
Hiroshima University Institutional Repository

Neighborhood Mutual Remainder: Self-Stabilizing Distributed Implementation and Applications*

Shlomi Dolev[1†], Sayaka Kamei[2†], Yoshiaki Katayama[3†], Fukuhito Ooshita[4†] and Koichi Wada[5†]

[1]Department of Computer Science, Ben-Gurion University of the Negev, Israel.
[2]Graduate School of Advanced Science and Engineering, Hiroshima University, Japan.
[3]Graduate School of Engineering, Nagoya Institute of Technology, Japan.
[4]Fukui University of Technology, Japan.
[5]Faculty of Science and Engineering, Hosei University, Japan.

Contributing authors: dolev@cs.bgu.ac.il; s10kamei@hiroshima-u.ac.jp; katayama@nitech.ac.jp; f-oosita@fukui-ut.ac.jp; wada@hosei.ac.jp;
†These authors contributed equally to this work.

Abstract

Motivated by the need to convert move-atomic assumption in *LOOK-COMPUTE-MOVE* (LCM) robot algorithms to be implemented in existing distributed systems, we define a new distributed fundamental task, the *Neighborhood Mutual Remainder* (NMR). Consider a situation where each process has a set of operations $\boldsymbol{O_p}$ and executes each operation in $\boldsymbol{O_p}$ infinitely often in distributed systems. Then, let $\boldsymbol{O_e} \subset \boldsymbol{O_p}$ be a subset of operations, which a process cannot execute while its closed neighborhood executes operations in $\boldsymbol{O_p} \setminus \boldsymbol{O_e}$. The NMR is defined for such a situation. A distributed algorithm that satisfies the NMR requirement should satisfy the following two properties: (1) Liveness is

*Preliminary brief announcement versions (one of five pages and the other of six pages) of this detailed complete paper can be found in [1, 2].

satisfied if a process executes each operation in $\boldsymbol{O_p}$ infinitely often, and (2) safety is satisfied if, when each process executes operations in $\boldsymbol{O_e}$, no process in its closed neighborhood executes operations in $\boldsymbol{O_p} \setminus \boldsymbol{O_e}$. We formalize the concept of NMR and give a simple self-stabilizing algorithm using the pigeon-hole principle to demonstrate the design paradigm to achieve NMR. A self-stabilizing algorithm tolerates transient faults (e.g., message loss, memory corruption, etc.) by its ability to converge from an arbitrary configuration to the legitimate one. In addition, we present an application of NMR to an LCM robot system for implementing a move-atomic property, where robots possess an independent clock that is advanced at the same speed. It is the first self-stabilizing implementation of the LCM synchronization for environments where each robot can have limited visibility and lights.

1 Introduction

In today's diverse distributed systems, there are many situations where exclusive control over operations (tasks) is required. That is, we need to control the schedule for executing certain specific operations so that no process can execute such operations when its closed neighborhood execute other operations. For example, while some discrete algorithms for the *LOOK-COMPUTE-MOVE* (LCM) robot systems [3] assume that each robot can move from a position to its destination instantaneously for simplicity. This means that when we consider their implementation, each robot cannot move while its neighbors observe the configuration. Therefore, the implementation must realize the exclusive control of MOVE operations and LOOK operations. Although it is very important to solve such a scheduling problem efficiently, it has not been formally formulated until now. Thus, in this paper, we define this scheduling problem as the *Neighborhood Mutual Remainder* (NMR) problem.

More formally, in a distributed system with a general, non-necessarily complete communication graph, consider the situation such that each process has some operations $O_p = \{op_1, op_2, \cdots, op_q\}$ and executes each operation in O_p infinitely often. Then, sometimes the distributed systems encounter mutually exclusive executions between operations, that is, operations in $O_e \subset O_p$ cannot be executed by a process concurrently with other operations in $O_p \setminus O_e$ by its closed neighborhood. We call such special operations in O_e, *exclusive operations*. The NMR allows up to all processes in the closed neighborhood to execute operations in $O_p \setminus O_e$ simultaneously, but requires a guarantee that the execution of the operations in O_e is mutually exclusive against the execution of the operations in $O_p \setminus O_e$. That is, distributed algorithm that satisfies the NMR requirement should satisfy the following two properties: (1) Liveness is satisfied if a process executes each operation in O_p infinitely often, and (2) safety is satisfied if, when each process executes operations in O_e, no process in its closed neighborhood executes operations in $O_p \setminus O_e$.

In this work, we give a self-stabilizing algorithm to demonstrate the design paradigm to achieve NMR. A self-stabilizing algorithm tolerates transient faults (e.g., message loss, memory corruption, etc.) by its ability to converge from an arbitrary configuration to the legitimate one.

To solve the NMR consistently, the processes should schedule the operations carefully. One may think we can apply *mutual exclusion* [4], *local mutual exclusion* [5–8], or *local group mutual exclusion* [9–11] to solve the local synchronization problem. Mutual exclusion (resp., local mutual exclusion) guarantees that no two processes (resp., no two neighboring processes) enter a *critical section* (CS) at the same time. Indeed, if processes execute operations in O_p only when they are in the CS, they can keep the consistency because no two neighboring processes execute any operations at the same time. However, this approach seems very expensive because the processes execute any operations sequentially, although they are allowed to execute operations in O_e or $O_p \setminus O_e$ at the same time. Although both of these concepts (i.e., (local) mutual exclusion) are similar to NMR, they differ from NMR in that these concepts provide exclusive control over processes, whereas NMR provides exclusive control over the execution of a set of operations. In particular, NMR offers a more efficient solution by enabling fully synchronized execution of operations. This means that all closed neighborhoods can simultaneously execute the same operation in O_e or $O_p \setminus O_e$. *Local group mutual exclusion* (LGME) guarantees that, if two neighboring processes execute their CS simultaneously, both are in the same group (i.e., using the same resource). This safety property is similar to NMR when the number of groups is only two. However, while LGME only cares about the CS, NMR requires that all processes will be out of the CS (i.e., all processes execute operations in O_e in the remainder) for a while. Also, to realize mutual exclusion (resp., local (group) mutual exclusion), processes should achieve symmetry breaking because one process should be selected to enter the CS among all processes (resp. the closed neighborhood for each process). However, in highly-symmetric distributed systems (e.g., the LCM robot systems), it is difficult or even impossible to achieve deterministic symmetry breaking and thus achieve mutual exclusion (resp., local (group) mutual exclusion).

1.1 Application Examples of NMR

As the first example, let consider a database system shared by multiple processes. An administrator may read the database (i.e., execute a backup database) while no process writes to the database. In this case, the read operation and the write operation are mutually exclusive, if these operations are assumed to be atomic. Then, by applying the NMR, an administrator can execute a read operation as an exclusive operation (O_e), while no process executes a write operation.

The next example is a sensor network. A sensor network has to periodically perform a data collection phase when it is used for environmental assessment purposes. In this situation, the data collection phase should be performed exclusively with other operations by closed neighborhood, such as observations

and data exchange with neighboring sensors. That is, the operations that comprise this phase correspond to exclusive operations.

The following example is an overlay network. In an overlay network, the configuration of the underlying network is basically unchanged, but the overlay network, which is a logical or virtual network built on top of it, may require frequent changes depending on the situation. For example, consider a system that builds a virtual grid network on an ad hoc network and performs routing on it [12]. In such a system, it is desirable to perform the routing reconstruction phase on the virtual grid network when the underlying system is not communicating. That is, the operations comprise the routing reconstruction on the overlay network corresponds to exclusive operations.

As the last example, as already mentioned, we can consider a LCM robot system, where each robot repeats executing cycles of LOOK, COMPUTE, and MOVE phases. Some algorithms in the LCM robot system assume the move-atomic property, that is, while robot r executes a MOVE phase, r's neighbors (i.e., robots in r's sight) cannot execute a LOOK phase. In this case, the move-atomic property can be achieved by NMR: Each robot executes a MOVE phase as O_e only when no robot in its closed neighborhood executes LOOK and COMPUTE phases. While a robot executes LOOK and COMPUTE phases, none of its neighbors can execute a MOVE phase.

Actually, to implement LCM synchronization, we can use ordinary global synchronizers [4, 13]. Based on a global synchronizer, each robot may be able to MOVE once every $\Delta + 1$ clock pulses where Δ is the maximum degree of the network. Note that keeping the correct value of Δ in the network, whose topology frequently changes, is very costly. Hence, to implement LCM synchronization by using ordinary global synchronizers is not efficient. On the other hand, based on our proposed NMR, each robot r_i is able to MOVE once every $\max\{|N[i]|\} + 1 \leq \Delta + 2$ clock pulses where $|N[i]|$ is the degree of the closed neighborhood of r_i and it can be locally computed.

1.2 Our Contributions

We first formalize the concept of NMR, and give a design paradigm to achieve NMR. To demonstrate the design paradigm, we consider synchronous distributed systems and give a simple self-stabilizing algorithm for NMR in static networks (Section 2).

Afterwards, to demonstrate applicability of NMR, by using the aforementioned design paradigm, we implement a self-stabilizing synchronization algorithm for an LCM robot system, where each robot can have limited visibility and lights (Section 3). As described above, in the LCM robot system, each robot repeats executing cycles of LOOK, COMPUTE, and MOVE phases. First, we realize the move-atomic property in a self-stabilizing manner on the assumption that robots repeatedly receive clock pulses at the same time points, where the move-atomic property guarantees that, while some robot executes LOOK phase, no robot in its sight can execute a MOVE phase (Section 3.4). Finally, we extend the self-stabilizing algorithm to the assumption that robots receive

(individual) clock pulses at different time points, but the interval between two pulses is identical for all robots (Section 3.5). This research presents the first self-stabilizing implementation of the LCM synchronization with move-atomic property, allowing the implementation in practice of any self-stabilizing or stateless robot algorithm, where robots possess independent clocks that are advanced in the same speed.

1.3 Technical Overview

To implement such NMR, first, we assume there is a global pulse, and each process v_i maintains a modulo ℓ_i counter as a local clock, where ℓ_i is some integer, to count the global pulse. Let each v_i execute operations in $O_p \setminus O_e$ when its local clock counter is 1, and execute operations in O_e when no counter of processes in its closed neighborhood is 1. Let us consider four processes positioned in a star form where v_1 is in the middle, and v_2, v_3, and v_4 can communicate with v_1, but not with each other. Then, if we assume that ℓ_1 is less than or equal to $3+1$ (i.e., its degree plus one), none of them may be able to execute operations in O_e, since every counter of each of its neighbors and itself may be 1 in distinct pulses. Thus, ℓ_1 should be greater than or equal to $3+2$ (i.e., its degree plus two). However, if ℓ_2, ℓ_3 and ℓ_4 are also their degree plus two (i.e., 3), then v_1 cannot execute operations in O_e because every time one of its neighbors may have a counter value 1. Thus, our algorithm uses the biggest degree among v_i's closed neighborhood plus two as ℓ_i (in our case all choose $\ell = 5$). This choice ensures self-stabilization [14], due to the pigeon-hole principle, when starting with arbitrary counter values. Further, note that, if a leaf in the star, say v_2, is connected to another remote process v_5, then v_5 may choose only $\ell_5 = 4$, as v_2 has only two neighbors. Thus, v_5 may enjoy more frequent opportunities to execute operations in O_e than others. In this paper, to relax the assumption of the existence of the global pulse and the perfect timing between pulses in the above discussion, we also consider a logical pulse of several consecutive real pulses.

1.4 Related Work

As one of the synchronization problems, for global mutual exclusion problem, much research has been devoted to self-stabilizing algorithms, e.g., [15] and [16]. Self-stabilizing distributed algorithms for the local (group) mutual exclusion problem are proposed in [6–8, 10]. Various generalized versions of mutual exclusion have been studied extensively, e.g., l-mutual exclusion [17, 18], mutual inclusion [19][1], l-mutual inclusion [19], critical section problem [20, 21].

Robots with globally observed lights were introduced in [22] and used to synchronize the LCM schedules among the robots. In [22], the authors show that asynchronous robots with lights can simulate any algorithm on semi-synchronous[2] robots with lights, and thus the asynchronous robots with lights

[1]The mutual inclusion problem guarantees that at least one process is in the CS.
[2]In the semi-synchronous model, one or more robots are activated in each global round.

have the same power as the semi-synchronous robots with lights. However, unlike our setting, their simulation algorithm is performed asynchronously on the same LCM robot system as the system where the simulated semi-synchronous algorithm works. On the other hand, in our setting, as an application of newly introduced NMR, robots utilizing lights can implement some LCM schedules such as asynchronous move-atomic ones in self-stabilizing manners. Although in [22] unlimited visibility is assumed, in our setting, limited visibility is assumed and only neighboring robots observe the light.

2 Neighborhood Mutual Remainder

In this section, we introduce a concept of NMR and give a design paradigm to achieve NMR. To explain the design paradigm simply, we consider fully-synchronous distributed systems whose topology is static and present a self-stabilizing algorithm as an example. Our design paradigm uses local clocks such that processes can keep different clock values but must increment the clock values synchronously. In fully-synchronous distributed systems, we can trivially implement the local clocks by using global pulses such that all processes regularly receive the pulse at the same time. In asynchronous distributed systems, we cannot use global pulses and hence we must design some mechanism to implement the local clocks. For simplicity, we omit the additional discussions concerning asynchronous distributed systems.

2.1 A System Model

A distributed system is represented by an undirected connected graph $G = (V, E)$, where $V = \{v_0, \ldots, v_{k-1}\}$ is a set of processes and $E \subseteq V \times V$ is a set of communication links between processes. Processes are anonymous and identical, that is, they have no unique identifiers and execute the same deterministic algorithm. Process v_i is a neighbor of v_j if $(v_i, v_j) \in E$ holds. A neighborhood of v_i is denoted by $N(i) = \{v_j \mid (v_i, v_j) \in E\}$, and the degree of v_i is denoted by $\delta(i) = |N(i)|$. Let $\Delta = \max\{\delta(i) \mid v_i \in V\}$. A closed neighborhood of v_i is denoted by $N[i] = N(i) \cup \{v_i\}$.

The local state of a process is defined by its set of local variables and whether it can execute operations in O_e or $O_p \setminus O_e$. Let Q_i be the local state of process $v_i \in V$. Then, a vector of local states $(Q_0, Q_1, \cdots, Q_{k-1})$ of all processes forms a *configuration* of the distributed system. We consider the *state-reading model* as a communication model. In this model, each process v_i can directly read states of all $v_j \in N[i]$ without delay and update its own state.

We assume that every process has an identical program for a round of NMR scheduling. We assume that every process repeats to execute the program in a parallel and synchronized manner. To this end, we assume global pulses. Processes operate synchronously based on the global pulses, that is, all processes regularly receive the pulse at the same time, and operate when they receive the pulse. The duration of local computation (including updates of its state) is

sufficiently small so that every process completes the local computation before receiving the next pulse. We call the interval between two global pulses a *round*.

2.2 Concept of Neighborhood Mutual Remainder

In this subsection, we introduce a concept of NMR. In a distributed system, processes have the same set of operations $O_p = \{op_1, op_2, \cdots, op_q\}$. In O_p, there is a subset $O_e \subset O_p$ such that each process should execute them not concurrently with operations in $O_p \setminus O_e$ by its closed neighborhood. Then, we require that, for each process v_i, *all* processes in the closed neighborhood of v_i may execute operations in O_e infinitely often and simultaneously for a while, while having the opportunity to execute operations in $O_p \setminus O_e$, possibly simultaneously with others, infinitely often too.

Definition 1 (Neighborhood mutual remainder (NMR)) Let O_p be a set of operations, and $O_e \subset O_p$ be a set of exclusive operations. The system achieves *neighborhood mutual remainder* if the following two properties hold.

- *Liveness*: Every process infinitely often executes each of the operations in O_p.
- *Safety*: For every process v_i, when v_i executes operations in O_e, no process in $N[i]$ executes operations in $O_p \setminus O_e$ in the same round.

2.3 Definition of Self-Stabilization

The self-stabilization property [14] is defined as the ability to converge to the correct system operation in finite time from an arbitrary initial configuration. Let $S = (\Gamma, F, \rightarrow)$, where Γ is the finite set of all configurations, F is the specification on a sequence of configurations (in this paper, F is defined in Definition 1.), and $\rightarrow$ is a binary relation on $\Gamma \times \Gamma$. The system $S = (\Gamma, F, \rightarrow)$ can be viewed as a transition system defined by the topology of a given network and algorithm. For any configuration $\gamma \in \Gamma$, let $\gamma' \in \Gamma$ be any configuration that follows γ by a single step of execution on every process. Note that every process executes synchronously based on global pulses. We denote this transition relation by $\gamma \rightarrow \gamma'$.

Definition 2 For a configuration γ_0, a *computation* starting from γ_0 is a maximal (possibly infinite) sequence of configurations $\gamma_0, \gamma_1, \gamma_2, \ldots$, where $\gamma_t \rightarrow \gamma_{t+1}$ for each $t \geq 0$.

Definition 3 A computation $\gamma_0, \gamma_1, \gamma_2, \cdots$ is *legal* with respect to the problem specification F on computations iff it satisfies F.

Definition 4 Configuration γ is *legitimate* for the problem specification F on computations iff any computation that starts from γ is legal for F. Let Λ_F be the set of all legitimate configurations.

Definition 5 A system $S = (\Gamma, F, \rightarrow)$ is self-stabilizing iff it satisfies the following two conditions:

- Convergence: Starting from an arbitrary configuration, the system eventually reaches a configuration in Λ_F, and
- Closure: For any configuration $\gamma \in \Lambda_F$, any configuration γ' that follows γ is also in Λ_F.

Note that, by these properties, self-stabilizing algorithms do not need any initialization, and the system starts convergence following the transient faults in any subsequent fault free execution for a long enough period.

2.4 A Self-Stabilizing Algorithm for Neighborhood Mutual Remainder

In this subsection, we give a design paradigm to achieve NMR. As an example, we realize a self-stabilizing algorithm to achieve NMR in static networks.

First, we give the underlying idea of the self-stabilizing algorithm. Temporarily, let us consider a simple setting where $|N[i]|$ is identical for any process v_i. Every process v_i maintains a local clock $Clock_i$ whose value is incremented by 1 modulo $(|N[i]| + 1)$ in every round. The value of $Clock_i$ may differ from the value of $Clock_j$, for a neighbor v_j of v_i. We assume that v_i can execute operations in $O_p \setminus O_e$ only when $Clock_i = 1$. When the values of all the above clocks are not equal to 1, all processes can execute operations in O_e. Using the pigeon-hole principle in every $|N[i]| + 1$ consecutive rounds, there must be a configuration in which no clock value of the neighboring processes is 1 and at the same time the value of $Clock_i$ is also not 1. Hence, the NMR must hold.

In general, since $|N[i]| \neq |N[j]|$ may hold for some v_i and $v_j \in N(i)$, we can use $MaxN_i = \max\{|N[j]| \mid v_j \in N[i]\}$ instead of $|N[i]|$. Since every process $v_j \in N[i]$ executes operations in $O_p \setminus O_e$ at most once in $MaxN_i + 1$ consecutive rounds, we can still use the pigeon-hole principle and hence the NMR must hold.

Algorithm 1 gives a self-stabilizing algorithm to achieve NMR. In addition to $Clock_i$, each process v_i has two variables N_i and $MaxN_i$. Process v_i sets N_i to $|N[i]|$, then each neighboring process in $N[i]$ reads it (line 1). After that, v_i computes $MaxN_i$ from N_j for all $v_j \in N[i]$ (line 2). Process v_i increments $Clock_i$ modulo $(MaxN_i + 1)$ in line 3, and executes operations in $O_p \setminus O_e$ if $Clock_i = 1$ (lines 4-5). Process v_i also exposes the value of $Clock_i$ to its neighbors. Thus, a process can execute operations in O_e when all the neighborhood clocks are not equal to 1 (lines 6-7).

Theorem 1 *Algorithm 1 achieves NMR in a self-stabilizing manner.*

Proof Every process v_i correctly assigns $|N[i]|$ to N_i at the round following the first pulse, and hence, it correctly assigns $\max\{|N[j]| \mid v_j \in N[i]\}$ to $MaxN_i$ at the next

Algorithm 1 Self-Stabilizing NMR Algorithm for v_i.

Variables for v_i:

- N_i: the size of $N[i]$.
- $MaxN_i$: the maximum value of N_j in $N[i]$.
- $Clock_i$: a local counter of global pulses.

Algorithm for v_i:

1: $N_i := |N[i]|$
2: $MaxN_i := \max\{N_j \mid v_j \in N[i]\}$
3: $Clock_i := (Clock_i + 1) \mod (MaxN_i + 1)$
4: **if** $Clock_i = 1$ **then**
5: Execute operations in $O_p \setminus O_e$ and finish before the next round
6: **else if** $\forall v_j \in N[i][Clock_j \neq 1]$ **then**
7: Execute operations in O_e

round. After that, variable $MaxN_i$ is never changed for any v_i since we assume that the topology does not change.

After the second pulse, v_i executes operations in $O_p \setminus O_e$ once in $MaxN_i + 1$ consecutive rounds.

For any $v_j \in N[i]$, since $MaxN_j \geq |N[i]|$ holds, v_j executes operations in $O_p \setminus O_e$ once in $MaxN_j + 1$ consecutive rounds. During $MaxN_j + 1 \geq |N[i]| + 1$ consecutive rounds, there is a configuration such that no process $v_j \in N[i]$ executes operations in $O_p \setminus O_e$ from the pigeon-hole principle. That is, each process can execute operations in O_e at most once during $MaxN_j + 1 \geq |N[i]| + 1$ consecutive rounds.

Because, in $MaxN_i + 1$ consecutive rounds, v_i executes operations in $O_p \setminus O_e$ once and operations in O_e at least once, liveness property is satisfied.

Thus, the theorem holds. □

3 Self-Stabilizing LCM Implementations

We show the effectiveness of the NMR by applying it to an ordinary distributed system composed of mobile terminals to implement traditional LCM robot models [3]. In the following subsections, we describe an underlying mobile terminal model, where the NMR is executed (Section 3.1) and a simulated robot model, where algorithms for the LCM robot model can be executed (Section 3.2). After discussing the relationship between these two models for the LCM implementations (Section 3.3), we propose two self-stabilizing algorithms for the NMR on the terminals to implement the LCM robot systems with move-atomic properties (Sections 3.4–3.5).

3.1 Underlying Mobile Terminal Model

The mobile terminal model is the model of hardware for the robots. In the system, k mobile terminals exist in a plane. No terminal knows the value of k. They do not have unique IDs and they execute the same deterministic

algorithm. No terminal has direct communication means, except lights which can emit a color to other terminals.

Additionally, the terminals have an observation device to obtain other terminals' positions and colors of lights within a fixed distance ϕ from its current position where ϕ can be infinite. Then, a communication graph (visibility graph) is defined as $G = (V, E)$ where V is a set of terminals and E is a set of terminal pairs that can observe each other. Note that the communication graph may change when terminals move. We assume that the communication graph is connected. We say terminal r_i is a neighbor of r_j if $(r_i, r_j) \in E$ holds, i.e., the distance from r_i to r_j is less than or equal to ϕ. A neighborhood of terminal r_i is denoted by $N(i) = \{r_j \mid (r_i, r_j) \in E\}$, and a closed neighborhood of r_i is denoted by $N[i] = N(i) \cup \{r_i\}$.

Every terminal operates based on pulses, which are generated in a partially synchronous manner. When a terminal receives a pulse, it instantaneously takes a snapshot by using its observation device and obtains positions and colors of neighboring terminals in $N(i)$. Then, it executes an algorithm based on the snapshot before the next pulse. Additionally, in the algorithm, whenever a terminal needs to obtain positions and colors of terminals in $N(i)$, the terminal instantaneously takes a snapshot by using its observation device. We consider two different pulses depending on the partial-synchronization assumptions.

- *Global pulses* are external pulses. All terminals receive these pulses at the same time points.
- *Local pulses* are generated locally. All terminals receive these pulses at different time points.

In both cases, we assume that the interval between two successive pulses is identical for all terminals. We regard the interval between two successive pulses as one round.

In a round, a terminal can travel the distance of at most y, where y can be infinite, by a move operation.

3.2 Simulated Robot Model

In this subsection, we describe the simulated LCM robot models that we will implement on the underlying mobile terminal model. The simulated robot model is a framework to simulate the traditional LCM (LOOK-COMPUTE-MOVE) robot model.

In the traditional LCM robot model, each robot repeats three-phase cycles: LOOK, COMPUTE, and MOVE.

- During the LOOK phase, the robot takes a snapshot to obtain other robots' positions and colors of lights within a fixed distance ϕ' from its current position where ϕ' can be infinite.
- During the COMPUTE phase, the robot computes its next state, color and movement according to the observation in the LOOK phase. The robot may change its state and color at the end of the COMPUTE phase.

- If the robot decides to move in the immediately preceding COMPUTE phase, it moves toward the target position during the MOVE phase. The robot may stop moving before arriving at the target position, however, if the distance from the current position to the target position is more than $\sigma' > 0$, where σ' is a constant, the robot travels the distance of at least σ'. If the distance from the current position to the target position is at most σ', then the robot always reaches the target position during this phase.

In the following, we simply describe that a robot executes LOOK (resp. COMPUTE, MOVE) instead of executing the LOOK (resp. COMPUTE, MOVE) phase.

In the literature [3], some synchronization models (scheduler) are considered in the LCM model. In this paper, we focus on the move-atomic model. The move-atomic model guarantees that, while a robot executes MOVE, none of its neighbors can execute LOOK under an asynchronous scheduler. That is, move-atomic is a sort of constraint for asynchronous scheduler models. We know that this constraint is helpful for designing algorithms on LCM model under asynchronous scheduler models.

3.3 Self-stabilizing Implementations of Move-atomic Model

We aim to implement the move-atomic model on the underlying terminal model in a self-stabilizing manner. That is, each robot in the LCM robot system is simulated by an underlying terminal. We define $robot(r_i)$ as the robot simulated by terminal r_i.

Definition 6 The system $S = (\Gamma, F, \rightarrow)$ implements a self-stabilizing move-atomic model if F is the following specifications:

- *Liveness*: Every robots executes each of the steps, LOOK, COMPUTE and MOVE infinitely often.
- *Safety*: For every robot, when it executes MOVE, the distance from other robots, which execute LOOK in the same round, is more than ϕ'.

To this end, we assign some rounds to execute LOOK and COMPUTE or to execute MOVE, and trigger the phases by terminals. We assume the terminal r_i and its robot $robot(r_i)$ satisfy the following conditions:

- $\sigma' \leq \min\{y, \phi - \phi'\}$, and
- $\phi' < \phi$.

In the following section, we explain the details of the structure of the robot system and the reasons of these assumptions.

We assume that each phase does not last beyond the next pulse. This implies that one round (i.e., the interval from a pulse to the next pulse) is sufficiently long so that the robots travel the distance of at least σ', i.e., $\sigma' \leq y$.

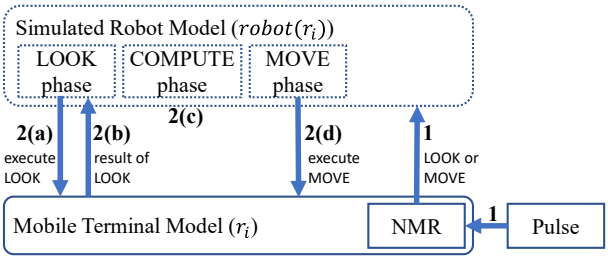

Fig. 1 Structure of the robot system. Arrows represent the data flow. "Pulse" represents an external (or local) synchronization mechanism. A mobile terminal r_i has a software "NMR" to decide which phase can be executed, while its simulated robot $robot(r_i)$ has its software "LOOK phase", "COMPUTE phase" and "MOVE phase" according to the algorithm for the robot.

Figure 1 shows the structure of a robot system to implement the move-atomic model.

1. When terminal r_i receives a pulse, by the NMR algorithm, r_i immediately tells $robot(r_i)$ which phase(s) can be executed in the round[3].
2. $robot(r_i)$ immediately executes the phase according to the instructions from r_i.
   (a) When $robot(r_i)$ wants to execute LOOK, it asks r_i to take a snapshot.
   (b) When r_i is asked to take a snapshot by $robot(r_i)$, it obtains a snapshot containing the positions and colors of other robots (terminals) within the distance $\phi' \leq \phi$, and sends the snapshot to $robot(r_i)$.
   (c) In the same round when $robot(r_i)$ receives the snapshot, $robot(r_i)$ executes COMPUTE.
   (d) When $robot(r_i)$ wants to execute MOVE, it asks r_i to travel toward the point computed by $robot(r_i)$, then r_i moves. Then, $robot(r_i)$ can travel the distance y', where $\sigma' < y' \leq \min\{y, \phi - \phi'\}$, in the round.
3. After that, r_i changes its state (i.e., its light colors and the values of its variables) by the NMR algorithm.

The above interactions between r_i and $robot(r_i)$ and the changing of the state of r_i are completed before the next pulse. Note that, for the NMR algorithm, while r_i observes its environment and changes its state at each round, these operations are independent of the LCM operations of $robot(r_i)$.

In the following sections, we present self-stabilizing algorithms for the underlying terminals, in order to realize the move-atomic model by NMR. First, we need to regard the dynamism of the neighbors of each terminal to maintain their NMR properties. Consider the case that some terminal r_j, which is not viewed by r_i, and r_j moves toward r_i and penetrates into the LOOK range of the $robot(r_i)$. Note that, $robot(r_i)$ (resp. $robot(r_j)$) can execute LOOK (resp. MOVE) because r_i and r_j are not neighbors with each other before the execution. In this situation, if r_j moved into the $r_i's$ LOOK range while $robot(r_i)$ was executing LOOK, then $robot(r_j)$ can be observed by $robot(r_i)$. This breaks

[3]Note that each robot can execute different phases with other robots.

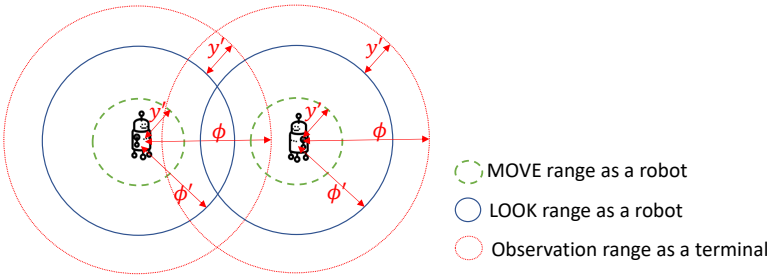

Fig. 2 The relationship between ϕ, ϕ' and y'.

a property of NMR. To prevent such situation, we assume $2y' < \phi = \phi' + y'$ holds in our algorithms (See Figure 2). The NMR algorithm of a terminal r_i is executed with all terminals within ϕ distance from r_i, i.e., it is executed on the communication graph by terminals (defined in Section 3.1). By this assumption, each simulated robot moves up to $y' < \phi/2$ in a single round and receives a snapshot within the distance up to $\phi' = \phi - y'$ when it executes LOOK. Thus, even if r_j travels towards r_i and becomes in $N[i]$, r_j does not penetrate into the LOOK range (ϕ') of $robot(r_i)$, while $robot(r_i)$ executes LOOK.

3.4 Self-Stabilizing Move-Atomic Algorithm with Global Pulses

In this subsection, we consider the implementation of the self-stabilizing move-atomic model, where we assume there is an external clock for global pulses. Note that, our algorithm is self-stabilizing with respect to the change of terminal state.

Firstly, we consider static networks, where the set of neighbors for each terminal does not change. The main idea of the implementation is to apply the NMR algorithm in Section 2.4 to the terminals that simulate robots. Terminal r_i can make $robot(r_i)$ execute LOOK and COMPUTE as $O_p \setminus O_e$, and it can make $robot(r_i)$ execute MOVE as O_e. By this behavior, we can achieve the move-atomic property: while a robot executes MOVE, none of its neighbors can execute LOOK.

A formal description of the algorithm is in Algorithm 2. Note that, the number of terminals is k. Each terminal r_i has the following two lights:

- $Nlight(i) \in \{1, \ldots, k\}$: the color represents $|N[i]|$.
- $Light(i) \in \{0, 1\}$: the color represents whether the local clock counter of global pulses is 1 or not. If $Light(i) = 1$, the clock value is 1.

Additionally, r_i maintains the following variables:

- $MaxN_i \in \{1, \ldots, k\}$: the maximum value of $Nlight$ among the closed neighborhood of r_i.
- $LookCompute_i \in \{true, false\}$: a Boolean value which represents whether the next operation is LOOK or not.

Algorithm 2 Self-Stabilizing Move-Atomic Algorithm with Global Pulses for r_i.

Lights for r_i:

- $Light(i) \in \{0, 1\}$: the color represents whether the value of $Clock_i$ is 1 or not.
- $Nlight(i) \in \{1, \ldots, k\}$: the color represents $|N[i]|$.

Variables for r_i:

- $LookCompute_i \in \{true, false\}$: the value represents whether the next operation is LOOK or not.
- $MaxN_i \in \{1, \ldots, k\}$: the maximum value of $Nlight$ in $N[i]$.
- $Clock_i \in \{0, \ldots, k\}$: a local counter of global pulses.

Algorithm for r_i:

1: **Upon** a global pulse
2: **if** $Light(i) = 1 \wedge LookCompute_i = true$ **then**{
// Execute operations in $O_p \setminus O_e$
3: make $robot(r_i)$ execute LOOK
4: make $robot(r_i)$ execute COMPUTE
5: $LookCompute_i := false$
6: } **else if** $\forall r_j \in N[i][Light(j) = 0] \wedge LookCompute_i = false$ **then**{
// Execute operations in O_e
7: make $robot(r_i)$ execute MOVE
8: $LookCompute_i := true$
9: }
10: $Nlight(i) := |N[i]|$
11: $MaxN_i := \max\{Nlight(j) \mid r_j \in N[i]\}$
12: $Clock_i := (Clock_i + 1) \mod (MaxN_i + 1)$
13: **if** $Clock_i = 1$ **then** $Light(i) := 1$
14: **else** $Light(i) := 0$

- $Clock_i \in \{0, \ldots, k\}$: a local counter of global pulses, not necessarily identical among terminals.

When r_i detects a global pulse, first, r_i tries to make $robot(r_i)$ execute LOOK and COMPUTE or MOVE. When $Light(i)$ is 1, r_i can make $robot(r_i)$ execute LOOK and COMPUTE as $O_p \setminus O_e$ (lines 2-5). Only immediately after all values of $Light$ of r_i's closed neighborhood becomes not 1, meaning no neighbor is planning to make its robot execute LOOK and COMPUTE, r_i can make $robot(r_i)$ execute MOVE as O_e, *i.e.*, safety is satisfied (lines 6-9).

After that, r_i obtains visible neighbors' $Nlight$ values and updates $MaxN_i$ (lines 10-11)[4]. The local counter of global pulses $Clock_i$ is bounded by $MaxN_i$, and maintained by each terminal r_i, that is, they are not necessarily the same

[4] Because the maximal number of neighboring terminals is typically much less than the total number of terminals k, the number of colors is typically much smaller than k.

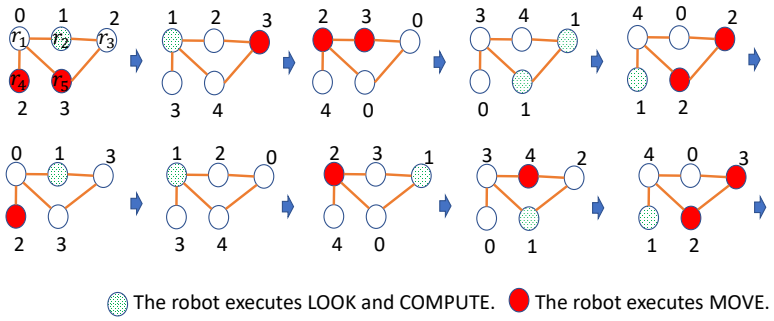

Fig. 3 An execution example of Algorithm 2.

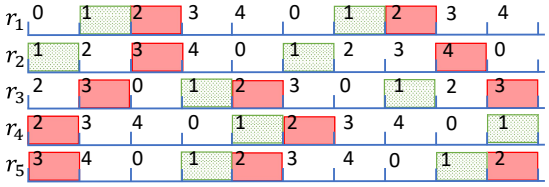

Fig. 4 Time diagram for the execution of Fig. 3.

(line 12). By the value of its counter, each terminal decides its color of $Light(i)$ (lines 13-14).

Because the $Clock_i$ value is $0, \ldots, MaxN_i$, even if the neighbors and itself have different light values, then there is a time when no light value is 1 among the neighbors and itself. The time means no neighbor is planning to make its robot execute LOOK in the next round. Just after the next pulse (i.e., the beginning of the next round), no one has a clock value of 1. Thus, if it has not yet made its robot execute MOVE, it can make the robot execute MOVE from the original LCM algorithm, when all are not 1.

Figures 3 and 4 describe the same execution in two different ways. The numbers beside nodes in Figure 3 and the numbers in Figure 4 are clock values at the beginning of the round. Figure 3 is an execution example of Algorithm 2, which allows a terminal to make its robot execute MOVE if none of the neighbors have a light value 1 and the robot wants to execute MOVE. Red filled (resp. green dotted) nodes represent that the terminals can make their robots execute MOVE (resp. LOOK and COMPUTE). Figure 4 shows the time diagram in the execution of Figure 3. Any time slot in Figure 4 with a red filled (resp. green dotted) box allows executing MOVE (resp. LOOK and COMPUTE) by the algorithm.

Lemma 1 *By Algorithm 2, eventually, whenever a robot executes* MOVE*, none of its neighbors can execute* LOOK *and vice versa.*

Proof Because the initial configuration is arbitrary, every robot starts their execution of the algorithm from arbitrary lines. However, upon the next pulse, every robot executes from line 1. Then, consider a terminal r_i and its neighbor $r_j \in N(i)$. By line 6, when each terminal r_i makes $robot(r_i)$ execute MOVE, $\forall r_j \in N[i][Light(j) = 0]$ must hold. By line 2, when r_i makes $robot(r_i)$ execute LOOK (and COMPUTE), $Light(i) = 1$ must hold. Therefore, when r_i makes $robot(r_i)$ execute MOVE, no terminal $r_j \in N[i]$ can make $robot(r_j)$ execute LOOK (and COMPUTE). Additionally, when r_i makes $robot(r_i)$ execute LOOK, no terminal $r_j \in N[i]$ can make $robot(r_j)$ execute MOVE.

Thus, the lemma holds. □

Lemma 2 *By Algorithm 2, each robot executes* LOOK, COMPUTE, *and* MOVE *infinitely often.*

Proof Consider the execution after the second round. Because we assume that the topology is not changed, the value of $MaxN_i$ for each terminal is not changed. Thus, $robot(r_i)$ executes MOVE once in $MaxN_i + 1$ consecutive rounds by the pigeon-hole principle, and then executes LOOK and COMPUTE when $Clock_i$ is 1. Thus, the lemma holds. □

By Lemmas 1 and 2, we derive Theorem 2.

Theorem 2 *Algorithm 2 implements a self-stabilizing move-atomic model under global pulses, if the topology is static.* □

Algorithm 2 guarantees the move-atomic property based on NMR, if there is no network topology change. However, in the mobile terminal model, it cannot be guaranteed. If adversarial topology changes occur, there may be some starvation terminals r_i, that is, $robot(r_i)$ cannot execute the LCM cycles. Because the clock value is incremented by 1 modulo $MaxN_i + 1 \leq k + 1$ in each round, the clock values are eventually reset to 1, that is, $robot(r_i)$ can eventually execute LOOK and COMPUTE. Then, when $MaxN_i \neq k$, if r_i is always neighboring to other terminals with light value 1, r_i cannot make $robot(r_i)$ execute MOVE. Because their clock values are incremented by 1 modulo $MaxN + 1$ in each round and their lights become 1 only when their clock values are 1, such a situation means that other terminals must take turns to come into the sight of r_i when their clock values are 0, and they change their clock values and light values to 1 in $N(i)$. Thus, to prevent such starvation states to MOVE, it is necessary to realize a time when the lights of all terminals in $N(i)$ become 0, even in the above situation. To solve the problem, in the case that topology changes occur frequently, the clock value is incremented by 1 modulo $k + 1$. Then, while the execution of LCM cycles becomes very slow, no starvation occurs by the pigeon-hole principle.

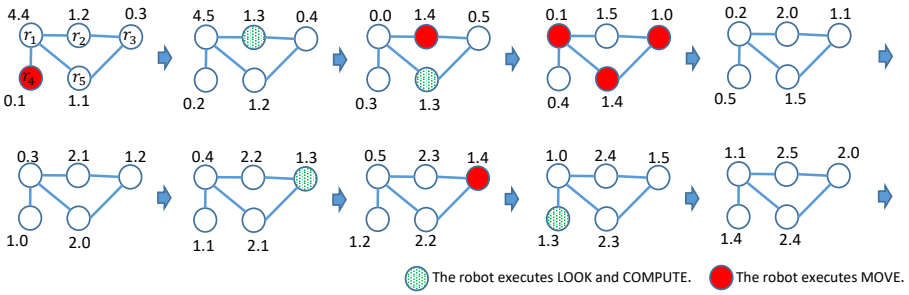

Fig. 5 An execution example of Algorithm 3.

3.5 Self-Stabilizing Move-Atomic Algorithm with Local Pulses

Now we consider the case that there is no global pulse in static networks. That is, each terminal executes the algorithm based only on local pulses.

We assume that the interval of local pulses is the same for each terminal. Let $Lclock_i$ be a local pulse counter for a terminal r_i. However, they are not ticking together. When pulses are not synchronized, they can be slightly less than one round apart. Thus, we sextuplicate the time reserved for LOOK and COMPUTE, and LOOK and COMPUTE are executed only in the middle round of these six. To this end, in Algorithm 3, we sextuplicate the value of clock $Clock_i = (\lfloor Lclock_i/6 \rfloor).(Lclock_i \mod 6)$. If all the lights of neighbors r_j are 0, r_i makes $robot(r_i)$ execute MOVE. When $Clock_i$ is 1.1, 1.2 and 1.3, the light value of r_j is set to 1. Then, if $Clock_i$ is 1.3, r_i makes $robot(r_i)$ execute LOOK and COMPUTE.

Each terminal r_i has following two lights:

- $Nlight(i) \in \{1, \ldots, k\}$: the color represents $|N[i]|$.
- $Light(i) \in \{0, 1\}$: the color represents whether the local clock value $Clock_i$ is 1.1, 1.2, 1.3 or not. If $Light(i) = 1$, the clock value is 1.1, 1.2 or 1.3.

Additionally, r_i maintains the following variables:

- $MaxN_i \in \{1, \ldots, k\}$: the maximum value of $Nlight$ among the closed neighborhood of r_i.
- $LookCompute_i \in \{true, false\}$: a Boolean, which represents whether the next operation is LOOK (and COMPUTE) or not.
- $Lclock_i \in \{0, \ldots, k\}$: a local counter of local pulses, not necessarily identical among terminals.
- $Clock_i$: a local clock by $(\lfloor Lclock_i/6 \rfloor).(Lclock_i \mod 6)$.

Figures 5 and 6 describe the same execution in two different ways. Figure 6 demonstrates the case that the timing of the local pulses drifted. Figure 5 shows the configurations in timing represented by the vertical dot-lines in Figure 6 (of course, the choice of the timing is one of many). When $Clock_i$ is

Algorithm 3 Self-Stabilizing Move-Atomic Algorithm with Local Pulses for r_i.

Lights for r_i:

- $Light(i) \in \{0,1\}$**: the color represents whether the value of** $Clock_i$ **is** 1.1**,** 1.2**,** 1.3 **or not.**
- $Nlight(i) \in \{1,\ldots,k\}$**: the color represents** $|N[i]|$**.**

Variables for r_i:

- $Clock_i$**: a local clock.**
- $LookCompute_i \in \{true, false\}$**: the value represents whether the next operation is LOOK or not.**
- $MaxN_i \in \{1,\ldots,k\}$**: the maximum value of** $Nlight$ **in** $N[i]$**.**
- $Lclock_i \in \{0,\ldots,k\}$**: a local counter of local pulses.**

Algorithm for r_i:

1: **if** $Clock_i = 1.3 \wedge LookCompute_i = true$ **then**{
2: make $robot(r_i)$ execute LOOK
3: make $robot(r_i)$ execute COMPUTE
4: $LookCompute_i := false$
5: }**else if** $\forall r_j \in N[i][Light(j) = 0] \wedge LookCompute_i = false$ **then**{
6: make $robot(r_i)$ execute MOVE
7: $LookCompute_i := true$
8: }
9: $Nlight(i) := |N[i]|$
10: $MaxN_i := \max\{Nlight(j) \mid r_j \in N[i]\}$
11: $Lclock_i := (Lclock_i + 1) \mod 6(MaxN_i + 1)$
12: $Clock_i := (\lfloor Lclock_i/6 \rfloor).(Lclock_i \mod 6)$
13: **if** $Clock_i \in \{1.1, 1.2, 1.3\}$ **then** $Light(i) := 1$
14: **else** $Light(i) := 0$

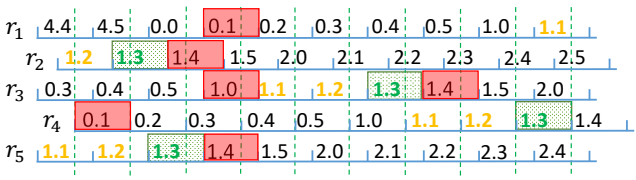

Fig. 6 Time Diagram for the execution of Fig. 5.

1.3, r_i makes $robot(r_i)$ execute LOOK and COMPUTE. If r_i did not observe $Light = 1$, r_i makes $robot(r_i)$ execute MOVE.

Lemma 3 *By Algorithm 3, eventually, whenever a robot executes* MOVE*, none of its neighbors can execute* LOOK *and vice versa.*

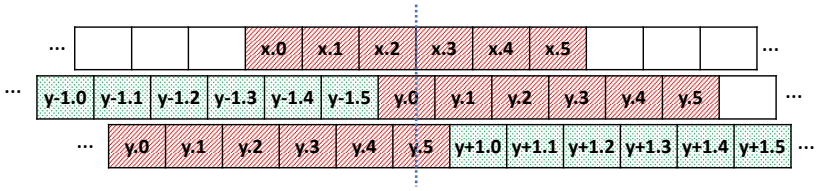

Fig. 7 Time Diagram for proof of Lemma 4.

Proof Because the initial configuration is arbitrary, every robot starts their execution of the algorithm from arbitrary lines. However, upon the next local pulse (i.e., the first local pulse), every robot executes from line 1. Thus, every robot r_i sets their values $Clock_i$ and $Light(i)$ in the round.

Then, consider a terminal r_i and its neighbor $r_j \in N(i)$ after the second local pulse. By line 5, when each terminal r_i makes $robot(r_i)$ execute MOVE, $\forall r_j \in N[i][Light(j) = 0]$ must hold. By line 1, $Clock_j = 1.3$ must hold when r_j makes $robot(r_j)$ execute LOOK (and COMPUTE). In addition, $Light(j) = 1$ holds by line 13 because r_j executed line 9–14 at least once and appropriately set its clock and light values. Therefore, no terminal $r_j \in N[i]$ makes $robot(r_j)$ execute LOOK (and COMPUTE) when r_i makes $robot(r_i)$ execute MOVE. Additionally, no terminal $r_i \in N[j]$ can make $robot(r_i)$ execute MOVE when r_j makes $robot(r_j)$ execute LOOK (and COMPUTE).

Thus, the lemma holds. □

Lemma 4 *By Algorithm 3, each robot executes* LOOK, COMPUTE, *and* MOVE *infinitely often.*

Proof Consider the execution of a terminal r_i after the second local pulse. Because we assume that the topology is not changed, the value of $MaxN_i$ is not changed. Thus, if $robot(r_i)$ executes MOVE, then $robot(r_i)$ executes LOOK and COMPUTE once in $6MaxN_i + 6$ consecutive rounds, i.e., when $Clock_i$ is 1.3.

In the following, we consider whether r_i can make $robot(r_i)$ execute MOVE infinitely often. When $robot(r_i)$ executes MOVE, $\forall r_j \in N[i][Light(j) = 0]$ must hold. Then, $\forall r_j \in N[i][Clock_j \notin \{1.1, 1.2, 1.3\}]$ must hold. Because $Clock_i = (\lfloor Lclock_i/6 \rfloor).(Lclock_i \mod 6)$ and clock interval is the same for all robots, we can take notice the group of time intervals $y = \lfloor Lclock_j/6 \rfloor$ for the time interval $x = \lfloor Lclock_i/6 \rfloor$ in which each y overlaps at least half of x (See Fig. 7). Note that if y overlaps just half of x, it overlaps the latter half of x. Then, by the pigeon-hole principle, there is a time interval t such that $x \neq 1$ and $y \neq 1$ for all $r_j \in N[i]$ in $6(MaxN_i + 1)$ consecutive rounds. In t, we consider whether r_i can make $robot(r_i)$ execute MOVE because the interval x overlaps with the interval $y - 1$ or $y + 1$. If x does not overlap with the interval such that $\lfloor Lclock_j/6 \rfloor = 1$ holds, then r_i makes $robot(r_i)$ execute MOVE. Thus, we have to consider the case that x overlaps with such an interval.

Let r_k be an arbitrary robot in $N[i]$. Let z be $\lfloor Lclock_k/6 \rfloor$ at t, which is not 1 from the assumption. Then, the interval z overlaps at least half of x at t by the definition of t. If $z+1 = 1$, since the interval 1 overlaps with x, then $Light(k)$ becomes 1 during the last half of x (i.e., $x.4$ or $x.5$). If $z - 1 = 1$, since the interval 1 overlaps

with x, then $Light(k)$ becomes 0 in $z.4$, i.e., during $x.0$ or $x.1$. That is, at $x.2$ or $x.3$, r_i does not see the light of r_k is 1, then r_i makes $robot(r_i)$ execute MOVE.

Thus, the lemma holds. □

By Lemmas 3 and 4, we derive Theorem 3.

Theorem 3 *Algorithm 3 implements a self-stabilizing move-atomic model under global pulses if the topology is static.* □

4 Conclusions

In this paper, we defined a new synchronization problem as the neighborhood mutual remainder (NMR) and proposed a self-stabilizing design paradigm to achieve NMR. NMR provides exclusive control over the execution of operations of the process and its neighbors in cases where there are some exclusive operations that must not be executed at the same time as other operations being executed by its neighbors. While (original) mutual exclusion provides exclusive control over execution processes, NMR provides exclusive control over operations.

In addition, in order to achieve move-atomic in LCM robot systems, each robot must not perform MOVE phase at the same time as its neighboring robot performs LOOK and COMPUTE phases. Since such a situation fits perfectly with NMR, we presented a practical implementation of a self-stabilizing LCM using NMR. Our results are described for the case where a round is identical across all local clocks.

As mentioned in section 1.1, NMR has various other applications that we hope to develop in the future.

References

[1] Dolev, S., Kamei, S., Katayama, Y., Ooshita, F., Wada, K.: Brief announcement: Self-stabilizing lcm schedulers for autonomous mobile robots using neighborhood mutual remainder. In: SSS, pp. 127–132 (2019)

[2] Dolev, S., Kamei, S., Katayama, Y., Ooshita, F., Wada, K.: Brief announcement: Neighborhood mutual remainder and its self-stabilizing implementation of look-compute-move robots. In: DISC, pp. 43–1433 (2019)

[3] Flocchini, P., Prencipe, G., Santoro, N.: Distributed Computing by Oblivious Mobile Robots. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, United States (2012)

[4] Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, United States (1996)

[5] Antonoiu, G., Srimani, P.K.: Mutual exclusion between neighboring nodes in a tree that stabilizes using read/write atomicity. In: Euro-Par, pp. 545–553 (1998)

[6] Kakugawa, H., Yamashita, M.: Self-stabilizing local mutual exclusion on networks in which process identifiers are not distinct. In: SRDS, pp. 202–211 (2002)

[7] Boulinier, C., Petit, F., Villain, V.: When graph theory helps self-stabilization. In: Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing (2004)

[8] Emek, Y., Keren, E.: A thin self-stabilizing asynchronous unison algorithm with applications to fault tolerant biological networks. In: Proceedings of the ACM Symposium on Principles of Distributed Computing (2021)

[9] Hadzilacos, V.: A note on group mutual exclusion. In: PODC '01 (2001)

[10] Boulinier, C., Petit, F., Villain, V.: When graph theory helps self-stabilization. In: PODC, pp. 150–159 (2004)

[11] Altisen, K., Devismes, S., Durand, A.: Concurrency in snap-stabilizing local resource allocation. Journal of Parallel and Distributed Computing (2016)

[12] Kim, Y., Shibata, M., Sudo, Y., Nakamura, J., Katayama, Y., Masuzawa, T.: Improved-zigzag: An improved local-information-based self-optimizing routing algorithm in virtual grid networks. In: Proceedings of the 21st International Symposium on Stabilization, Safety, and Security of Distributed Systems (2019)

[13] Dolev, S.: Self-Stabilization. MIT Press, United States (2000)

[14] Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Communications of the ACM **17**(11), 643–644 (1974)

[15] Israeli, A., Jalfon, M.: Token management schemes and random walks yield self stabilizing mutual exclusion. In: PODC, pp. 119–131 (1990)

[16] Kakugawa, H., Yamashita, M.: Uniform and self-stabilizing fair mutual exclusion on unidirectional rings under unfair distributed daemon. Journal of Parallel and Distributed Computing **62**(5), 885–898 (2002)

[17] Abraham, U., Dolev, S., Herman, T., Koll, I.: Self-stabilizing l-exclusion. Theoretical Computer Science **266**(1–2), 653–692 (2001)

[18] Flatebo, M., Datta, A.K., Schoone, A.A.: Self-stabilizing multi-token rings. Distributed Computing **8**(3), 133–142 (1995)

[19] Kakugawa, H.: Self-stabilizing distributed algorithm for local mutual inclusion. Information Processing Letters **115**(6), 562–569 (2015)

[20] Kamei, S., Kakugawa, H.: Self-stabilizing algorithm for dynamically maintaining two disjoint dominating sets. In: PDAA, pp. 278–284 (2018)

[21] Kamei, S., Kakugawa, H.: A self-stabilizing distributed algorithm for the local $(1, \mid N_i \mid)$-critical section problem. Concurrency and Computation: Practice and Experience, 5628 (2019)

[22] Das, S., Flocchini, P., Prencipe, G., Santoro, N., Yamashita, M.: Autonomous mobile robots with lights. Theoretical Computer Science **609**, 171–184 (2016)