Theoretical Computer Science 412 (2011) 6469–6483

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Passively mobile communicating machines that use restricted space[☆,☆☆]

Ioannis Chatzigiannakis [a,b], Othon Michail [a], Stavros Nikolaou [a,b,*], Andreas Pavlogiannis [c], Paul G. Spirakis [a,b]

[a] *Research Academic Computer Technology Institute (CTI), Patras, Greece*
[b] *Computer Engineering and Informatics Department (CEID), University of Patras, Greece*
[c] *Department of Computer Science, University of California, Davis (UCDavis), United States*

A R T I C L E   I N F O

Article history:
Received 6 January 2011
Received in revised form 15 June 2011
Accepted 1 July 2011
Communicated by D. Peleg

Keywords:
Distributed computing
Pervasive environments
Diffuse computation
Passive mobility

A B S T R A C T

We propose a new theoretical model for passively mobile wireless sensor networks, called *PM*, standing for *passively mobile machines*. The main modification w.r.t. the population protocol model (Angluin et al., 2006) [30] is that agents now, instead of being automata, are Turing Machines. We provide general definitions for unbounded memories, but we are mainly interested in computations upper-bounded by plausible space limitations. However, we prove that our results hold for more general cases. We focus on *complete interaction graphs* and define the complexity classes **PMSPACE**$(f(n))$ parametrically, consisting of all predicates that are stably computable by some PM protocol that uses $\mathcal{O}(f(n))$ memory in each agent. We provide a protocol that generates unique identifiers from scratch only by using $\mathcal{O}(\log n)$ memory, and use it to provide an exact characterization of the classes **PMSPACE**$(f(n))$ when $f(n) = \Omega(\log n)$: *they are precisely the classes of all symmetric predicates in* **NSPACE**$(nf(n))$. As a consequence, we obtain a space hierarchy of the PM model when the memory bounds are $\Omega(\log n)$. We next explore the computability of the PM model when the protocols use $o(\log\log n)$ space per machine and prove that **SEM** $=$ **PMSPACE**$(f(n))$ when $f(n) = o(\log\log n)$, where **SEM** denotes the class of the semilinear predicates. Finally, we establish that the minimal space requirement for the computation of non-semilinear predicates is $O(\log\log n)$.

1. Introduction: population protocols

Theoretical models for wireless sensor networks (WSNs) have received great attention over the past few years. Recently, Angluin et al. [3] proposed the *population protocol* (*PP*) model. Their aim was to model sensor networks consisting of tiny computational devices (called *agents*) with sensing capabilities that follow some unpredictable and uncontrollable mobility pattern. Due to the minimalistic nature of their model, the class of computable predicates was proven to be fairly small: it is the class of *semilinear predicates* [4], which does not e.g. support multiplications, exponentiations, and many other important operations on input variables. Additionally, according to the work of Delporte-Gallet et al. [5], we only know how to transform any protocol that computes a function in the failure-free model into a protocol that can tolerate

☆ 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 Chatzigiannakis et al. (2010) [1] and Chatzigiannakis et al. (2010) [2].

* Corresponding address: Research Academic Computer Technology Institute (CTI), N. Kazantzaki Str., Patras University Campus, Rio, P.O. Box 1382, 26500, Greece. Tel.: +30 2610 960200; fax: +30 2610 960490.

E-mail addresses: ichatz@cti.gr (I. Chatzigiannakis), michailo@cti.gr (O. Michail), nikolaou@cti.gr, snikolaou@ceid.upatras.gr (S. Nikolaou), apavlogiannis@ucdavis.edu (A. Pavlogiannis), spirakis@cti.gr (P.G. Spirakis).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.07.001

$\mathcal{O}(1)$ crash failures.[1] Moreover, Guerraoui and Ruppert [6] showed that any function computable by a population protocol tolerating one Byzantine agent is trivial. On the other hand, Angluin, Aspnes, and Eisenstat [7] 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 briefly discussed in Section 1.1).

The work of Angluin et al. shed light and opened the way toward 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. These models draw most of their beauty precisely from their inability to organize interactions in a convenient and predetermined way. In fact, the PP model was the minimalistic starting point of this area of research. Most efforts are now toward 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 [8,6].

In this work, we want to allow the agents to use $f(n)$ space for various f, where n is the population size (i.e. the number of agents), while preserving the *uniformity* and *anonymity* properties of PPs. We think of each agent as being a Turing Machine.[2] This leads us to propose a new theoretical model for passively mobile sensor networks, called the *PM* model. It is a model of Passively mobile Machines (that we have been calling agents) with sensing capabilities, equipped with two-way communication. We initially focus on PM protocols that use $\mathcal{O}(\log n)$ memory, which is an interesting space bound since (as we shall prove) it allows the assignment of unique identifiers[3] to the agents of the population and plays a major role on establishing the computational power of the model. In addition, we explore the computability of the PM model on different space bounds in order to get an insight of the trade-off between computational power and resource (memory) availability. For example, does more available memory to the agents imply increased computational power? How are the computational capabilities affected under modifications of the available memory? As we shall see, in PM protocols that use $f(n) = \Omega(\log n)$ space, agents can be organized into a distributed NTM that makes use of all the available space. In the case, where $f(n) = o(\log \log n)$ however, we show that the PM protocols are computationally equal to population protocols. Thus, we provide exact characterizations for the input symmetric computations performed by communicating TMs using the above space bounds. Some preliminary versions of the results in this paper have also appeared in [1,2].

1.1. Other previous work

In [3], the *probabilistic population protocol* model was proposed, in which the scheduler selects randomly and uniformly the next pair to interact. Some recent work has concentrated on performance, supported by this random scheduling assumption (see e.g. [9]). [10] 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. [11,12] considered a huge population hypothesis (population going to infinity), and studied the dynamics, stability and computational power of probabilistic population protocols by exploiting the tools of continuous nonlinear dynamics.

In addition, several extensions of the basic model have been proposed in order to more accurately reflect the requirements of practical and more powerful systems. The *mediated population protocol* (*MPP*) model of [8] was based on the additional assumption that each edge of the interaction graph is a finite storage. It has been recently proven [13] that in the case of complete graphs the corresponding class of stably computable predicates is the symmetric subclass of **NSPACE**(n^2), rendering the MPP model extremely powerful (for a thorough presentation of the MPP model see [14]). Guerraoui and Ruppert [6] made another natural assumption: each agent has its own unique id and can store up to a constant number of other agents' ids. In this model, which they named the *Community Protocol* model, the only permitted operation on ids is comparison. It was proven that the corresponding class consists of all symmetric predicates in **NSPACE**$(n \log n)$. In [15], Angluin et al. studied what properties of restricted interaction graphs are stably computable, 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. In [16], MPP's ability to decide graph properties was studied and it was proven that connectivity is undecidable. Another direction is to allow some heterogeneity in the model, so that some agents have more computational power or additional capabilities than others. For example, a base station can be an additional part of the network with which the agents are allowed to communicate [17]. Such an addition allowed for self-stabilizing algorithms that count the number of agents in the network [17]. Recently, Bournez et al. [18] investigated the possibility of studying population protocols via gametheoretic approaches. For some introductory texts to the subject of PPs see [19–21] and for a survey on mediated population protocols see [22]. Finally, the *Static Synchronous Sensor Field* (SSSF) [23,24] 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 [25] for a joint survey on population-protocol-like models and static synchronous sensor fields.

---

1 Although the letter 'O' is usually used in the Complexity Theory literature for the big-O notation, we have chosen here to use its calligraphic version '$\mathcal{O}$' in order to avoid confusion with the output function of protocols.

2 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.

3 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. Our results: roadmap

In Section 3, we begin with a formal definition of the PM model. The section proceeds with a thorough description of the functionality of the systems under consideration and then provides definitions of *configurations* and *fair executions*. In Section 4, first *stable computation* and the family of classes **PMSPACE**($f(n)$) (stably computable predicates by the PM model using $\mathcal{O}(f(n))$ space in each agent) are defined. In Section 5, we give two examples of PM protocols where $\mathcal{O}(\log n)$ space is used in each agent; since those compute non-semilinear predicates, it is established that PM protocols using $\mathcal{O}(\log n)$ space are strictly stronger than population protocols. In Section 6, we show that the PM model using $\mathcal{O}(f(n))$ space can simulate a NTM (Theorem 4) of space $\mathcal{O}(nf(n))$ for any $f(n) = \Omega(\log n)$. This along with Theorem 5, where we prove that **PMSPACE**($f(n)$) is a subset of the symmetric subclass of **NSPACE**($nf(n)$), **SNSPACE**($nf(n)$), provide the following exact characterizations: **PMSPACE**($f(n)$) = **SNSPACE**($nf(n)$) for all $f(n) = \Omega(\log n)$. Based on the results of this section, we establish a space hierarchy theorem for the PM model, when the corresponding protocols use $\Omega(\log n)$ space (Theorem 9). In Section 7, we examine the interesting case of the $o(\log\log n)$ space bounded protocols, showing that this particular bound acts as a computability threshold. In fact, we show that **PMSPACE**($f(n)$) is equal to the class of semilinear predicates when $f(n) = o(\log\log n)$ and a proper superset of the semilinear predicates when $f(n) = \Omega(\log\log n)$. Finally, in Section 8 we conclude and discuss some interesting open problems.

3. The model

In this section, we formally define the PM model and describe its functionality. In what follows, we denote by $G = (V, E)$ the (directed) interaction graph: V is the set of agents, or *population*, and E is the set of permissible ordered pairwise interactions between these agents. We provide definitions for general interaction graphs and unbounded memories, although in this work we deal with complete interaction graphs only and we are mainly interested in computations that are space bounded by a logarithm of the population size. We generally denote by n the population size (i.e. $n = |V|$).

Definition 1. A *PM* protocol is a 6-tuple $(X, \Gamma, Q, \delta, \gamma, q_0)$ where X, Γ and Q are all finite sets and

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

Each agent is equipped with the following:

- A *sensor* in order to sense its environment and receive a piece of the input.
- Four read/write *tapes*: the *working tape*, the *output tape*, the *incoming message tape* and the *outgoing message tape*. We assume that all tapes are bounded to the left and unbounded to the right.
- A *control unit* that contains the state of the agent and applies the transition functions.
- Four *heads* (one for each tape) that read from and write to the cells of the corresponding tapes and can move one step at a time, either to the left or to the right, or remain stationary.
- A binary *working flag* either set to 1 meaning that the agent is *working* internally or to 0 meaning that the agent is *ready* for interaction.

Initially, all agents are in state q_0, their working flag is set to 1, and all their cells contain the *blank symbol* $\sqcup$. We assume that all agents concurrently receive their sensed input (different agents may sense different data) as a response to a global start signal. The input to each agent is a symbol from X and is written on the leftmost cell of its working tape. We call an *input assignment* to the population, any string $x = \sigma_1\sigma_2 \ldots \sigma_n \in X^*$, with n being the size of the population. If we assume an ordering on V, the input to agent i is the symbol σ_i, $1 \le i \le n$.

When its working flag is set to 1 we can think of an agent working as a usual multitape TM (with the additional step of writing the working flag). In particular, while the working flag is set to 1 the internal transition function δ is applied, the control unit reads the symbols under the heads and its own state, updates all of them, moves each head one step to the left or to the right or keeps it stationary, and sets the working flag to 0 or 1, according to δ.

As it is common in the PP literature, an *adversary* selects ordered pairs of agents (edges from E) to interact. Assume now that two agents u and υ are about to interact with u being the *initiator* of the interaction and υ being the *responder*, i.e. the interacting pair is (u, υ). Let $f : V \to \{0, 1\}$ be a function returning the current value of each agent's working flag. If at least one of $f(u)$ and $f(\upsilon)$ is equal to 1, then nothing happens, because at least one agent is still working internally. Otherwise, both agents are ready and an *interaction* is established. In the latter case, the external transition function γ is applied, the states of the agents are updated accordingly, the outgoing message of the initiator is copied to the leftmost cells of the incoming message tape of the responder (replacing its contents and writing $\sqcup$ to all other previously non-blank cells) and vice versa (we call this the *message swap*), and finally the working flags of both agents are again set to 1.[4] These operations

[4] These operations could be handled by the protocols themselves, but then protocol descriptions would become awkward. So, we simply think of them as automatic operations performed by the hardware.

are also considered as atomic, which intuitively means that the interacting agents cannot take part in another interaction before the completion of these operations.

Since each agent is a TM, we use the notion of a configuration to capture its "state". An *agent configuration* is a tuple $(q, l_w, r_w, l_o, r_o, l_{im}, r_{im}, l_{om}, r_{om}, f)$, where $q \in Q$, $l_j, r_j \in \Gamma^*$ for $j \in \{w, o, im, om\}$, and $f \in \{0, 1\}$. We denote by q the state of the control unit, by l_w (l_o, l_{im}, l_{om}) the string of the working (output, incoming message, outgoing message) tape to the left of the head (including the symbol scanned), by r_w (r_o, r_{im}, r_{om}) the string of the working (output, incoming message, outgoing message) tape to the right of the head (excluding the blank cells), and by f the working flag which indicates whether the agent is ready to interact ($f = 0$) or carrying out some internal computation ($f = 1$). We call an agent configuration *initial* if the agent is in state q_0, all its tape cells contain the blank symbol except for the leftmost cell of the working tape that contains its input symbol and the flag bit is 0. Let $\mathcal{B}$ be the set of all agent configurations. Given two agent configurations $A, A' \in \mathcal{B}$, we say that A *yields* A' if A' follows A by a single application of δ.

A *population configuration* is a mapping $C : V \to \mathcal{B}$, specifying the agent configuration of each agent in the population. A population configuration specifying the initial agent configuration of each of the population's agents is called *initial population configuration*. Note that every input assignment corresponds to an initial configuration of the population in which each agent is in state q_0 and has a symbol of the input assignment written in its working tape. Let C, C' be population configurations and let $u \in V$. We say that C *yields* C' via *agent transition u*, denoted $C \stackrel{u}{\to} C'$, if $C(u)$ yields $C'(u)$ and $C'(w) = C(w), \forall w \in V - \{u\}$.

Denote by $q(A)$ the state component of an agent configuration A and similarly for the other components (e.g. $l_w(A)$, $r_{im}(A)$, $f(A)$, and so on). Let $s_{tp}(A) = l_{tp}(A)r_{tp}(A)$, that is, we obtain by concatenation the whole contents of tape $tp \in \{w, o, im, om\}$. Given a string s and $1 \le i, j \le |s|$ denote by $s[\ldots i]$ its prefix $s_1 s_2 \ldots s_i$ and by $s[j \ldots]$ its suffix $s_j s_{j+1} \ldots s_{|s|}$. If $i, j > |s|$ then $s[\ldots i] = s\sqcup^{i-|s|}$ (i.e. $i - |s|$ blank symbols appended to s) and $s[j \ldots] = \varepsilon$. For any external transition $\gamma(q_1, q_2) = (q_1', q_2')$ define $\gamma_1(q_1, q_2) = q_1'$ and $\gamma_2(q_1, q_2) = q_2'$. Given two population configurations C and C', we say that C *yields* C' via *encounter* $e = (u, \upsilon) \in E$, denoted $C \stackrel{e}{\to} C'$, if one of the following two cases holds:
Case 1 (only for this case, we define $C^u \equiv C(u)$ to avoid excessive number of parentheses):

- $f(C(u)) = f(C(\upsilon)) = 0$, which guarantees that both agents u and υ are ready for interaction under the population configuration C.
- $C'(u) = (\gamma_1(q(C^u), q(C^\upsilon)), l_w(C^u), r_w(C^u), l_o(C^u), r_o(C^u), s_{om}(C^\upsilon)[\ldots |l_{im}(C^u)|], s_{om}(C^\upsilon)[|l_{im}(C^u)| + 1 \ldots], l_{om}(C^u), r_{om}(C^u), 1)$,
- $C'(\upsilon) = (\gamma_2(q(C^u), q(C^\upsilon)), l_w(C^\upsilon), r_w(C^\upsilon), l_o(C^\upsilon), r_o(C^\upsilon), s_{om}(C^u)[\ldots |l_{im}(C^\upsilon)|], s_{om}(C^u)[|l_{im}(C^\upsilon)| + 1 \ldots], l_{om}(C^\upsilon), r_{om}(C^\upsilon), 1)$, and
- $C'(w) = C(w), \forall w \in V - \{u, \upsilon\}$.

Case 2:

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

Generally, we say that C *yields* (or *can go in one step to*) C', and write $C \to C'$, if $C \stackrel{e}{\to} C'$ for some $e \in E$ (via encounter) or $C \stackrel{u}{\to} C'$ for some $u \in V$ (via agent transition). We say that C' is *reachable* from C, and write $C \stackrel{*}{\to} C'$ if there is a sequence of population configurations $C = C_0, C_1, \ldots, C_t = C'$ such that $C_i \to C_{i+1}$ holds for all $i \in \{0, 1, \ldots, t-1\}$. An *execution* is a finite or infinite sequence of population configurations $C_0, C_1 \ldots$, where C_0 is an initial configuration and $C_i \to C_{i+1}$. An infinite execution is *fair* if for all population configurations C, C' such that $C \to C'$, if C appears infinitely often in an execution then so does C'. This global *fairness condition* is a restriction imposed on the adversary to ensure that the protocol makes progress. A *computation* is an infinite fair execution.

The *space used by an agent running any protocol* $\mathcal{A}$ is the number of tape cells used to store its configuration, that is the sum of the number of tape cells for the contents of its four tapes. In addition, we say that a *PM protocol* $\mathcal{A}$ *uses* $f(n)$ *space* if the maximum space used by any agent for storing any configuration over all computations is $f(n)$. A (N)TM is called $f(n)$ space bounded if for every input of size n (and in any of its computation paths in the case of a NTM) it scans at most $f(n)$ tape cells on any of its (working) tapes. Note that in our simulations throughout Section 6, we use the TM model with one tape, which includes input and working tapes. We do this because the space bounds discussed in that section are all in $\Omega(n)$, thus the $\Omega(n)$ complexity for scanning the input is included. We call a *protocol* $\mathcal{A}$, $f(n)$ *space bounded* if it uses $f(n)$ space.

We assume that the input alphabet X, the tape alphabet Γ, and the set of states Q are all sets whose cardinality is fixed and independent of the population size. Thus, protocol descriptions have also no dependence on the population size and the PM model *preserves uniformity*. Moreover, PM protocols are *anonymous*, they do not have any id. Uniformity and anonymity are two outstanding properties of the basic population protocol model [3].

4. Stably computable predicates

Any mapping $p : X^* \to \{0, 1\}$ is a *predicate on input assignments*.

Definition 2. A predicate on input assignments p is called *symmetric* if for every $x \in X^*$ and any x' which is a permutation of x's symbols, it holds that $p(x) = p(x')$.

In words, permuting the input symbols does not affect the symmetric predicate's outcome. From each predicate p a language L_p is derived that is the set of all strings that make p true or equivalently, $L_p = \{x \in X^* \mid p(x) = 1\}$. In other words, L_p is equal to the support of p, that is $p^{-1}(1)$. A language L_p is symmetric iff predicate p is symmetric, that is, *for each input string $x \in L_p$ any permutation of x's symbols x' also belongs in L_p*. Note that symmetric languages are also known as *commutative languages* [26].

A population configuration C is called *output stable* if for every configuration C' that is reachable from C it holds that $O(C') = O(C)$, where $O(C) \in \{0, 1\}$ according to the output value that all agents agree to. In other words, the system does not change its overall output in any subsequent step and no matter how the computation proceeds. A predicate on input assignments p is said to be *stably computable* by a PM protocol $\mathcal{A}$ in a graph family $\mathcal{U}$ if, for any input assignment $x \in X^*$, any computation of $\mathcal{A}$, on any interaction graph from $\mathcal{U}$ of order $|x|$, contains an output stable configuration in which all agents have $p(x)$ written on their output tape. In what follows, we always assume that the graph family under consideration contains only complete interaction graphs.

We say that a predicate p over X^* belongs to **SPACE**$(f(n))$ (**NSPACE**$(f(n))$) if there exists some deterministic (nondeterministic, resp.) TM that decides L_p using $\mathcal{O}(f(n))$ space, [27]. A computation path of a NTM accepts if it halts in the accept state and rejects if it halts in the reject state. A nondeterministic TM, M, decides a language L_p if for every input x of size n, there is at least one computation path that accepts (i.e. M accepts) if $x \in L_p$ whereas if $x \notin L_p$, all computation paths of M reject (i.e. M rejects). A NTM decides a language L_p using $f(n)$ space if the maximum number of tape cells scanned/used for any input of size n and in any branch of its computation is $f(n)$. These definitions are similar for deterministic TMs; the difference is that there is only one computation path. Throughout this work, we use **SSPACE**$(f(n))$ and **SNSPACE**$(f(n))$ to denote the **SPACE**$(f(n))$'s and **NSPACE**$(f(n))$'s restrictions to symmetric languages, respectively. In addition, we denote by **SEM**, the class of the semilinear predicates, consisting of all predicates definable by first-order logical formulas of Presburger arithmetic (see, e.g., [4]).

Definition 3. Let **PMSPACE**$(f(n))$ be the class of all predicates that are stably computable by some PM protocol that uses $\mathcal{O}(f(n))$ space.

Note that all agents are initially identical (they do not have unique ids) and since the interaction graph is complete and the executions are fair, all predicates in **PMSPACE**$(f(n))$ are symmetric for any function $f(n)$.

5. Two examples

5.1. Multiplication of variables

We present now a PM protocol that stably computes the predicate $(N_c = N_a \cdot N_b, N_c > 0)$ using $\mathcal{O}(\log n)$ space (on the complete interaction graph of n nodes) that is, all agents eventually decide whether the number of cs (N_c) in the input assignment is the product of the number of as (N_a) and the number of bs (N_b). We give a high-level description of the protocol.

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

Note that the number of cs is at most n which means that the c-counter will become at most $\lceil \log n \rceil$ bits long, and the same holds for the a and b counters, so $\mathcal{O}(\log n)$ memory is required in each tape.

Theorem 1. *The above PM protocol stably computes the predicate $(N_c = N_a \cdot N_b)$ using $\mathcal{O}(\log n)$ space.*

Proof. Given a fair execution, eventually all as and bs become $\bar{a}$s and $\bar{b}$s and only one agent in state c will remain, its a-counter containing the total number of as, its b-counter the total number of bs, and its c-counter the total number of cs.

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

Corollary 1. **SEM** $\subsetneq$ **PMSPACE**$(\log n)$

Proof. PM protocols using $\mathcal{O}(\log n)$ space can simulate population protocols and $(N_c = N_a \cdot N_b) \in$ **PMSPACE**$(\log n)$, which is non-semilinear. □

Note that the previously described protocol can be easily extended to also take into account the case $N_c = 0$ by running in parallel (in different state components within their working tape) a population protocol that check the following cases: (a) $N_c = N_a = 0$ and (b) $N_c = N_b = 0$. These two cases are the only ones that $N_c = 0$ and the protocol should accept. Otherwise, due to the fact that all agents have their outputs initialized to 0 and that there are no effective interactions (interactions that change the state of the participating agents) between agents in a and/or b, the protocol will correctly reject. The predicates described in the previous two cases can be computed by population protocols and thus can be simulated in constant space by the agents. If any of the previous predicates hold then the output tapes are set to 1. In any other case, the output tape is written by the agents in state c. We omit the technical details of the previous construction to avoid further confusion.

In the following subsection, we present another PM protocol using $\mathcal{O}(\log n)$ space that computes the non-semilinear predicate $(N_1 = 2^t)$, which provides an alternative route to the previous corollary.

5.2. Power of 2

Here, we present a PM protocol that, using $\mathcal{O}(\log n)$ memory, stably computes the non-semilinear predicate $(N_1 = 2^t)$, where $t \in \mathbb{Z}_{\geq 0}$, on the complete interaction graph of n nodes, that is, all agents eventually decide whether the number of 1s in the input assignment is a power of 2.

The idea is similar to the one presented in the previous section. The set of input symbols is binary. The protocol counts in binary the number of 1s in the input. The sum of 1s is eventually aggregated in one *awake* agent and all other *sleeping* agents copy the former's output value (see e.g. the parity protocol in [3]). The awake agent can easily recognize whether its counter holds a power of 2 and performs this check every time the counter is incremented. Eventually, the awake agent will know the correct answer to the predicate and the rest of population will obtain it.

Note that the counter of 1s can be at most n. Thus, it requires at most $\lceil \log n \rceil$ bits of memory. In addition, the check of whether the counter is a power of 2 can be easily computed by an agent in $\mathcal{O}(\log n)$ space.

6. Space hierarchy of the PM model

In this section, we study the behavior of the PM model for various space bounds. Such a study is of particular interest since it is always important to know what computations is a model capable of dispatching according to the capabilities of the available hardware.

6.1. A lower bound

We prove here that, for space functions $f(n) = \Omega(\log n)$, the PM model can simulate a NTM of space $\mathcal{O}(nf(n))$ using $\mathcal{O}(f(n))$ space in each agent.

The intuition behind the proof is that with at least $\log n$ memory per agent we can assign unique ids and propagate the size of the population to all agents. The assignment process is presented in Section 6.1.1. Since the agents do not know when the process terminates, the simulation is reinitialized in a fashion similar to the one described in [13]. The agents line up according to their ids and the simulation accesses their tapes in a modular way. The nondeterministic choices are made by exploiting the inherent nondeterminism of the interaction pattern. The full proof is presented in Section 6.1.2.

6.1.1. Assigning unique IDs by reinitializing computation

In this subsection, we prove that PM protocols can assume the existence of unique consecutive ids and knowledge of the population size at the space cost of $\mathcal{O}(\log n)$ (Theorem 2). In particular, we present a PM protocol that correctly assigns unique consecutive ids to the agents and informs them of the correct population size using only $\mathcal{O}(\log n)$ memory, without assuming any initial knowledge of none of them. We show that this protocol can simulate any PM protocol that assumes the existence of these ids and knows the population size.

Definition 4. Let **PLM** $\equiv$ **PMSPACE**$(\log n)$. In words it is the class of all predicates that are stably computable by some PM protocol that uses $\mathcal{O}(\log n)$ space in each agent (and in all of its tapes, excluding the space used for the read-only tape).

Definition 5. Let *IPM* ('I' standing for "Ids") be the extension of the PM model in which the agents have additionally the unique ids $\{0, 1, \ldots, n-1\}$ and in which each agent knows the population size (these are read-only information stored in a separate read-only tape).

Definition 6. Let **IPMSPACE**$(f(n))$ be the class of all predicates that are stably computable by some IPM protocol that uses $\mathcal{O}(f(n))$ space in every agent (and in all of its tapes, excluding the space used for the read-only tape) and denote by **SIPMSPACE**$(f(n))$ its symmetric subclass. Similarly to **PLM** define **IPLM** $\equiv$ **IPMSPACE**$(\log n)$ and **SIPLM** $\equiv$ **SIPMSPACE**$(\log n)$.

Pick any $p \in$ **SIPLM**. Let $\mathcal{A}$ be the IPM protocol that stably computes it in $\mathcal{O}(\log n)$ space. We now present a PM protocol $\mathcal{I}$, containing protocol $\mathcal{A}$ as a subroutine (see Protocol 1), that stably computes p, by also using $\mathcal{O}(\log n)$ space. $\mathcal{I}$ is always executed by every agent and its job is to assign unique ids to the agents, to inform them of the correct population size and to control $\mathcal{A}$'s execution (e.g. restarts its execution if needed). $\mathcal{A}$, when $\mathcal{I}$ allows its execution, simply reads the unique ids and the population size provided by $\mathcal{I}$ and executes itself normally. We first present $\mathcal{I}$ and then prove that it eventually correctly assigns unique ids and correctly informs the agents of the population size, and that when this process comes to a successful end, it restarts $\mathcal{A}$'s execution in all agents without allowing non-reinitialized agents to communicate with the reinitialized ones. Therefore, at some point, $\mathcal{A}$ will begin its execution reading the correct unique ids and the correct population size (provided by $\mathcal{I}$), thus, it will get correctly executed and will stably compute p.

Protocol 1 $\mathcal{I}$

```
if rid == id then // when interacting with an agent of the same id
    if initiator == 1 then // if the agent is the initiator of the interaction
        // it increases its id by one and stores it in the outgoing message
        id ← id + 1, sid ← id
        // it sets the population size equal to its updated id + 1
        ps ← id + 1, sps ← ps
    else // if it is the responder
        // it updates the population size to the same value as the initiator
        ps ← id + 2, sps ← ps
    end if
    // in either case it clears its working block and copies its
    // input symbol into it; it also clears its output tape
    working ← binput, output ← ∅
else // if the other participant in the interaction has different id
    if rps > ps then // in case the agent has an outdated population size
        working ← binput, output ← ∅ // it gets reinitialized
        // and updates its population size to the greater value
        ps ← rps, sps ← ps
    else if rps == ps then // in case the other agent knows the same population size
        // it must have also been reinitialized and thus
        // the agent can proceed executing A
        execute A for 1 step
    end if
end if
```

We begin by describing $\mathcal{I}$'s variables. The variable *id* is for storing the id of the agent (from which $\mathcal{A}$ reads the agents' ids), *sid* the variable for storing the *id* that an agent writes in its outgoing message tape in order to send it, and *rid* the variable for storing the *id* that an agent receives via interaction. The model's definition implies that all variables used for sending information, like *sid*, preserve their value in future interactions unless altered by the agent. Initially, $id = sid = 0$ for all agents. All agents have an input backup variable *binput* which they initially set to their input symbol and make it read-only. Thus, each agent has always available its input via *binput* even if the computation has proceeded. *working* represents the block of the working tape that $\mathcal{A}$ uses for its computation and *output* represents the contents of the output tape. *initiator* is a binary flag that becomes true after every interaction if the agent was the initiator of the interaction and false otherwise (this is easily implemented by exploiting the external transition function). We denote by *ps*, the variable storing the population size, by *sps*, the one used to put it in a outgoing message, and by *rps*, the received one. Initially, $ps = sps = 1$.

We now describe $\mathcal{I}$'s functionality. Whenever a pair of agents with the same id interact, the initiator increases its id by one and both update their population size value to the greater id plus one. Whenever two agents with different ids and population size values interact, they update their population size variables to the greater size. Thus the correct size (greatest id plus one) is propagated to all agents. Both interactions described above reinitialize the participating agents (restore their input and erase all data produced by the subroutine $\mathcal{A}$, without altering their ids and population sizes). Whenever two agents of different ids and same population sizes interact, $\mathcal{A}$ runs as a subroutine using those data provided by $\mathcal{I}$.

The following lemmas provide some important properties of Protocol 1. Lemma 1 shows that $\mathcal{I}$ correctly assigns unique consecutive ids and propagates the correct population size to the agents of the population in a finite number of steps, whereas Lemma 3 guarantees the fairness of $\mathcal{A}$'s execution.

Lemma 1. (i) *No agent id becomes greater than $n-1$, and no ps variable becomes greater than n.* (ii) $\mathcal{I}$ *assigns the ids* $\{0, 1, \ldots, n-1\}$ *in a finite number of interactions.* (iii) $\mathcal{I}$ *sets the ps variable of each agent to the correct population size in a finite number of interactions.*

Proof. (i) By an easy induction, in order for an *id* to reach the value v, there have to be at least $v+1$ agents present in the population. Thus, whenever an *id* becomes greater than $n-1$, there have to be more than n agents present, which creates a contradiction. Similar arguments hold for the *ps* variables
(ii) Assume on the contrary that it does not. Because of (i), at each point of the computation there will exist at least two agents, u, v such that $id_u = id_v$. Due to fairness, an interaction between such agents shall take place infinitely many times, creating an arbitrarily large *id* which contradicts (i).
(iii) The correctness of the id assignment ((i), (ii)) guarantees that after a finite number of steps two agents, u, v will set their *ps* variables to the correct population size (upon interaction in which $id_u = id_v = n-2$). It follows from (i) that no agent will have its *ps* variable greater than n. Fairness guarantees that each other agent will interact with u or v, updating its *ps* to n. □

Note that once a agent learns the correct population size this value does not change and thus no further reinitialization can take place. The following lemma show that only agents that know the correct population size can have effective interactions w.r.t. $\mathcal{A}$'s execution.

Lemma 2. *After the unique consecutive ids* $\{0, 1, \ldots, n-1\}$ *have been assigned, only agents that know the correct population size have* effective interactions *with each other w.r.t. to* $\mathcal{A}$*'s execution.*

Proof. After the unique ids have been successfully assigned, it is like the population is partitioned in two classes, the class *FR* of finally reinitialized agents which know the correct population size and *NFR* of the non finally reinitialized ones (which do not know the correct population size). Initially (just after the unique ids have been successfully assigned), $FR = \{n-2, n-1\}$ and $NFR = \{0, 1, \ldots, n-3\}$, that is, all agents except $n-1$ and $n-2$, who know the correct population size n, are considered as non finally reinitialized. An agent $i \in NFR$ moves to *FR* iff it interacts with an agent in *FR*. This interaction reinitializes *i* for the last time since its *ps* value is updated with the correct population size and by Lemma 1(i, iii) cannot change any further. Therefore, no interaction between agents in different classes can be effective. Similarly, by inspecting Protocol 1 it is easy to see that only agents in *FR*, that is agents with proper ids and population size values, have effective interactions with each other. □

Lemma 3. *Given that* $\mathcal{I}$*'s execution is fair,* $\mathcal{A}$*'s execution is fair as well.*

Proof. Due to the fact that the id-assignment process and the population size propagation are completed in a finite number of steps, it suffices to study fairness of $\mathcal{A}$'s execution after their completion. The state of each agent may be thought of as containing an $\mathcal{I}$-subcomponent and an $\mathcal{A}$-subcomponent, with obvious contents. Denote by $C_{\mathcal{A}}$ the unique subconfiguration of C consisting only of the $\mathcal{A}$-subcomponents of all agents and note that some $C_{\mathcal{A}}$ may correspond to many superconfigurations C. Assume that $C_{\mathcal{A}} \to C'_{\mathcal{A}}$ and that $C_{\mathcal{A}}$ appears infinitely often (since here we consider $\mathcal{A}$'s configurations, this '$\to$' refers to a step of $\mathcal{A}$'s execution). $C_{\mathcal{A}} \to C'_{\mathcal{A}}$ implies that there exist superconfigurations C, C' of $C_{\mathcal{A}}$, $C'_{\mathcal{A}}$, respectively, such that $C \to C'$ (via some step of $\mathcal{A}$ in the case that $C_{\mathcal{A}} \neq C'_{\mathcal{A}}$). Due to $\mathcal{I}$'s fairness, if C appears infinitely often, then so does C' and so does $C'_{\mathcal{A}}$ since it is a subconfiguration of C'. Thus, it remains to show that C appears infinitely often. Since $C_{\mathcal{A}}$ appears infinitely often, then the same must hold for all of its superconfigurations. The reasoning is as follows. All those superconfigurations differ only in the $\mathcal{I}$-subcomponents, that is, they only differ in some variable checks performed by $\mathcal{I}$ (after the id-assignment process and the population size propagation have come to an end, nothing else is performed by $\mathcal{I}$). But all of them are reachable from and can reach a common superconfiguration of $C_{\mathcal{A}}$ in which no variable checking is performed by $\mathcal{I}$, thus, they only depend on which pair of agents is selected for interaction and they are all reachable from one another. Since at least one of them appears infinitely often then, due to the fairness of $\mathcal{I}$'s execution, all of them must also appear infinitely often and this completes the proof. □

By combining the above lemmas we can prove the following:

Theorem 2. **PLM** $=$ **SIPLM**.

Proof. **PLM** $\subseteq$ **SIPLM** holds trivially, so it suffices to show that **SIPLM** $\subseteq$ **PLM**. We have presented a **PLM** protocol (protocol 1) that assigns the agents unique consecutive ids after a finite number of interactions and informs them of the population size (Lemma 1). It follows directly from the protocol that after that point, further fair execution of $\mathcal{I}$ will result in execution of protocol $\mathcal{A}$ which can take into account the existence of unique ids. Moreover, execution of $\mathcal{A}$ is guaranteed to be fair (Lemma 3). □

6.1.2. **SNSPACE**$(nf(n)) \subseteq$ **PMSPACE**$(f(n))$ *for any* $f(n) = \Omega(\log n)$

We now show that for space functions $f(n) = \Omega(\log n)$, the PM model can simulate a deterministic TM of space $\mathcal{O}(nf(n))$ using $\mathcal{O}(f(n))$ in each agent. This is formally stated by the following theorem:

Theorem 3. **SSPACE**$(nf(n)) \subseteq$ **PMSPACE**$(f(n))$ *for any* $f(n) = \Omega(\log n)$.

Proof. Let $p : X^* \to \{0, 1\}$ be any predicate in **SSPACE**$(nf(n))$ and $\mathcal{M}$ be the deterministic TM that decides p by using $\mathcal{O}(nf(n))$ space. We can construct a PM protocol $\mathcal{A}$ that uses $f(n) = \Omega(\log n)$ space on each agent and that stably computes p by exploiting its knowledge of unique ids and the population size. Such knowledge can be obtained by the protocol $\mathcal{I}$ of Theorem 2 (see Section 6.1.1). Note that *protocol $\mathcal{I}$ can be executed by any PM protocol whose agents use $\Omega(\log n)$ space*. Let x be any input assignment in X^*. Each agent receives its input symbol according to x (e.g. u receives symbol $x(u)$). We assume for the sake of simplicity that the agents are equipped with an extra tape, the *simulation tape* that is used during the simulation. The agent that has obtained the unique id 0 starts simulating $\mathcal{M}$.

In the general case, assume that currently the simulation is carried out by an agent u having the id i_u. Agent u uses its simulation tape to write symbols according to the transition function of $\mathcal{M}$. Any time the head of $\mathcal{M}$ moves to the right, u moves the head of the simulation tape to the right, pauses the simulation, writes the current state of $\mathcal{M}$ to its outgoing message tape, and passes the simulation to the agent v having id $i_v = (i_u + 1) \mod n$. Any time the head of $\mathcal{M}$ moves to the left, u pauses the simulation, writes the current state of $\mathcal{M}$ to its outgoing message tape, and passes the simulation to the agent v having id $i_v = (i_u - 1) \mod n$. From agent v's perspective, in the first case it just receives the state of $\mathcal{M}$, copies it to its working tape and starts the simulation, while in the second case it additionally moves the head of the simulation tape one cell to the left before it starts the simulation.

It remains to cover the boundary case in which the head of the simulation tape is over the special symbol that indicates the beginning of the tape. In that case, the agent moves the head to the right and continues the simulation himself (notice that this can only happen to the agent that begins the simulation, that is, the one having the id 0).

Whenever, during the simulation, $\mathcal{M}$ accepts, then $\mathcal{A}$ also accepts; that is, the agent that detects $\mathcal{M}$'s acceptance, writes 1 to its output tape and informs all agents to accept. If $\mathcal{M}$ rejects, it also rejects. Finally, note that $\mathcal{A}$ simulates $\mathcal{M}$ not necessarily on input $x = (\sigma_0, \sigma_1, \ldots, \sigma_{n-1})$ but on some x' which is a permutation of x. The reason is that agent with id i does not necessarily obtain σ_i as its input. The crucial remark that completes the proof is that $\mathcal{M}$ accepts x if and only if it accepts x', because p is symmetric.

Because of the above process, it is easy to verify that the kth cell of the simulation tape of any agent u having the id i_u corresponds to the $(n(k-1) + i_u + 1)$th cell of $\mathcal{M}$. Thus, whenever $\mathcal{M}$ alters $l = \mathcal{O}(nf(n))$ tape cells, any agent u will alter $l' = \frac{l - i_u - 1}{n} + 1 = \mathcal{O}(f(n))$ cells of its simulation tape. □

The next theorem shows how the above approach can be generalized to include NTMs.

Theorem 4. *For any $f(n) = \Omega(\log n)$ it holds that* **SNSPACE**$(nf(n)) \subseteq$ **PMSPACE**$(f(n))$.

Proof. We have already shown that the PM model can simulate a deterministic TM $\mathcal{M}$ of $\mathcal{O}(nf(n))$ space, where $f(n) = \Omega(\log n)$, by using $\mathcal{O}(f(n))$ space (Theorem 3). We now present some modifications that will allow us to simulate a NTM $\mathcal{N}$ of the same memory size. Keep in mind that $\mathcal{N}$ halts for every input, that is it decides any language corresponding to some predicate in **SNSPACE**$(nf(n))$. Upon initialization, each agent enters a reject state (writes 0 to its output tape) and the simulation is carried out as in the case of $\mathcal{M}$.

Whenever a nondeterministic choice has to be made, the corresponding agent gets ready and waits to participate in an interaction. The id of the other participant will provide the nondeterministic choice to be made. One possible implementation of this idea is the following. Since there is a fixed upper bound on the number of nondeterministic choices (independent of the population size), the agents can store them in their memories. Any time a nondeterministic choice has to be made between k candidates the agent assigns the numbers $0, 1, \ldots, k-1$ to those candidates and becomes ready for interaction. Assume that the next interaction is with an agent whose id is i. Then the nondeterministic choice selected by the agent is the one that has been assigned the number $i \mod k$. It follows directly from the fairness constraint that if the computation reaches any state S infinitely many times, all the possible nondeterministic choices from S will be followed. In what follows, we will see that this is sufficient for the population to simulate the behavior of $\mathcal{N}$.

Any time the simulation reaches an accept state, all agents change their output to 1 and the simulation halts. Moreover, any time the simulation reaches a reject state, it is being reinitiated. The correctness of the above procedure is captured by the following two cases.

1. *If $\mathcal{N}$ rejects then every agent's output stabilizes to* 0. Upon initialization, each agent's output is 0 and can only change if $\mathcal{N}$ reaches an accept state. But all branches of $\mathcal{N}$'s computation reject, thus, no accept state is ever reached, and every agent's output forever remains to 0.
2. *If $\mathcal{N}$ accepts then every agent's output stabilizes to* 1. Since $\mathcal{N}$ accepts, there is a sequence of configurations S, starting from the initial configuration C that leads to a configuration C' in which each agent's output is set to 1 (by simulating directly the branch of $\mathcal{N}$ that accepts). Notice that when an agent sets its output to 1 it never alters its output tape again, so it suffices to show that the simulation will eventually reach C'. Assume on the contrary that it does not. Since $\mathcal{N}$ always halts the simulation will be at the initial configuration C infinitely many times. Due to fairness, by an easy induction on the configurations of S, C' will also appear infinitely many times, which leads to a contradiction. Thus the simulation will eventually reach C' and the output will stabilize to 1.

6.2. Upper bounds

We first prove that **PMSPACE**$(f(n)) \subseteq$ **SNSPACE**$(nf(n))$.

Theorem 5. *For any function $f(n)$ it holds that* **PMSPACE**$(f(n)) \subseteq$ **SNSPACE**$(nf(n))$.

Proof. We will now present a NTM $\mathcal{M}$ of space $\mathcal{O}(nf(n))$ that can decide a language L_p corresponding to any predicate $p \in$ **PMSPACE**$(f(n))$. To accept the input (assignment) x, $\mathcal{M}_{\mathcal{A}}$ must verify two conditions: That there exists a configuration C reachable from the initial configuration corresponding to x in which the output tape of each agent indicates that p holds, and that there is no configuration C' reachable from C under which p is violated for some agent.

The first condition is verified by guessing and checking a sequence of configurations. Starting from the initial configuration, each time $\mathcal{M}_{\mathcal{A}}$ guesses configuration C_{i+1} and verifies that C_i yields C_{i+1}. This can be caused either by an agent transition u, or an encounter (u, v). In the first case, the verification can be carried out as follows: $\mathcal{M}_{\mathcal{A}}$ guesses an agent u so that C_i and C_{i+1} differ in the configuration of u, and that $C_i(u)$ yields $C_{i+1}(u)$. It then verifies that C_i and C_{i+1} differ in no other agent configurations. Similarly, in the second case $\mathcal{M}_{\mathcal{A}}$ nondeterministically chooses agents u, v and verifies that encounter (u, v) leads to C' by ensuring that: (a) both agents have their working flags cleared in C, (b) the tape exchange takes place in C', (c) both agents update their states according to γ and set their working flags to 1 in C' and (d) that C_i and C_{i+1} differ in no other agent configurations. In each case, the space needed is $\mathcal{O}(nf(n))$ for storing C_i, C_{i+1}, plus $\mathcal{O}(f(n))$ extra capacity for ensuring the validity of each agent configuration in C_{i+1}.

If the above hold, $\mathcal{M}_{\mathcal{A}}$ replaces C_i with C_{i+1} and repeats this step. Otherwise, $\mathcal{M}_{\mathcal{A}}$ drops C_{i+1}. Any time a configuration C is reached in which p holds, $\mathcal{M}_{\mathcal{A}}$ computes the complement of a similar reachability problem: it verifies that there exists no configuration reachable from C in which p is violated. Since **NSPACE** is closed under complement for all space functions $g(n) = \Omega(\log n)$ (see the Immerman–Szelepcsényi theorem [28]), this condition can also be verified in $\mathcal{O}(n \log n)$ space. Thus, L_p can be decided in $\mathcal{O}(nf(n))$ space by some NTM, which implies that $L_p \in$ **SNSPACE**$(nf(n))$. □

Using a different representation of population configurations we can improve the above upper bound to **SNSPACE**$(2^{f(n)}(f(n) + \log n))$ for $f(n) = o(\log n)$.

Theorem 6. *For any function* $f : \mathbb{N} \to \mathbb{N}$, *any predicate in* **PMSPACE**$(f(n))$ *is also in* **SNSPACE**$(2^{f(n)}(f(n) + \log n))$.

Proof. Take any $p \in$ **PMSPACE**$(f(n))$. Let $\mathcal{A}$ be the PM protocol that stably computes predicate p in space $\mathcal{O}(f(n))$. $L_p = \{(\sigma_1\sigma_2 \ldots \sigma_n) \mid \sigma_i \in X \text{ for all } i \in \{1, \ldots, n\} \text{ and } p(\sigma_1\sigma_2 \ldots \sigma_n) = 1\}$ is the language corresponding to p ($X \subset \Sigma^*$ is the set of input strings). We describe a NTM $\mathcal{N}$ that decides L_p in $g(n) = \mathcal{O}(2^{f(n)}(f(n) + \log n))$ space.

Note that each agent uses memory of size $\mathcal{O}(f(n))$. So, by assuming a binary tape alphabet $\Gamma = \{0, 1\}$ (the alphabet of the agents' tapes), an assumption which is w.l.o.g., there are $2^{\mathcal{O}(f(n))}$ different agent configurations (internal configurations) each of size $\mathcal{O}(f(n))$. $\mathcal{N}$ stores a population configuration by storing all these agent configurations, consuming for this purpose $\mathcal{O}(f(n)2^{f(n)})$ space, together with a number per agent configuration representing the number of agents in that agent configuration under the current population configuration. These numbers sum up to n and each one of them requires $\mathcal{O}(\log n)$ tape cells, thus, $\mathcal{O}(2^{f(n)} \log n)$ extra space is needed, giving a total of $\mathcal{O}(2^{f(n)}(f(n) + \log n))$ space needed to store a population configuration. The reason that such a representation of population configurations suffices is that when k agents are in the same internal configuration there is no reason to store it k times. The completeness of the interaction graph allows us to store it once and simply indicate the number of agents that are in this common internal configuration, that is, k.

Now $\mathcal{N}$ does the same as the NTM of Theorem 5 does. The main difference is that it now stores the population configurations according to the new representation we discussed above. □

The upper bounds shown in Theorem 5 are obviously better for functions $f(n) = \Omega(\log n)$ than those established by Theorem 6. Note however, that for $f(n) = o(\log n)$ the upper bounds of Theorem 5 are worse than those of Theorem 6. In order to realize this, consider the function $f(n) = c$ (the memory of each agent is independent of the population size, thus this corresponds to the PP model). According to Theorem 5 the upper bound is the trivial **SNSPACE**(n), whereas the Theorem 6 decreases the upper bound to **SNSPACE**$(\log n)$. This behavior is expected due to the configuration representation of the population used by those theorems. When the configuration is stored as n-vector where each element of the vector holds the internal configuration of an agent (representation used in Theorem 5) then as the memory size grows the additional space needed is a factor n of that growth. On the other hand, when a configuration is represented as a vector of size equal to the number of all possible internal configurations where each element is the number of agents that are in the corresponding internal configuration (as in Theorem 6) then the size of the vector grows exponentially to the memory growth. Therefore tighter upper bounds are obtained by Theorem 5 for functions $f(n) = \Omega(\log n)$ and by Theorem 6 for $f(n) = o(\log n)$. Note that for $f(n) = \log n$ the bounds by both theorems are the same.

6.3. *An exact characterization and a space hierarchy*

In this section we use the previously shown lower and upper bounds to provide exact characterizations for **PMSPACE**$(f(n))$, when $f(n) = \Omega(\log n)$ and to formally state the *Space Hierarchy Theorem of the PM model*.

From Theorems 4 and 5 we get exact characterizations for all **PMSPACE**$(f(n))$, when $f(n) = \Omega(\log n)$. It is formally stated as:

Theorem 7. *For any* $f(n) = \Omega(\log n)$ *it holds that* **PMSPACE**$(f(n)) =$ **SNSPACE**$(nf(n))$.

The following theorem states a space hierarchy for classes of symmetric languages.

Theorem 8 (*Symmetric Space Hierarchy Theorem*)**.** *For each* $h(n)$ *and each recursive* $l(n)$, *separated by a nondeterministically fully space constructible function* $f(n)$, *with* $h(n) \in \Omega(f(n))$ *but* $l(n) \notin \Omega(f(n))$, $\exists$ *a language* L *that is in* **SNSPACE**$(h(n)) -$ **SNSPACE**$(l(n))$.

Proof. Follows immediately from the unary (tally) separation language presented in [29] and the fact that any unary language is symmetric. □

The previous theorem is used next for establishing a similar hierarchy on the classes of symmetric predicates that we discuss in this work.

Theorem 9 (*PM Space Hierarchy*)**.** *For each $h(n) \in \Omega(\log n)$ and each recursive $l(n)$, separated by a nondeterministically fully space constructible function $g(n)$, with $h(n) \in \Omega(g(n))$ but $l(n) \notin \Omega(g(n))$, there is a language in* $\mathbf{PMSPACE}(h(n)) - \mathbf{PMSPACE}(l(n))$.

Proof. Since $l(n)$ is recursive so is $nl(n)$ and since $g(n)$ is nondeterministically fully space constructible so is $ng(n)$. Moreover, $h(n) \in \Omega(g(n))$ and $l(n) \notin \Omega(g(n))$ imply $nh(n) \in \Omega(ng(n))$ and $nl(n) \notin \Omega(ng(n))$, respectively. Now we may apply Theorem 8 to the functions $nh(n)$, $nl(n)$, and $ng(n)$ to obtain a language L in $\mathbf{SNSPACE}(nh(n)) - \mathbf{SNSPACE}(nl(n))$. Note that $h(n) \in \Omega(\log n)$ implies (Theorem 7) $\mathbf{SNSPACE}(nh(n)) = \mathbf{PMSPACE}(h(n))$. Thus, $L \in \mathbf{PMSPACE}(h(n))$. Moreover, $L \notin \mathbf{SNSPACE}(nl(n))$ implies $L \notin \mathbf{PMSPACE}(l(n))$, otherwise we could apply Theorem 5 to obtain a contradiction. Thus, L is in $\mathbf{PMSPACE}(h(n)) - \mathbf{PMSPACE}(l(n))$. □

In simple words, Theorem 9 says that for the space bounds discussed in this section, protocols using more memory can compute more things.

7. A threshold in the computability of the PM model

In this section, we explore the computability of the PM model when the protocols use $o(\log \log n)$ space. We show that $\log \log n$ acts as a threshold under which PM protocols become computationally equivalent to PPs. In particular, we prove that $\mathbf{PMSPACE}(f(n)) = \mathbf{SEM}$ when $f(n) = o(\log \log n)$. Moreover, we prove that $\mathbf{SEM} \subsetneq \mathbf{PMSPACE}(f(n))$ when $f(n) = \Omega(\log \log n)$ by showing that $O(\log \log n)$ space suffices for computing a non-semilinear predicate.

7.1. $\log \log n$ *threshold*

Here, we prove an interesting limitation on the computability of the PM model when the memory bounds are too restrictive.

Theorem 10. *PM protocols using $f(n) = o(\log \log n)$ space can only compute semilinear predicates.*

Proof Idea. The result lies on the fact that *populations of different size share common executions at the beginning of their computation.* Indeed, the set of initial states is identical for any two populations A, B, $|A| = n < |B|$. In the first step of the execution, any non-initial state can occur by an interaction of agents being in the initial states. Thus, the sets of states that occur by such interactions are also the same in the two populations (for non-trivial values of n). Proceeding inductively this way, we can see that in order for a new state w to occur in B, but not in A, there has to be an interaction between two states u, v, which can be present in both populations. But since w appears only in B, *u and v cannot exist in A at the same time*, otherwise w would occur in A too. Based on this observation, one can establish that protocols that use $o(\log \log n)$ memory restrict the state space so much, that any two states u, v can occur concurrently in any population A, so that any state that appears in such a B has to be present in A too.

In the following, we formalize this proof idea, showing that Theorem 10 holds.

Definition 7. Let $\mathcal{A}$ be a PM protocol executed in a population V of size n. Define an *agent configuration graph*, $R_{\mathcal{A},V} = \{U, W, F\}$ with components described as follows:

- U is the set of the agent configurations that can occur in any execution of $\mathcal{A}$ such that the working flag is set to 0.
- W is the set of edges (u, v), $u, v \in U$ so that there exists an edge (u, v) when there exists an agent configuration w so that an interaction between two agents with configurations u, w will lead the first one to configuration v.
- $F : W \to \{u_1, u_2, \dots\}, u_i \in U \times \{i, r\}$ is an edge labeling function so that when an agent k being in configuration u enters configuration v via a single interaction with an agent being in configuration w, and k acts as $x \in \{i, r\}$ (initiator–responder) in the interaction, then $\{w, x\} \in F((u, v))$.

In other words, U contains the configurations that an agent may enter in any possible execution, when we do not take into consideration the ones that correspond to internal computation, while W defines the transitions between those configurations through interactions defined by F. Note that the model's description excludes infinite sequences of blank cells from the agent configurations. Also, notice that in general, $R_{A,V}$ depends not only on the protocol $\mathcal{A}$, but also on the population V. We call a $u \in U$ *initial node* iff it corresponds to an initial agent configuration.

Because of the uniformity property, we can deduce the following theorem:

Lemma 4. *Let $R_{\mathcal{A},V}$, $R_{\mathcal{A},V'}$ be two agent configuration graphs corresponding to a protocol $\mathcal{A}$ for any two different populations V, V' of size n and n' respectively, where $n < n'$. Then, there exists a subgraph R^* of $R_{\mathcal{A},V'}$ such that $R^* = R_{\mathcal{A},V}$, and whose initial nodes contains all the initial nodes of $R_{\mathcal{A},V'}$.*

Proof. Indeed, let V_1', V_2' be a partitioning of V' such that $V_1' = V$, and observe the agent configuration graph that is yielded by the execution of $\mathcal{A}$ in V_1'. Since both populations execute the same protocol A the transitions are the same, thus all edges in $R_{\mathcal{A},V}$ will be present in $R_{\mathcal{A},V_1'}$ between the common pairs of nodes and their F labels will be equal as well since $V_1' = V$. Therefore $R_{\mathcal{A},V} = R_{\mathcal{A},V_1'}$. Moreover, since the initial nodes are the same for both populations, they must be in $R_{\mathcal{A},V_1'}$. Finally, $R_{\mathcal{A},V_1'}$ is a subgraph of $R_{\mathcal{A},V'}$, as $V_1' \subset V'$, and the proof is complete. □

The above lemma states that while we explore populations of greater size, the corresponding agent configuration graphs are only enhanced with new nodes and edges, while the old ones are preserved.

Given an agent configuration graph, we associate each node a with a value $r(a)$ inductively, as follows:

Base Case For any initial node a, $r(a) = r_{init} = 1$.

Inductive Step For any other node a, $r(a) = min(r(b) + r(c))$ such that a is reachable from b through an edge that contains c in its label, and b, c have already been assigned an r value.

Lemma 5. *Let $R_{\mathcal{A},V} = \{U, W, F\}$ be an agent configuration graph. Every node in $R_{\mathcal{A},V}$ gets associated with an r value.*

Proof. Assume for the sake of the contradiction that there is a maximum, nonempty set of nodes $U' \subset U$ such that $\forall v \in U'$, v does not get associated with an r value. Then $B = U - U'$, and $C = (B, U')$ defines a cut, with all the initial nodes being in B. We examine any edge (u, v) with label L that crosses the cut, having an arbitrary $(w, x) \in L$. Since no initial node can be in U' (initial nodes are assigned the r-value 1) and each node of the nonempty U' is reachable from some initial configuration (by the definition of $R_{\mathcal{A},V}$) there must be at least one such edge. Obviously $u \in B$ and $v \in U'$, and u is associated with a value $r(u)$. Since v is not associated with any r value, the same must hold for node w (otherwise $r(v) = r(u) + r(w)$). We now examine the first agent c that enters in some execution a configuration corresponding to some $v \in U'$. Because of the above observation, this could only happen through an interaction with an agent being in a configuration that is also in U' which creates the contradiction. □

Note that for any given protocol and population size, the r values are *unique* since the agent configuration graph is unique. The following lemma captures a bound in the r values when the corresponding protocol uses $f(n) = o(\log\log n)$ space.

Lemma 6. *Let r_{max-i} be the ith greatest r value associated with any node in an agent configuration graph. For any protocol $\mathcal{A}$ that uses $f(n) = o(\log\log n)$, there exists a n_0 such that for any population of size $n > n_0$, $r_{max} < \frac{n}{2}$.*

Proof. Since $f(n) = o(\log\log n)$, $\lim_{n\to\infty} \frac{f(n)}{\log\log n} = 0$, so $\lim_{n\to\infty} \frac{\log\log n}{f(n)} = \infty$ and $\lim_{n\to\infty} \frac{\log n}{2^{f(n)}} = \infty$. It follows from the last equation that there exists a fixed n_0 such that $\frac{\log n}{2^{f(n)}} > 2$ for any $n > n_0$.

Fix any such n and let $k = |U| \le 2^{f(n)}$ in the corresponding agent configuration graph. Since any node is associated with an r value, there can be at most k different such values. Now observe that $r_{max} \le 2 \cdot r_{max-1} \le \cdots \le 2^k \cdot r_{init} \le 2^{2^{f(n)}} < 2^{\frac{\log n}{2}} \le \sqrt[2]{n} \le \frac{n}{2}$ for $n > max(n_0, 2)$. □

Note that these r-values are a part of our theoretical analysis and are not stored on the population (there is not enough space on the agents to store them).

Lemma 7. *Let a be a node in the agent configuration graph RAV. Then for every subpopulation of V of size r(a) there is an input and an execution of the protocol A that leads to the configuration a.*

Proof. We prove the above lemma by generalized induction in the r values.

Base Case The lemma holds for any initial node u, since $r_{init} = 1$.

Inductive Step We examine any non-initial node u that has been associated with a value $r(u) = r(a) + r(b)$, for some a, b. The inductive hypothesis guarantees that a and b can be reached in two separate subpopulations of size $r(a)$ and $r(b)$. Then an interaction between those agents will take one of them to the configuration u, so the lemma holds for u too. □

Lemmas 6 and 7 lead to the following:

Lemma 8. *For any protocol $\mathcal{A}$ that uses $f(n) = o(\log\log n)$ there exists a fixed n_0 such that for any population of size $n > n_0$ and any pair of agent configurations u, v, there exists an execution in which the interaction (u, v) takes place.*

Proof. Indeed, because of the Lemma 6, there exists a n_0 such that for any $n > n_0$, $r(a) < \frac{n}{2}$ for any a. With that in mind, Lemma 7 guarantees that in any such population, any interaction (u, v) can occur since any of the agent configurations u, v can occur independently, by partitioning the population in two subpopulations of size $\frac{n}{2}$ each. □

We can now complete our proof of Theorem 10:

Proof. Because of the uniformity constraint, A can be executed in any population of arbitrary size. We choose a fixed n_0 as defined in Lemma 6 and examine the population L of size $n = n_0$. Let $R_{\mathcal{A},L}$ be the corresponding agent configuration graph.

Let L' be any population of size $n' > n$ and $R_{\mathcal{A},L'}$ the corresponding agent configuration graph. Because of Lemma 4, $R_{\mathcal{A},L'}$ contains a subgraph K, such that $K = R_{\mathcal{A},L}$, and the initial nodes of $R_{\mathcal{A},L'}$ are in K. Let $U^* = U' - U$, and k the first agent configuration that appears in L' such that $k \in U^*$ through an interaction (u, v)(k cannot be an initial configuration, thus it occurs through some interaction). Then $u, v \in U$, and the interaction (u, v) can occur in the population L too (Lemma 8), so that $k \in U$, which refutes our choice of k creating a contradiction. So, $U^* = \emptyset$, and the set of agent configurations does not change as we examine populations of greater size. Since the set of agent configurations remains described by the *fixed* $R_{\mathcal{A},L}$, the corresponding predicate can be computed by the PP model, thus it is semilinear. □

Theorem 10 guarantees that for any protocol that uses only $f(n) = o(\log\log n)$ space in each agent, there exists a population of size n_0 in which it stops using extra space. Since n_0 is fixed, we can construct a protocol based on the agent configuration graph which uses constant space,[5] and thus can be executed in the PP model.

So far, we have established that **PMSPACE**$(f(n)) \subseteq$ **SEM** when $f(n) = o(\log\log n)$. Since the inverse direction holds trivially, we can conclude that **PMSPACE**$(f(n)) =$ **SEM**.

Theorem 10 practically states that when the memories available to the protocols are strictly smaller than $\log\log n$ (asymptotically) then these PM protocols are nothing more than PPs, and although their memory is still dependent on the population size, they cannot exploit it as such; instead they have to use it as a constant memory much like PPs do.

7.2. The power of 2 predicate

We will now present the non-semilinear power of 2 predicate, and devise a PM protocol that computes it using $\mathcal{O}(\log\log n)$ space in each agent.

The predicate's definition is slightly different to the one described in Section 5.2. We here define the power of 2 as follows: During the initialization, each agent receives an input symbol from $X = \{a, 0\}$, and let N_a denote the number of agents that have received the symbol a. We want to compute whether $\log N_a = t$ for some natural t. We give a high-level protocol that computes this predicate, and prove that it can be correctly executed using $\mathcal{O}(\log\log n)$ space.

Each agent u maintains a variable x_u, and let out_u be the variable that u uses to write its output. Initially, any agent u that receives a as his input symbol sets $x_u = 1$ and $out_u = 1$, while any other agent v sets $x_v = 0$ and $out_v = 1$.

The main protocol consists of two subprotocols, $\mathcal{A}$ and $\mathcal{B}$, that are executed concurrently. Protocol $\mathcal{A}$ does the following: whenever an interaction occurs between two agents, u, v, with u being the initiator, if $x_u = x_v > 0$, then $x_u = x_u + 1$ and $x_v = 0$. Otherwise, nothing happens. Protocol $\mathcal{B}$ runs in parallel, and computes the semilinear predicate of determining whether there exist 0, two or more agents having $x > 0$. If so, it outputs 0, otherwise it outputs 1. Observe that $\mathcal{B}$ is executed on *stabilizing inputs*, as the x-variables fluctuate before they stabilize to their final value. However, it is well known that the semilinear predicates are also computable under this constraint [30].

Lemma 9. *The main protocol uses* $\mathcal{O}(\log\log n)$ *space.*

Proof. As protocol $\mathcal{B}$ computes a semilinear predicate, it only uses $\mathcal{O}(c)$ space, with c being a constant. To examine the space bounds of $\mathcal{A}$, pick any agent u. We examine the greatest value that can be assigned to the variable x_u. Observe that in order for x_u to reach value k, there have to be at least 2 pre-existing x-variables with values $k-1$. Through an easy induction, it follows that there have to be at least 2^k pre-existing variables with the value 1. Since $2^k \leq N_a$, $k \leq \log N_a \leq \log n$, so x_u is upper-bounded by $\log n$, thus consuming $\mathcal{O}(\log\log n)$ space. □

Lemma 10. *For every agent u, eventually* $out_u = 1$ *if* $\log N_a = t$ *for some arbitrary t, and* $out_u = 0$ *otherwise.*

Proof. Indeed, the execution of protocol $\mathcal{B}$ guarantees that all agents will set $out = 1$ iff eventually there exists only one agent u that has a non-zero value assigned in x_u. Assume that $x_u = k$ for some k. Then, because of the analysis of Lemma 9 during the initialization of the population will exist 2^k x-variables set to 1. Since each of those variables corresponds to one a assignment, $N_a = 2^k \Rightarrow \log N_a = k$. On the other hand, if the answer of the protocol is 0 then there are $t > 1$ agents in the population with x-variables set to different values $x_1, x_2, \ldots, x_t$ otherwise they could have effective interactions with each other. Therefore, there should have initially existed $2^{x_1-1} + 2^{x_2-1} + \cdots + 2^{x_t-1}$ agents with input a. This, however, means that $N_a \neq 2^k$ for any k since each number can be uniquely expressed as a sum of distinct powers of 2. Thus the protocol correctly outputs 0. □

Thus, we have presented a non-semilinear predicate that can be computed by a PM protocol using $\mathcal{O}(\log\log n)$ space. Combining this result with Theorem 10, we obtain the following theorem:

Theorem 11 (*Threshold Theorem*)**.** **SEM** $=$ **PMSPACE**$(f(n))$ *when* $f(n) = o(\log\log n)$ *and* **SEM** $\subsetneq$ **PMSPACE**$(f(n))$ *when* $f(n) = \Omega(\log\log n)$.

Theorem 11 resembles a similar well-known result of Computational Complexity for the class of regular languages **REG**, according to which **REG** $=$ **SPACE**$(o(\log\log n)) \subsetneq$ **SPACE**$(\Omega(\log\log n))$ (see [27,31] and Theorem 5.1.3, pages 29–30, of [32]). However, the model under consideration here and, consequently, the proof that we provide are quite different.

---

[5] Notice that this fixed agent configuration graph can be viewed as a deterministic finite automaton.

8. Conclusions: future research directions

We proposed the PM model, an extension of the PP model, in which the agents are communicating TMs. Throughout our work, we studied the computational power of the new model when the space used by each agent is bounded by a function $f(n)$ of the population size. To do so, we presented protocols in which the number of states used by any execution on n agents is bounded by $\mathcal{O}(c^{f(n)})$ (so that each state can be represented by $\mathcal{O}(f(n))$ tape cells), where c constant, and the new states of the interacting agents are computable in $f(n)$ space by a TM. Although the model preserves uniformity and anonymity, interestingly, we have been able to prove that the agents can *organize themselves into a NTM* that makes full use of the agents' total memory (i.e. of $\mathcal{O}(nf(n))$ space) when $f(n) = \Omega(\log n)$. The agents are initially identical and have no global knowledge of the system, but by executing an *iterative reinitialization process* they are able to assign *unique consecutive ids* to themselves and get informed of the population size. In this manner, we showed that **PMSPACE**$(f(n))$, which is the class of predicates stably computable by the PM model using $\mathcal{O}(f(n))$ memory, contains all symmetric predicates in **NSPACE**$(nf(n))$. Moreover, by proving that **PMSPACE**$(f(n)) \subseteq$ **SNSPACE**$(nf(n))$, we concluded that for $f(n) = \Omega(\log n)$, it is precisely equal to the class consisting of all symmetric predicates in **NSPACE**$(nf(n))$. We also explored the behavior of the PM model for space bounds $f(n) = o(\log n)$ and proved that **SEM** = **PMSPACE**$(f(n))$ when $f(n) = o(\log \log n)$. Finally, we showed that this bound acts as a threshold, that is, **SEM** $\subsetneq$ **PMSPACE**$(f(n))$ when $f(n) = \Omega(\log \log n)$.

Many interesting questions remain open. Is the PM model fault-tolerant? What preconditions are needed in order to achieve satisfactory fault tolerance? What is the behavior of the model when the agents use $\mathcal{O}(f(n))$ memory, where $f(n) = o(\log n)$ and $f(n) = \Omega(\log \log n)$? Does a space hierarchy similar to the one presented in Section 6.3, hold for functions $o(\log n)$?

Acknowledgements

We wish to thank some anonymous reviewers for their very useful comments on a previous version of this work.

References

[1] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, P.G. Spirakis, Passively mobile communicating logarithmic space machines, Technical Report FRONTS-TR-2010-16, RACTI, Patras, Greece, 2010. http://fronts.cti.gr/aigaion/?TR=154, arXiv/1004.3395v1.
[2] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, P.G. Spirakis, Passively mobile communicating machines that use restricted space, Technical Report, RACTI, Patras, Greece, 2010. http://arxiv.org/abs/1012.2440v1.
[3] D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, R. Peralta, Computation in networks of passively mobile finite-state sensors, Distributed Computing (2006) 235–253.
[4] S. Ginsburg, E.H. Spanier, Semigroups, presburger formulas, and languages, Pacific Journal of Mathematics 16 (1966) 285–296.
[5] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, E. Ruppert, When birds die: Making population protocols fault-tolerant, in: IEEE 2nd Intl Conference on Distributed Computing in Sensor Systems, DCOSS, in: Lecture Notes in Computer Science, vol. 4026, Springer-Verlag, 2006, pp. 51–66.
[6] R. Guerraoui, E. Ruppert, Names trump malice: tiny mobile agents can tolerate byzantine failures, in: 36th International Colloquium on Automata, Languages and Programming, ICALP, in: Lecture Notes in Computer Science, vol. 5556, Springer-Verlag, 2009, pp. 484–495.
[7] D. Angluin, J. Aspnes, D. Eisenstat, A simple population protocol for fast robust approximate majority, Distributed Computing 21 (2008) 87–102.
[8] I. Chatzigiannakis, O. Michail, P.G. Spirakis, Mediated population protocols, in: 36th International Colloquium on Automata, Languages and Programming, ICALP, in: Lecture Notes in Computer Science, vol. 5556, Springer-Verlag, 2009, pp. 363–374.
[9] D. Angluin, J. Aspnes, D. Eisenstat, Fast computation by population protocols with a leader, Distributed Computing 21 (2008) 183–199.
[10] I. Chatzigiannakis, S. Dolev, S.P. Fekete, O. Michail, P.G. Spirakis, Not all fair probabilistic schedulers are equivalent, in: 13th International Conference on Principles of Distributed Systems, OPODIS, in: Lecture Notes in Computer Science, vol. 5923, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 33–47.
[11] O. Bournez, P. Chassaing, J. Cohen, L. Gerin, X. Koegler, On the convergence of population protocols when population goes to infinity, Applied Mathematics and Computation (2009).
[12] I. Chatzigiannakis, P.G. Spirakis, The dynamics of probabilistic population protocols, in: 22nd international symposium on Distributed Computing, DISC, in: Lecture Notes in Computer Science, vol. 5218, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 498–499.
[13] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, 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, in: Lecture Notes in Computer Science, vol. 6281, Springer-Verlag, 2010, pp. 270–281.
[14] O. Michail, I. Chatzigiannakis, P.G. Spirakis, Mediated population protocols, Theoretical Computer Science 412 (2011) 2434–2450.
[15] D. Angluin, J. Aspnes, M. Chan, M.J. Fischer, H. Jiang, R. Peralta, Stably computable properties of network graphs, in: V.K. Prasanna, S. Iyengar, P. Spirakis, M. Welsh (Eds.), Distributed Computing in Sensor Systems: First IEEE International Conference, DCOSS 2005, Marina del Rey, CA, USA, June/July, 2005, Proceedings, in: Lecture Notes in Computer Science, vol. 3560, Springer-Verlag, 2005, pp. 63–74.
[16] I. Chatzigiannakis, O. Michail, P.G. Spirakis, Stably decidable graph languages by mediated population protocols, in: 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS, in: Lecture Notes in Computer Science, vol. 6366, Springer-Verlag, 2010, pp. 252–266.
[17] J. Beauquier, J. Clement, S. Messika, L. Rosaz, B. Rozoy, Self-stabilizing counting in mobile sensor networks, in: Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC'07, ACM, New York, NY, USA, 2007, pp. 396–397.
[18] O. Bournez, J. Chalopin, J. Cohen, X. Koegler, Playing With Population Protocols, ArXiv e-prints, 2009. http://arxiv.org/abs/0906.3256.
[19] J. Aspnes, E. Ruppert, An introduction to population protocols, Bulletin of the European Association for Theoretical Computer Science 93 (2007) 98–117.
[20] P.G. Spirakis, Theoretical Aspects of Distributed Computing in Sensor Networks, Springer-Verlag.
[21] O. Michail, I. Chatzigiannakis, P.G. Spirakis, New Models for Population Protocols, in: N.A. Lynch (Ed.), Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool, 2011.
[22] I. Chatzigiannakis, O. Michail, P.G. Spirakis, Recent advances in population protocols, in: 34th International Symposium on Mathematical Foundations of Computer Science, MFCS, in: Lecture Notes in Computer Science, vol. 5734, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 56–76.
[23] C. Àlvarez, A. Duch, J. Gabarro, M. Serna, Sensor field: a computational model, in: 5th Intl Workshop on Algorithmic Aspects of Wireless Sensor Networks, ALGOSENSORS, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 3–14.
[24] C. Àlvarez, M. Serna, P.G. Spirakis, On the computational power of constant memory sensor fields, Technical Report FRONTS-TR-2010-10, 2010.
[25] C. Àlvarez, I. Chatzigiannakis, A. Duch, J. Gabarró, O. Michail, S. Maria, P.G. Spirakis, Computational models for networks of tiny artifacts: a survey, Computer Science Review 5 (2011) 7–25.

[26] P.C. Fischer, A.R. Meyer, A.L. Rosenberg, Counter machines and counter languages, Mathematical Systems Theory 2 (1968) 265–283.
[27] R.E. Stearns, J. Hartmanis, P.M. Lewis, Hierarchies of memory limited computations, in: Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design, SWCT 1965, FOCS'65, IEEE Computer Society, Washington, DC, USA, 1965, pp. 179–190.
[28] N. Immerman, Nondeterministic space is closed under complementation, SIAM Journal on Computing 17 (1988) 935–938.
[29] V. Geffert, Space hierarchy theorem revised, Theoretical Computer Science 295 (2003) 171–187.
[30] D. Angluin, J. Aspnes, D. Eisenstat, Stably computable predicates are semilinear, in: 25th annual ACM Symposium on Principles of Distributed Computing, PODC, ACM Press, New York, NY, USA, 2006, pp. 292–299.
[31] M. Alberts, Space complexity of alternating turing machines, in: L. Budach (Ed.), Fundamentals of Computation Theory, in: Lecture Notes in Computer Science, vol. 199, Springer, Berlin, Heidelberg, 1985, pp. 1–7. 10.1007/BFb0028785.
[32] A. Szepietowski, Turing Machines with Sublogarithmic Space, Springer-Verlag New York, Inc., 1994.