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

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

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

Abstract. This work focuses on the computational power of the *Mediated Population Protocol* model on complete communication graphs and initially identical edges (*SMPP*). In particular, we investigate the class MPS of all predicates that are stably computable by SMPP. Let Σ be an alphabet. A predicate $p : \Sigma^* \rightarrow \{0, 1\}$ is *symmetric* if, $\forall x, x' \in \Sigma^*$ such that x' is a permutation of x's symbols, it holds that $p(x) = p(x')$. Any language $L \subseteq \Sigma^*$ corresponds to a unique predicate p_L defined as $p_L(x) = 1$ iff $x \in L$. Such a language is symmetric iff p_L is symmetric. Due to this bijection we use the term *symmetric predicate* for both predicates and languages. It is already known that MPS is in the symmetric subclass of $NSPACE(n^2)$. Here we prove that this inclusion holds with equality, thus, providing an exact characterization for MPS. To do so, we first prove that any symmetric predicate in $SPACE(n)$ is in MPS and then extend those ideas and improve this to all symmetric predicates in $NSPACE(n^2)$. Thus, we arrive at the following exact characterization: *A predicate is in MPS iff it is symmetric and is in $NSPACE(n^2)$.*

Note: Due to space restrictions, we have moved some proofs and some additional material to a clearly marked Appendix, to be read at the discretion of the Program Committee.

1 Introduction - Population Protocols

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

One reason for studying extremely limited computational devices is that in many real WSNs' application scenarios having limited resources is inevitable. For example, power supply limitations may render strong computational devices useless due to short lifetime. In other applications, mote's size is an important constraint that thoroughly determines the computational limitations. The other reason is that the population protocol model constitutes the starting point of a brand new area of research and in order to provide a clear understanding and foundation of the laws and the inherent properties of the studied systems it ought to be minimalistic. In terms of computational characterization each node is simply a finite-state machine additionally equipped with sensing and communication capabilities and is usually called an *agent*. A *population* is the collection of all agents that constitute the distributed computational system.

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

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

As expected, due to the minimalistic nature of the population protocol model, the class of computable predicates was proven [1, 2] to be fairly small: it is the class of *semilinear predicates* [9] (or, equivalently, all predicates definable by first-order logical formulas in *Presburger arithmetic* [11]) which does not include multiplication of variables, exponentiations, and many other important and natural operations on input variables. Moreover, Delporte-Gallet *et al.* [8] showed that population protocols can tolerate only $\mathcal{O}(1)$ crash failures and not even a single Byzantine agent.

2 Enhancing the Model

The next big step is naturally to strengthen the population protocol model with extra realistic and implementable assumptions, in order to gain more computational power and/or speed-up the time to convergence and/or improve fault-tolerance. Several promising attempts have appeared towards this direction. In each case, the model enhancement is accompanied by a logical question: What is exactly the class of predicates computable by the new model?

An interesting extension was the *community protocol* model of Guerraoui and Ruppert [10] in which the agents have read-only industrial unique ids picked from an infinite set of ids. Moreover, each agent can store up to a constant number of other agents' ids. In this model, agents are only allowed to compare ids, that is, no other operation on ids is permitted. The community protocol model was proven to be extremely strong: the corresponding class consists of all symmetric predicates in $NSPACE(n \log n)$. The proof was based on a simulation of a modified version of Nondeterministic Schönhage's *Storage Modification Machine* (*NSMM*). It was additionally shown that if faults cannot alter the unique ids and if some necessary preconditions are satisfied, then community protocols can tolerate $\mathcal{O}(1)$ Byzantine agents.

The *PALOMA* model [4] made the assumption that each agent is a Turing Machine whose memory is logarithmic in the population size. Interestingly, it turned out that the agents are able to assign unique ids to themselves, get informed of the population size and, by exploiting these, organize themselves into a distributed Nondeterministic TM (*NTM*) that makes full use of the agents' memories. The TM draws its nondeterminism by the nondeterminism inherent in the interaction pattern. The main result of that work was an exact characterization for the class PLM, of all predicates stably computable by the PALOMA model: it is precisely the class of all symmetric predicates in $NSPACE(n \log n)$.

3 Our Results - Roadmap

This work focuses on the computational power of another extension of the population protocol model that was proposed in [6] (see also [7] and [5]) and is called the *Mediated Population Protocol*

(*MPP*) model. The main additional feature in comparison to the population protocol model is that the agents communicate through fixed links which are capable of storing limited information. We think of each link (u, υ) joining two agents u and υ as being itself an agent that only participates in the interaction (u, υ). The agents u and υ can exploit this joint memory to store pairwise information and to have it available during some future interaction. Another way to think of this system is that agents store pairwise information into some global storage, like, e.g., a base station, called the *mediator*, that provides a small fixed slot to each pair of agents. Before interacting, the agents communicate with the mediator to get informed of their collective information and this information is updated according to the result of the interaction.

From [6] we know that the MPP model is strictly stronger than the population protocol model since it can simulate it and can additionally compute a non-semilinear predicate. Moreover, we know that any predicate that is stably computable by the MPP model is also in $NSPACE(n^2)$. In this work, we show that this inclusion holds with equality, thus, providing the following exact characterization for the computational power of the MPP model in the fully symmetric case: *A predicate is stably computable by the MPP model iff it is symmetric and is in* $NSPACE(n^2)$. We show in this manner that the MPP model is surprisingly strong.

In section 4 we give a formal definition of the MPP model and introduce a special class of graphs (the *correctly labeled line graphs*) that comes up in our proof later on. Section 5 holds the actual proof. In particular, subsection 5.1 presents some basic ideas that help us establish a first inclusion. In subsection 5.2 we show how to extend these ideas in order to prove our actual statement. Finally, as the ideas we present are somewhat complex, we essentially give high level descriptions in the main body of the paper. The reader is reffered to the Appendix for some missing proofs, a *formal constructive proof of the main result*, and some graphical step by step examples.

4 The Mediated Population Protocol Model

4.1 Formal Definition

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

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

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

A *network configuration*, or simply a *configuration*, is a mapping $C : V \cup E \to Q \cup S$ specifying the state of each agent in the population and each edge in the set of permissible interactions. Let C and C' be configurations, and let u, υ be distinct agents. We say that C goes to C' via encounter $e = (u, \upsilon)$, denoted $C \overset{e}{\to} C'$, if $C'(u) = \delta_1(C(u), C(\upsilon), C(e))$, $C'(\upsilon) = \delta_2(C(u), C(\upsilon), C(e))$, $C'(e) = \delta_3(C(u), C(\upsilon), C(e))$, and $C'(z) = C(z)$, for all $z \in (V - \{u, \upsilon\}) \cup (E - e)$; 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 $C \rightarrow C'$, 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 $C_i \rightarrow C_{i+1}$ 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 $C_i \rightarrow C_{i+1}$, 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 partition the agents into non-communicating clusters. If that's the case, then it is easy to see that no meaningful computation is possible. To avoid this unpleasant scenario, a strong global *fairness condition* is imposed on the adversary to ensure the protocol makes progress. An infinite execution is *fair* if for every pair of configurations C and C' such that $C \rightarrow C'$, 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 $C' \neq C$).

4.2 Stable Computation

Throughout this work we assume that the communication graph is complete and that all edges are initially in a common state s_0, that is, $\iota(e) = s_0$ for all $e \in E$. Call this for sake of simplicity the SMPP model ('S' standing for "Symmetric"). An SMPP may run on any such communication graph $G = (V, E)$ and its input (also called an *input assignment*) is any $x = \sigma_1 \sigma_2 \cdots \sigma_n \in X^*$ such that $n = |V|$.[3] In particular, by assuming an ordering over V, the input to agent i is the symbol σ_i, for all $i \in \{1, 2, \ldots, n\}$. Let $p : X^* \rightarrow \{0, 1\}$ be any predicate over X^*. 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 predicate's outcome).

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, no agent changes its output in any subsequent step and no matter how the computation proceeds.

A predicate p over X^* is said to be *stably computable* by the SMPP model, if there exists an SMPP $\mathcal{A}$ such that for any input assignment $x \in X^*$, any computation of $\mathcal{A}$ on the complete communication graph of $|x|$ nodes beginning from the initial configuration corresponding to x reaches an output stable configuration in which all agents output $p(x)$.

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

[3] 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).

4.3 Correctly Labeled Line Graphs

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

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

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

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

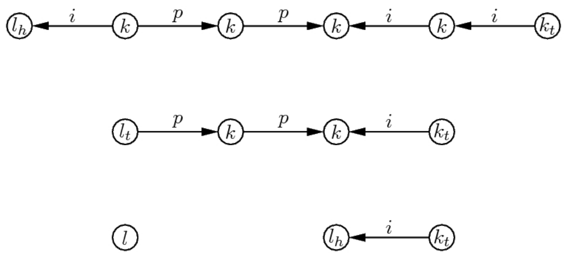


Fig. 1. Some correctly labeled line subgraphs. We assume that all edges not appearing are in state 0 (inactive).

5 The Computational Power of the SMPP Model

In [6] it was shown that MPS is a proper superset of the set of semilinear predicates. Here we are going to establish a much better inclusion. In particular, in Section 5.1 we show that any predicate in $SSPACE(n)$ is also in MPS. In other words, the SMPP model is at least as strong as a linear space TM that computes symmetric predicates. Then in Section 5.2 we extend the ideas used in the proof of this result in order to establish that $SSPACE(n^2)$ is a subset of MPS showing that

MPS is a surprisingly wide class. Finally, we improve to $SNSPACE(n^2)$, thus, arriving at an exact characterization for MPS (the inverse inclusion already exists from [6]).

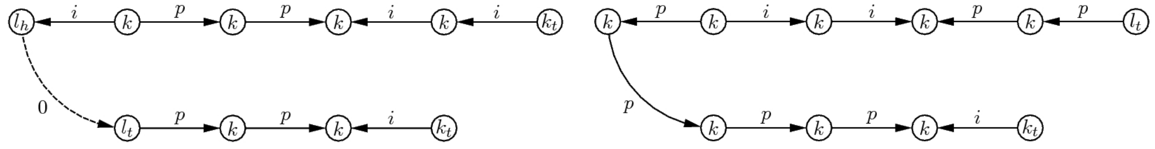
5.1 A First Inclusion: $SSPACE(n) \subseteq MPS$

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

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

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 line subgraphs (see again their definition and Figure 1).

We now describe how two such line graphs L_1 and L_2 are pieced together. Denote by $l(L) \in V$ and by $k_t(L) \in V$ the leader and tail non-leader endpoints of a correctly labeled line graph L, respectively. When $l(L_1) = u$ interacts as the initiator with $l(L_2) = \upsilon$, through an inactive edge, υ becomes a non-leader with a special mark, e.g. k', the edge becomes proper with a special mark, and u becomes a leader in a special state l' indicating that this state will travel towards $k_t(L_1)$ while making all proper edges that it meets inverse and all inverse edges proper. In order to know its direction, it marks each edge that it crosses. When it, finally, arrives at the endpoint, it goes to another special state and walks the same path in the inverse direction until it meets υ again. This walk can be performed easily, without using the marks, because now all edges have correct labels (states). To diverge from L_1's endpoint it simply follows the proper links as the initiator (moving from their tail to their head) and the inverse links as the responder (moving from their head to their tail) while erasing all marks left from its previous walk. When it reaches υ it erases its mark, making its state k, and goes to another special state indicating that it again must walk towards $k_t(L_1)$ for the last time, performing no other operation this time. To do that, it follows the proper links as the responder (from their head to their tail) and the inverse links as the initiator (from their tail to their head). When it, finally, arrives at $k_t(L_1)$ it becomes a normal tail leader and now it is easy to see that L_1 and L_2 have been merged correctly into a common correctly labeled line graph. See Figure 2 for an example. The correctness of this process, called the *merging process*, is captured by Lemma 1 (in the Appendix [A] we also present a graphical step by step example).



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

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

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

Recall that we consider isolated simple leaders as trivial line graphs. Thus, initially, G is partitioned into n correctly labeled trivial line graphs. It is easy to see that correctly labeled line graphs never become smaller (see Appendix [C]) and, according to Lemma 1, when their leaders interact they are merged into a new line graph containing all nodes of the interacting line graphs plus an additional edge joining them. Moreover, given that there are two correctly labeled line subgraphs in the current configuration there is always the possibility (due to fairness) that these line graphs may get merged, 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 line graphs. In simple words, two line graphs can only get merged and there is always the possibility that merging actually takes place. It is easy to see that this process has to end, due to fairness, in a finite number of steps having constructed a correctly labeled spanning line subgraph of G (for simplicity, we call this process the *spanning process*). □

Theorem 2. *Assume that the communication graph $G = (V, E)$ is a correctly labeled line graph of n agents, where each agent takes its input symbol in a second* state component *(the first component is used for the labels of the spanning process and is called* label component*). Then there is an MPP $\mathcal{A}$ that when running on such a graph simulates a deterministic TM $\mathcal{M}$ of $\mathcal{O}(n)$ (linear) space that computes symmetric predicates.*

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

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 line subgraph constructed in some intermediate step (it can be proven by symmetry arguments together with the fact that the agents cannot count up to the population size). Fortunately, we can overcome the impossibility of detecting termination by applying the *reinitialization* technique of [10, 4].

Let us first outline the approach that will be followed in Theorem 3. Whenever two correctly labeled line subgraphs get merged, we know that a new correctly labeled line graph will be constructed in a finite number of steps. Moreover, termination of the merging process can be detected (see the proof of Lemma 1 in the Appendix [B]). When merging comes to an end, the unique leader of the new line 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 line 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 do the same. Restoring the input 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 line subgraphs get merged into the final spanning line 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 2 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 line subgraph is active and all other edges are inactive).

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

Proof. Take any $p \in SSPACE(n)$. By Theorem 2 we know that there is an MPP $\mathcal{A}$ that stably computes p on a line graph of n nodes. We have to show that there exists an SMPP $\mathcal{B}$ that stably computes p. We construct $\mathcal{B}$ to be the composition of $\mathcal{A}$ and another protocol $\mathcal{I}$ that is responsible for executing the spanning and reinitialization processes.

Each agent's state consists of three components: a read-only *input backup*, one used by $\mathcal{I}$, and one used by $\mathcal{A}$. Thus, $\mathcal{A}$ and $\mathcal{I}$ are, in some sense, executed in parallel in different components.

Protocol $\mathcal{I}$ does the following. It always executes the spanning process and when two line 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_t^*. Recall that the new line graph has also correct labels. When u meets its right neighbor, u sets its $\mathcal{A}$ component to its input symbol (by copying it from the input backup), becomes unmarked, and passes the mark to its right neighbor (correct edge labels guarantee that each agent distinguishes its right and left neighbors). When the newly marked agent interacts with its own right neighbor, it does the same, and so on, until the two rightmost agents interact, in which case they are both reinitialized at the same time and the special mark is lost. It is easy to see that this process guarantees that all agents in the line graph become reinitialized and before completion non-reinitialized agents do not have effective interactions with reinitialized ones (the special marked agent acts always as the separator between reinitialized and non-reinitialized agents). Note that if other reinitialization processes are pending from previous reinitialization steps, then the new one 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 while it is executing a reinitialization process. This approach is also correct: fairness guarantees that reinitialization will terminate in a finite number of steps, thus, the merging process will not be blocked forever.

From Theorem 1 we know that the spanning process executed by $\mathcal{I}$ results in a correctly labeled spanning line subgraph of G. The spanning process, as already mentioned, terminates when the merging of the last two line subgraphs takes place and merging also correctly terminates in a finite number of steps (Lemma 1). Moreover, from the above discussion we know that, when this happens, the reinitialization process will correctly reinitialize all agents of the spanning line subgraph, thus, all agents in the population. But then, independently of its computation so far (in its own component), $\mathcal{A}$ will run from the beginning on a correctly labeled line graph of n nodes (this line graph will not be modified again in the future), thus, it will stably compute p. Finally, if we assume that $\mathcal{B}$'s output is $\mathcal{A}$'s output then we conclude that the SMPP $\mathcal{B}$ also stably computes p, thus, $p \in MPS$. See the Appendix [E] for a graphical step by step example. $\square$

5.2 An Exact Characterization: $\boldsymbol{MPS = SNSPACE(n^2)}$

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

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

Proof. For simplicity and w.l.o.g. we assume that $\mathcal{A}$ begins its execution from the leader endpoint 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 line 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 line 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 line graph can be identified because the line 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 2 the simulation also makes use of the inactive edges. The agent in control of the simulation is in a special state denoted with a star '$*$'. Since the simulation starts from the left endpoint (tail leader), its state will be l_t^*. When the star-marked leader interacts with its unique right neighbor on the line graph, the neighbor's state is updated to a *r-marked* state (i.e. k^r). The k^r agent then interacts with its own right neighbor which is unmarked and the neighbor updates its state to a special *dot* state (i.e. $\dot{k}$) whereas the other agent (in state k^r) is updated to k. Then the only effective interaction is between the star-marked leader (l_t^*) and the dot non-leader ($\dot{k}$) 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 1 for the special states traversing through the line graph. The dot mark's (state's) position in the line graph determines which outgoing inactive edge of l_t^* will be used. The sequence in which the dot mark is traversing the graph is the sequence in which l_t^* visits its outgoing inactive edges. Therefore if it has to visit the next inactive edge it moves the dot mark to the right (via a k^r state) or to the left (via a k^l state) if it has to visit the previous one. It should be noted that the dot marked agent plays the role of the TM's head since it points the edge (which would correspond to a tape's cell in $\mathcal{M}$) that is visited. As stated above only the inactive edges hold the contents of the TM's tape. The active ones are used for allowing the special states (symbols) traverse the line graph.

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

When the control of the simulation reaches a non-leader agent (e.g. from the left leader endpoint side) in order to visit its first edge it places the dot mark to the left leader endpoint and then to the next (on the right) non-leader and so forth. If the dot mark reaches the star-marked agent (in the previous example from the left endpoint side) then it moves the dot to the closer (in the line graph) agent that can "see" via an inactive edge towards the right non-leader endpoint. In this way, each agent visits its outgoing edges in a specific sequence (from leader to non-leader when the simulation moves right and the reverse when it moves left) providing the $\mathcal{O}(n^2)$ space needed for the simulation. See Appendix [F] for a graphical example, [G] for a thorough discussion about how the simulation is controlled by intermediate agents, and [H] for a more formal treatment.

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. □

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

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

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

Now reinitialization of the inactive edges begins. When the leader in l^* interacts with its unique right neighbor (via the active edge joining them) it updates its neighbor's state to a special *bar* state (e.g. $\bar{k}$). When the agent with the bar state interacts with its own right neighbor, which is unmarked, the neighbor updates its state to a special *dot* state (e.g. $\dot{k}$). Now the bars cannot propagate and the only effective interaction is between the star leader and the dot non-leader. This interaction reinitializes the state component of the edge used for the simulation and makes the responder non-leader a bar non-leader. Then the new bar non-leader turns its own right neighbor to a dot non-leader, the second outgoing edge of the leader is reinitialized in this manner, and so on, until the edge joining the star leader (left endpoint) with the dot tail non-leader (right endpoint) is reinitialized. What happens then is that the bars are erased one after the other from right to left (keep in mind that all outgoing edges of the leader have been reinitialized) 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. Though it is clear that the above process is executed correctly when L is spanning, because all outgoing edges have their heads on the line graph, it is not so clear that it correctly terminates when L is not spanning. The reason is that in the latter case there will always be inactive edges between agents of L and agents of some other line subgraph L'. It is crucial to prove that no problem will occur, because otherwise there is a clear risk that the reinitialization process on L will not terminate. This would render the spanning process incapable of merging L with some other line graph and a spanning line graph would never get formed.

Lemma 2. *Let L and L' be two distinct line subgraphs of G, and assume that L runs a reinitialization process. The process 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 line 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 ready in finite time to participate in another merging process. □

Lemma 2 guarantees that the spanning process terminates with a spanning line subgraph with active edges, while all remaining edges in G are inactive. In this case, since a unique line 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 4 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 4 stating that this construction is all that $\mathcal{A}$ needs to get executed correctly. Due to its length the formal constructive proof is given in the Appendix [H]. $\square$

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

Proof Idea. We modify the deterministic TM described in Appendix [H] by adding another component in each agent's state which stores a non-negative integer of value at most equal to the greatest number of non-deterministic choices that the new NTM $\mathcal{N}$ can face at any time. Note that this number is independent of the population size. In every reinitialization each agent obtains this value from its neighbors according to its position (which depends on the distance from the leader endpoint) in the line graph. Nondeterministic choices are mapped to these values and whenever such a choice has to be made, the agent in control of the simulation uses the value of the agent with whom it has the next arbitrary interaction. The inherent nondeterminism of the interaction pattern ensures that choices are made nondeterministically. If the accept state is reached all agents accept whereas if the reject state is reached the TM's computation is reinitialized. Fairness guarantees that all paths in the tree representing $\mathcal{N}$'s nondeterministic computation will eventually (although maybe after a long time) be followed, that is, ensures the correctness of the simulation. See Appendix [I] for a full proof. $\square$

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

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

Proof. One direction follows from Theorem 6 and the inverse from Theorem 8 of [6]. $\square$

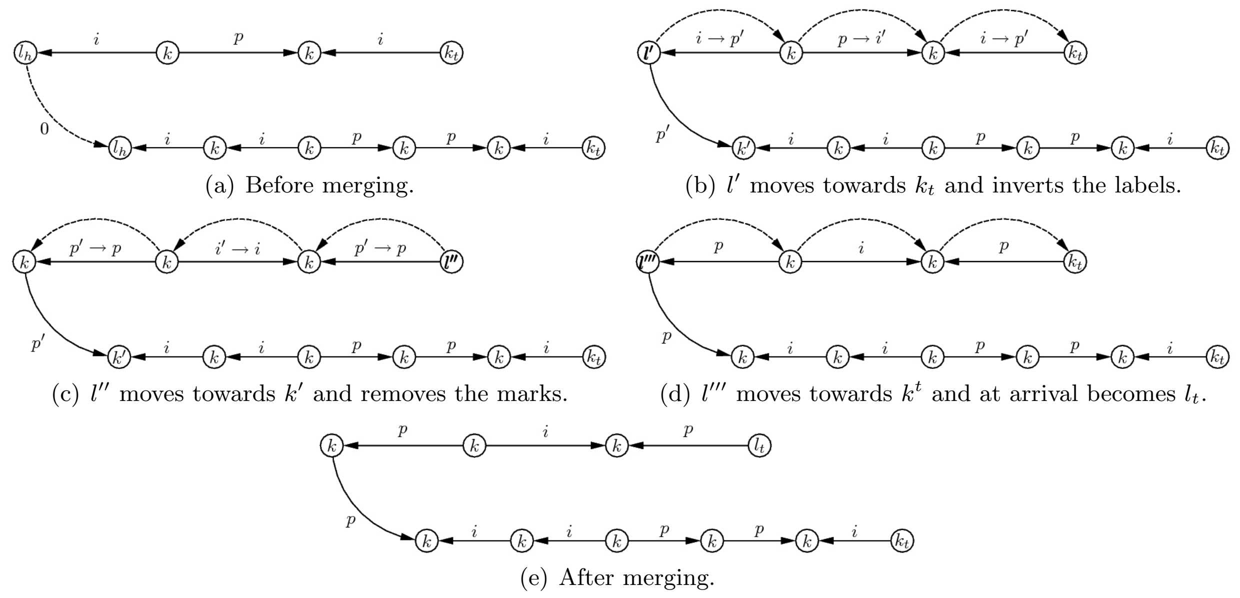
References

1. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, mar 2006.
2. Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *PODC '06: Proceedings of the 25th annual ACM Symposium on Principles of Distributed Computing*, pages 292–299, New York, NY, USA, 2006. ACM Press.
3. James Aspnes and Eric Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science*, 93:98–117, October 2007.
4. Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. Passively mobile communicating logarithmic space machines. Technical Report FRONTS-TR-2010-16, RACTI, Patras, Greece, 2010.
5. Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. Brief announcement: Decidable graph languages by mediated population protocols. In *DISC*, pages 239–240, 2009.
6. Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. Mediated population protocols. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2, pages 363–374, 2009.
7. Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. Recent advances in population protocols. In *MFCS '09: Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science 2009*, pages 56–76, Berlin, Heidelberg, 2009. Springer-Verlag.
8. Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When birds die: Making population protocols fault-tolerant. In *DCOSS*, pages 51–66, 2006.

9. S. Ginsburg and E. H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.
10. Rachid Guerraoui and Eric Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 484–495, 2009.
11. 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.

Appendix

A An Example of the Merging Process



(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.

Fig. 3. Two line subgraphs are merged together.

B Proof of Lemma 1

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

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 line 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 line 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 line graph L_3. Additionally, the edge via which the line 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 line 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 to 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.

□

C Line Graphs Never Become Smaller

Lemma 3. *Correctly labeled line graphs never become smaller.*

Proof. Let r denote the number of nodes of a line graph L_1. For $r = 1$, the trivial line graph consists of an isolated simple leader. The only effective interaction is with another leader which is either an isolated node or the leader of another line graph L_2. In both cases, clearly, no line graph becomes smaller, because either L_1 obtains another node, or L_1 and L_2 get merged to form a new line graph whose number of nodes is L_2's nodes plus one. For $r > 1$ L_1 is a normal line graph with a leader, either a tail leader or a head leader. If L_1 is the result of some merging and is still running the merging process, then, according to Lemma 1, this process will eventually come to an end, and L_1 will then have a head or a tail leader. Then the only effective interaction is between L_1's leader and another leader, which either adds another node to L_1 or merges L_1 with another line graph, and in both cases no line graph becomes smaller. □

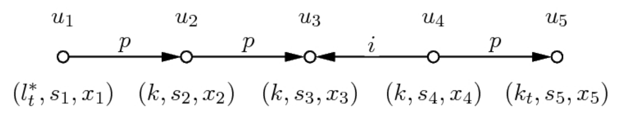
D Proof of Theorem 2

Assume that the communication graph $G = (V, E)$ is a correctly labeled line graph of n agents, where each agent takes its input symbol in a second state component *(the first component is used for the labels of the spanning process and is called* label component*). Then there is an MPP $\mathcal{A}$ that when running on such a graph simulates a deterministic TM $\mathcal{M}$ of $\mathcal{O}(n)$ (linear) space that computes symmetric predicates.*

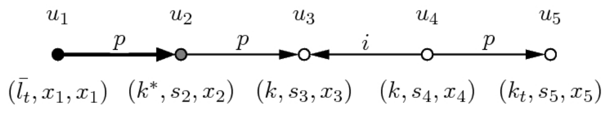
Proof. $\mathcal{A}$ exploits the fact that G is correctly labeled. It knows that there is a unique leader in one endpoint, called the *left endpoint*, which is either a head leader or a tail leader. Moreover, it knows that the other endpoint, the *right endpoint*, is a tail non-leader and that all other (internal) agents are simple non-leaders. All these are w.r.t. to $\mathcal{A}$'s label component. $\mathcal{A}$ also knows that the agents' second components contain initially their initial state, which is the same as their input symbol. It additionally knows that the edges have correct direction labels, which implies that those following the direction of the path from the leader endpoint to the non-leader endpoint are proper, while the remaining are inverse.

The leader starts simulating the TM. Since $\mathcal{M}$ computes symmetric predicates, it gives the same output for all permutations of any input vector's symbols. From [1] and [3] it is known that if all edges were proper then there is a population protocol, and, thus, an MPP, that simulates $\mathcal{M}$. It remains to show that having inverse edges cannot not harm the simulation. The explanation is as follows. Assume that an agent u has $\mathcal{M}$'s head over the last symbol of its state component (each agent can use a constant number of such symbols due to 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 is easy to check that in this manner the correct directions can always be followed. Finally, $\mathcal{A}$ can detect the endpoints by their distinct labels. In fact, since the MPP model can also store states on the edges, it can use an additional edge component for the simulation in order to double the available space. □

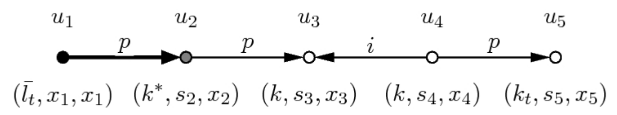
E An Example of the Reinitialization Process


u_1 u_2 u_3 u_4 u_5
p p i p
(l_t^*, s_1, x_1) (k, s_2, x_2) (k, s_3, x_3) (k, s_4, x_4) (k_t, s_5, x_5)

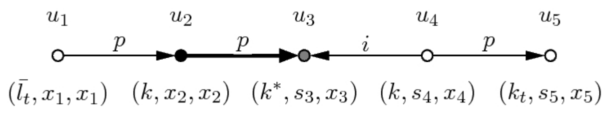
(a) Just after merging. The leader endpoint has the special star mark. The reinitialization process begins.


u_1 u_2 u_3 u_4 u_5
p p i p
$(\bar{l}_t, x_1, x_1)$ (k^*, s_2, x_2) (k, s_3, x_3) (k, s_4, x_4) (k_t, s_5, x_5)

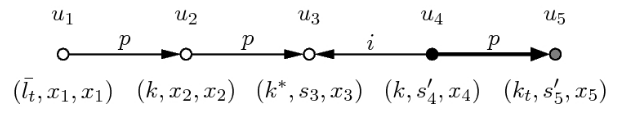
(b) u_1 becomes reinitialized and u_2 obtains the star mark. Note that u_1's first component obtains a bar mark to block u_1 from participating in another merging process.


u_1 u_2 u_3 u_4 u_5
p p i p
$(\bar{l}_t, x_1, x_1)$ (k^*, s_2, x_2) (k, s_3, x_3) (k, s_4, x_4) (k_t, s_5, x_5)

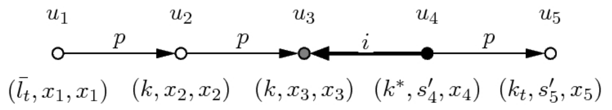
(c) Nothing happens, since only u_1 has been reinitialized (u_2 still has the star mark).


u_1 u_2 u_3 u_4 u_5
p p i p
$(\bar{l}_t, x_1, x_1)$ (k, x_2, x_2) (k^*, s_3, x_3) (k, s_4, x_4) (k_t, s_5, x_5)

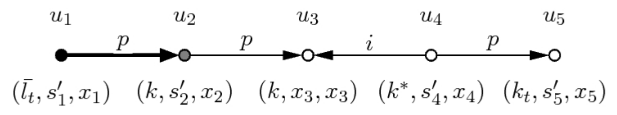
(d) u_2 becomes reinitialized and passes the mark to u_3.


u_1 u_2 u_3 u_4 u_5
p p i p
$(\bar{l}_t, x_1, x_1)$ (k, x_2, x_2) (k^*, s_3, x_3) (k, s'_4, x_4) (k_t, s'_5, x_5)

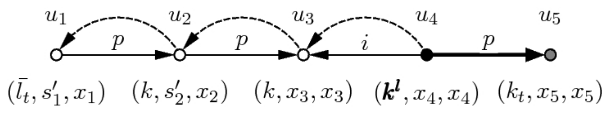
(e) Both u_4 and u_5 have not been reinitialized yet. A simulation step is executed but this is unimportant since both u_4 and u_5 will be reinitialized in future steps and cannot communicate with reinitialized agents (the star mark acts as a separator).


u_1 u_2 u_3 u_4 u_5
p p i p
$(\bar{l}_t, x_1, x_1)$ (k, x_2, x_2) (k, x_3, x_3) (k^*, s'_4, x_4) (k_t, s'_5, x_5)

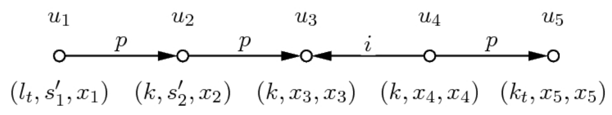
(f) u_3 becomes reinitialized and passes the mark to u_4.


u_1 u_2 u_3 u_4 u_5
p p i p
$(\bar{l}_t, s'_1, x_1)$ (k, s'_2, x_2) (k, x_3, x_3) (k^*, s'_4, x_4) (k_t, s'_5, x_5)

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


u_1 u_2 u_3 u_4 u_5
p p i p
$(\bar{l}_t, s'_1, x_1)$ (k, s'_2, x_2) (k, x_3, x_3) $(\boldsymbol{k^l}, x_4, x_4)$ (k_t, x_5, x_5)

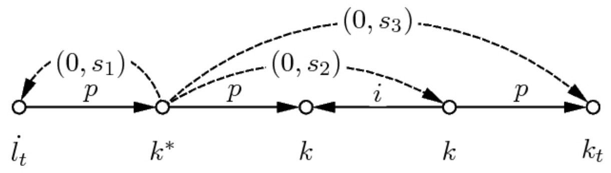
(h) Both become reinitialized and u_4 goes to k^l. The "left" mark will travel to u_1 to remove its mark and indicate the end of the reinitialization process.


u_1 u_2 u_3 u_4 u_5
p p i p
(l_t, s'_1, x_1) (k, s'_2, x_2) (k, x_3, x_3) (k, x_4, x_4) (k_t, x_5, x_5)

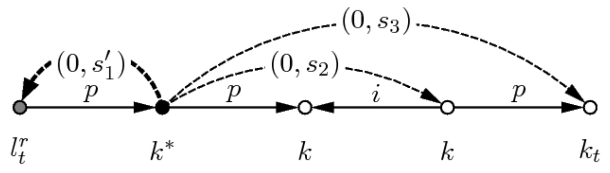
(i) After the completion of the reinitialization process. The leader is again ready for merging.

Fig. 4. An example of the reinitialization process just after two line 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.

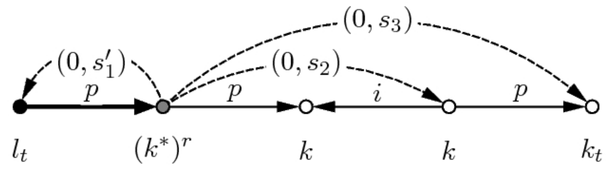
F An Example: Simulating a $\mathcal{O}(n^2)$-space Deterministic TM



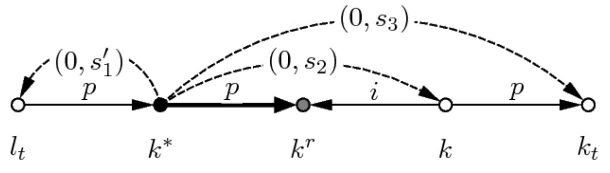
(a) The agent in k^* controls now the simulation.



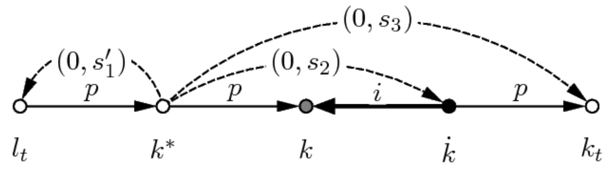
(b) A step of the simulation is executed on the inactive edge. The TM says "right" so k^* must next run the simulation on the first inactive edge to the right.



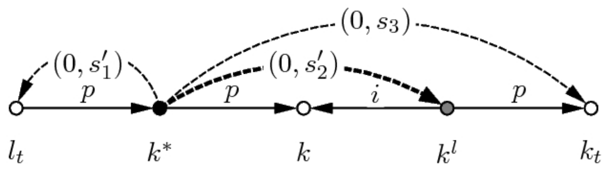
(c) Mark r travels to the right until it meets the first agent that has an incoming inactive edge from k^*.



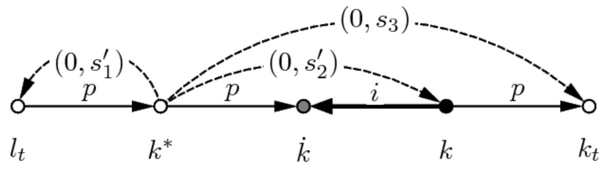
(d) The mark still travels.



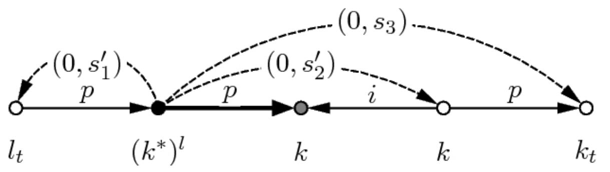
(e) The correct agent was found. The special dot mark will make the simulation run on the next inactive edge.



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

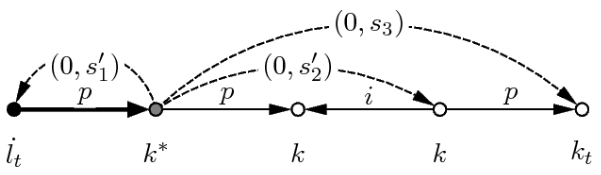


(g) The dot mark is placed on the wrong agent (because we only use for the simulation the inactive outgoing edges of k^*).

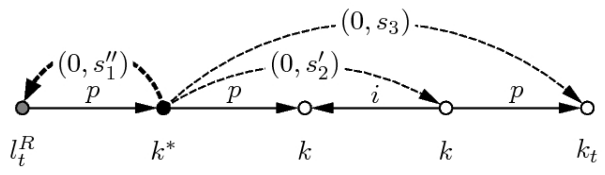


(h) The error is detected because the interaction happened through an active edge. The l mark continues its travel to the left.

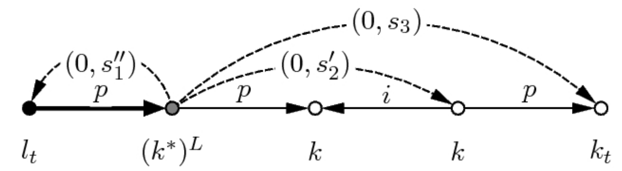
Fig. 5. An example of simulating a $\mathcal{O}(n^2)$-space deterministic TM. The simulation is performed on the second (state) component of the inactive edges (those whose first component is 0). The bold edge indicates the pair that has just interacted. The black agent is the initiator and the grey 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...).



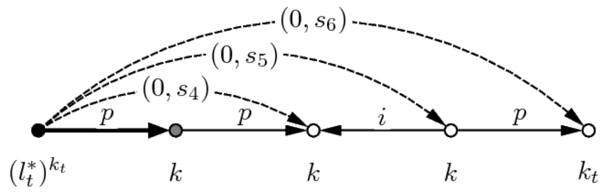
(i) The dot mark is placed at the leader endpoint.



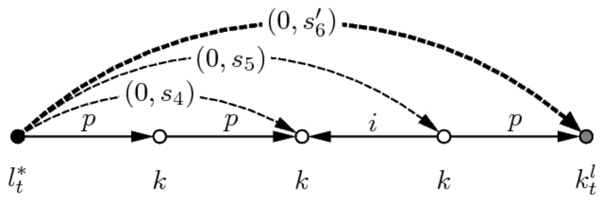
(j) A step of the simulation is executed. The TM says again "left" but it is already at the leftmost agent. A special R mark is created to change the agent that has control of the simulation.



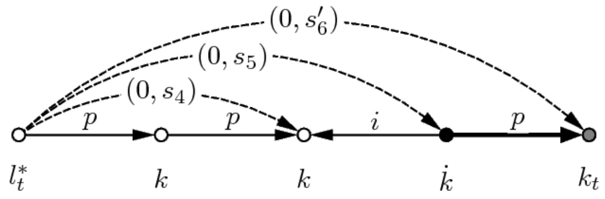
(k) The agent that has control was detected. The L mark will pass control to the left neighbor.



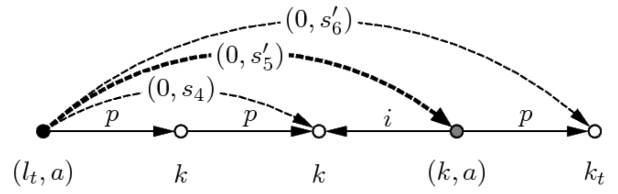
(l) The star mark was passed to the left. The left endpoint has now control of the simulation and will use its own inactive outgoing edges. The 'k_t' mark indicates that it must continue from its last outgoing edge (acts like an artificial dot mark over the non-leader endpoint.



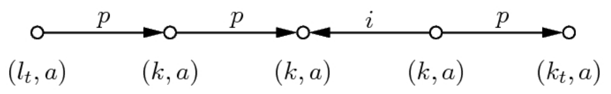
(m) A step of the simulation is executed. The TM says again "left".



(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.

Fig. 5. 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.

G Simulating a $\mathcal{O}(n^2)$-space Deterministic TM: Intermediate Agents

When the control of the simulation reaches a non-leader agent the situation is a little more complex. Since the sequence in which the inactive outgoing edges are accessed is the sequence in which the TM accesses its tape cells, there are two distinct cases: The first one is that an agent gets the star mark from its neighbor closer (w.r.t. the line graph path) to the leader endpoint (hence for the example described in Theorem 4 towards the right non-leader endpoint). The second is that the mark is passed from the neighbor closer to the non-leader endpoint (in the same example towards the leader endpoint). These two cases are different and should be handled accordingly (for the simulation to be correct) as explained in the following paragraphs.

The first one implies that the agent towards the leader has reached the last inactive edge and the TM's simulation needs to proceed to the next tape cell. Note that we consider an order of outgoing edges from any agent to everyone else in the line graph, so that the first edge is the one incident to the leader endpoint, the second the one incident to the leader's neighbor and so forth until the (last) one incident to the non-leader endpoint. This is the case described in Theorem 4 and therefore, the agent that has now the star mark can only have an effective interaction with the leader endpoint (in order to follow the ordering described above).

The second case on the other hand, implies that the agent towards the non-leader endpoint has reached its first inactive edge (using the same ordering with the first case) and the TM's simulation needs to go to the previous tape cell. In this case the agent that has now the star mark can only have an effective interaction with the non-leader endpoint. This is because, for any agent, the first inactive edge of its neighbor towards the non-leader endpoint corresponds to the tape cell next to the cell represented by the last inactive edge of the agent (due to the ordering of inactive edges described above). This last inactive edge is, in this case, the outgoing inactive edge of the currently star-marked agent to the non-leader endpoint. Therefore the star-marked agent must interact with the non-leader endpoint to continue the simulation to the correct tape cell. Note that these cases concern the intermediate agents because both endpoints (leader and tail non-leader) already know the sequence in which they should access their inactive edges (and thus the endpoint with which they should interact) once the control of the simulation is passed to them.

As a result, the two cases must be handled separately. This separation is simple: In the first case the agent that takes the star symbol updates its state to $(k^*)^{l_t}$ (meaning that it has to interact with the leader endpoint) and in the second the updated state is $(k^*)^{k_t}$ (meaning that it has to interact with the non-leader endpoint). Then the only available interactions would be $((k^*)^{l_t}, l)$ and $((k^*)^{k_t}, k_t)$. In both interactions the first star-marked agent updates its state to k^* and the second updates to $\dot{l}$ and $\dot{k}_t$ respectively. Then the simulation proceeds as in the leaders case described in Theorem 4 (by moving the dot mark according to the TM's transitions). Note that if the dot mark reaches the star-marked agent then it moves the dot towards the same direction, to the first agent that can be reached via an inactive edge. This is done because only inactive edges are used for the TM's execution. Therefore, whenever a star-marked agent has to move the dot to a neighbor with whom it has an active outgoing edge that neighbor does not get the dot but updates its state to either k^r or k^l according to the direction that the dot is moving. The above mechanism ensures that each agent only visits its inactive outgoing edges in the following sequence: From leader to non-leader when the simulation moves right and the reverse when it moves left. Moreover the group of all agents' inactive outgoing edges provides the $\mathcal{O}(n^2)$ space needed for the simulation.

H $SSPACE(n^2) \subseteq MPS$: A Formal Constructive Proof

Let Σ be an alphabet and $L \subseteq \Sigma^*$ a language such that L is *symmetric* and $L \in SPACE(n^2)$. A language $L \subseteq \Sigma^*$ is symmetric if $x \in L$ and x' is a permutation of x's symbols implies that $x' \in L$. Moreover, since $L \in SPACE(n^2)$ there exists a deterministic TM $\mathcal{M}$ deciding L in $\mathcal{O}(n^2)$ space (that is, on any input $x \in \Sigma^*$, $\mathcal{M}$ accepts if $x \in L$ and rejects if $x \notin L$ in $\mathcal{O}(|x|^2)$ space).

Let $p_L : \Sigma^* \to \{0,1\}$ be the predicate corresponding to L and defined as $p_L(x) = 1$ iff $x \in L$. Generally, such a predicate $p : \Sigma^* \to \{0,1\}$ is symmetric if for all $x, x' \in \Sigma^*$ such that x' is a permutation of x it holds that $p(x) = p(x')$.

Lemma 4. *p_L is symmetric iff L is symmetric.*

Proof. If p_L is symmetric then no permutation is missing from its support $p_L^{-1}(1)$. But $L = p_L^{-1}(1)$ (precisely those strings that make the predicate true), thus, L is symmetric. If L is symmetric then $p_L^{-1}(1)$ is also symmetric and so must be $p_L^{-1}(0)$, otherwise $p_L^{-1}(1)$ wouldn't be either (some permutation missing from $p_L^{-1}(0)$ would belong to $p_L^{-1}(1)$). □

So, let $\mathcal{M} = (Q_m, \Sigma, \Gamma, \delta_m, q_1, q_{accept}, q_{reject})$ be the TM deciding L. Let $Q_i = Q_m - \{q_{accept}, q_{reject}\}$. Given a transition $\delta_m(q, \gamma) = (q', \gamma', D)$, where $q, q' \in Q_m$, $\gamma, \gamma' \in \Gamma$, and $D \in \{L, R\}$, we define $\delta_m^1(q,\gamma) = q'$, $\delta_m^2(q,\gamma) = \gamma'$, and $\delta_m^3(q,\gamma) = D$. We construct an SMPP $\mathcal{B} = (X, Y, Q, S, I, O, \delta)$ that stably computes p_L. Formally, this means that for all $\sigma_1\sigma_2\cdots\sigma_n = x \in \Sigma^*$, where $n \geq 3$, when $\mathcal{B}$ runs on the complete communication graph of n agents and on input assignment $(\sigma_1, \sigma_2, \ldots, \sigma_n)$, all agents eventually output $p_L(x)$. In fact, we will straightly consider any $\sigma_1\sigma_2\cdots\sigma_n = x \in \Sigma^*$ as an input assignment for the complete communication graph of $|x| = n$ agents, where agent i gets input σ_i (by assuming an arbitrary ordering on the set of agents).

The input alphabet of $\mathcal{B}$ is Σ (that is, $X = \Sigma$). Thus, given a population of n agents $\{1, 2, \ldots, n\}$, the input to those agents may be any $x \in \Sigma^*$ of length n. We want all agents to output 1 if $p_L(x) = 1$ and 0 if $p_L(x) = 0$. The idea is to construct $\mathcal{B}$ such that it simulates $\mathcal{M}$. There is obviously enough space to run this simulation, because $\mathcal{M}$ on input x, where $|x| = n$, uses $\mathcal{O}(n^2)$ space and since $\mathcal{B}$, for the same input, runs on the complete communication graph of n nodes, it has $\mathcal{O}(n^2)$ edges to use as cells. Then, if we manage to make this construction, if $\mathcal{M}$ ever answers "accept" all agents output 1 and if it ever answers "reject" all agents output 0 (at least one of these answers will be provided by $\mathcal{M}$ in a finite number of steps).

The output alphabet Y is $\{0,1\}$, since we consider predicates. Each agent state $c \in Q$ consists of three components (c_1, c_2, c_3). c_1 is the *label* used by the spanning, merging, and reinitialization processes, c_2 is the *state of the simulation*, and c_3 is the *input backup*. Each edge state $s \in S$ consists of either one or two components. Active edges only use one component that identifies them as proper/inverse and inactive edges have states of the form $(0, s_2)$, where s_2 plays the role of a cell of $\mathcal{M}$, that is, it is a symbol from Γ.

Let $G = (V, E)$ be the communication graph. For all $e \in E$ we have that $\iota(e) = (0, \sqcup)$, that is, initially all edges are inactive and their cells contain the empty symbol $\sqcup$ (ι is the *edge initialization function*). For all $\sigma \in \Sigma$ we have that $I(\sigma) = (l, q_0, \sigma)$, that is, initially all agents are simple leaders, the state of the simulation is q_0 (some state not appearing in Q_m) in all agents, and all agents contain a read-only backup of their input. It remains to define O and δ. We begin from the latter.

Remark 1. The states (symbols) used here and some implementation choices are in general different from those used in the main text and in the graphical examples presented so far in the Appendix.

This is, mainly, due to the fact that here we provide a formal version of the whole transition function. Though, the main ideas are precisely the same. Moreover, *ineffective transitions* (also called *identity transitions*) [4] are omitted.

Spanning Process:

$$\begin{aligned}(l,\cdot,\cdot),(l,\cdot,\cdot),(0,\cdot) &\to (k_t,\cdot,\cdot),(l_h,\cdot,\cdot),i\\(l_h,\cdot,\cdot),(l,\cdot,\cdot),(0,\cdot) &\to (k^b,q_0,\cdot),(l_h^b,q_0,\cdot),i\\(l,\cdot,\cdot),(l_h,\cdot,\cdot),(0,\cdot) &\to (l_t^b,q_0,\cdot),(k^b,q_0,\cdot),p\\(l_t,\cdot,\cdot),(l,\cdot,\cdot),(0,\cdot) &\to (k^b,q_0,\cdot),(l_h^b,q_0,\cdot),i\\(l,\cdot,\cdot),(l_t,\cdot,\cdot),(0,\cdot) &\to (l_t^b,q_0,\cdot),(k^b,q_0,\cdot),p\end{aligned}$$

Merging Process:

$$(l_j,\cdot,\cdot),(l_k,\cdot,\cdot),(0,\cdot) \to (l',\cdot,\cdot),(k',\cdot,\cdot),p' \text{ for } j,k\in\{h,t\}$$

// l' moves towards the non-leader endpoint of the first
// line graph inverting and marking all the active edges

$$\begin{aligned}(k,\cdot,\cdot),(l',\cdot,\cdot),i &\to (l',\cdot,\cdot),(k,\cdot,\cdot),p'\\(l',\cdot,\cdot),(k,\cdot,\cdot),p &\to (k,\cdot,\cdot),(l',\cdot,\cdot),i'\\(k_t,\cdot,\cdot),(l',\cdot,\cdot),i &\to (l''_0,\cdot,\cdot),(k,\cdot,\cdot),p'\\(l',\cdot,\cdot),(k_t,\cdot,\cdot),p &\to (k,\cdot,\cdot),(l''_0,\cdot,\cdot),i'\end{aligned}$$

// once the non-leader tail is reached, a new special state is generated that travels
// towards the initially marked agent, erasing all marks from the active edges

$$\begin{aligned}(l''_0,\cdot,\cdot),(k,\cdot,\cdot),p' &\to (k_t,\cdot,\cdot),(l'',\cdot,\cdot),p\\(l'',\cdot,\cdot),(k,\cdot,\cdot),p' &\to (k,\cdot,\cdot),(l'',\cdot,\cdot),p\\(k,\cdot,\cdot),(l'',\cdot,\cdot),i' &\to (l'',\cdot,\cdot),(k,\cdot,\cdot),i\end{aligned}$$

// when the initially marked agent is reached a new state l''' is generated
// and traverses the line graph again towards the opposite direction

$$\begin{aligned}(l'',\cdot,\cdot),(k',\cdot,\cdot),p' &\to (l''',\cdot,\cdot),(k,\cdot,\cdot),p\\(k,\cdot,\cdot),(l''',\cdot,\cdot),p &\to (l''',\cdot,\cdot),(k,\cdot,\cdot),p\\(l''',\cdot,\cdot),(k,\cdot,\cdot),i &\to (k,\cdot,\cdot),(l''',\cdot,\cdot),i\end{aligned}$$

// when the non-leader tail is reached it becomes the leader of the newly merged line graph;
// both the leader endpoint and its neighbor are in blocked state q^b after termination

$$(k_t,\cdot,\cdot),(l''',\cdot,\cdot),p \to (l_t^b,q_0,\cdot),(k^b,q_0,\cdot),p$$

[4] A transition $(a,b,c)\to(a',b',c')$ is called *ineffective* or *identity* if $a'=a$, $b'=b$, and $c'=c$.

Reinitialization Process [5]*:*

// leader's blocked neighbor propagates its block state to the rest of the line
// graph towards the non-leader endpoint resetting the simulation state

$$(k^b,\cdot,\cdot),(k,\cdot,\cdot),p \to (k^b,\cdot,\cdot),(k^b,q_0,\cdot),p$$
$$(k,\cdot,\cdot),(k^b,\cdot,\cdot),i \to (k^b,q_0,\cdot),(k^b,\cdot,\cdot),i$$

// once the non-leader tail is reached, k^b_0 state is generated and moves towards the leader endpoint

$$(k_t,\cdot,\cdot),(k^b,\cdot,\cdot),i \to (k^b_t,q_0,\cdot),(k^b_0,\cdot,\cdot),i$$
$$(k^b,\cdot,\cdot),(k^b_0,\cdot,\cdot),p \to (k^b_0,\cdot,\cdot),(k^b,\cdot,\cdot),p$$
$$(k^b_0,\cdot,\cdot),(k^b,\cdot,\cdot),i \to (k^b,\cdot,\cdot),(k^b_0,\cdot,\cdot),i$$

// when k^b_0 reaches the leader it gives it the star mark and its neighbor
// points to the agent that will get the dot mark; the dot mark determines
// which outgoing edge of the star-marked agent is going to be reinitialized;
// it moves from the leader towards the non-leader endpoint

$$(l^b_t,\cdot,\cdot),(k^b_0,\cdot,\cdot),p \to (l^*_t,\cdot,\cdot),(k^b_r,\cdot,\cdot),p$$
$$(k^b_r,\cdot,\cdot),(k^b,\cdot,\cdot),p \to (k^b,\cdot,\cdot),(\dot{k}^b,\cdot,\cdot),p$$
$$(k^b,\cdot,\cdot),(k^b_r,\cdot,\cdot),i \to (\dot{k}^b,\cdot,\cdot),(k^b,\cdot,\cdot),i$$

// reinitialize the outgoing edge from the star-marked leader to the dotted agent

$$(l^*_t,\cdot,\cdot),(\dot{k}^b,\cdot,\cdot),(0,\gamma) \to (l^*_t,\cdot,\cdot),(k^b_r,\cdot,\cdot),(0,\sqcup) \text{ for all } \gamma\in\Gamma$$
$$(k^b_t,\cdot,\cdot),(k^b_r,\cdot,\cdot),i \to (\dot{k}^b_t,\cdot,\cdot),(k^b,\cdot,\cdot),i$$

// reinitialize the final outgoing edge from the star-marked leader to the dotted tail
// non-leader; once the special bar mark given to the tail non-leader reaches the leader endpoint
// the latter will pass its star mark to the next agent of the line graph towards the non-leader
// endpoint

$$(l^*_t,\cdot,\cdot),(\dot{k}^b_t,\cdot,\cdot),(0,\gamma) \to (l^*_t,\cdot,\cdot),(k^{\bar{b}}_t,\cdot,\cdot),(0,\sqcup) \text{ for all } \gamma\in\Gamma$$

// the bar mark is moving towards the star-marked agent (on the leader's direction)

$$(k^{\bar{b}}_t,\cdot,\cdot),(k^b,\cdot,\cdot),i \to (k^b_t,\cdot,\cdot),(k^{\bar{b}},\cdot,\cdot),i$$
$$(k^b,\cdot,\cdot),(k^{\bar{b}},\cdot,\cdot),p \to (k^{\bar{b}},\cdot,\cdot),(k^b,\cdot,\cdot),p$$
$$(k^{\bar{b}},\cdot,\cdot),(k^b,\cdot,\cdot),i \to (k^b,\cdot,\cdot),(k^{\bar{b}},\cdot,\cdot),i$$

[5] We present it only for a tail leader but similar ideas are applied in the case of a head leader.

// once the leader sees the bar mark via its proper edge it passes the star mark to its neighbor;
// the leader also gets the dot mark since the outgoing edge of the newly star-marked
// agent towards him is inactive (this is the case for the tail leader);
// the reinitialization continues similar to the star-marked leader's case

$$(l_t^*, \cdot, \cdot), (k^{\bar{b}}, \cdot, \cdot), p \rightarrow (\dot{l}_t^b, \cdot, \cdot), (k^*, \cdot, \cdot), p$$
$$(k^*, \cdot, \cdot), (\dot{l}_t^b, \cdot, \cdot), (0, \gamma) \rightarrow (k^*, \cdot, \cdot), (l_t^r, \cdot, \cdot), (0, \sqcup) \text{ for all } \gamma \in \mathit{\Gamma}$$
$$(l_t^r, \cdot, \cdot), (k^b, \cdot, \cdot), p \rightarrow (l_t^b, \cdot, \cdot), (\dot{k}^b, \cdot, \cdot), p$$

// if the dot mark reaches the star-marked agent the latter gets to a special state k_1^*

$$(l_t^r, \cdot, \cdot), (k^*, \cdot, \cdot), p \rightarrow (l_t^b, \cdot, \cdot), (k_1^*, \cdot, \cdot), p$$
$$(k^*, \cdot, \cdot), (\dot{k}^b, \cdot, \cdot), i \rightarrow (k_1^*, \cdot, \cdot), (k^b, \cdot, \cdot), i$$
$$(k_r^b, \cdot, \cdot), (k^*, \cdot, \cdot), p \rightarrow (k^b, \cdot, \cdot), (k_1^*, \cdot, \cdot), p$$

// since only inactive edges are reinitialized, if the agent in k_1^* has a proper edge to its neighbor
// towards the tail non-leader then it passes the r mark to this neighbor; this way the dot mark
// will reach the neighbor of the agent with the r mark towards the non-leader endpoint;
// if not then the neighbor of the agent in k_1^* towards the tail non-leader will get the dot mark;
// in any case the star-marked agent will definitely have an inactive edge
// to the new dot-marked agent and the process can continue

$$(k_1^*, \cdot, \cdot), (k^b, \cdot, \cdot), p \rightarrow (k^*, \cdot, \cdot), (k_r^b, \cdot, \cdot), p$$
$$(k^b, \cdot, \cdot), (k_1^*, \cdot, \cdot), i \rightarrow (\dot{k}^b, \cdot, \cdot), (k^*, \cdot, \cdot), i$$
$$(k_t^b, \cdot, \cdot), (k_1^*, \cdot, \cdot), i \rightarrow (\dot{k}_t^b, \cdot, \cdot), (k^*, \cdot, \cdot), i$$
$$(k^*, \cdot, \cdot), (\dot{k}^b, \cdot, \cdot), (0, \gamma) \rightarrow (k^*, \cdot, \cdot), (k_r^b, \cdot, \cdot), (0, \sqcup) \text{ for all } \gamma \in \mathit{\Gamma}$$

// as in the leader's case when a star-marked agent reinitializes
// its last inactive outgoing edge the bar mark is generated

$$(k^*, \cdot, \cdot), (\dot{k}_t^b, \cdot, \cdot), (0, \gamma) \rightarrow (k^*, \cdot, \cdot), (k_t^{\bar{b}}, \cdot, \cdot), (0, \sqcup) \text{ for all } \gamma \in \mathit{\Gamma}$$

// when all star-marked agent's inactive outgoing edges have been reinitialized and the bar mark
// has reached the star-marked agent, the star mark is passed to the neighbor towards the
// non-leader endpoint and a new special state is generated (k_2^b); then this
// special state traverses the line graph towards the leader

$$(k^*, \cdot, \cdot), (k^{\bar{b}}, \cdot, \cdot), p \rightarrow (k_2^b, \cdot, \cdot), (k^*, \cdot, \cdot), p$$
$$(k^{\bar{b}}, \cdot, \cdot), (k^*, \cdot, \cdot), i \rightarrow (k^*, \cdot, \cdot), (k_2^b, \cdot, \cdot), p$$
$$(k_t^{\bar{b}}, \cdot, \cdot), (k^*, \cdot, \cdot), i \rightarrow (k_t^*, \cdot, \cdot), (k_2^b, \cdot, \cdot), i$$
$$(k^b, \cdot, \cdot), (k_2^b, \cdot, \cdot), p \rightarrow (k_2^b, \cdot, \cdot), (k^b, \cdot, \cdot), p$$
$$(k_2^b, \cdot, \cdot), (k^b, \cdot, \cdot), i \rightarrow (k^b, \cdot, \cdot), (k_2^b, \cdot, \cdot), i$$

// once it finds the leader it gives him the dot mark and a new round of reinitializations begins

$$(l_t^b, \cdot, \cdot), (k_2^b, \cdot, \cdot), p \to (\dot{l}_t^b, \cdot, \cdot), (k^b, \cdot, \cdot), p$$
$$(k^*, \cdot, \cdot), (\dot{l}_t^b, \cdot, \cdot), (0, \gamma) \to (k^*, \cdot, \cdot), (l_t^r, \cdot, \cdot), (0, \sqcup) \text{ for all } \gamma \in \Gamma$$
$$(k_t^*, \cdot, \cdot), (\dot{l}_t^b, \cdot, \cdot), (0, \gamma) \to (k_t^*, \cdot, \cdot), (l_t^r, \cdot, \cdot), (0, \sqcup) \text{ for all } \gamma \in \Gamma$$
$$(k_t^*, \cdot, \cdot), (\dot{k}^b, \cdot, \cdot), (0, \gamma) \to (k_t^*, \cdot, \cdot), (k_r^b, \cdot, \cdot), (0, \sqcup) \text{ for all } \gamma \in \Gamma$$

// once all the inactive outgoing edges of the tail non-leader have been reinitialized another
// special state k_f^b is generated and traverses the line graph from the non-leader endpoint to
// the leader endpoint erasing all the b marks from the agents; when it reaches the leader endpoint
// the special state disappears indicating that reinitialization process has come to an end

$$(k_t^*, \cdot, \cdot), (\dot{k}^b, \cdot, \cdot), i \to (k_t, \cdot, \cdot), (k_f^b, \cdot, \cdot), i$$
$$(k^b, \cdot, \cdot), (k_f^b, \cdot, \cdot), p \to (k_f^b, \cdot, \cdot), (k, \cdot, \cdot), p$$
$$(k_f^b, \cdot, \cdot), (k^b, \cdot, \cdot), i \to (k, \cdot, \cdot), (k_f^b, \cdot, \cdot), i$$
$$(l_t^b, \cdot, \cdot), (k_f^b, \cdot, \cdot), p \to (l_t, \cdot, \cdot), (k, \cdot, \cdot), p$$

Simulation [6]*:*

// inputs of all agents are stored/written to the cell components of the first inactive outgoing
// edges of the line graph considering an ordering of edges that starts from the edges of the
// leader endpoint and continues to those the neighboring agents towards the non-leader endpoint;
// initially the leader writes the tail non-leader's input to its inactive outgoing edges towards the
// non-leader endpoint;this results in the generation of a dot mark which travels towards the
// leader endpoint indicating each time agent's input and the corresponding edge that this input
// is written

$$(l_t, q_0, \cdot), (k_t, q_0, \sigma), (0, \sqcup) \to (l_t, q_0', \cdot), (k_t, (q_0, l), \sigma), (0, \sigma) \text{ for all } \sigma \in \Sigma$$
$$(k_t, (q_0, l), \cdot), (k, q_0, \cdot), i \to (k_t, q_0, \cdot), (k, (q_0, d), \cdot), i$$
$$(l_t, q_0', \cdot), (k, (q_0, d), \sigma), (0, \sqcup) \to (l_t, q_0, \cdot), (k, (q_0, l), \sigma), (0, \sigma) \text{ for all } \sigma \in \Sigma$$
$$(k, q_0, \cdot), (k, (q_0, l), \cdot), p \to (k, (q_0, d), \cdot), (k, q_0, \cdot), p$$
$$(k, (q_0, l), \cdot), (k, q_0, \cdot), i \to (k, q_0, \cdot), (k, (q_0, d), \cdot), i$$

// the leader writes its input to the first inactive outgoing edge of its
// neighbor (the one towards the leader since it is the tail leader)

$$(l_t, q_0', \cdot), (k, (q_0, d), \cdot), p \to (l_t, (q_0, d), \cdot), (k, q_0', \cdot), p$$
$$(k, q_0', \cdot), (l_t, (q_0, d), \sigma), (0, \sqcup) \to (k, (q_0, r), \cdot), (l_t, (q_1, *), \sigma), (0, \sigma) \text{ for all } \sigma \in \Sigma$$

// the input symbol of the leader's neighbor is written on its second inactive outgoing edge; this
// edge is the one with its neighbor (3rd agent in the line graph counting from the leader)
// towards the non-leader endpoint if this edge is active

$$(k, q_0, \cdot), (k, (q_0, r), \cdot), i \to (k, (q_0, d), \cdot), (k, q_0', \cdot), i$$
$$(k, q_0', \sigma), (k, (q_0, d), \cdot), (0, \sqcup) \to (k, R, \sigma), (k, q_0, \cdot), (0, \sigma) \text{ for all } \sigma \in \Sigma$$

[6] We present it only for a tail leader but similar ideas are applied in the case of a head leader.

// otherwise the leader's neighbor (2nd agent) writes its input symbol to the
// outgoing edge towards its neighbor's neighbor (4th agent in the line graph)

$$
\begin{aligned}
(k,(q_0,r),\cdot),(k,q_0,\cdot),p &\rightarrow (k,q_0',\cdot),(k,(q_0,r'),\cdot),p\\
(k,(q_0,r'),\cdot),(k,q_0,\cdot),p &\rightarrow (k,q_0,\cdot),(k,(q_0,d),\cdot),p\\
(k,q_0,\cdot),(k,(q_0,r'),\cdot),i &\rightarrow (k,(q_0,d),\cdot),(k,q_0,\cdot),i
\end{aligned}
$$

// any star-marked (in control of the simulation) agent who "sees" (interacts as an intiator with)
// a dot-marked agent via an inactive edge applies the transition function of the simulated NTM
// using the star-marked agent's state as the state of the TM and the edge's cell component as the
// tape cell under TM's head; the resulting TM's state is stored in the star-marked agent's state
// component, the symbol written to the tape is stored on the inactive edge's cell component and
// the direction of the head (R or L) is marked on the dot-marked agent who loses its dot mark

$$
\begin{aligned}
(l_t,(q,*),\cdot),(k,d,\cdot),(0,\gamma) &\rightarrow (l_t,(\delta^1_m(q,\gamma),*),\cdot),(k,\delta^3_m(q,\gamma),\cdot),(0,\delta^2_m(q,\gamma)) \text{ for all } \gamma\in\Gamma,\ q\in Q_i\\
(l_t,(q,*),\cdot),(k_t,d,\cdot),(0,\gamma) &\rightarrow (l_t,(\delta^1_m(q,\gamma),*),\cdot),(k_t,\delta^3_m(q,\gamma),\cdot),(0,\delta^2_m(q,\gamma)) \text{ for all } \gamma\in\Gamma,\ q\in Q_i\\
(k,(q,*),\cdot),(k,d,\cdot),(0,\gamma) &\rightarrow (k,(\delta^1_m(q,\gamma),*),\cdot),(k,\delta^3_m(q,\gamma),\cdot),(0,\delta^2_m(q,\gamma)) \text{ for all } \gamma\in\Gamma,\ q\in Q_i\\
(k,(q,*),\cdot),(l_t,d,\cdot),(0,\gamma) &\rightarrow (k,(\delta^1_m(q,\gamma),*),\cdot),(l_t,\delta^3_m(q,\gamma),\cdot),(0,\delta^2_m(q,\gamma)) \text{ for all } \gamma\in\Gamma,\ q\in Q_i\\
(k,(q,*),\cdot),(k_t,d,\cdot),(0,\gamma) &\rightarrow (k,(\delta^1_m(q,\gamma),*),\cdot),(k_t,\delta^3_m(q,\gamma),\cdot),(0,\delta^2_m(q,\gamma)) \text{ for all } \gamma\in\Gamma,\ q\in Q_i\\
(k_t,(q,*),\cdot),(k,d,\cdot),(0,\gamma) &\rightarrow (k_t,(\delta^1_m(q,\gamma),*),\cdot),(k,\delta^3_m(q,\gamma),\cdot),(0,\delta^2_m(q,\gamma)) \text{ for all } \gamma\in\Gamma,\ q\in Q_i\\
(k_t,(q,*),\cdot),(l_t,d,\cdot),(0,\gamma) &\rightarrow (k_t,(\delta^1_m(q,\gamma),*),\cdot),(l_t,\delta^3_m(q,\gamma),\cdot),(0,\delta^2_m(q,\gamma)) \text{ for all } \gamma\in\Gamma,\ q\in Q_i
\end{aligned}
$$

// according to the L or R mark that an agent may has, it gives its left or right
// neighbor respectively a dot mark depending on the direction of NTM's head and resets its state

$$
\begin{aligned}
(k,\cdot,\cdot),(k,L,\cdot),p &\rightarrow (k,d,\cdot),(k,q_0,\cdot),p\\
(k,L,\cdot),(k,\cdot,\cdot),i &\rightarrow (k,q_0,\cdot),(k,d,\cdot),i\\
(k,R,\cdot),(k,\cdot,\cdot),p &\rightarrow (k,q_0,\cdot),(k,d,\cdot),p\\
(k,\cdot,\cdot),(k,R,\cdot),i &\rightarrow (k,d,\cdot),(k,q_0,\cdot),i\\
(k_t,\cdot,\cdot),(k,R,\cdot),i &\rightarrow (k_t,d,\cdot),(k,q_0,\cdot),i\\
(l_t,\cdot,\cdot),(k,L,\cdot),p &\rightarrow (l_t,d,\cdot),(k,q_0,\cdot),p
\end{aligned}
$$

// if a star-marked agent "sees" a dot-marked agent via an active edge then it gets to a special // state (R^* or L^*) according to the direction of dot-marked agent; if the dot-marked agent is // closer to the leader endpoint then the dot is passed to the closest agent towards the non-leader // endpoint that can be seen via an inactive edge; otherwise the dot is passed to the closest agent // towards the leader endpoint that can be seen via an inactive edge

$(l_t,(q,*),\cdot),(k,d,\cdot),p \rightarrow (l_t,(q,*),\cdot),(k,R,\cdot),p$ for all $q \in Q_i$

$(k,(q,*),\cdot),(k,d,\cdot),p \rightarrow (k,(q,L^*),\cdot),(k,q_0,\cdot),p$ for all $q \in Q_i$

$(k,\cdot,\cdot),(k,(q,L^*),\cdot),p \rightarrow (k,d,\cdot),(k,(q,*),\cdot),p$ for all $q \in Q_i$

$(k,(q,L^*),\cdot),(k,\cdot,\cdot),i \rightarrow (k,(q,*),\cdot),(k,L,\cdot),i$ for all $q \in Q_i$

$(k,(q,*),\cdot),(k,d,\cdot),i \rightarrow (k,(q,R^*),\cdot),(k,q_0,\cdot),p$ for all $q \in Q_i$

$(k,\cdot,\cdot),(k,(q,R^*),\cdot),i \rightarrow (k,d,\cdot),(k,(q,*),\cdot),i$ for all $q \in Q_i$

$(k,(q,R^*),\cdot),(k,\cdot,\cdot),p \rightarrow (k,(q,*),\cdot),(k,R,\cdot),p$ for all $q \in Q_i$

$(k,L,\cdot),(k,(q,*),\cdot),i \rightarrow (k,q_0,\cdot),(k,(q,L^*),\cdot),i$ for all $q \in Q_i$

$(k,R,\cdot),(k,(q,*),\cdot),p \rightarrow (k,q_0,\cdot),(k,(q,R^*),\cdot),p$ for all $q \in Q_i$

// if a star-marked agent has reached its final inactive outgoing edge (corresponding to the one // towards the tail non-leader if the star mark is not on the tail non-leader or to the one towards // the 3rd agent counting from the non-leader endpoint otherwise) and the simulation needs to // proceed to the right then the star marked agent gets to a special state R' which indicates // that the next interaction is with the leader endpoint; the next interaction gives the dot mark // to the leader endpoint and brings the R' agent to special state R''; in the next interaction the // R'' agent passes the star mark to the neighbor towards the tail non-leader and resets its state

$(k,(q,*),\cdot),(k_t,R,\cdot),(0,\cdot) \rightarrow (k,(q,R'),\cdot),(k_t,q_0,\cdot),(0,\cdot)$ for all $q \in Q_i$

$(k,(q,R'),\cdot),(l_t,\cdot,\cdot),(0,\cdot) \rightarrow (k,(q,R''),\cdot),(l_t,d,\cdot),(0,\cdot)$ for all $q \in Q_i$

$(k,(q,R''),\cdot),(k,\cdot,\cdot),p \rightarrow (k,q_0,\cdot),(k,(q,*),\cdot),p$ for all $q \in Q_i$

$(k,\cdot,\cdot),(k,(q,R''),\cdot),i \rightarrow (k,(q,*),(k,q_0,\cdot),\cdot),i$ for all $q \in Q_i$

$(l_t,(q,*),\cdot),(k_t,R,\cdot),(0,\cdot) \rightarrow (l_t,(q,R'),\cdot),(k_t,q_0,\cdot),(0,\cdot)$ for all $q \in Q_i$

$(l_t,(q,R'),\cdot),(k,\cdot,\cdot),p \rightarrow (l_t,d,\cdot),(k,(q,*),\cdot),p$ for all $q \in Q_i$

// the case is similar when the star-marked agent reaches its first inactive // outgoing edge and the simulation needs to proceed to the left

$(k,(q,*),\cdot),(l_t,L,\cdot),(0,\cdot) \rightarrow (k,(q,L'),\cdot),(l_t,q_0,\cdot),(0,\cdot)$ for all $q \in Q_i$

$(k,(q,L'),\cdot),(k_t,\cdot,\cdot),(0,\cdot) \rightarrow (k,(q,L''),\cdot),(k_t,d,\cdot),(0,\cdot)$ for all $q \in Q_i$

$(k,\cdot,\cdot),(k,(q,L''),\cdot),p \rightarrow (k,(q,*),\cdot),(k,q_0,\cdot),p$ for all $q \in Q_i$

$(k,(q,L''),\cdot),(k,\cdot,\cdot),i \rightarrow (k,q_0,\cdot),(k,(q,*),\cdot),i$ for all $q \in Q_i$

$(k_t,(q,*),\cdot),(l_t,L,\cdot),(0,\cdot) \rightarrow (k_t,(q,L'),\cdot),(l_t,q_0,\cdot),(0,\cdot)$ for all $q \in Q_i$

$(k_t,(q,L'),\cdot),(k,\cdot,\cdot),i \rightarrow (k_t,d,\cdot),(k,(q,*),\cdot),i$ for all $q \in Q_i$

// if any agent reaches an accept or reject state this
// state is propagated to the rest of the population

$$
\begin{aligned}
(j,(q',\cdot),\cdot),(k,(q,\cdot),\cdot),\cdot &\to (j,q',\cdot),(k,q',\cdot),\cdot \text{ for all } q' \in \{q_{accept}, q_{reject}\}, q \in Q_i, j,k \in \{l_t,k,k_t\}\\
(j,(q,\cdot),\cdot),(k,(q',\cdot),\cdot),\cdot &\to (j,q',\cdot),(k,q',\cdot),\cdot \text{ for all } q' \in \{q_{accept}, q_{reject}\}, q \in Q_i, j,k \in \{l_t,k,k_t\}\\
(j,(q',\cdot),\cdot),(k,q_0,\cdot) &\to (j,q',\cdot),(k,q',\cdot),\cdot \text{ for all } q' \in \{q_{accept}, q_{reject}\}, j,k \in \{l_t,k,k_t\}\\
(j,q_0,\cdot),(k,(q',\cdot),\cdot),\cdot &\to (j,q',\cdot),(k,q',\cdot),\cdot \text{ for all } q' \in \{q_{accept}, q_{reject}\}, j,k \in \{l_t,k,k_t\}\\
(j,q',\cdot),(k,(q,\cdot),\cdot),\cdot &\to (j,q',\cdot),(k,q',\cdot),\cdot \text{ for all } q' \in \{q_{accept}, q_{reject}\}, q \in Q_i, j,k \in \{l_t,k,k_t\}\\
(j,(q,\cdot),\cdot),(k,q',\cdot),\cdot &\to (j,q',\cdot),(k,q',\cdot),\cdot \text{ for all } q' \in \{q_{accept}, q_{reject}\}, q \in Q_i, j,k \in \{l_t,k,k_t\}\\
(j,q',\cdot),(k,q_0,\cdot),\cdot &\to (j,q',\cdot),(k,q',\cdot),\cdot \text{ for all } q' \in \{q_{accept}, q_{reject}\}, j,k \in \{l_t,k,k_t\}\\
(j,q_0,\cdot),(k,q',\cdot),\cdot &\to (j,q',\cdot),(k,q',\cdot),\cdot \text{ for all } q' \in \{q_{accept}, q_{reject}\}, j,k \in \{l_t,k,k_t\}
\end{aligned}
$$

It remains to define the output function O. We have $O(\cdot, q_{accept}, \cdot) = 1$ and $O(q) = 0$ for all $q \in Q - \{(\cdot, q_{accept}, \cdot)\}$.

I Proof of Theorem 6

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

Proof. In Theorems 4 and 5 it has been shown that the SMPP model can simulate a deterministic TM $\mathcal{M}$ of $\mathcal{O}(n^2)$ space. Some modifications are now presented that will allow the simulation of a nondeterministic TM $\mathcal{N}$ of the same memory size. $\mathcal{N}$ is similar to $\mathcal{M}$ defined in Appendix [H]. Note that $\mathcal{N}$ is a decider for some predicate in $SNSPACE(n^2)$, thus, it always halts.The modifications for $\mathcal{N}$ concern the agents states' components and the reinitialization process after each line graph merging.

Each agent state $c \in Q$ has an additional (fourth) component c_4 (from now on it will be referred as *choice component*) which is initialized to $t_{max} - 1$, where t_{max} is the greatest number of nondeterministic choices that $\mathcal{N}$ will ever face. It should be noted that any agent has to be able to store $\mathcal{N}$'s transition function (which includes the nondeterministic transitions) in order to perform a part of $\mathcal{N}$'s execution because that requires updating the corresponding components of its inactive outgoing edges ($\mathcal{N}$'s tape cells) according to this function. Also note that this assumption holds for the deterministic TM $\mathcal{M}$ presented in Appendix [H] and it is vital for both simulations. Obviously, there is a fixed upper bound on the number of nondeterministic choices (independent of the population size) and thus the agents can store them in their memories. In addition, t_{max} can be easily extracted from $\mathcal{N}$'s transition function stored in any agent.

As regards the reinitialization process it changes as follows: Every time a leader is reinitialized it sets its state's choice component to $t_{max} - 1$. Each time during the reinitialization that the star mark passes from an agent v to its right neighbor u (considering w.l.o.g. that reinitialization takes place from left to right) u sets its choice component to v's choice component -1. If the newly computed choice component is less than 0 then the corresponding agent sets this component to $t_{max} - 1$. This component's update is additional to the work each agent does during those interactions as described for the deterministic TM's simulation. From Theorem 5 we have that the reinitializations will eventually stop and that the final reinitialization takes place on a correctly labeled spanning line subgraph of the communication graph G. Since the reinitialization process is performed from the leader endpoint to the non-leader endpoint, the last reinitialization ends up with a line graph where the choice components of the agents, starting from the leader, will be $t_{max}-1, t_{max}-2, \dots, 1, 0, t_{max}-1, \dots$ and so forth. After that last reinitialization the choice components will not change any further.

To view it more systematically consider that each agent in the line graph has a position according to its distance from the leader endpoint. Leader has the position 0 its neighbor has 1, the non-leader neighbor of leader's neighbor has 2 and so forth. Then the formula that determines the choice component's value for each agent is $(t_{max}-1)-(position \mod t_{max})$. Note however that the agents do not know this formula nor their position and assign their choice components value during the reinitialization process as described in the previous paragraph. Position is dependent of the population size and therefore cannot be stored in a constant memory agent. The formula is given purely for clarifying the choice components' values after the termination of a reinitialization.

Whenever a nondeterministic choice has to be made between t candidates where $t \leq t_{max}$ by definition of t_{max}, the agent in control of $\mathcal{N}$'s simulation maps the numbers $0, 1, \dots, t-1$ to those candidates and waits for an arbitrary interaction. The choice component of the agent participating in the next interaction with the agent in control will provide the nondeterministic choice. This is achieved by having the agent in control of the simulation choose the candidate mapped to $v(c_4)$ mod t where v is the other participant of the interaction and $v(c_4)$ denotes the choice component of agent v. Note that since t_{max} is the greatest number of nondeterministic choices that can appear during $\mathcal{N}$'s execution, any time a nondeterministic choice has to be made all possible choices will be available provided that there are enough agents ($2t_{max} \leq n$ so that it is possible to see any choice number from $0, \dots, t_{max}-1$ no matter which agent has control of the simulation, since an agent cannot interact with itself).

As in $\mathcal{M}$, 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 reinitialized since the fact that a branch of the nondeterministic computation tree rejects does not mean that all branches reject. Fairness guarantees that, by making the nondeterministic choices in the previously described way, all possible paths in the tree representing $\mathcal{N}$'s nondeterministic computation will eventually be followed. Correctness lies in 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 since $\mathcal{N}$'s output function is the same as $\mathcal{M}$'s in Appendix [H] 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 from $\mathcal{M}$'s description that when an agent reaches the q_{accept} state it never alters its state again and therefore its output remains on 1, 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, 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.

□