# Computational Power of Energy-Constrained Autonomous Robots Under Sequential Schedulers

**Caterina Feletti**
School of Computer Science, Carleton University, Ottawa, Canada
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

**Paola Flocchini**
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

**Nicola Santoro**
School of Computer Science, Carleton University, Ottawa, Canada

## Abstract

We consider the distributed framework of swarms of mobile robots. A swarm is a set of computational, anonymous, indistinguishable, homogeneous, and autonomous entities that operate in the Euclidean plane through infinite sequences of Look-Compute-Move cycles. The goal of a swarm is to collaborate to solve a given problem. The ability to solve a problem depends on the swarm features and its setting $X^S$, where $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$ denotes the memory/communication model and $S$ denotes the class of schedulers (e.g., fully-synchronous, sequential, asynchronous) that activate the robots. Given a pool of settings, prior research has characterized the relations (dominance, equivalence, or orthogonality) among their computational powers, recently extending this analysis to the class of *sequential* schedulers (i.e., activating only one robot per round), and of the *restricted* ones (i.e., never activating a robot twice consecutively).

In this paper, we extend the study on sequential schedulers (`SEQ`, `PERM`, and `RROBIN`) by defining two classes of *sequential restricted* schedulers `R-SEQ` and `R-PERM`. In particular, we analyze how the computational power of each model $\mathcal{OBLOT}$, $\mathcal{LUMI}$, and $\mathcal{FCOM}$ is affected by considering both sequential schedulers and their restricted variants; for $\mathcal{FSTA}$, we only provide the relation between `RROBIN` and `R-PERM`. We establish both equivalence and dominance results: some settings are computationally equivalent, while others can be separated by problems solvable in one setting but not in the other.

**2012 ACM Subject Classification** Theory of computation → Distributed algorithms

**Keywords and phrases** Autonomous mobile robots, Look-Compute-Move, Computational power, Sequential schedulers, Energy-constrained

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2026.22

**Funding** This work was partly supported by NSERC through the Discovery Grant program.

## 1 Introduction

The theoretical framework of mobile robots operating through Look-Compute-Move (LCM) cycles has been extensively studied in distributed computing by mobile entities [17]. According to the LCM framework, a *swarm of robots* is a mobile distributed system formed by a finite set $\mathcal{R} = \{r_1, \ldots, r_n\}$ of computational entities that collaborate to solve a given task by executing the same algorithm. Typically, robots are autonomous (i.e., without any central control), anonymous and indistinguishable (i.e., without any internal or external ids), and punctiform entities operating on the Euclidean plane $\mathbb{R}^2$. Each robot $r$ is equipped with a local coordinate system and a sensor system by which $r$ can perceive the positions of all the robots in $\mathcal{R}$. By default, each robot is idle and can be activated by a *scheduler*. When activated, a robot $r$ performs an LCM cycle: it takes a snapshot of the plane which contains the positions of all the robots according to the local coordinate system of $r$ (*Look*),


executes the algorithm giving it the snapshot as input (*Compute*), and moves to the computed destination along a straight trajectory (*Move*). All operations are assumed to be executed with infinite precision. Generally, there is no or only partial agreement on the local coordinate systems among the robots; in other words, robots are supposed to be completely disoriented or, for example, to agree only on the chirality (i.e., the clockwise direction) of $\mathbb{R}^2$ [6].

Robots are asked to solve a common task, called *problem*, by executing the shared algorithm at each LCM cycle. Typical problems include `Pattern Formation` [6, 16, 18, 24], `Gathering` [1, 7, 9, 21], `Scattering` [2, 22]. The goal is to find algorithms solving a problem under the most restricted combinations of robot features, called *settings*. Unsurprisingly, not all the problems are solvable under a setting; thus, the research has tried to define more powerful or different variants and to check if they enable the solution of otherwise unsolvable problems. This has led, in parallel to the development of new algorithms, to the establishment of a line of research investigating the *computational power* of a setting (i.e., the set of problems solvable under such a setting) and outlining the hierarchical relations (dominance, equivalence, or orthogonality) among different settings [3, 5, 10, 12, 14, 15, 19, 20].

A setting $X^S$ is mainly defined by two components: the memory/communication model of the robots $X$, and the class of the schedulers $S$. Generally, $X$ can take one of the four values corresponding to the four models mainly considered in the literature. In the foundational model [24], $\mathcal{OBLOT}$, robots are assumed to be oblivious and silent, i.e., devoid of any persistent memory and direct communication means. The $\mathcal{LUMI}$ model, instead, assumes robots are embedded with a $O(1)$-size light whose color, both internally and externally visible, can be changed at each activation and thus used as an internal persistent memory and a communication means. The intermediate models, $\mathcal{FSTA}$ and $\mathcal{FCOM}$, assume the light of each robot is only internally and only externally visible, respectively, thus modeling robots with only memory and only communication means. The second factor that mainly determines the computational power of a setting is the scheduler class $S$, which regulates how robots are activated [11]. Typically, all classes assume *fair* schedulers (i.e., activating each robot infinitely often). The most general class, `ASYNCH`, includes the *asynchronous* schedulers which assume nothing of robots synchronization; instead, a scheduler is *semi-synchronous* (`SSYNCH`) if time is divided into atomic rounds and a non-empty arbitrary subset of robots is activated at each round and they all execute an LCM cycle in perfect synchrony; if all the robots are activated at each round, the scheduler is *fully-synchronous* (`FSYNCH`). Indeed, these three classes are in a set-inclusion relation $\texttt{FSYNCH} \subset \texttt{SSYNCH} \subset \texttt{ASYNCH}$. More recently, research has also focused on the class of *sequential* schedulers (`SEQ`), which activate only one robot at a round. A subclass of `SEQ` is `PERM`, which contains the so-called *permutation* schedulers: they activate all the robots in any permutation of $\mathcal{R}$ before starting a new, possibly different, permutation. Self-evidently, a *round-robin* scheduler (`RROBIN`) is a `PERM` scheduler in which the robots are always activated in the same permutation. Thus, $\texttt{RROBIN} \subset \texttt{PERM} \subset \texttt{SEQ} \subset \texttt{SSYNCH}$.

**Related works and contribution.** The computational powers of different settings have been largely studied and compared in the last decade. This has been done by developing *simulators* to prove equivalences among settings, or *separators* (namely, problems which are solved in one setting but not in another one) to prove dominance or orthogonality relations. The series of papers [3, 10, 19, 20] provides the complete landscape of the computational power of the twelve settings $\{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\} \times \{\texttt{FSYNCH}, \texttt{SSYNCH}, \texttt{ASYNCH}\}$. Among the results, the relations $\mathcal{LUMI}^{\texttt{SSYNCH}} \equiv \mathcal{LUMI}^{\texttt{ASYNCH}}$ and $\mathcal{FCOM}^{\texttt{FSYNCH}} \equiv \mathcal{LUMI}^{\texttt{FSYNCH}}$ are worth noting. Other works have studied the same twelve settings with extra features (e.g., [15] considers robots with obstructed visibility, [12] considers robots operating on graphs).

The work [14] is the first to focus on sequential schedulers; it studies the intra-model relations between the settings $X^{S_1}$ and $X^{S_2}$ with $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$, and $S_1, S_2 \in \{\texttt{SEQ}, \texttt{PERM}, \texttt{RROBIN}\}$. It is worth mentioning the proved equivalence $\mathcal{LUMI}^{\texttt{SEQ}} \equiv \mathcal{LUMI}^{\texttt{PERM}}$, which has highlighted the intrinsic capability of $\mathcal{LUMI}$ sequential robots to use their lights to implement a counter. Such a capability has then been exploited in [13] for making a swarm perform a choreography (i.e., a sequence of patterns), and hopefully it may be used in other contexts. In [8], the computational power of sequential schedulers is studied with respect to two specific agreement problems (fault detection and identification).

The recent works [4, 5, 23] extend the analysis by proposing a new class of schedulers. Notably, these works assume *energy-constrained* robots, i.e., robots that need one idle round after an LCM cycle to recharge their energy and be able to perform a new cycle. This results in formalizing the class of *restricted schedulers* RSYNCH, which is a subclass of SSYNCH with the constraint that a robot cannot be activated twice consecutively[1]. Indeed, it is evident that the energy constraint of robots reduces the adversarial power of the scheduler; therefore, robots activated by a restricted class of schedulers are at least as powerful as robots activated by the unrestricted class. Among the results, we cite the equivalence $\mathcal{LUMI}^{\texttt{SSYNCH}} \equiv \mathcal{LUMI}^{\texttt{RSYNCH}}$ [4], later optimized in the number of colors in [23]. Instead, for all other models, it is shown that the energy constraint of robots actually provides them with a computational advantage w.r.t. the non-constrained robots [4].

In this paper, we extend the study on sequential schedulers (i.e., SEQ, PERM, and RROBIN) by considering also their restricted variants. In particular, we define the two classes of *sequential restricted* schedulers R-SEQ = SEQ ∩ RSYNCH and R-PERM = PERM ∩ RSYNCH. We establish how the computational power of each model $\mathcal{OBLOT}$, $\mathcal{LUMI}$, $\mathcal{FCOM}$ is affected by considering both sequential schedulers and their restricted variants; for $\mathcal{FSTA}$, we only establish the difference between RROBIN and R-PERM.

Firstly, we prove that, for each $X \in \{\mathcal{OBLOT}, \mathcal{FCOM}, \mathcal{FSTA}, \mathcal{LUMI}\}$, $X^{\texttt{RROBIN}}$ is more powerful than $X^{\texttt{R-PERM}}$. Indeed, this result confirms the power of the RROBIN scheduler. Then, under $\mathcal{LUMI}$, we provide an *ad hoc* simulator which extends the equivalence $\mathcal{LUMI}^{\texttt{SEQ}} \equiv \mathcal{LUMI}^{\texttt{PERM}}$ proved in [14] to the class $\mathcal{LUMI}^{\texttt{R-PERM}}$. In other terms, $\mathcal{LUMI}^{\texttt{SEQ}}$ robots are able to perform like under a restricted-permutation scheduler.

In $\mathcal{FCOM}$, which has only communication means, we prove that a computational equivalence holds only for $\mathcal{FCOM}^{\texttt{R-SEQ}} \equiv \mathcal{FCOM}^{\texttt{R-PERM}}$. The related simulator shows how the lack of memory in $\mathcal{FCOM}$ robots can be compensated by the repetition-restricted nature of the schedulers, i.e., by the fact that at least one robot will be activated between two activations of the same robot; in other words, the repetition-restricted scheduling enhances the communication power of the $\mathcal{FCOM}$ model. In $\mathcal{OBLOT}$ and in the remaining settings of $\mathcal{FCOM}$, we provide separators establishing the computational dominance of one scheduler class over the other ones, showing that the pattern of activation of the robots can affect the ability of a swarm to solve problems. Table 1 summarizes the found relations in this paper.

[1] The original definition of RSYNCH [5] allows schedulers that may begin with an arbitrarily long prefix behaving like FSYNCH, before eventually following a strictly RSYNCH schedule.

**Table 1** Computational relations within the models $\mathcal{LUMI}$, $\mathcal{FCOM}$, $\mathcal{OBLOT}$.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\mathcal{LUMI}$ | SEQ | $\equiv$ / $\equiv$ | R-SEQ / PERM | $\equiv$ / $\equiv$ | R-PERM | $<$ | RROBIN |
| $\mathcal{FCOM}$ | SEQ | $<$ / $<$ | R-SEQ / PERM | $\equiv$ / $<$ | R-PERM | $<$ | RROBIN |
| $\mathcal{OBLOT}$ | SEQ | $<$ / $<$ | R-SEQ / PERM | $<$ / $<$ | R-PERM | $<$ | RROBIN |

# 2 Preliminaries

## 2.1 Settings

We now present in detail the settings considered in this paper. We say that a swarm of robots $\mathcal{R}$ adheres to a setting $X^S$ if $\mathcal{R}$ satisfies all the core features listed below, it has $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$ as memory-communication capability, and it is activated by schedulers of class $S \in \{\texttt{SEQ}, \texttt{R-SEQ}, \texttt{PERM}, \texttt{R-PERM}, \texttt{RROBIN}\}$.

**Core robot features.** A swarm of robots is a set $\mathcal{R} = \{r_1, \ldots, r_n\}$ of $n$ autonomous and punctiform entities that operate in the Euclidean plane $\mathbb{R}^2$ in synchronous and atomic rounds. Robots are *autonomous* (i.e, without any central control), *anonymous* and *indistinguishable* (i.e., without internal or external ids), and *homogeneous*, i.e., they execute the same given deterministic algorithm $\mathbb{A}$ in a distributed fashion.

Time is marked in atomic rounds ($t = 0, 1, \ldots$). At each round, a robot can be idle or activated. In the second case, it executes a *Look-Compute-Move* (LCM) cycle within the round: in particular, the robot takes a snapshot of the swarm (*Look*), executes $\mathbb{A}$ using the snapshot as input (*Compute*), and then moves straight towards the computed destination point (*Move*). Each robot $r$ activated at time $t$ is equipped with a local coordinate system $\Xi_r(t)$. This coordinate system may differ from those of other robots in terms of axes orientation, unit distance, and clockwise orientation of the plane, and it may change at each activation of the same robot. The snapshot taken by a robot $r$ at time $t$ contains the positions of all the $n$ robots according to $\Xi_r(t)$. For the sake of simplicity, we assume that each $\Xi_r(t)$ is egocentric, i.e., has the origin in the position of $r$ at time $t$.

More than one robot can occupy the same position at the same time, forming a *multiplicity*. We assume robots have *strong multiplicity detection*: in the *Look* step, they can distinguish the number of robots in a multiplicity.

**Memory and communication.** We consider the well-studied models $\mathcal{OBLOT}$, $\mathcal{FSTA}$, $\mathcal{FCOM}$, $\mathcal{LUMI}$, which differ in the robots' memory and/or communication capabilities. The $\mathcal{OBLOT}$ model assumes *oblivious* and *silent* robots, namely devoid of any direct means to store or communicate some information. On the other hand, the $\mathcal{LUMI}$ model equips each robot $r$ with $O(1)$ bits of persistent memory, called *light* and denoted $\texttt{lig}_r$. The value of $\texttt{lig}_r$, a.k.a. *color*, can be updated[2] at the end of each *Compute* step, choosing from a fixed palette of colors $\mathfrak{L}$, defined by $\mathbb{A}$. We assume that $\mathfrak{L}$ always contains the default color $\mathsf{OFF}$, to

[2] A robot can choose to not change the value of its light.

which all robots' lights are initialized at time 0. In the $\mathcal{LUMI}$ model, the snapshot taken by each robot also includes the colors of all robots' lights[3]; hence, lights serve both as internal memory and as a means of communication. The $\mathcal{FSTA}$ model (silent but finite-state) differs from $\mathcal{LUMI}$ in that the light of each robot $r$ is only internally visible; thus, the color of $\texttt{lig}_r$ appears only in the snapshot taken by $r$ in its *Look* step and only serves as internal memory. In contrast, in the $\mathcal{FCOM}$ model (oblivious but finite-communication), the color of $\texttt{lig}_r$ is visible to all the other $n-1$ robots in their snapshots, but not to $r$ itself.

Note that the strong multiplicity detection assumption makes $\mathcal{LUMI}$ and $\mathcal{FCOM}$ robots able to see all the (visible) colors of the robots forming a multiplicity.

**Sequential schedulers.** We here consider only sequential schedulers, namely those that activate exactly one robot at each round. Given a swarm $\mathcal{R} = \{r_1, \ldots, r_n\}$, a sequential scheduler can be defined as a function $\mathfrak{S} : \mathbb{N} \to \mathcal{R}$ or, equivalently, the infinite string $\mathfrak{S} = r_{i_0} r_{i_1} \ldots$ describing for any round $t \in \mathbb{N}$ the activated robot $\mathfrak{S}(t) = r_{i_t} \in \mathcal{R}$. We always assume the *fairness condition*: for any $r \in \mathcal{R}$ and any $t \in \mathbb{N}$, there exists a time $t' > t$ such that $\mathfrak{S}(t') = r$. This fairness condition enables us to measure time in terms of successive sequences of activations of robots, called *epochs*: the first epoch starts with the first activation in the swarm; each epoch ends as soon as all robots have been activated; the next epoch starts with the next activation.

We consider five classes of sequential schedulers. The most general class, SEQ, assumes nothing except that robots are activated sequentially and in a fair fashion. The $\texttt{PERM} \subset \texttt{SEQ}$ class includes the so-called *permutation* schedulers, namely where, for a swarm $\mathcal{R}$ of $n$ robots, each epoch consists of $n$ rounds (thus, a permutation of $\mathcal{R}$). The $\texttt{R-SEQ} \subset \texttt{SEQ}$ class is composed of the *restricted* sequential schedulers, where $\mathfrak{S}(t) \neq \mathfrak{S}(t+1)$ for any $t \in \mathbb{N}$. The $\texttt{R-PERM} = \texttt{R-SEQ} \cap \texttt{PERM}$ class is the class of the restricted permutation schedulers. The $\texttt{RROBIN} \subset \texttt{R-PERM}$ class (*round-robin*) is composed of the permutation schedulers where robots are activated following the same order within each epoch. Figure 1 depicts the set hierarchy of the sequential schedulers.

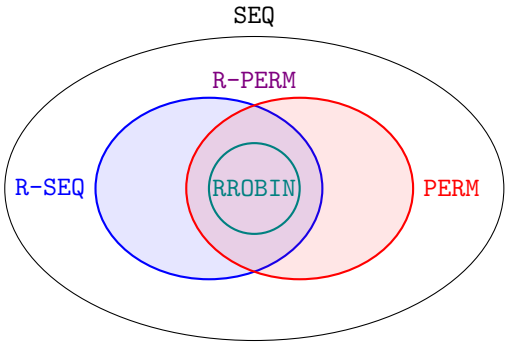



**Figure 1** Set hierarchy of the scheduler classes under SEQ.

## 2.2 Computational Power

A problem is a task that a swarm of robots must achieve. In the LCM framework, a problem $P$ requires the robots of a swarm to form (possibly infinite sequences of) specific configurations, starting from defined initial configurations, possibly moving along allowed

[3] A robot can always distinguish its own light in the snapshot.

trajectories or avoiding forbidden configurations. As a matter of fact, these problems are also called as *configuration-related problems* [8]. To be well-defined, a problem cannot impose any restrictions on light colors, absolute times or absolute positions, or the number of rounds needed to achieve a configuration; in fact, $P$ might in principle be solved with $\mathcal{OBLOT}$ and disoriented robots, and any scheduler class. However, despite the robots' anonymity, $P$ may require certain actions to be executed by robots chosen based on their relative positions in the current or previous configurations.

An algorithm $\mathbb{A}$ *solves* a problem $P$ in a setting $M = X^S$ if, for each swarm of robots belonging to $M$, in all executions[4], the sequence of configurations produced by the swarm satisfies the requirements of $P$. If such an algorithm exists, the problem $P$ is said to be *solvable* in $M$. We denote with $\mathcal{P}(M)$ the *computational power* of setting $M$, i.e., the set of problems solvable in $M$. Given two settings $M_1, M_2$, we have the following relations:

- (**computational equi-dominance**) $M_1 \geq M_2$, if $\mathcal{P}(M_1) \supseteq \mathcal{P}(M_2)$;
- (**computational dominance**) $M_1 > M_2$, if $\mathcal{P}(M_1) \supset \mathcal{P}(M_2)$;
- (**computational equivalence**) $M_1 \equiv M_2$, if $\mathcal{P}(M_1) = \mathcal{P}(M_2)$;
- (**computational orthogonality**) $M_1 \perp M_2$, if $\mathcal{P}(M_1) \setminus \mathcal{P}(M_2) \neq \varnothing$ and $\mathcal{P}(M_2) \setminus \mathcal{P}(M_1) \neq \varnothing$.

We define the relation $<$ ($\leq$, resp.) as the converse relation of $>$ ($\geq$, resp.).

## 3 Intra-model relations

In this section, we investigate the computational relations within the models $\mathcal{OBLOT}$, $\mathcal{LUMI}$, $\mathcal{FCOM}$, and $\mathcal{FSTA}$. The following equi-dominances follow directly from the set-inclusion hierarchy of scheduler classes and from the definition of the models.

▶ **Theorem 1.** *For any $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$, we have that*

$$X^{SEQ} \leq \begin{cases} X^{PERM} \\ X^{R\text{-}SEQ} \end{cases} \leq X^{R\text{-}PERM} \leq X^{RROBIN}.$$

▶ **Theorem 2.** *For any class of schedulers $S$, we have that*

$$\mathcal{OBLOT}^S \leq \begin{cases} \mathcal{FSTA}^S \\ \mathcal{FCOM}^S \end{cases} \leq \mathcal{LUMI}^S.$$

### 3.1 Dominance of RROBIN

In this section, we provide a problem which is solved under $\mathcal{OBLOT}^{\texttt{RROBIN}}$ (and, *a fortiori*, under any $X^{\texttt{RROBIN}}$), but not under $\mathcal{LUMI}^{\texttt{R-PERM}}$ (and, *a fortiori*, under any $X^{\texttt{R-PERM}}$). This proves the intra-model dominance of the RROBIN schedulers w.r.t. the other scheduler classes.

▶ **Problem 3** (Fan)**.** Refer to Figure 2. Let $\mathcal{C}$ be an initial configuration where $n+2$ robots form a scalene square triangle $\triangle abc$, where $n > 3$ robots lie on the square vertex $a$, while the other two robots separately lie on the other vertices. The problem requires all the robots in $a$ to scatter in distinct positions and thus to reach a configuration $\mathcal{C}'$ where no multiplicity exists in the swarm. Then, from $\mathcal{C}'$, the robots have to form $\mathcal{C}$, thus re-composing the original triangle. Perpetually, the problem asks to form $\mathcal{C}'$ and $\mathcal{C}$, where each robot has to maintain the same scattered position in $\mathcal{C}'$. Note that the robots in $b$ and $c$ must never change their position.


**(a)** Configuration $\mathcal{C}$. The multiplicity in $a$ is represented through a yellow-shaded diamond.

**(b)** Movements of the robots to reach configuration $\mathcal{C}'$.

**Figure 2** A possible solution for `Fan` with six robots.

▶ **Lemma 4.** $\mathit{Fan} \in \mathcal{P}\left(\mathcal{OBLOT}^{\mathit{RROBIN}}\right)$.

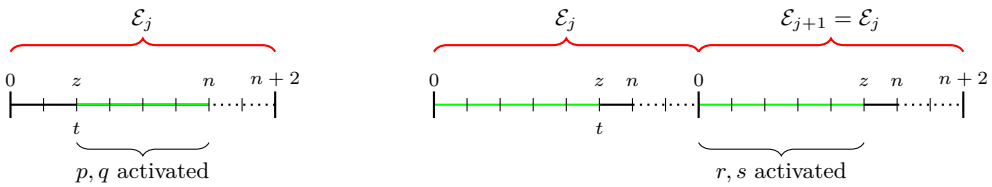
**Proof.** We provide a simple algorithm to solve `Fan` (shown in Figure 2). Assume $\overline{ab}$ is the shortest edge of the triangle. Let $\angle b'ac'$ be the opposite (square) angle centered in $a$ and formed by the extensions of the edges $\overline{ba}$ and $\overline{ca}$, so that $b, a, b'$ are aligned and $dist(a, b') = dist(a, c') = \frac{dist(a,b)}{2}$. Our strategy is to form $\mathcal{C}'$ by making the $n$ robots in $a$ sequentially distribute on the uniform positions along the arc $\overset{\frown}{b'c'}$. Thus, as soon as a robot in $a$ is activated, it moves to the first uniform position available along $\overset{\frown}{b'c'}$ starting from $b'$. The robots in $b$ and $c$ do nothing. Note that, from $\mathcal{C}'$, the (now empty) vertex $a$ is always unambiguously computable, since it corresponds to the center of the circle on which $n > 3$ robots are uniformly arranged. So, in the second epoch, all the robots along $\overset{\frown}{b'c'}$ come back to the vertex $a$ sequentially to form $\mathcal{C}$ again. Now, the whole procedure is repeated. The round-robin activation scheme ensures each robot activated in $a$ always occupies the same position in $\mathcal{C}'$. ◀

▶ **Lemma 5.** $\mathit{Fan} \notin \mathcal{P}\left(\mathcal{LUMI}^{\mathit{R\text{-}PERM}}\right)$.

**Proof.** The proof is similar to that in Lemma 5 in [14] and exploits the pigeonhole principle. Suppose, by contradiction, that there exists an algorithm $\mathbb{A}$ solving `Fan` under $\mathcal{LUMI}^{\texttt{R-PERM}}$ using $k \geq 2$ colors. Consider an instance of the problem where $n \geq 2k + 1$ robots are initially positioned on $a$ in configuration $\mathcal{C}$. Let $\mathcal{S} : [n] \to \mathbb{R}^2$ be an injection, chosen by $\mathbb{A}$, defining the position to be reached in $\mathcal{C}'$ by each robot in $a$. Let $\mathfrak{S} = \mathcal{E}_1\mathcal{E}_2\ldots$ be a `R-PERM` scheduler where $\mathcal{E}_i$ is a permutation string of the swarm (and, hence, corresponds to an epoch). For the sake of simplicity, assume that each $\mathcal{E}_i$ activates first the $n$ robots in $a$, and finally the two robots in $b$ and $c$. Let $t$ be the first time when all $n$ robots have returned to $a$ from $\mathcal{C}'$, and now they have to scatter again in their previous positions in $\mathcal{S}([n])$. Let $t$ belong to $\mathcal{E}_j$, and let $z = t \bmod n + 2$ (i.e., the round within $\mathcal{E}_j$ corresponding to time $t$). Since we assume that the robots of $a$ are activated in the first $n$ rounds of $\mathcal{E}_j$, we have that $z < n$. Suppose that $\mathcal{E}_{j+1} = \mathcal{E}_j$ in $\mathfrak{S}$. We distinguish two scenarios (see Figure 3):

- if $z \leq k$ (see Figure 3a), then $n - z > k$. This means that, at time $t$, there are at least two robots $p, q$ in $a$ with the same color and they will be activated in the next $n - z$ rounds of $\mathcal{E}_j$ from $t$ onward;
- otherwise, (see Figure 3b), there are at least two robots $r, s$ in $a$ with the same color at time $t$ and they will be activated in the first $z$ rounds of $\mathcal{E}_{j+1}$ (which is equal to $\mathcal{E}_j$).

[4] An execution depends on many system factors. In our case, it depends on the scheduler (which must belong to class $S$), and the local coordinate systems of the activated robots.




**(a)** Case $z \leq k$. **(b)** Case $z > k$.

**Figure 3** Epochs in $\mathfrak{S}$ for `Fan`. Solid (dotted, resp.) lines represent the activation of robots in $a$ ($b$ and $c$, resp.).

For each scenario, we construct a `R-PERM` scheduler, namely $\mathfrak{S}'$ (for $z \leq k$) and $\mathfrak{S}''$ (for $z > k$), under which $\mathbb{A}$ fails to solve the problem. In particular, $\mathfrak{S}'$ is the same as $\mathfrak{S}$ except that the activations of $p$ and $q$ are swapped starting from epoch $\mathcal{E}_j$; similarly, $\mathfrak{S}''$ is obtained by $\mathfrak{S}$ by swapping the activations of $r$ and $s$ starting from $\mathcal{E}_{j+1}$. Note that, in both cases, the obtained scheduler still satisfies the `R-PERM` mode, since the robots in $b$ and $c$ are always the last ones activated in each epoch. Thus, in the first scenario, when executing $\mathfrak{S}'$, from time $t$ onward, robot $p$ ($q$, resp.) will take the same snapshot as $q$ ($p$, resp.) under $\mathfrak{S}$, and therefore it will move to the target position intended for $q$ ($p$, resp.). This contradicts the problem's request. The same contradiction arises in the scenario $z > k$ using $\mathfrak{S}''$. ◀

▶ **Theorem 6.** *For any $X \in \{\mathcal{OBLOT}, \mathcal{FSTA}, \mathcal{FCOM}, \mathcal{LUMI}\}$, we have that:*

$$X^{\textit{R-PERM}} < X^{\textit{RROBIN}}.$$

**Proof.** By Theorem 1, we know that $X^{\texttt{R-PERM}} \leq X^{\texttt{RROBIN}}$. Combining Theorem 2 with Lemma 4 and Lemma 5, we derive that `Fan` $\in \mathcal{P}(X^{\texttt{RROBIN}})$ and `Fan` $\notin \mathcal{P}(X^{\texttt{R-PERM}})$. Thus, `Fan` separates the two settings within each model $X$. ◀

## 3.2 Relations in $\mathcal{LUMI}$

This section presents the relations in the $\mathcal{LUMI}$ model, which are illustrated in Figure 4.

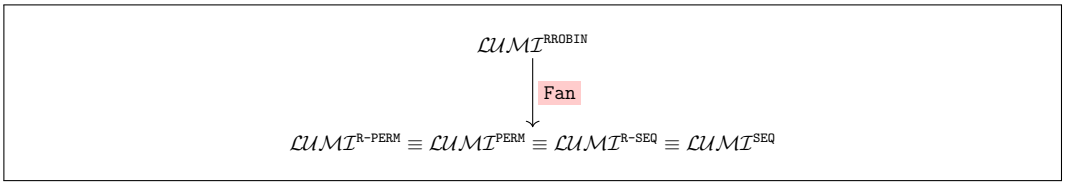



**Figure 4** $\mathcal{LUMI}$ hierarchy. Black arrows represent dominances.

▶ **Theorem 7.** $\mathcal{LUMI}^{\textit{SEQ}} \equiv \mathcal{LUMI}^{\textit{R-PERM}}$.

**Proof.** By Theorem 1, we know that $\mathcal{LUMI}^{\texttt{SEQ}} \leq \mathcal{LUMI}^{\texttt{R-PERM}}$. We now prove that $\mathcal{LUMI}^{\texttt{SEQ}} \geq \mathcal{LUMI}^{\texttt{R-PERM}}$ by showing that any problem solved in $\mathcal{LUMI}^{\texttt{R-PERM}}$ can be solved under the most generic sequential scheduler. This implies that the settings $\mathcal{LUMI}^{\texttt{R-PERM}}$, $\mathcal{LUMI}^{\texttt{PERM}}$, $\mathcal{LUMI}^{\texttt{R-SEQ}}$, and $\mathcal{LUMI}^{\texttt{SEQ}}$ coincide in terms of computational power.

The proof is constructive: we provide a simulator $\mathbb{A}_{SIM}$ which works in $\mathcal{LUMI}^{\texttt{SEQ}}$ and simulates the behavior of any $\mathcal{LUMI}^{\texttt{R-PERM}}$ algorithm $\mathbb{A}$. Let $\mathfrak{L}$ be the palette of colors used by $\mathbb{A}$. We set that $\mathbb{A}_{SIM}$ uses the palette $\mathfrak{L}_{SIM} = \mathfrak{L} \times \{\mathsf{OFF}, \mathsf{MOVED}, \mathsf{LAST}, \mathsf{NEWFIRST}\}$

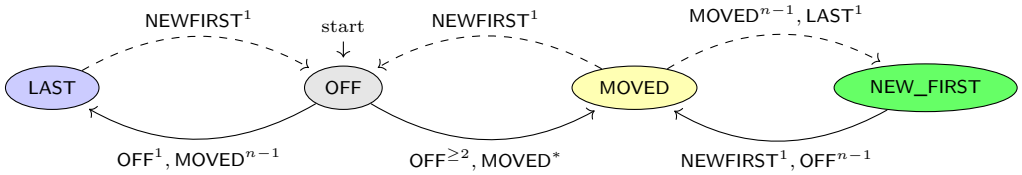



**Figure 5** Color transition of $\texttt{aux}_r$ in Theorem 7. Some regex are abbreviated when unambiguous. Unlike dashed arrows, solid arrows represent transitions where $r$ executes also $\mathbb{A}$.

and that each robot $r$ has a pair of lights: the primary light $\texttt{lig}_r$ (supposed for containing the original colors in $\mathfrak{L}$) and the auxiliary light $\texttt{aux}_r$ (supposed for containing the auxiliary colors $\{\mathsf{OFF}, \mathsf{MOVED}, \mathsf{LAST}, \mathsf{NEWFIRST}\}$, where $\mathsf{OFF}$ is the default one).

Note that, being in $\mathcal{LUMI}$, each robot sees the (primary and auxiliary) lights of all the $n$ robots during each Look step. For simplicity, we represent the snapshot of the auxiliary colors seen by a robot $r$ using a regex-like notation; e.g., the expression $\mathsf{OFF}^+, \mathsf{MOVED}^*, \mathsf{LAST}^1$ means that $r$ sees at least one $\mathsf{OFF}$ robot, possibly some $\mathsf{MOVED}$ ones, exactly one $\mathsf{LAST}$ robot, and nothing else[5]. We now explain how $\mathbb{A}_{SIM}$ simulates one $\texttt{R-PERM}$ epoch in a so-called *mega-epoch*, i.e., a sequence of multiple epochs of $\texttt{SEQ}$: specifically, each mega-epoch is composed of one epoch for the execution of $\mathbb{A}$, at most two rounds for the election of the new first robot, and one epoch for the mega-epoch reset. We now describe the phases of one mega-epoch. See Figure 5 for the transition diagram of the colors in $\texttt{aux}_r$.

- **Epoch simulation**: This phase starts with either the initial configuration $\mathsf{OFF}^n$ (first mega-epoch), or with the configuration $\mathsf{NEWFIRST}^1, \mathsf{OFF}^{n-1}$ (subsequent mega-epochs). During this phase, each $\mathsf{OFF}$ or $\mathsf{NEWFIRST}$ robot switches its $\texttt{aux}$ light to either $\mathsf{MOVED}$ or $\mathsf{LAST}$, and simulates *exactly one* execution of $\mathbb{A}$. We now describe our strategy for the first mega-epoch: we will then clarify how the subsequent mega-epochs begin.

  Let $r$ be an activated robot, and let $\sigma$ be the taken snapshot. If $r$ is $\mathsf{OFF}$ and $\sigma$ satisfies $\mathsf{OFF}^{\geq 2}, \mathsf{MOVED}^*$, then $r$ sets $\texttt{aux}_r \leftarrow \mathsf{MOVED}$ and executes $\mathbb{A}(\sigma_{\mathbb{A}})$, where $\sigma_{\mathbb{A}}$ is obtained by $\sigma$ by removing the $\texttt{aux}$ colors. Note that this is the only step in $\mathbb{A}_{SIM}$ where $r$ may update $\texttt{lig}_r$ and its position. If a $\mathsf{MOVED}$ robot is reactivated during this first epoch (i.e., as long as an $\mathsf{OFF}$ robot exists), it does nothing. If $r$ is the last $\mathsf{OFF}$ robot in the swarm (thus $\sigma$ satisfies $\mathsf{OFF}^1, \mathsf{MOVED}^{n-1}$), $r$ sets $\texttt{aux}_r \leftarrow \mathsf{LAST}$ and executes $\mathbb{A}(\sigma_{\mathbb{A}})$. The snapshot $\mathsf{MOVED}^{n-1}, \mathsf{LAST}$ marks the end of this phase.
- **First-robot setting and reset**: Now, we have to prepare the swarm for simulating the next $\texttt{R-PERM}$ epoch, which, by definition, must start with any robot but the $\mathsf{LAST}$ one. Thus, as soon as a $\mathsf{MOVED}$ robot is activated, it sets $\texttt{aux}_r \leftarrow \mathsf{NEWFIRST}$. After the setting of the $\mathsf{NEWFIRST}$ robot, all the others reset their auxiliary light to $\mathsf{OFF}$. The snapshot $\mathsf{NEWFIRST}^1, \mathsf{OFF}^{n-1}$ marks the start of a new mega-epoch. In this case, as soon as the $\mathsf{NEWFIRST}$ robot sees $\mathsf{NEWFIRST}^1, \mathsf{OFF}^{n-1}$, it sets $\texttt{aux} \leftarrow \mathsf{MOVED}$ and executes $\mathbb{A}$. From this point onward, the process proceeds as in the first mega-epoch.

$\mathbb{A}_{SIM}$ executes an infinite sequence of mega-epochs, each simulating the execution of $\mathbb{A}$ in one $\texttt{R-PERM}$ epoch. In fact, it is easy to see that $\mathbb{A}_{SIM}$ ensures that all the robots execute $\mathbb{A}$ only once in a mega-epoch, and that each simulated epoch does not start with the last robot of the previous simulated epoch. ◀

[5] The order of the colors in the regex is irrelevant.

## 3.3 Relations in $\mathcal{OBLOT}$

This section presents the relations in the $\mathcal{OBLOT}$ model, which are illustrated in Figure 6.

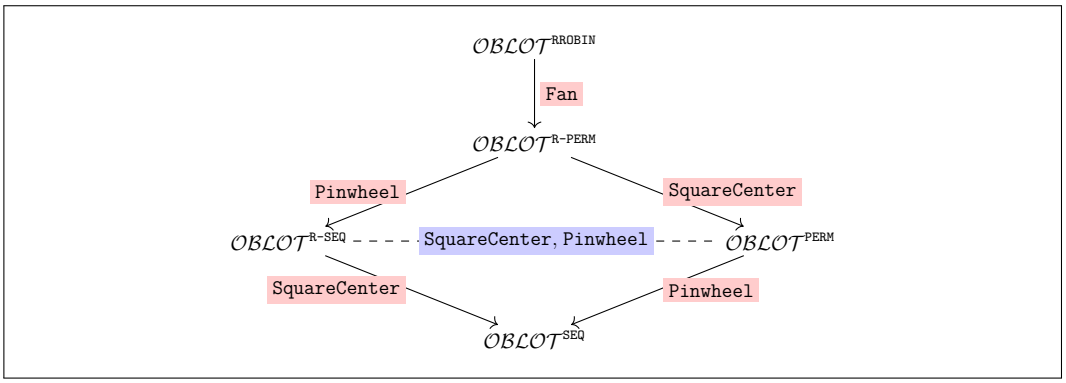
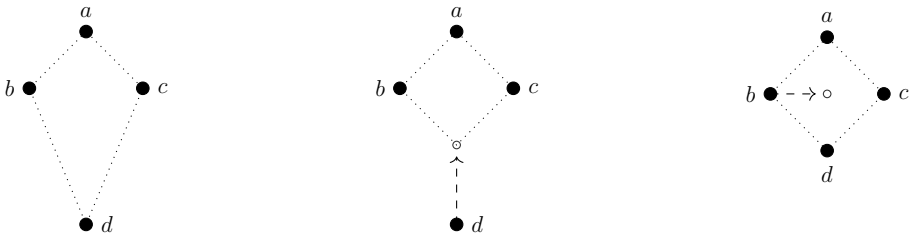



**Figure 6** $\mathcal{OBLOT}$ hierarchy. Black arrows represent dominances. Dashed lines represent orthogonalities.

▶ **Problem 8** (`SquareCenter`)**.** Refer to Figure 7. Let $a, b, c, d$ be four robots forming a non-degenerate kite, where the reflection axis lies on $\overline{ad}$, and where $\angle bac = \frac{\pi}{2}$ and $\angle bdc < \frac{\pi}{2}$. The problem requires $d$ to move along the axis and stop to form a square with $a, b, c$. Then, one robot among $a, b, c$ must reach the center of the square. No other movement is allowed.




**(a)** Initial configuration. **(b)** Square configuration. **(c)** A robot (not $d$) reaches the center of the square.

**Figure 7** Configurations in `SquareCenter`.

▶ **Lemma 9.** $\texttt{SquareCenter} \in \mathcal{P}\left(\mathcal{OBLOT}^{\textit{R-SEQ}}\right)$.

**Proof.** The claim is easily proved by the fact that, after the movement of $d$ towards its target vertex on the square, another robot in $\{a, b, c\}$, is activated. Such a robot, whichever it is, recognizes the square configuration and moves towards its center. No lights are needed. ◀

▶ **Lemma 10.** $\texttt{SquareCenter} \notin \mathcal{P}\left(\mathcal{OBLOT}^{\textit{PERM}}\right)$.

**Proof.** Suppose by contradiction that there is an algorithm $\mathbb{A}$ solving the problem, and suppose the swarm is activated by the scheduling $\mathfrak{S} = abcd\ dcba\ldots$. Then, after having reached the square's vertex in the first epoch, $d$ is activated first in the second epoch. Thus, $d$ executes $\mathbb{A}(\sigma)$, where $\sigma$ only contains the positions of the swarm, and stays still by hypothesis. However, due to the disorientation of the robots and the symmetry of the configuration, the same snapshot $\sigma$ could be taken by any other robot among $\{a, b, c\}$. Thus, $\mathbb{A}$ never makes the swarm reach the final configuration. Contradiction achieved. ◀

▶ **Theorem 11.** $\mathcal{OBLOT}^{PERM} < \mathcal{OBLOT}^{R\text{-}PERM}$ *and* $\mathcal{OBLOT}^{SEQ} < \mathcal{OBLOT}^{R\text{-}SEQ}$.

**Proof.** By Theorem 1, we know that $\mathcal{OBLOT}^{\texttt{PERM}} \leq \mathcal{OBLOT}^{\texttt{R-PERM}}$ and $\mathcal{OBLOT}^{\texttt{SEQ}} \leq \mathcal{OBLOT}^{\texttt{R-SEQ}}$. `SquareCenter` separates the two pairs of settings (Lemmas 9 and 10). ◀

▶ **Problem 12** (`Pinwheel` [14])**.** Refer to Figure 8. The problem is perpetual, and it is defined recursively. Let $\mathcal{C}$ be a configuration in which a swarm of robots forms two different acute scalene triangles, denoted as $\triangle ABC$ and $\triangle abc$, having the same shape, orientation, and circumcenter. Assume the circumcircle of $\triangle abc$ is strictly contained in $\triangle ABC$. Exactly one robot lies on each vertex of $\triangle abc$, while a total of $m \geq 3$ robots lie on the vertices of $\triangle ABC$. On the circumcenter of the two triangles, $m + 1$ robots form a multiplicity. Assume that $\angle a > \angle b > \angle c$, and fix the clockwise orientation of the swarm as $\overset{\frown}{abc}$. The problem requires the robots in $\triangle ABC$ to *perform a spin*, i.e., to reach a configuration $\mathcal{C}'$ where each multiplicity of $\triangle ABC$ has moved to the next vertex of the triangle following the common clockwise orientation. During the transition from $\mathcal{C}$ to $\mathcal{C}'$, each robot of $\triangle ABC$ must travel only along the edge connecting itself with the related target vertex, following the clockwise direction, without stopping except at its destination. No other movement is allowed. Recursively, a new spin must be performed starting from $\mathcal{C}'$.

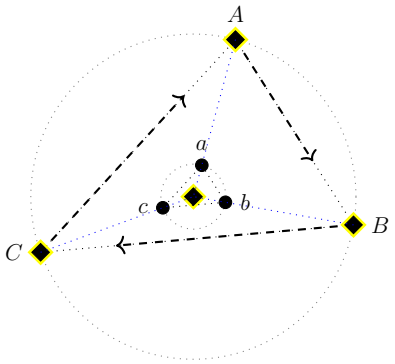



**Figure 8** `Pinwheel`. Multiplicities are represented through yellow-shaded diamonds.

▶ **Lemma 13** ([14])**.** $\texttt{Pinwheel} \in \mathcal{P}\left(\mathcal{OBLOT}^{PERM}\right)$.

**Proof.** When a robot on a vertex of $\triangle ABC$ is activated, it reaches the next vertex as required. The `PERM` scheduler ensures each robot moves only once in a spin. ◀

▶ **Lemma 14.** $\texttt{Pinwheel} \notin \mathcal{P}\left(\mathcal{OBLOT}^{R\text{-}SEQ}\right)$.

**Proof.** As soon as a robot moves from a vertex to another, say from $A$ to $B$, the new multiplicity in $B$ contains two groups of robots: $\mathcal{R}_0$ containing the not-yet-moved robots within the current spin, and $\mathcal{R}_1$ containing the already-moved robots within the current spin. Let $t$ be the time of this first movement. Eventually, all the robots in $\mathcal{R}_0$ must perform their movement to complete the current spin (from $\mathcal{C}$ to $\mathcal{C}'$). Suppose a robot in $C$ is activated at time $t + 1$. Let $t' > t + 1$ be the time when a robot $r$ from $\mathcal{R}_0$ is activated and moves. However, if the scheduler had instead activated a robot from $\mathcal{R}_1$ at time $t'$, it would have had the same snapshot as $r$ and moved. This contradicts the problem's request. ◀

▶ **Theorem 15.** $\mathcal{OBLOT}^{SEQ} < \mathcal{OBLOT}^{PERM}$ *and* $\mathcal{OBLOT}^{R\text{-}SEQ} < \mathcal{OBLOT}^{R\text{-}PERM}$.

**Proof.** By Theorem 1, we know that $\mathcal{OBLOT}^{\texttt{SEQ}} \leq \mathcal{OBLOT}^{\texttt{PERM}}$ and $\mathcal{OBLOT}^{\texttt{R-SEQ}} \leq \mathcal{OBLOT}^{\texttt{R-PERM}}$. `Pinwheel` separates the two pairs of settings (Lemmas 13 and 14). ◀

▶ **Theorem 16.** *$\mathcal{OBLOT}^{PERM} \perp \mathcal{OBLOT}^{R\text{-}SEQ}$.*

**Proof.** By Lemmas 9 and 10, we have $\texttt{SquareCenter} \in \mathcal{P}\left(\mathcal{OBLOT}^{\texttt{R-SEQ}}\right) \setminus \mathcal{P}\left(\mathcal{OBLOT}^{\texttt{PERM}}\right)$. By Lemmas 13 and 14, we have $\texttt{Pinwheel} \in \mathcal{P}\left(\mathcal{OBLOT}^{\texttt{PERM}}\right) \setminus \mathcal{P}\left(\mathcal{OBLOT}^{\texttt{R-SEQ}}\right)$. ◀

## 3.4 $\mathcal{FCOM}$ relations

This section presents the relations in the $\mathcal{FCOM}$ model, which are illustrated in Figure 9.

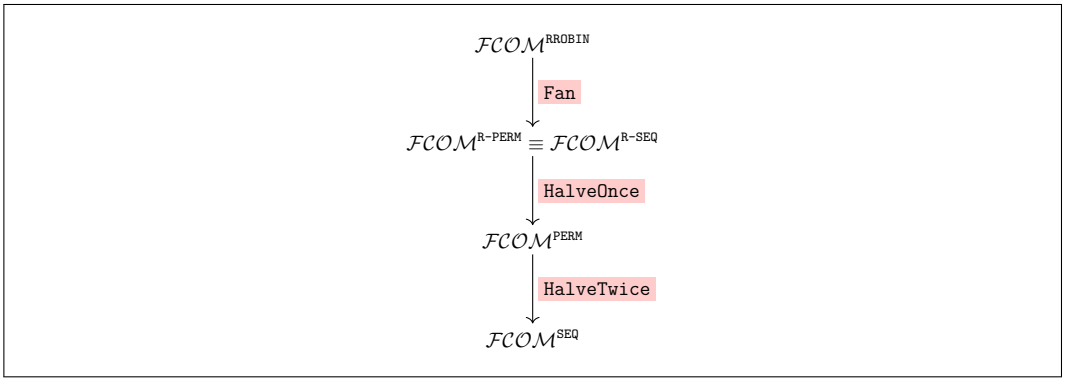



**Figure 9** $\mathcal{FCOM}$ hierarchy. Black arrows represent dominances.

▶ **Theorem 17.** *$\mathcal{FCOM}^{R\text{-}PERM} \equiv \mathcal{FCOM}^{R\text{-}SEQ}$.*

**Proof.** We already know that $\mathcal{FCOM}^{\texttt{R-PERM}} \geq \mathcal{FCOM}^{\texttt{R-SEQ}}$. As in Theorem 7, we prove the reverse equi-dominance by providing a simulator $\mathbb{A}_{SIM}$ that simulates the behavior of an algorithm $\mathbb{A}$ working in $\mathcal{FCOM}^{\texttt{R-PERM}}$ in the $\mathcal{FCOM}^{\texttt{R-SEQ}}$ setting. We assume that each robot, in addition to the primary light `lig` which contains the original color used by $\mathbb{A}$, is equipped with a vector of three auxiliary independent lights in the form:

(`round_state`, `epoch_state`, `parity`)

which are used by $\mathbb{A}_{SIM}$ to simulate an `R-PERM` epoch. The default value of the auxiliary lights is $(\mathsf{OFF}, \mathsf{START}, 0)$. In contrast to the simulator for $\mathcal{LUMI}^{\texttt{SEQ}} \equiv \mathcal{LUMI}^{\texttt{R-PERM}}$ in Theorem 7 in which each robot sets the state $\mathsf{MOVED}$ in its auxiliary light to keep track of the fact that it has already moved in the simulated epoch or not, here robots cannot see their own lights. However, we exploit the communication power of $\mathcal{FCOM}$: we make robots collectively maintain the parity of the number of $\mathsf{MOVED}$ robots, so that a robot $r$, when activated, can understand if it has already moved or not by comparing the `parity` value of the other robots with the number of $\mathsf{MOVED}$ robots $r$ actually sees.

Note that an activated robot can independently modify any subset of its lights during a Compute step[6]. We now explain how $\mathbb{A}_{SIM}$ uses the auxiliary lights to simulate one `R-PERM` epoch in a `R-SEQ` mega-epoch[7]. More specifically:

[6] Even if not relevant in the synchronous schedulers, the updates of the lights within each robot are always simultaneous.
[7] We have prioritized clarity over minimizing the number of colors.

- `round_state` $\in$ {OFF, JUST_MOVED, I_SEE_MOVED, RND_RESET, NEWFIRST} is used to make one robot simulate a round by executing $\mathbb{A}$, namely an $\mathbb{A}$-round: OFF is the default color, JUST_MOVED is set by a robot $r$ which has just moved[8] (i.e., performed an $\mathbb{A}$-round), I_SEE_MOVED is set by the first robot activated after $r$ has just moved, RND_RESET is used to reset the current round, NEWFIRST is used by the robot which will start a new mega-epoch;
- `epoch_state` $\in$ {START, MOVED, LAST, EPC_RESET} indicates the state of a robot within each simulated epoch (i.e., each mega-epoch): START is used to initialize a mega-epoch, MOVED is set by a robot $r$ to indicate that $r$ has already performed its $\mathbb{A}$-round within the simulated epoch, LAST is used by the last robot performing an $\mathbb{A}$-round within the simulated epoch, and EPC_RESET is used to reset the `R-SEQ` mega-epoch and prepare the simulation of a new `R-PERM` epoch;
- `parity` $\in \{0, 1\}$ marks the parity of the number of already moved robots within a mega-epoch. Thus, `parity` $= 0$ (1, resp.) indicates that the number of robots with `epoch_state` $=$ MOVED is even (odd, resp.).

Simulator $\mathbb{A}_{SIM}$ executes an infinite sequence of mega-epochs, each one consisting of the following phases:

- **Mega-epoch starting:** each mega-epoch starts with two possible configurations. The first mega-epoch starts with all robots in $(\mathsf{OFF}, \mathsf{START}, 0)$, while the subsequent mega-epochs start with one robot in $(\mathsf{NEWFIRST}, \mathsf{START}, 0)$ and all the others in $(\mathsf{OFF}, \mathsf{START}, 0)$. We now explain how the first mega-epoch works, when all the robots are $(\mathsf{OFF}, \mathsf{START}, 0)$:
- **Simulation of a round:**
  - **Round execution:** as soon as a robot $r$ is activated and sees that the `parity` corresponds to the number of MOVED robots it sees, $r$ is aware that it has not yet simulated an $\mathbb{A}$-round within the current mega-epoch. Thus, $r$ executes the original algorithm $\mathbb{A}$. Moreover, $r$ sets its `round_state` $\leftarrow$ JUST_MOVED, `epoch_state` $\leftarrow$ MOVED, and updates its value of `parity`.
  - **Movement notice:** being `R-SEQ`, the scheduler will activate a robot $r' \neq r$ in the subsequent round. Thus, $r'$ sees that there is a JUST_MOVED robot and no I_SEE_MOVED-colored robot. Thus, $r'$ sets its `round_state` $\leftarrow$ I_SEE_MOVED and updates its `parity` (by copying the `parity` of the JUST_MOVED-colored robot);
  - **Parity broadcast:** all the other robots (except $r$) must update their `parity` value and set `round_state` $\leftarrow$ I_SEE_MOVED as well;
  - **Round reset:**
    - as soon as $r$ sees all the other robots with `round_state` $=$ I_SEE_MOVED, then $r$ sets `round_state` $\leftarrow$ RND_RESET;
    - all the other robots set `round_state` $\leftarrow$ RND_RESET;
    - as soon as a robot sees only robots with `round_state` $=$ RND_RESET, then it sets `round_state` $\leftarrow$ OFF;
    - all the other robots reset their `round_state` $\leftarrow$ OFF, thus preparing the swarm for simulating a new round.
- **Last round:** The previous phases are repeated until all the robots except for one have `epoch_state` $=$ MOVED. Remind that a robot $r$ can determine its own `epoch_state` value by comparing the `parity` value of the other robots with the number of robots with `epoch_state` $=$ MOVED that $r$ sees. If $r$ understands it is the last not-yet-moved robot, it simulates an $\mathbb{A}$-round as usual, but sets `epoch_state` $\leftarrow$ LAST.

[8] The movement of a robot caused by $\mathbb{A}$ can even be null.

- **Epoch reset:** Let $r'$ be the first robot activated after the LAST one. Then, $r'$ realizes that an `R-PERM` epoch has been correctly simulated, and now it is time to reset the swarm for a new epoch. Being under `R-PERM`, the next epoch cannot be started by the LAST robot. Then,
  - $r'$ sets (`round_state`, `epoch_state`, `parity`) $\leftarrow$ (NEWFIRST, EPC_RESET, 0). It will be the first robot to start the next simulated epoch.
  - All the other robots (except the NEWFIRST one, which can always detect its color by exclusion) will set (`round_state`, `epoch_state`, `parity`) $\leftarrow$ (OFF, EPC_RESET, 0).
  - As soon as the NEWFIRST robot sees all the other robots have reset their values, it sets `epoch_state` $\leftarrow$ START.
  - All the other robots will set `epoch_state` $\leftarrow$ START as well.
- **Start of a new mega-epoch:** As soon as a robot $r$ sees all the robots in (OFF, START, 0), it realizes it is the (only) robot in (NEWFIRST, START, 0). Then, $r$ will simulate the first $\mathbb{A}$-round by executing it, and setting `round_state` $\leftarrow$ JUST_MOVED and `epoch_state` $\leftarrow$ MOVED, and updating the parity. From now on, the mega-epoch follows the strategy explained above.

  In all the other cases, robots do nothing.

Our strategy guarantees that each mega-epoch makes all the robots execute only one $\mathbb{A}$-round and that the last robot executing an $\mathbb{A}$-round in a mega-epoch does not coincide with the first one of the next mega-epoch. Thus, each mega-epoch simulates an `R-PERM` epoch, thus completing the proof. ◀

▶ **Problem 18** (`HalveTwice`)**.** See Figure 10. Let $\mathcal{C}$ be a configuration with three robots $p, q, r$, where $p, q$ lie on the same point while $r$ lies on a distinct point. Let $d = dist(p, r) > 0$ be their distance (according to an absolute coordinate system), and let $\gamma$ be the line where the robots lie. Robot $r$ is required to move along $\gamma$ and to stop in order to halve the distance with $p, q$ at least once, at most twice. Formally, it must form a new stable configuration $\mathcal{C}'$ where the new distance of the robots, say $d'$, is such that $d' \in \{\frac{d}{2}, \frac{d}{4}\}$. Robot $r$ is not allowed to stop along $\gamma$ at other distances other than $\frac{d}{2}$ or $\frac{d}{4}$. No other movement is allowed.

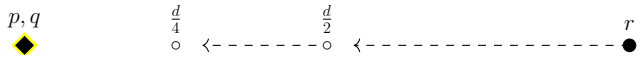



**Figure 10** `HalveTwice` problem: $r$ must halve the distance with $p, q$ at least once, at most twice.

▶ **Lemma 19.** *HalveTwice* $\in \mathcal{P}\left(\mathcal{FCOM}^{PERM}\right)$.

**Proof.** The problem can be solved using only two colors, say {OFF, MOVED}, where OFF is the default one. As soon as $r$ is activated and sees that $p, q$ are OFF, it sets its light to MOVED, and moves along $\gamma$ so that the new distance is $\frac{d}{2}$, where $d$ was the distance of the initial configuration. Now, let us suppose that $r$ is activated in the next round: it sees that $p, q$ are OFF, so it moves again to halve the current distance. In this case, the resulting new distance is $\frac{d}{4}$. After at most two consecutive activations (and movements) of $r$, another robot, $p$ or $q$, is activated. Seeing the MOVED light of $r$, this robot sets its light as MOVED too. This marks the end of the algorithm. ◀

▶ **Lemma 20.** *HalveTwice* $\notin \mathcal{P}\left(\mathcal{FCOM}^{SEQ}\right)$.

**Proof.** By contradiction, let $\mathbb{A}$ be an algorithm solving `HalveTwice` under $\mathcal{FCOM}^{\texttt{SEQ}}$. Let $t$ be the first time $r$ moves from $\mathcal{C}$ after having executed $\mathbb{A}(\sigma)$ and reaches a distance $d'$. By request, $d' \in \{\frac{d}{2}, \frac{d}{4}\}$. Suppose now that $r$ is activated other two consecutive times, and

suppose the local coordinate system of $r$ is such that $r$ perceives the distance $dist(r,p)$ as always $d$. Thus, the snapshots taken at times $t+1$ and $t+2$ will be $\sigma$. As a result, $r$ executes the same action (i.e., halves the distance) both at time $t+1$ and at time $t+2$. However, this contradicts the problem request, which asks $r$ to halve the distance at most twice. ◀

▶ **Theorem 21.** $\mathcal{FCOM}^{SEQ} < \mathcal{FCOM}^{PERM}$.

**Proof.** By Theorem 1, we know that $\mathcal{FCOM}^{\texttt{SEQ}} \leq \mathcal{FCOM}^{\texttt{PERM}}$. `HalveTwice` separates the two settings (Lemmas 19 and 20). ◀

▶ **Problem 22** (`HalveOnce`)**.** As `HalveTwice`, but now $r$ is required to move along $\gamma$ and stop into a new stable configuration $\mathcal{C}'$ where the new distance $d' = \frac{d}{2}$. No other stops are allowed.

▶ **Lemma 23.** $HalveOnce \in \mathcal{P}\left(\mathcal{FCOM}^{R\text{-}PERM}\right)$.

**Proof.** The proof is similar to that in Lemma 19. As soon as $r$ is activated and sees that $p, q$ are OFF, it sets its light to MOVED, and moves along $\gamma$ to halve the distance with the multiplicity. At the next round, $p$ or $q$ is activated; seeing the MOVED light of $r$, the activated robot sets its light as MOVED too. This marks the end of the algorithm. ◀

▶ **Lemma 24.** $HalveOnce \notin \mathcal{P}\left(\mathcal{FCOM}^{PERM}\right)$.

**Proof.** The proof is similar to that in Lemma 20 for `HalveTwice`. By contradiction, suppose an algorithm $\mathbb{A}$ solves `HalveOnce` under $\mathcal{FCOM}^{\texttt{PERM}}$. Let $\mathfrak{S}$ be a `PERM` scheduler that always activates $r$ as the last one in each epoch. Let $t$ be the time $r$ moves from $\mathcal{C}$ and halves its distance with $p, q$, by executing $\mathbb{A}(\sigma)$. Suppose now to have a `PERM` scheduler $\mathfrak{S}'$ which is identical to $\mathfrak{S}$ until time $t$, but it activates $r$ at time $t+1$. Thus, since $r$ cannot see its light, and since $r$ can perceive the new distance as still $d$, the snapshots taken at time $t+1$ under $\mathfrak{S}'$ will be $\sigma$. As a result, $r$ halves the distance with $p, q$ both at time $t$ and $t+1$ under $\mathfrak{S}'$. Contradiction achieved. ◀

▶ **Theorem 25.** $\mathcal{FCOM}^{PERM} < \mathcal{FCOM}^{R\text{-}PERM}$.

**Proof.** By Theorem 1, we know that $\mathcal{FCOM}^{\texttt{PERM}} \leq \mathcal{FCOM}^{\texttt{R-PERM}}$. `HalveOnce` separates the two settings (Lemmas 23 and 24). ◀

## 4 Conclusions

This paper analyzes whether and how the computational power of mobile robot models – $\mathcal{OBLOT}$, $\mathcal{LUMI}$, $\mathcal{FCOM}$, and $\mathcal{FSTA}$ – changes depending on the class of sequential schedulers that activate the robots. In particular, we have focused on the three already studied classes of schedulers `SEQ` (sequential), `PERM` (permutation), and `RROBIN` (round-robin), and the *restricted* variants (`R-SEQ` and `R-PERM`), which vary from the original classes in that no robot can be activated twice consecutively.

For each $X \in \{\mathcal{OBLOT}, \mathcal{FCOM}, \mathcal{FSTA}, \mathcal{LUMI}\}$, we have shown that $X^{\texttt{RROBIN}} > X^{\texttt{R-PERM}}$. Then, we have proved that $\mathcal{LUMI}^{\texttt{SEQ}} \equiv \mathcal{LUMI}^{\texttt{R-SEQ}} \equiv \mathcal{LUMI}^{\texttt{PERM}} \equiv \mathcal{LUMI}^{\texttt{R-PERM}}$, thus showing how lights can be exploited to make sequential robots operate as under the restricted class `R-PERM`. Instead, for the $\mathcal{FCOM}$ model, we have proved that the equivalence holds only in the restricted case, i.e., $\mathcal{FCOM}^{\texttt{R-SEQ}} \equiv \mathcal{FCOM}^{\texttt{R-PERM}}$; this result highlights how the restricted schedulers enhance the communication power of $\mathcal{FCOM}$. In $\mathcal{OBLOT}$ and

in the remaining settings of $\mathcal{FCOM}$, we have instead proved that a more specific (i.e., less adversarial) class of schedulers strictly increases the computational power of the model.

A first future work may complete this investigation for the $\mathcal{FSTA}$ model, and then consider the inter-model relations (e.g., what is the relation between $\mathcal{FSTA}^{\texttt{R-SEQ}}$ and $\mathcal{OBLOT}^{\texttt{R-PERM}}$?). Another interesting research line is to generalize the classes of the restricted schedulers, including, for example, schedulers that can activate a robot for a bounded number of consecutive rounds.

## References


1 Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006. doi:10.1137/050645221.

2 Quentin Bramas and Sébastien Tixeuil. The random bit complexity of mobile robots scattering. *International Journal of Foundations of Computer Science*, 28(2):111–117, 2017. doi:10.1007/978-3-319-19662-6_15.

3 Kevin Buchin, Paola Flocchini, Irina Kostitsyna, Tom Peters, Nicola Santoro, and Koichi Wada. Autonomous mobile robots: Refining the computational landscape. In *Proc. of 35th International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 576–585. IEEE, 2021. doi:10.1109/IPDPSW52791.2021.00091.

4 Kevin Buchin, Paola Flocchini, Irina Kostitsyna, Tom Peters, Nicola Santoro, and Koichi Wada. On the computational power of energy-constrained mobile robots: Algorithms and cross-model analysis. In *Proc. of 29th International Colloquium On Structural Information and Communication Complexity (SIROCCO)*, pages 42–61. Springer, 2022. doi:10.1007/978-3-031-09993-9_3.

5 Kevin Buchin, Paola Flocchini, Irina Kostitsyna, Tom Peters, Nicola Santoro, and Koichi Wada. On the computational power of energy-constrained mobile robots. *Information and Computation*, 303:105280, 2025. doi:10.1016/J.IC.2025.105280.

6 Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Solving the pattern formation by mobile robots with chirality. *IEEE Access*, 9:88177–88204, 2021. doi:10.1109/ACCESS.2021.3089081.

7 Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012. doi:10.1137/100796534.

8 Stefano Clemente and Caterina Feletti. Fault detection and identification by autonomous mobile robots. In *Proc. of 4th Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, volume 330 of *LIPIcs*, pages 10:1–10:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.SAND.2025.10.

9 Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. Certified universal gathering in $\mathbb{R}^2$ for oblivious mobile robots. In *Proc. of 30th International Symposium on Distributed Computing (DISC)*, pages 187–200, 2016. doi:10.1007/978-3-662-53426-7_14.

10 Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. Autonomous mobile robots with lights. *Theoretical Computer Science*, 609:171–184, 2016. doi:10.1016/J.TCS.2015.09.018.

11 Xavier Défago, Maria Potop-Butucaru, and Sébastien Tixeuil. Fault-tolerant mobile robots. In *Chapter 10 of [17]*, pages 234–251. Springer, 2019. doi:10.1007/978-3-030-11072-7_10.

12 Mattia D'Emidio, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Characterizing the computational power of mobile robots on graphs and implications for the euclidean plane. *Information and Computation*, 263:57–74, 2018. doi:10.1016/J.IC.2018.09.010.

13 Caterina Feletti, Paola Flocchini, Debasish Pattanayak, Giuseppe Prencipe, and Nicola Santoro. Universal dancing by luminous robots under sequential schedulers. In *Proc. of 33rd International Colloquium On Structural Information and Communication Complexity (SIROCCO)*, pages 292–312. Springer, 2026. doi:10.1007/978-3-032-26465-7_16.

**14** Caterina Feletti, Paola Flocchini, and Nicola Santoro. On the computational power of mobile robots under sequential schedulers. In *Proc. of 27th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 16350 of *LNCS*. Springer Nature Switzerland, 2025. doi:10.1007/978-3-032-11127-2_18.

**15** Caterina Feletti, Lucia Mambretti, Carlo Mereghetti, and Beatrice Palano. Computational power of autonomous robots: Transparency vs. opaqueness. *Theorerical Computer Science*, 1036:115153, 2025. doi:10.1016/J.TCS.2025.115153.

**16** Paola Flocchini, Alfredo Navarra, Debasish Pattanayak, Francesco Piselli, and Nicola Santoro. Universal pattern formation by oblivious robots under sequential schedulers. *Distributed Computing*, 39(2):11, 2026. doi:10.1007/S00446-026-00504-3.

**17** Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors. *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *LNCS*. Springer, 2019. doi:10.1007/978-3-030-11072-7.

**18** Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1):412–447, 2008. doi:10.1016/j.tcs.2008.07.026.

**19** Paola Flocchini, Nicola Santoro, Yuichi Sudo, and Koichi Wada. On asynchrony, memory, and communication: Separations and landscapes. In *Proc. of 27th International Conference on Principles of Distributed Systems (OPODIS)*, volume 286 of *LIPIcs*, pages 28:1–28:23, 2023. doi:10.4230/LIPIcs.OPODIS.2023.28.

**20** Paola Flocchini, Nicola Santoro, and Koichi Wada. On memory, communication, and synchronous schedulers when moving and computing. In *Proc. of 23rd International Conference on Principles of Distributed Systems (OPODIS)*, volume 153 of *LIPIcs*, pages 25:1–25:17, 2019. doi:10.4230/LIPIcs.OPODIS.2019.25.

**21** Fabian Frei and Koichi Wada. Brief announcement: Distinct gathering under round robin. In *Proc. of 38th International Symposium on Distributed Computing (DISC)*, volume 319 of *LIPIcs*, pages 48:1–48:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.DISC.2024.48.

**22** Taisuke Izumi, Daichi Kaino, Maria Gradinariu Potop-Butucaru, and Tixeuil Sébastien. On time complexity for connectivity-preserving scattering of mobile robots. *Theoretical Computer Science*, 738:42–52, 2018. doi:10.1016/j.tcs.2018.04.047.

**23** Keita Nakajima, Kaito Takase, and Koichi Wada. Efficient self-stabilizing simulations of energy-restricted mobile robots by asynchronous luminous mobile robots. In *Proc. of 26th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 141–155. Springer, 2024. doi:10.1007/978-3-031-74498-3_10.

**24** Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999. doi:10.1137/S009753979628292X.