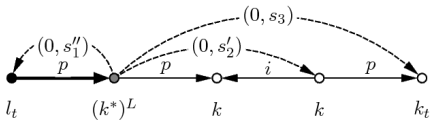
We present now an SMPP that simulates a deterministic TM by using asymptotically all its distributed memory as its tape cells. The correctness of the simulation is proved in Theorem 9 which concludes that $\mathbf{SSPACE}(n^2) \subseteq \mathbf{MPS}$. The main idea is similar to that in the proof of Theorem 7 (based again on the reinitialization technique). We assume that the edge states now consist of two



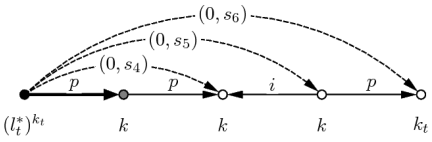
(i) The dot mark is placed at the leader endpoint.



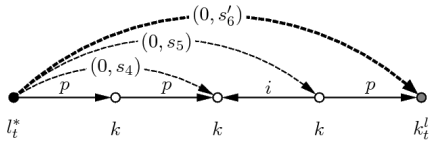
(j) A step of the simulation is executed. The TM says again "left" but it is already at the leftmost agent. A special R mark is created to change the agent that has control of the simulation.



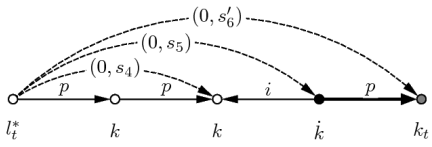
(k) The agent that has control was detected. The L mark will pass control to the left neighbor.



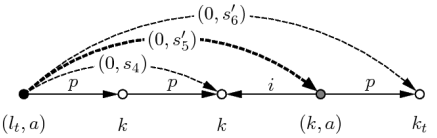
(l) The star mark was passed to the left. The left endpoint has now control of the simulation and will use its own inactive outgoing edges. The 'k_t' mark indicates that it must continue from its last outgoing edge (acts like an artificial dot mark over the non-leader endpoint.



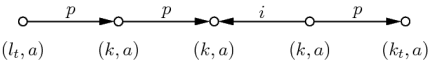
(m) A step of the simulation is executed. The TM says again "left".



(n) The dot mark was passed to the left.



(o) A step of the simulation is executed. The TM accepts and an accepting component is created in both agents.



(p) The accepting component is in a finite number of steps (due to fairness) propagated to all agents and the population also accepts.

Figure 4: An example of simulating a $\mathcal{O}(n^2)$-space deterministic TM.

components, one used to identify them as active/inactive and the other used by the simulation (protocol $\mathcal{A}$ from Theorem 8).

This time, the reinitialization process attempts to reinitialize not only all agents of a pseudo-path graph but also all of their outgoing edges. We begin by describing the reinitialization process in detail. Whenever the merging process of two pseudo-path graphs comes to an end, resulting in a new pseudo-path 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 pseudo-path 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 pseudo-path graph). When it isn't, the correctness of the process is captured by the following lemma.

Lemma 3. *Let L and L' be two distinct pseudo-path 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'. The reason is that the reinitialization process has some effective interaction via an inactive edge only when the edge's tail is in a star state and its head is in a dot state. But these states can only appear in a

pseudo-path graph while it is executing a reinitialization process. Thus, if this is the case, L's reinitialization process will get executed as if L' didn't exist.

If L' is also running its own reinitialization process, then there are two possible conflicts:

1. *A star agent of L interacts with a dot agent of L'*: In this case, the dot agent of L' simply becomes a bar non-leader, and the star agent of L maintains its state. Thus, L's reinitialization process is not affected.
2. *A star agent of L' interacts with a dot agent of L*: Now the opposite happens and L's reinitialization process is clearly affected. But what really happens is that the dot agent of L becomes a bar non-leader via a wrong interaction. But this does not delay the progress of the reinitialization process; it only makes it take one step forward without reinitializing the correct edge.

In the first case the process is not affected at all and in the second the process cannot be delayed (it simply takes some steps without reinitializing the corresponding edges), thus, it always terminates in a finite number of steps (due to fairness and by taking into account the discussion preceding this lemma) and L will be in finite time ready to participate in another merging process. □

Theorem 9. SSPACE$(n^2) \subseteq$ MPS.

Proof. Lemma 3 guarantees that the spanning process terminates with a spanning pseudo-path subgraph with active edges, while all remaining edges in G are inactive. In this case, since a unique pseudo-path subgraph exists (the spanning one), there can be no conflict and it must be clear that all agents and all edges will get correctly reinitialized. When the last reinitialization process ends, protocol $\mathcal{A}$ starts its last execution, this time on the correct reinitialized system. We finally ensure that the simulation does not ever alter the agent labels used by the spanning and reinitialization processes. These labels are read-only from the viewpoint of $\mathcal{A}$. In the proof of Theorem 8 we made $\mathcal{A}$ put marks on the labels in order to execute 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. The theorem follows by taking into account Theorem 8 stating that this construction is all that $\mathcal{A}$ needs to get executed correctly. □

We kept the above discussion and proofs somewhat descriptive in order to avoid their many low-level details. A formal constructive proof can be found in the corresponding Technical Report [CMN$^+$10b].

We next show how one can add some nondeterminism to the above simulation and, as a consequence, further improve the inclusion of Theorem 9.

Theorem 10. SNSPACE$(n^2) \subseteq$ MPS.

Proof. We modify the deterministic TM of Theorem 9 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 pseudo-path 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 now deal with the inverse direction of Theorem 1. That is, we are going to show that **MPS** $\subseteq$ **SNSPACE**(n^2). This, as alluded to in the proof idea of Theorem 1, is a not so difficult task. First recall that m denotes the number of edges of the interaction graph.

Definition 3. *Let* **DMP** *(***UMP***) be the class of predicates stably computable by the MPP model in any family $\mathcal{G}$ of directed (undirected) and connected interaction graphs.*

Theorem 11. *All predicates that belong to the classes* **DMP** *and* **UMP** *are also in* **NSPACE**(m).

Proof. Let $\mathcal{A}$ be a mediated protocol that stably computes such a predicate p in a family of graphs $\mathcal{G}$, and let $G \in \mathcal{G}$ be any graph of this family. Since G is always connected, we have that $m \geq n-1$. A network configuration can be represented explicitly, by storing a state per node and a state per edge of G. This takes $\mathcal{O}(m)$ space (in fact it is $m+n$, but since $m \geq n-1$, m dominates n). The language corresponding to p is defined as $L = \{x \mid x \in X^* \text{ and } p(x) = 1\}$.

We present a nondeterministic Turing machine $M_{\mathcal{A}}$ that decides L in space $\mathcal{O}(m)$. $M_{\mathcal{A}}$ works as follows: To accept input x, $M_{\mathcal{A}}$ must verify two conditions: (i) that there exists a configuration C reachable from $I(x)$ (the initial configuration corresponding to x), in which all agents output 1, and (ii) that there is no configuration C' reachable from C, in which at least one agent outputs 0. The first condition is verified by guessing and checking a sequence of network configurations, starting from $I(x)$ and reaching such a C. $M_{\mathcal{A}}$ guesses a C_{i+1} each time, verifies that $C_i \rightarrow C_{i+1}$ (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 $\geq \log n$ (see Immerman-Szelepcsényi theorem [Imm88, Sze87] or [Pap94] pages $151-153$). Thus, $M_{\mathcal{A}}$ decides L in $\mathcal{O}(m)$ space. □

Theorem 11 has the following immediate consequence.

Corollary 2. **MPS** $\subseteq$ **SNSPACE**(n^2).

Proof. Any $p \in \mathbf{MPS}$ is symmetric (see Lemma 1) and according to Theorem 11 belongs to $\mathbf{NSPACE}(m)$. Finally, notice that **MPS** deals with complete interaction graphs, in which $m = \mathcal{O}(n^2)$. □

We have now arrived at the exact characterization of **MPS** stated in Theorem 1, that is, $\mathbf{MPS} = \mathbf{SNSPACE}(n^2)$. One direction follows from Theorem 10 and the inverse direction from Corollary 2.

6. Conclusions - Future Research Directions

We have proposed the mediated population protocol model, an enhancement of the population protocol model in which also the edges of the interaction graph are capable of storing fixed-size information. Our model not only preserves the most significant properties of population protocols, namely, uniformity and anonymity, but also dramatically extends the class of computable predicates, from semilinear to all symmetric predicates in $\mathbf{NSPACE}(n^2)$. In other words, we have been able to show that the MPP model in the fully symmetric case is equivalent to a NTM of $\mathcal{O}(n^2)$ space that computes symmetric predicates. To show this, we first demonstrated how the MPP model can organize the agents into a spanning pseudo-path subgraph. We then showed that, by reinitializing computation, the agents are able to simulate a NTM on the edges that are not part of the pseudo-path graph. Each agent can order its outgoing edges by exploiting the fact that their heads are agents of the pseudo-path graph, who, in turn, are ordred according to their distance from the "left" endpoint of the pseudo-path graph.

Many interesting problems remain open in the rapidly growing body of the population protocols literature. First of all, there is an interesting open question left open by this article. Can the exact characterization for the complete case be generalized to connected interaction graphs? In particular, is it possible to show that the corresponding class of computable predicates is equal to $\mathbf{SNSPACE}(n + m)$ (recall that m denotes the number of edges)? Note that in this case the pseudo-path construction cannot be applied, since the interaction graph may contain any tree structure. Possibly the solution lies in the construction of some sort of spanning tree, but again one has to be very careful, because node degrees that depend on the population size may prevent the protocol from keeping an ordering on its incident edges. Moreover, are the MPP and PM models *fault-tolerant*? What preconditions are needed in order to achieve satisfactory fault-tolerance? Additionally, as pointed out by [AAC$^+$05] and [CMS10b], population-protocol-like models have the ability to stably decide properties of the interaction graph, which is of outstanding importance for almost any distributed system. What are the exact characterizations of the classes of stably decidable graph languages by the MPP and PM models? What are the most appropriate real-life scenarios for applying the MPP model? What is the computational power of the PM model for space bounds $f(n)$ between $\log \log n$ and $\log n$? As alluded to in Section 2, the only things that we do know about this particular space bound is that above $\log \log n$ and below $\log n$ the semilinear and

the $nf(n)$ behaviors, respectively, of the model cease. [CDF+09] 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*. Moreover, the time complexity of protocols based on some probabilistic operational assumption concerning the scheduler has not been studied yet for none of the MPP, PM, and Community Protocol models. On the other hand, some works have performed such a study for population protocols [AAD+04, AAD+06, AAE08a, AAE08b] and it is expected that some of the existing methods may also be applicable to these models. Are there more efficient, possibly logic-based, verification solutions for population protocols than those presented in [CMS10a]? Verification methods for MPPs, Community Protocols, and PM protocols are still totally unknown, although some of the ideas of [CMS10a] may also be applicable to these models. Finally, one can study a variant of the classical model in which the agents interact in groups of $k > 2$ agents and not in pairs (like a broadcast medium). Of course, assuming a constant state space, the computational power of the model is semilinear (see, e.g., Theorem 9, [AAER07]). However, the time efficiency of this variant is open.

7. Acknowledgements

We would like to thank Stavros Nikolaou and Andreas Pavlogiannis for their valuable contribution to some preliminary versions of this work. We would also like to thank some anonymous reviewers for their valuable comments that helped us to improve our work substantially.

References

[AAC+05] D. Angluin, J. Aspnes, M. Chan, M. J. Fischer, H. Jiang, and R. Peralta. Stably computable properties of network graphs. In V. K. Prasanna, S. Iyengar, P. Spirakis, and M. Welsh, editors, *Distributed Computing in Sensor Systems: First IEEE International Conference, DCOSS 2005, Marina del Rey, CA, USE, June/July, 2005, Proceedings*, volume 3560 of *Lecture Notes in Computer Science*, pages 63–74. Springer-Verlag, June 2005.

[AAD+04] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 290–299, New York, NY, USA, 2004. ACM.

[AAD+06] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *Distributed Computing* [AAD+04], pages 235–253.

[AAE06] D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear. In *PODC '06: Proceedings of the 25th annual ACM Symposium on Principles of Distributed Computing*, pages 292–299, New York, NY, USA, 2006. ACM Press.

[AAE08a] D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21[3]:183–199, September 2008.

[AAE08b] D. Angluin, J. Aspnes, and D. Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21[2]:87–102, July 2008.

[AAER07] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20[4]:279–304, November 2007.

[ACD+11] C. Àlvarez, I. Chatzigiannakis, A. Duch, J. Gabarró, O. Michail, S. Maria, and P. G. Spirakis. Computational models for networks of tiny artifacts: A survey. *Computer Science Review*, 5[1], January 2011.

[ADGS09] C. Àlvarez, A. Duch, J. Gabarro, and M. Serna. Sensor field: A computational model. In *Algorithmic Aspects of Wireless Sensor Networks: 5th International Workshop, ALGOSENSORS 2009, Rhodes, Greece, July 10-11, 2009. Revised Selected Papers*, pages 3–14, Berlin, Heidelberg, 2009. Springer-Verlag.

[AR07] J. Aspnes and E. Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science*, 93:98–117, October 2007.

[ASS10] C. Àlvarez, M. Serna, and P. G. Spirakis. On the computational power of constant memory sensor fields. Technical Report FRONTS-TR-2010-10, 2010.

[BCCK08] O. Bournez, J. Chalopin, J. Cohen, and X. Koegler. Playing with population protocols. In *CSP*, pages 3–15, 2008.

[BCK+09] O. Bournez, P. Chassaing, X. Koegler, L. Gerin, and J. Cohen. On the convergence of population protocols when population goes to infinity. *Applied Mathematics and Computation*, 215:1340–1350, 2009.

[BCM+07] J. Beauquier, J. Clement, S. Messika, L. Rosaz, and B. Rozoy. Self-stabilizing counting in mobile sensor networks. In *PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 396–397, New York, NY, USA, 2007. ACM.

[CDF+09] 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)*, volume 5923 of *Lecture Notes in Computer Science*, pages 33–47, Berlin, Heidelberg, 2009. Springer-Verlag.

[CMN+10a] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. G. Spirakis. All symmetric predicates in $NSPACE(n^2)$ are stably computable by the mediated population protocol model. In *35th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 6281 of *Lecture Notes in Computer Science*, pages 270–281. Springer-Verlag, August 23–27 2010.

[CMN+10b] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. 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, Patras, Greece, 2010.

[CMN+10c] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. G. Spirakis. Passively mobile communicating logarithmic space machines. *CoRR*, abs/1004.3395, 2010. Also FRONTS-TR-2010-16.

[CMN+10d] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. G. Spirakis. Passively mobile communicating machines that use restricted space. *CoRR*, abs/1012.2440, 2010.

[CMS09a] I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Mediated population protocols. In *36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, volume 5556 of *LNCS*, pages 363–374, July 2009.

[CMS09b] I. Chatzigiannakis, O. Michail, and P. 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, Berlin, Heidelberg, 2009. Springer-Verlag.

[CMS10a] I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Algorithmic verification of population protocols. In *12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 6366 of *Lecture Notes in Computer Science*, pages 221–235. Springer-Verlag, September 2010.

[CMS10b] I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Stably decidable graph languages by mediated population protocols. In *12th International Symposium on Stabilization, Safety, and Security of*

Distributed Systems (SSS), volume 6366 of *Lecture Notes in Computer Science*, pages 252–266. Springer-Verlag, September 2010.

[CS08] I. Chatzigiannakis and P. G. Spirakis. The dynamics of probabilistic population protocols. In *22nd international symposium on Distributed Computing (DISC)*, volume 5218 of *Lecture Notes in Computer Science*, pages 498–499, Berlin, Heidelberg, 2008. Springer-Verlag.

[DGFGR06] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. When birds die: Making population protocols fault-tolerant. In *IEEE 2nd Intl Conference on Distributed Computing in Sensor Systems (DCOSS)*, volume 4026 of *Lecture Notes in Computer Science*, pages 51–66. Springer-Verlag, June 2006.

[GR09] 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)*, volume 5556 of *Lecture Notes in Computer Science*, pages 484–495. Springer-Verlag, 2009.

[GS66] S. Ginsburg and E. H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17[5]:935–938, 1988.

[MCS10] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. *New Models for Population Protocols*. N. A. Lynch (Ed), Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2010. To appear.

[Pap94] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[Pre29] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes-Rendus du I Congrès de Mathématiciens des Pays Slaves*, pages 92–101, 1929.

[Spi10] P. G. Spirakis. *Theoretical Aspects of Distributed Computing in Sensor Networks*, chapter Population Protocols and Related Models. Springer-Verlag, 2010.

[Sze87] R. Szelepcsényi. The method of forcing for nondeterministic automata. *Bulletin of the EATCS*, 33:96–99, 1987.