Comparison of scheduler models for distributed systems of luminous robots

Peters, Tom

Award date:
2020

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computer Science

Comparison of scheduler models for distributed systems of luminous robots

Master Thesis

Tom Peters

Supervisors:
Irina Kostitsyna
Kevin Buchin

Eindhoven, August 2020

Abstract

We investigate the computational power of multi-agent systems of autonomous point robots operating in the Euclidean plane. In particular, we compare four previously established models using pattern formation problems. In the first, the Oblot model, robots are oblivious and do not have persistent memory. They cannot communicate and have to base their movement entirely on the observed locations of the other robots. In the Lumi model, the robots are enhanced with a light that is visible both internally and externally, providing a constant amount of both a form of persistent memory as well as communication. In the last two models, FState and FComm, the light is only visible internally or externally respectively, each providing one of the two aspects this light offers: memory and communication. We investigate the relations between these models for two synchronous scheduler models: FSync and SSync. In the first model, all robots activate synchronously in rounds, in the second, the robots still activate in rounds, but not every robot activates each round. We show the computational relations between these four models under these two scheduler models when the robots are non-rigid, i.e. can be halted halfway during their movement, and are disoriented.

Furthermore, in depth exploration of the relation between Lumi and FComm under SSync, has revealed an additional scheduler model that is marginally stronger than SSync. For this new scheduler model the relation between Lumi and FComm becomes the same as under FSync.

Contents

Chapter 1

Introduction

In the field of distributed computing, autonomous mobile robots have been the topic of many investigations. The ultimate goal for these robots is to be deployed in situations and environments that are difficult to reach or hazardous to work in, for example inside a human body. In these places, it is either difficult or not desirable to setup more classical infrastructure for these multi-agent systems such as a central control mechanism. Therefore, these robots must be able to operate and organize themselves without such facilities. Furthermore, it would be undesirable to have a single controller. This would lead to a single point of failure, which would be hard to solve in the circumstances these robots operate in. In contrast to these practical applications, a lot of theoretical research has been conducted to develop a fundamental understanding of the possibilities and capabilities of these robots. We will try to extend this research and theoretically explore the computational capabilities of different models for multi-agent systems.

One of the most used models to research the computational capabilities of multi-agent systems is the OBLOT model, first introduced in [30]. In this model, the robots are oblivious, they forget everything they previously did. They are anonymous and homogeneous, i.e. they are indistinguishable from each other and they all execute the same algorithm. Moreover, there is no direct ability to communicate. The only information robots have to plan their movement comes from observing their peers. We will use point robots that do not have mass. Furthermore, we will not consider collisions, although previous research has sometimes considered fat robots and collisions, e.g. in [5, 6].

Robots usually operate in *Look-Compute-Move* cycles, in which they first observe their peers, then compute a new target location using an algorithm and lastly move towards that computed target. The algorithm is the same for all robots. Together, robots operating under these conditions can perform tasks and solve problems. Generally, these problems consist of a series of configurations the robots need to create, such as all robots gathering on the exact same location, or forming a uniform circle.

In the OBLOT model, a robot can either always reach its target location in the *Move* phase, or it can happen that it is interrupted anywhere on the path towards that target. In the first case, the movement is called *rigid*, in the second it is called *non-rigid*. In the *Look* phase, a robot plots the other robots in its own coordinate system. Each robot has its own coordinate system and they are not necessarily the same between robots. One distinction that can be made is if the coordinate systems of all robots have the same chirality or not.

Which robot activates when is decided by a scheduler. There are three common types of schedulers. The first is the *fully synchronous* scheduler (FSYNC), in which the robots activate in rounds. They operate completely synchronously and at every timestamp, every robot is in the same phase, either *Look*, *Compute* or *Move*. The second is the *semi synchronous* scheduler (SSYNC), in which the robots still operate synchronously in rounds, but not every robot will be activated every round. These schedulers were first studied in [31]. The last scheduler is the *asynchronous* scheduler (ASYNC), first used in [18]. Here, the robots do not have a common notion of time and the activation of the robots is unpredictable. Other schedulers, such as bounded schedulers [9] or

round-robin schedulers [8, 9], have also been studied.

Due to the limited abilities of the OBLOT model, algorithms are usually difficult and oftentimes an algorithm does not exist. An example of a difficult problem is breaking symmetry. The robots observe the same snapshot, and therefore will always act the same. Because of this, a new model was introduced: LUMI [7]. The idea was to enhance the robots in the OBLOT model by giving them limited memory and communicative abilities. In LUMI every robot is equipped with a light of which the robots can set the color. The number of colors the robot can choose from is constant and does not depend on the number of robots present. This light is persistent through cycles and can thus not only be used to transmit a limited amount of information to other robots, but it can also be used to remember the state the robot was in last cycle.

To fully comprehend this model and its abilities, two submodels have been created [21]. Each of these submodels has exactly one of the abilities of LUMI: communication or persistent memory. In the FCOMM model, the lights can only be observed by other robots and in the FSTATE model, the lights can only be observed internally. In the latter case, we will not speak about lights but rather about the internal state.

In this thesis we will try to give a better understanding about the computational power these four models have in relation to each other under the assumption of non-rigidity.

1.1 Related work

Since the introduction of the OBLOT model in [31], much research has been done with regards to gathering oblivious robots [1–4, 12, 19, 24, 25, 28, 29, 31], creating a uniform circle [10, 13–17, 26, 27], and general pattern formation [20, 23, 31, 33] to name a few. This has been done under various conditions such as using fat robots or robots with limited visibility.

For the LUMI model, the same problems have been studied. An overview of the results for the gathering problem with regards to LUMI can be found here [32]. Because LUMI is an enhancement of OBLOT, one would expect it to be more powerful than OBLOT. This relation has been studied and it has been proven that LUMI is indeed more powerful than OBLOT [7, 8, 11]. Moreover, it has also been shown that some problems are still not solvable under this more powerful model. For example it is still impossible for a group of LUMI robots that start at the same location with the same colored light to spread out in a deterministic manner.

For the submodels FCOMM and FSTATE not much is known yet. Algorithms for gathering two rigid robots have been found [21], but it is not trivial to adapt these algorithms to an environment in which robots can be halted at any moment on their route towards their target.

The computational relations between these four models have been studied for rigid robots under SSYNC and FSYNC schedulers [22]. The results are shown in Tables 1.1a, 1.1b and 1.1c, where M^{Sched} means model M under scheduler *Sched*, S stands for SSYNC, and F stands for FSYNC. To be read from left to top, $A > B$ means A is more powerful than B, $A \equiv B$ means A is computationally equivalent to B, and $A \perp B$ means models A and B are computationally incomparable. For non-rigid robots not much work has been done, but a few proofs and results can be easily transferred over from the works on rigid robots. For example, the requirement of rigidity can be lifted from the proof that gathering two robots to the exact same location is not possible under the SSYNC scheduler in the OBLOT model [31], whereas optimal algorithms are known for this problem in the LUMI model, even for non-rigid robots [32]. Another result for rigid robots that can be extended to non-rigid robots is that in the LUMI model, robots under an ASYNC scheduler have the same computational power as under an SSYNC scheduler. Moreover, robots in the LUMI model under the ASYNC scheduler can be shown to be more powerful than OBLOT robots under SSYNC schedulers [8], even when the requirement of rigidity is removed. Although these three results have been proven for rigid robots, they stay valid when we consider non-rigid robots.

	FCOMM^F	FSTATE^F	OBLOT^F
LUMI^F	$\equiv$	$>$	$>$
FCOMM^F	-	$>$	$>$
FSTATE^F	-	-	$>$

(a) Relations for FSYNC, rigid robots.

	FCOMM^S	FSTATE^S	OBLOT^S
LUMI^S	$>$	$>$	$>$
FCOMM^S	-	$\perp$	$>$
FSTATE^S	-	-	$>$

(b) Relations for SSYNC, rigid robots.

	LUMI^S	FCOMM^S	FSTATE^S	OBLOT^S
LUMI$^F \equiv$ FCOMM^F	$>$	$>$	$>$	$>$
FSTATE^F	$\perp$	$\perp$	$>$	$>$
OBLOT^F	$\perp$	$\perp$	$\perp$	$>$

(c) Relations between FSYNC and SSYNC.

Table 1.1: Relations between models for rigid robots, assuming common chirality [22]. To be read from left to top. For example, the topleft cell of (b) is to be read as LUMI$^S >$ FCOMM^S.

1.2 Results

We investigate the computational relations between the four models OBLOT, FCOMM, FSTATE, and LUMI specifically under the FSYNC and SSYNC schedulers and expand upon the work of [22]. First we consider problems that only contain a single robot in Chapter 3 and we show how the models relate to each other in this scenario. Next, we analyze problems for more than one robot in Chapter 4. We do this by subdividing the relations per scheduler. The proofs shown are valid for rigid robots as well as non-rigid robots. Moreover, they hold when we consider robots with the same chirality or robots where the chirality may differ. As it turns out, the relations already established for rigid robots with the same chirality shown in Tables 1.1a and 1.1b are the same if we consider non-rigid robots. It is still unknown if the same holds for the relations between a model under FSYNC on the one hand and a model under SSYNC on the other (Table 1.1c). The relations are also the same if we consider robots with the same chirality or robots where the chirality is different. The fact that LUMI^F is computationally equivalent to FCOMM^F and the fact that LUMI is more powerful than OBLOT under both SSYNC and FSYNC could be extended from previous work using already established formation problems. To prove the other relations, we constructed new problems. Furthermore, we investigate the relation between LUMI and FCOMM deeper. Under FSYNC, these two models are computationally equivalent, but under SSYNC, there is a strict dominance of LUMI over FCOMM. We show the point between SSYNC and FSYNC where this strict dominance of LUMI over FCOMM turns into an equivalence in Chapter 5.

Chapter 2

Preliminaries

The models considered in this thesis all consider a group of point robots. The robots operate in the Euclidean plane $\mathbb{R}^2$. In this plane they can move freely, and collisions are not considered. The robots are identical and can therefore not be distinguished from one another. They execute the same algorithm. The speed of the robots may vary and may not even be homogeneous while moving. Every robot has its own coordinate system. These are not necessarily consistent with each other, hence the chirality, unit distance and rotation may vary. The robots are equipped with sensors that allow a robot to observe the locations of its peers in their local coordinate systems. A robot will always observe itself at the origin of these coordinate systems. A robot is able to observe its peers at any moment and place, its vision is never impaired. The robots operate in fixed *Look-Compute-Move* cycles. At any point in time, every robot is in one of 4 phases, which it cycles through in order.

Look The robot observes its peers. This observation is an instantaneous snapshot containing the locations of all other robots within the local coordinate system of the observing robot. The observing robot will always put itself at the origin.

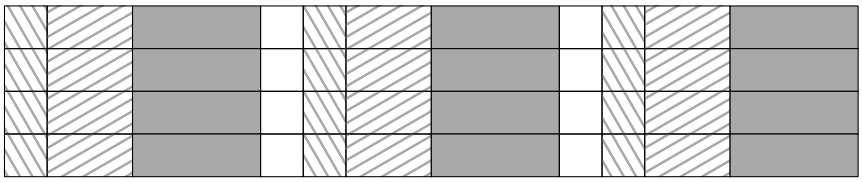
Compute The robot executes the algorithm and computes a target location and a path from the current location to this target location. This path can be a straight line or some curve. The input for the algorithm is the snapshot from the preceding *Look* phase.

Move The robot moves towards the target location calculated in the *Compute* phase along the computed path.

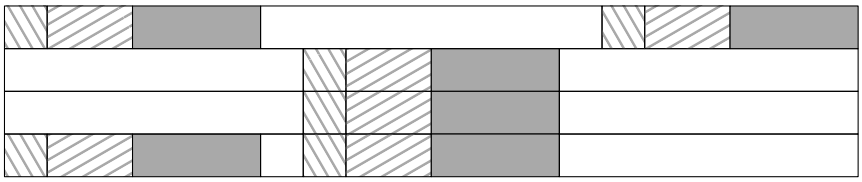
Sleep The robot does nothing and is not active. It is possible for a robot to skip this phase and immediately go to the next *Look* phase.

Whenever a robot is not in the *Sleep* phase, it is *active*. The robots cycle through these 4 phases in order based on a schedule defined by the *scheduler*. The scheduler acts as an adversary to the robots and decides when each robot goes into the next phase of the *Look-Compute-Move* cycle. In the literature, three main schedulers are considered, see Figure 2.1.

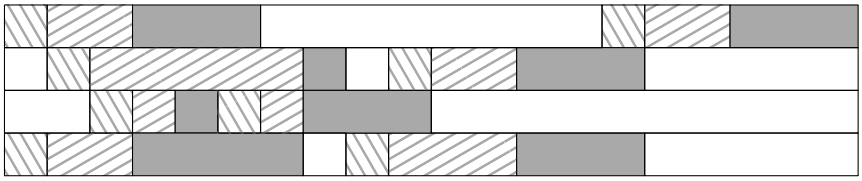
The first is the FSync scheduler. In these schedules, all robots go through all phases synchronously and at every moment in time, all robots are in the same phase. We say that the robots operate in *rounds*. Every round is one full cycle through the 4 phases. An example of an FSync schedule is shown in Figure 2.1a. The second scheduler is the SSync scheduler. This is a less strict version of FSync, where all active robots are still operating synchronously, but it is not required for every robot to be active every round. An example of an SSync schedule is shown in Figure 2.1b. The last scheduler is the ASync scheduler. In these schedules, the robots can activate at any point in time and there is no guarantee about synchronous behaviour at all. An example can be seen in Figure 2.1c. Note that this can create schedules in which a robot takes a snapshot in the *Look* phase, while other robots are moving. This means that as soon as the robot



(a) An example of a FSYNC schedule.



(b) An example of an SSYNC schedule.



(c) An example of an ASYNC schedule.

Figure 2.1: An example of the three main scheduler models. Each row is the schedule for a single robot. Time flows from left to right.

starts moving, its computation has been done with information that can be outdated. The only requirement for all these schedules is that they are *fair*, i.e. at any point in time, every robot will activate infinitely often in the future.

There are a couple of variations on these basic rules that will be considered here. First of all, the literature distinguishes between cases where the robots can detect *dense* points, or *multiplicities*, and cases where the robots are not able to detect these points. A dense point is a point where there are two or more robots on the exact same location. We will assume strict multiplicity detection here, meaning the snapshot will always contain all n robots and it will be a multiset. The next variation is whether or not the handedness of the internal coordinate systems of the robots is the same or not. If they are the same for every robot we say there is a common *chirality*, if they are not all the same, we say there is no common chirality. The last variation is in the *Move* phase. If a robot will always reach its target location, we will call the movement (and in extent the robot) *rigid*. In this case, the scheduler is not allowed to schedule the next phase if the robot has not yet reach its target. In synchronous schedulers, this means that all robots extend their *Move* phase until the last robot has reached its target. If the robot may be interrupted on its path, we call the robots non-rigid. To let non-rigid robots always make progress, they will always move at least some distance δ, unknown to the robots. If the distance a robot wants to move is less than δ, the robot will always reach its target.

2.1 Models

In the literature, various models have been defined based on the rules listed above. The first is the OBLOT model. In this model, the robots have no means to communicate. Furthermore, they are *oblivious*. When a robot ends its *Move* phase, its memory is wiped such that it does not have any knowledge of previous activations or computations.

An extension of this model is the LUMI model. Robots have a single light of which they can set the color. This color is picked from a finite colorset C, which usually includes the special color *Off*. All robots start with their light set to the color *Off*. If $|C| = 1$, the light is not used and this model is equivalent to OBLOT. The robots can set the color of the light at the start of the *Move* phase. The color of this light is persistent over cycles. This means it is not reset in between two cycles, as opposed to other memory, which is still wiped after each *Move* phase. It can be observed by all robots, including the owner of the light. The snapshot in the *Look* phase now becomes a multiset of pairs (*location*, *color*).

In some situations, it might be useful to describe a robot r as not having a single light, but having $\ell > 1$ lights instead, each with its own finite colorset C_ℓ. This is equivalent to r having a single light that can choose from $\prod_{i=1}^{\ell} |C_i|$ different colors.

This light functions as a limited form of both communication and memory. Two models have been designed that each offer one of these capabilities. The first is FCOMM. In this model the lights are only externally visible, meaning the colors of the lights can only be observed by robots other than the owner of the light. Each robot can communicate to the other robots, but at the end of each cycle, the robot forgets what it communicated. A snapshot in this model is the same as a snapshot in the LUMI model where the color of the observing robot is omitted.

In the second model, FSTATE, the color of the light can only be observed by the owner of the light. It is a persistent variable that can be set by the robot. In this case we also speak about the state of the robot. The colors are referred to as states. The robots have no means to communicate. A snapshot in this model is the same as a snapshot in the OBLOT model, with the addition of the current state of the robot.

2.2 Problems

In the literature, no real consensus has been reached yet about what is considered to be a problem. Take for example the following definition.

Definition 2.1 (OSCILLATING POINTS [8])**.** Two robots, x and y, initially in distinct locations, alternately come closer and move further from each other. More precisely, let $d(t)$ denote the distance of the two robots at time t. The OSCILLATING POINTS problem requires the two robots, starting from an arbitrary distance $d(t_0) > 0$ at timestamp 0, to move so that there exists a monotonically increasing infinite sequence of time instants t_0, t_1, t_2, t_3, $\cdots$ such that:

1. if $d(t_{2i+1}) < d(t_{2i})$ then $\forall h', h'' \in [t_{2i}, t_{2i+1}]$ if $h' < h''$ then $d(h'') \leq d(h')$; and
2. if $d(t_{2i}) > d(t_{2i-1})$ then $\forall h', h'' \in [t_{2i-1}, t_{2i}]$ if $h' < h''$ then t $d(h'') \geq d(h')$.

In this problem, we need to treat with care in how we approach these time instants. Take for example robots in SSYNC. We could easily assume that this problem requires the robots to alternate distances every other round of the scheduler. However, defined in this way, the problem does have knowledge about the inner workings of the robots, namely about the rounds. Problems defined with this kind of extra knowledge are a subclass of the set of problems defined without this knowledge. Therefore, any result proven using problems from this subclass only holds as long as we consider only problems from this subclass. Because we want to prove results for the more general case, we will not use any problems defined using extra knowledge. This includes knowledge about the color of the lights. We do not allow a problem to specify what color the lights should have and leave this solely up to the algorithm.

A problem is therefore only defined as a finite or infinite sequence of *configurations* for the robots to form. If the sequence is finite, the problem may require *termination*, meaning that the robots should stay in the last configuration and not move anymore. Moreover, in between every two configurations in this sequence, the problem declares a (possibly empty) set of configurations that should be avoided. The goal of the robots is now to form the sequence of configurations in order, while not forming any of the forbidden configurations. A configuration is a set of pointsets, usually defined by a predicate. These predicates should have the same evaluation under transformations such as translation, scaling and rotation. Therefore, although each robot has its own coordinate system, they all will evaluate the predicate the same. An example would be a regular hexagon. As soon as robots form a regular hexagon, every robot can acknowledge this, regardless of its coordinate system. Moreover, a configuration can have an extra requirement based on a previous configuration in the sequence. For example, if some configuration in the sequence requires the robots to form a regular hexagon, later in the sequence the problem might have a configuration that requires the robots to not only form a regular hexagon, but the exact same one. The robots might not always be able to evaluate this and enforcing this requirement is the task of the algorithm. We will only use the synchronous scheduler models (FSync and SSync). In those models, a configuration will only be considered to be formed if the locations of the robots fulfill the requirements at the end of a round of the scheduler, i.e. as soon as all robots finish their *Move* phase. Note that for non-rigid robots, it is still forbidden for robots to even pass through a forbidden configuration while moving, as the scheduler always has the option to halt the robots in exactly this configuration. Even if the robots would pass through this forbidden configuration at different timestamps during their *Move* phase, the scheduler could still interrupt the robots in such a way that a forbidden configuration could be formed. A problem can therefore only be solved by a specific algorithm, if there is no option for the scheduler to halt the robots in such a way that they would form a forbidden configuration.

The time or number of rounds it takes the robots to form each configuration in the sequence of configurations cannot be restricted by the problem definition. We will therefore interpret the problem description of the Oscillating points problem as follows. The timestamps t_0, t_1, $\cdots$, are at the end of a scheduler round. Furthermore, in between these timestamps, there could be any number of scheduler rounds.

2.3 Relation between models

We will use M^{Sched} to capture model $M \in \{\text{Oblot}, \text{FState}, \text{FComm}, \text{Lumi}\}$ under scheduler $Sched \in \{\text{FSync}, \text{SSync}, \text{ASync}\}$. We will use shorthand F, S, and A to express FSync, SSync, and ASync respectively. We will use $P \in M^{Sched}$ to express that there exists a team of robots that can solve problem P in model M under scheduler $Sched$. This means that problem P is solvable in M^{Sched}.

We define the following relations between two models X and Y.

- X is computationally not less powerful, or at least as powerful as Y ($X \geq Y$), if $\forall P \in Y$ we have $P \in X$. All problems solvable in Y are also solvable in X. This usually follows from the definition of the models.

- X is computationally strictly more powerful than Y ($X > Y$), if $X \geq Y$ and $\exists P \in X$ such that $P \notin Y$. Model X can solve every problem Y can solve, and moreover there exists at least one problem that is solvable by X but not by Y.

- X is computationally equivalent to Y ($X \equiv Y$), if $X \geq Y$ and $Y \geq X$. Both models can solve exactly the same problems.

- X is computationally incomparable to Y ($X \perp Y$), if $\exists P_1 \in X$ such that $P_1 \notin Y$, and $\exists P_2 \in Y$ such that $P_2 \notin X$. Both models X and Y can solve a problem that the other model cannot solve.

Because of their definitions, we can immediately conclude that any problem that is solvable in some model M under SSync is also solvable under the same model M under FSync. This holds because any FSync schedule is also an SSync schedule. Similarly, any problem that is solvable in Oblot, will also be solvable in Lumi by the same algorithm by just not using the light. The following relations all follow immediately from the definitions of the models and schedulers in the same way:

- $M^F \geq M^S \geq M^A$ for every model $M \in \{\text{Oblot}, \text{FState}, \text{FComm}, \text{Lumi}\}$.
- $\text{Lumi}^{Sched} \geq \text{FState}^{Sched} \geq \text{Oblot}^{Sched}$ for every scheduler $Sched \in \{\text{FSync}, \text{SSync}, \text{ASync}\}$.
- $\text{Lumi}^{Sched} \geq \text{FComm}^{Sched} \geq \text{Oblot}^{Sched}$ for every scheduler $Sched \in \{\text{FSync}, \text{SSync}, \text{ASync}\}$.

Chapter 3

A single robot

In the next chapters, we will explore the computational relationships between Lumi, FState, FComm and Oblot under the various schedulers. Because of the distributed nature of these models, usually problems for these models require at least 2 robots. However, when there is only a single robot, the relations are different. To the best of our knowledge, this has not been researched before. To give a complete overview of the relations, we will in this chapter show first that when only a single robot is considered, schedulers and rigidity do not really matter anymore. Furthermore, we show that Lumi is equivalent to FState, FComm is equivalent to Oblot and the first two are computationally more powerful than the last two.

When only a single robot is considered, there are no differences between schedulers. Any schedule fulfills the requirements to be an FSync schedule. Every snapshot taken by the robot will be the same, since a robot always puts itself at the origin. This means that even if the robot is non-rigid and will be interrupted, it will wake up and its snapshot will be the same as if it was not interrupted. Problems defined for a single robot cannot set a single target for that robot. Defining a problem in this way requires extra knowledge about the coordinate system of the robot, such as rotation and unit distance, which we do not allow. Problems defined for a single robot can therefore only require infinite movement, which can be done by a non-rigid robot, or they define that the robot should stay still forever, which can also be done by a non-rigid robot. Therefore, it does not matter if the robot is rigid or not.

It is trivial to see that FComm becomes equivalent to Oblot, because lights are only external in FComm and there are no other robots to see the color, this information gets lost. Lumi becomes equivalent to FState, as for Lumi the communicative abilities are lost and the only feature that remains, is the memory.

Lemma 3.1. FComm $\equiv$ Oblot and Lumi $\equiv$ FState.

The following problem is used to show the difference between these models when there is only a single robot.

Definition 3.2 (Step)**.** Let the single robot be r. The Step problem requires r to reach a location that is not its starting location. From then on, it is not allowed to move again.

Lemma 3.3. Step $\in$ FState, using 2 states.

Proof. An algorithm is shown that will solve this problem using the two colors *Off* and *Stop*. On wake up, the robot detects it is in the initial *Off* state and moves a little bit. The robot changes its state to *Stop*. If the robot wakes up in state *Stop*, it does not move. Thus a single robot in FState can solve Step. This also works in the non-rigid scenario, as the robot is guaranteed to move at least a little bit, changing its location. □

Lemma 3.4. Step $\notin$ Oblot.

Proof. With only one robot, that robot will always have the same snapshot every *Look* phase. Therefore, the action taken by the robot will be the same every activation. The robot can choose to either move or stay still. In the first case, the robot will move every round and never stop. In the second case, the robot will stop on its initial location. Both will not solve the problem. This is also true in the non-rigid scenario. Therefore there does not exist an algorithm that would solve the problem in OBLOT. □

From these lemmas, we can directly conclude the following.

Theorem 3.5. *For a single robot,* LUMI *is computationally equivalent to* FSTATE*. Both are computationally strictly more powerful than* FCOMM*.* FCOMM *is computationally equivalent to* OBLOT*. Hence* LUMI ≡ FSTATE > FCOMM ≡ OBLOT*.*

Proof. The equivalences follow from Lemma 3.1. Lemmas 3.3 and 3.4 show there exists a problem solvable in FSTATE but not in OBLOT. Together with FSTATE ≥ OBLOT by definition, the theorem follows. □

Chapter 4

General relations

In this chapter we will focus on the computational relationships between the four described models under FSync and SSync. These are already known for rigid robots when there is a common chirality Table 1.1 [22]. The problems used to show these relations usually require robots to reach some location and they will not hold for non-rigid robots anymore. We will prove that the same relations hold when the robots are non-rigid as when they are rigid. Furthermore, we will prove that the relations are the same when we consider a situation where the chirality is not the same for every robot.

4.1 SSync

We will first investigate the relations under the SSync scheduler, which lets the robots operate in rounds, but not every robot activates every round. However, we will prove all the impossibility results using models under the FSync scheduler. Any FSync schedule is also a valid SSync schedule. Therefore, if a problem is not solvable under FSync, there is also at least one SSync schedule for which the same problem is not solvable, namely the FSync schedule. Therefore any problem that is impossible under FSync for a specific model will also be impossible under SSync for the same model, this also proves the impossibility under SSync. In this way we can later use the same results when investigating FSync.

4.1.1 Oblot vs FComm/FState

First the relations between Oblot^S on the one hand and FComm^S and FState^S on the other will be shown. A classic example of a problem not solvable in Oblot^S would be the following [31].

Definition 4.1 (Rendezvous [31])**.** Let 2 robots initially be on distinct locations. They now need to gather in the exact same location and stay there infinitely.

Although algorithms have been found that solve this problem in FComm^S and FState^S under the assumption of rigid robots [21], to this date no research has been done to find out whether or not Rendezvous is solvable in FComm^S and/or FState^S using non-rigid robots. We use the following problem instead to make the distinction. This problem relies on the fact that in Oblot it is not possible to break symmetry once the robots reach a fully symmetric configuration, whereas both FComm and FState robots can set their lights prior to reaching this configuration to prevent total symmetry.

Definition 4.2 (N-GON Round-Trip)**.** Take $n > 2$ robots that start in an initial configuration that can be constructed as follows: Take a regular n-gon. Take one arbitrary vertex a of this n-gon. Put a robot at every vertex except at a. Also place a robot at the center. From this starting location, the robots should first create a regular n-gon and then reach the starting location again, see Figure 4.1.

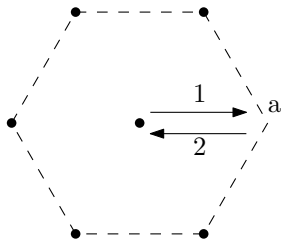


Figure 4.1: N-GON ROUND-TRIP. The supporting regular polygon is shown.

Lemma 4.3. N-GON ROUND-TRIP $\notin$ OBLOT^F.

Proof. To reach a contradiction, assume there exists an algorithm A that would solve this problem. This algorithm would at some point have the robots create a regular n-gon. As soon as the robots reach this configuration, there is no way for the robots to find out which vertex was empty in the starting configuration. Therefore they cannot recreate the starting configuration and A would not solve the problem. □

From this it also follows that N-GON ROUND-TRIP $\notin$ OBLOT^S.

Lemma 4.4. N-GON ROUND-TRIP $\in$ FSTATE^S, with 4 states.

Proof. The proof is constructive. The robots start in a state *Off*. As soon as they wake up, the robots can uniquely identify the regular polygon. The robots at the vertices immediately go to a state *Stop*, indicating that they should never move again. The robot in the center moves towards the empty vertex on the polygon and sets its state to *Move out*, such that if it gets interrupted by the scheduler, the robot recognizes that the second pattern has not yet been formed. As soon as the robot reaches the empty vertex, possibly after multiple rounds of activation, it switches to a state *Move in* and goes back to the center. Again, if the scheduler interrupts the robot, it still knows it was heading to the center. It is possible for the robots on the polygon to wake up for the first time only after the regular polygon has been formed. However, these robots know that they do not have to go to the center because their state is *Off*, and not *Move out*. □

Lemma 4.5. N-GON ROUND-TRIP $\in$ FCOMM^S, with 3 colors.

Proof. The proof is constructive. The robots start with their lights off. As soon as they wake up, the robots can uniquely identify the regular polygon. The robot in the center, let us call this robot r, sets its light to color *Don't move*. Every robot that now sees another robot with light *Don't move* will not move. If r has not yet been active, they can still detect that they are on the regular polygon and they will stay there. Now if r gets activated, it will move outwards because it is the only robot that does not see another robot with color *Don't move*. Even if r gets interrupted during its movement, it will keep moving outwards.

As soon as r reaches the polygon, it will wait for at least one other robot to complete a cycle. The robots that now see a regular polygon and a robot with color *Don't move*, will switch their light to *Polygon formed*, while not moving. Now robot r sees at least 1 robot with color *Polygon formed* and no robots with color *Don't move*, therefore it knows that the second pattern has been formed and that it can move. Therefore, it can move to the center of the polygon, creating the third pattern. Even if it gets interrupted, it sees the same colors and as a result it will continue its journey to the center. Robots on the polygon will not move, because they still see another robot with *Don't move*.

Robot r will be the only robot that moves. The robots on the polygon always see a robot in the center or at least on robot on the polygon with color *Don't move*, indicating that they should not move. It is not possible for the middle configuration to be skipped, because r will only move back to the center as soon as it sees another robot with color *Polygon formed*. This can only happen after the second configuration has been formed. □

From these lemmas, we can conclude the following.

Theorem 4.6.

1. FCOMM^S *is computationally strictly more powerful than* OBLOT^S. *Hence* FCOMM^S > OBLOT^S.

2. FSTATE^S *is computationally strictly more powerful than* OBLOT^S. *Hence* FSTATE^S > OBLOT^S.

Proof.

1. Lemmas 4.3 and 4.5 show there exists a problem solvable in FCOMM^S but not in OBLOT^S. Together with FCOMM$^S \geq$ OBLOT^S by definition, the theorem follows.

2. Lemmas 4.3 and 4.4 show there exists a problem solvable in FSTATE^S but not in OBLOT^S. Together with FSTATE$^S \geq$ OBLOT^S by definition, the theorem follows. □

4.1.2 FComm vs FState

For the relation between FCOMM^S and FSTATE^S, we already introduced a problem in Definition 3.2: STEP. However, we do not want to allow single robot problems as the relations are completely different in that scenario. One might think that extending this problem to two robots might give the same result. However, this is not true as the extended problem is solvable in FCOMM^S.

Definition 4.7 (2 ROBOT STEP)**.** Let 2 robots initially start on distinct locations. The 2 ROBOT STEP problem requires both of these 2 robots to reach a location that is not their respective starting location. From then on, they are not allowed to move again.

Lemma 4.8. 2 ROBOT STEP $\in$ FCOMM^S, using 3 colors.

Proof. The proof is constructive. Let each robot start with a color *Not moved*. As soon as a robot wakes up it will move and change its color to *I moved*. If it sees the other robot already has its color at *I moved*, it puts its color to *You moved* instead of *I moved*, indicating to the other robot that it has activated at least once and thus moved, and at the same time indicating that this robot has moved. As soon as a robot sees its partner in *You moved*, it will not move anymore, and only change its own light to *You moved*. To prevent the robots ending on their starting location, the move steps will only be directly away from each other. They will eventually both reach *You moved* and they will not move anymore.

This algorithm will be correct if both robots eventually reach color *You moved* at the same time and keep that color each activation. It is impossible for a robot to reach *Not moved* after the first time it has activated. The only time a robot r_2 that has color *You moved* will change its color to something else is when the other robot r_1 would have color *Not moved*. This is not possible because r_2 only did go to *You moved* because r_1 had either color *I moved* or *You moved*, indicating it has activated at least once and will therefore never have *Not moved* again. This means that as soon as a robot has color *You moved*, it will keep this color each activation. Because both robots will have *You moved* eventually and because the robots cannot get another color as soon as they do so, the algorithm is correct. □

Note that the robots will possibly move a couple of times before they reach their final location. This is allowed by the problem. We could create a problem where this behaviour is not allowed, but it would require the problem to be described in terms of the inner workings of the robots, which we do not allow.

It is easy to see that it also holds that 2 ROBOT STEP $\in$ FSTATE^S, using the same algorithm as for STEP (Lemma 3.3). Because 2 ROBOT STEP $\in$ FCOMM^S and 2 ROBOT STEP $\in$ FSTATE^S,

(a) The movement for REPELLING POINTS

(b) Robot r is not allowed to go closer than half the initial distance.

Figure 4.2: REPELLING POINTS, note the dense point on the left.

this does not give the separation between FSTATE and FCOMM. However, it does show the difference in relations when we consider problems defined for a single robot or problems defined for multiple robots.

Nevertheless, there are other problems that result in this separation. The first will be solvable in FSTATE, but not in FCOMM and relies on the fact that if a single robot activates twice in FCOMM, it will not have had the time to communicate.

Definition 4.9 (REPELLING POINTS)**.** Consider 3 robots such that 2 are on the same location and the third robot r is on a different location. Let distance $d(i)$ be the distance between r and the dense point after round i, such that $d(0)$ is the starting distance. Now r needs to first move towards the dense point such that at some round i, it holds that $d(i) < d(0)$. Next, r needs to move away from the dense point such that at some round $j > i$, it holds that $d(j) > d(i)$. All configurations in which the distance is less than half the starting distance are not allowed, so $d(k) \geq \frac{1}{2}d(0)$ must hold for all k. All configurations in which any robot other than r has moved are also not allowed. This is visualized in Figure 4.2.

Lemma 4.10. REPELLING POINTS $\in$ FSTATE^S, using 2 states.

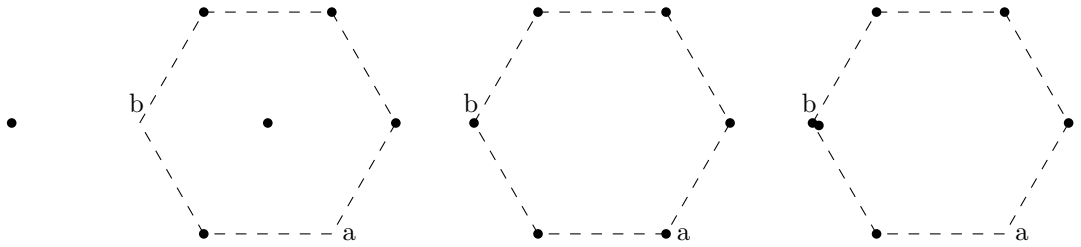
Proof. The proof shows an algorithm that would solve the problem. All robots can uniquely identify their role at all times because of the dense point. As soon as r wakes up, it switches its color from the initial color to *Moved in* and moves towards the other two robots for less than half the distance, satisfying the first condition. As soon as r wakes up with color *Moved in*, it will detect that it has already moved inwards and it will move outwards, solving the second configuration. □

Lemma 4.11. REPELLING POINTS $\notin$ FCOMM^S.

Proof. To reach a contradiction, assume an algorithm A exists that would solve this problem under any SSYNC schedule, but specifically under schedule S. We will now create another SSYNC schedule S' for which algorithm A would not solve REPELLING POINTS.

Because A solves the problem under S, robot r will have to move towards the dense point for some fraction of their distance at some round i. Let S' be the same as S up until round i. Now for some finite number of rounds, S' will only let r activate. If r is the only robot that activates for a couple of rounds, the other robots will not have been activated to communicate and tell r that it has moved. Because it is impossible for r to obtain this information on its own, this results in r moving in the same direction for the same fraction of the remaining distance as long as no other robot gets activated. Robot r will keep moving inwards every round for a fixed fraction of the distance, because the target point is always calculated relative to the other robots in the snapshot. If the target point would be calculated based on the unit distance of the coordinate system of r, it could be possible that r will come too close to the dense point in the first round. Because of the assumption that A solves the problem, this is not possible. Therefore r will try to move for the same fraction every time it activates. It could be that r is interrupted during this movement, but the next activation it will still compute a target location that is a fixed fraction of the remaining distance away.

Eventually, after moving inwards for a finite number of rounds x, the distance $d(i + x)$ will be less than half the starting distance $d(0)$. Therefore, if S' lets only r be active for x rounds, $d(i + x) < \frac{1}{2}d(0)$. This is an invalid configuration and REPELLING POINTS would not be solved by A under S'. Schedule S' is still a fair schedule, because x is a finite number of rounds. Therefore,



(a) First configuration (b) Second configuration (c) Third configuration, note the 2 robots at b

Figure 4.3: An example of the N-GON problem for $n = 6$ robot. The supporting regular n-gon is also drawn.

the other robots will eventually get activated again. Repelling points is not solvable by A under S', which is a valid SSync schedule. This contradicts the assumption that A would solve the problem under any SSync schedule. As a result, a valid algorithm that solves Repelling points in FComm under an SSync scheduler does not exist. □

Lemma 4.10 and Lemma 4.11 show that there exists a problem solvable in FState^S, but not in FComm^S. The opposite also holds.

Definition 4.12 (N-GON)**.** Let $n > 3$ robots, start in an initial configuration that can be constructed as follows: Start with a regular n-gon. Take two arbitrary vertices a and b. Put a robot at every vertex of the n-gon, except for vertices a and b. Put a robot at the exact center and a robot on the ray from the center to b such that the distance to the center is twice the distance from b to the center. An example is shown in Figure 4.3a.

From this starting configuration, the robots must first form a regular n-gon (Figure 4.3b) after which the robots must form a regular n-gon with no robot at location a and two robots at location b (Figure 4.3c). Between the second and third configuration, no configuration is allowed in which robots have moved that are not on location a in the second configuration.

Lemma 4.13. N-GON $\notin$ FState^F.

Proof. To reach a contradiction, assume that there exists an algorithm A that would solve this problem. At some point, the robots following A have to create the regular n-gon. Now the robots are in a completely symmetric configuration and although the robot at location a can remember that it has to move, there is no way to distinguish the other robots and determine which of the other robots was previously outside of the n-gon and is currently at b, using only a constant amount of memory. It would be possible if the robots could count the amount of robots, for example by saying remembering that it was the third counterclockwise robot. However, this would take a logarithmic amount of information, and the robots can only use a constant amount of colors. The robot at location b does know that there is a robot that should move towards b, but it has no means of communicating this information as it is not allowed to move anymore. Therefore, the robots cannot create the third pattern and there exists no algorithm that would solve the problem. □

Lemma 4.14. N-GON $\in$ FComm^S, using 3 colors.

Proof. The proof shows an algorithm that solves the problem. Let all robots start with color *Off*. The robot in the center and the one outside the n-gon can uniquely identify their own location and the original n-gon. As soon as they wake up, the center robot r_a sets its light to color A, while the outside robot r_b sets its light to color B. They can go to their respective location, forming the second configuration. Now every robot that sees another robot with color A will stay still, while the single robot r_a that does not see this color will move to the single robot r_b with color B, creating the third pattern. Intermediate configurations may occur when interrupted. However, r_a

will move outwards as long as it is on the line segment from the center to a and it will move to the robot with color B when it is on the polygon or on the line segment from a to b. In between configuration 2 and 3, the only robot that is allowed to move is this single robot, so even if it gets interrupted, it will still be a valid configuration. □

From these two problems, we can conclude the following.

Theorem 4.15. FCOMM^S *is computationally incomparable with* FSTATE^S. *Hence* FCOMM^S $\perp$ FSTATE^S.

Proof. This follows from Lemmas 4.10 and 4.11 which show there exists a problem solvable in FSTATE^S but not in FCOMM^S, and Lemmas 4.13 and 4.14 showing the opposite: there exists a problem solvable in FCOMM^S but not in FSTATE^S. □

Furthermore, from Lemma 4.14 it follows that N-GON $\in$ LUMI^S and from Lemma 4.10 it follows that REPELLING POINTS $\in$ LUMI^S. This also concludes the relations for LUMI^S with regards to FSTATE^S and FCOMM^S.

Theorem 4.16.

1. LUMI^S *is computationally strictly more powerful than* FSTATE^S. *Hence* LUMI^S $>$ FSTATE^S.
2. LUMI^S *is computationally strictly more powerful than* FCOMM^S. *Hence* LUMI^S $>$ FCOMM^S.

Proof.

1. LUMI^S $\geq$ FSTATE^S holds by definition. N-GON $\in$ LUMI^S follows from Lemma 4.14. This problem is not solvable in FSTATE^S (Lemma 4.13).
2. LUMI^S $\geq$ FCOMM^S holds by definition. REPELLING POINTS $\in$ LUMI^S (Lemma 4.10), but REPELLING POINTS $\notin$ FCOMM^S (Lemma 4.11). □

It also follows that LUMI^S $>$ OBLOT^S. This was already known for rigid robots [8] and that proof would also hold for non-rigid robots.

4.2 FSync

For FSYNC schedules, where the robots get activated in rounds and every round every robot is active, a lot of the same problems and lemmas can be used to show the computational relations between the models. First of all, we can show that the relations between FCOMM^F, FSTATE^F and OBLOT^F are the same as for SSYNC. From Lemma 4.3, Lemma 4.4 and Lemma 4.5 the same relations as for SSYNC can be concluded.

Theorem 4.17.

1. FCOMM^F *is computationally strictly more powerful than* OBLOT^F. *Hence* FCOMM^F $>$ OBLOT^F.
2. FSTATE^F *is computationally strictly more powerful than* OBLOT^F. *Hence* FSTATE^F $>$ OBLOT^F.

Proof.

1. Lemmas 4.3 and 4.5 show there exists a problem solvable in FCOMM^F but not in OBLOT^F. FCOMM^F $\geq$ OBLOT^F holds by definition. Together, they prove the theorem.
2. Lemmas 4.3 and 4.4 show there exists a problem solvable in FSTATE^F but not in OBLOT^F. FSTATE^F $\geq$ OBLOT^F holds by definition. Together, they prove the theorem. □

For rigid robots, it is known that $\text{LUMI}^F \equiv \text{FCOMM}^F$ [22]. This proof simulates LUMI^F robots using FCOMM^F robots. This is done by letting the FCOMM^F robots communicate their own colors to the other robots, which the other robots can then communicate back. For this proof, the assumption of rigid robots can be lifted without any problem because this communication can still happen if the robots are non-rigid.

Theorem 4.18. LUMI^F *is computationally equivalent to* FCOMM^F. *Hence* $\text{LUMI}^F \equiv \text{FCOMM}^F$.

The other relations for LUMI^F can now be deducted.

Theorem 4.19. LUMI^F *is computationally strictly more powerful than* FSTATE^F. *Hence* $\text{LUMI}^F > \text{FSTATE}^F$.

Proof. $\text{LUMI}^F \geq \text{FSTATE}^F$ holds by definition. From Lemma 4.14 it follows that N-GON $\in \text{LUMI}^F$. This problem is not solvable in FSTATE^F (Lemma 4.13). □

Again we can also conclude $\text{LUMI}^F > \text{OBLOT}^F$. However, this was already known [22] for rigid robots and this result would still hold for non-rigid robots.

Due to the equivalence we can also list the relations for FCOMM^F.

Theorem 4.20.

1. FCOMM^F *is computationally strictly more powerful than* FSTATE^F. *Hence* $\text{FCOMM}^F > \text{FSTATE}^F$.
2. FCOMM^F *is computationally strictly more powerful than* OBLOT^F. *Hence* $\text{FCOMM}^F > \text{OBLOT}^F$.

Proof.

1. Theorem 4.18 shows $\text{LUMI}^F \equiv \text{FCOMM}^F$. Theorem 4.19 shows $\text{LUMI}^F > \text{FSTATE}^F$. Together this proves the statement.
2. $\text{LUMI}^F > \text{OBLOT}^F$ as proven by [22] for rigid robots. This result still holds for non-rigid robots. Due to the equivalence with FCOMM^F (Theorem 4.18), this result also holds for FCOMM^F. □

The relations for rigid robots for models under the FSYNC and SSYNC schedulers where already known when there is a common chirality. The proofs here work for both rigid robots as well as for non-rigid robots. Moreover, they also hold when there is no common chirality. Therefore, Tables 1.1a and 1.1b also give an overview of the relations between these four models under synchronous schedulers under any of these variations.

4.3 FSync vs SSync

For rigid robots, the distinction between FSYNC and SSYNC was made using the CENTER OF GRAVITY EXPANSION problem.

Definition 4.21 (CENTER OF GRAVITY EXPANSION [22])**.** Let R bet a set of robots. The CENTER OF GRAVITY EXPANSION problem requires each robot $r_i \in R$ to move from its initial location (x_i, y_i) directly to $(f(x_i, c_x), f(y_i, c_y))$, where $f(a, b) = \lfloor 2a - b \rfloor$ and (c_x, c_y) is the center of gravity of the initial configuration.

For rigid robots, this problem is solvable in any model under FSYNC, but not by any model under SSYNC, because in SSYNC as soon as a robot moves, it is impossible to maintain information about the original center of gravity. However, this is possible in FSYNC, because all robots move at the same time and reach a new configuration in which the center of gravity is the same as in the old configuration. When switching to non-rigid robots, this problem does not give the distinction

anymore between FSYNC and SSYNC. In FSYNC robots can now be interrupted and create a configuration in which the center of gravity is not equal anymore to the center of gravity in the previous configuration. The only difference between the two models is now that in FSYNC, all robots will move at least a little bit every round, while in SSYNC some robots can stay still. We have not been able to find a problem solvable in any model under FSYNC using non-rigid robots that is unsolvable under SSYNC, nor have we been able to prove that all problems solvable in FSYNC for non-rigid robots are also solvable in SSYNC. Therefore, this is still an open problem for future work.

Chapter 5

Lumi vs FComm

Under the FSync scheduler, Lumi and FComm are computationally equivalent, see Theorem 4.18. The difference between Lumi and FComm is that in FComm robots do not have access to the color of their own light, while in Lumi they do. Hence, to simulate any algorithm A designed for Lumi robots using FComm robots, the FComm robots somehow need to be able to get information about the color of their own light. This is done using two activation rounds. A specific light with two values is used to let the robots know in which of the two rounds the robots currently are. In the first round, robots look at each other, see the state every robot is in and set their light accordingly. In the second round, robots can use this information to calculate the color of their own light. The information the FComm robots now have is equal to the information Lumi robots have. Therefore the FComm robots can execute the same algorithm in the second round [22]. This works because every round, every robot is active. Moreover, as soon as a robot changes its state in the second round, the next round robots are active to process this state change and send it back. This is not true for robots under the SSync scheduler. Because if a single robot is the only robot that is activated in two consecutive rounds, this robot will not be able to gain knowledge about its internal state as no other robots have been active to communicate with. This raises the question what the minimal scheduler requirements are to still be able to gain knowledge about the robot's own internal state. As it turns out, it is sufficient to disallow any two consecutive rounds i and $i+1$ when in round i only a single robot r is active and r is also active in round $i+1$.

Definition 5.1 (StricterSSync)**.** Let this scheduler be exactly the same as the normal SSync scheduler, but disallow any schedules which have two consecutive rounds i and $i+1$ such that in round i some robot r is the only active robot and r is also active in round $i+1$. An example of a non-valid schedule is shown in Figure 5.1.

We will prove that under the StricterSSync scheduler robots in the FComm model can gain knowledge about their internal state and solve the same problems as Lumi robots. As for FSync,

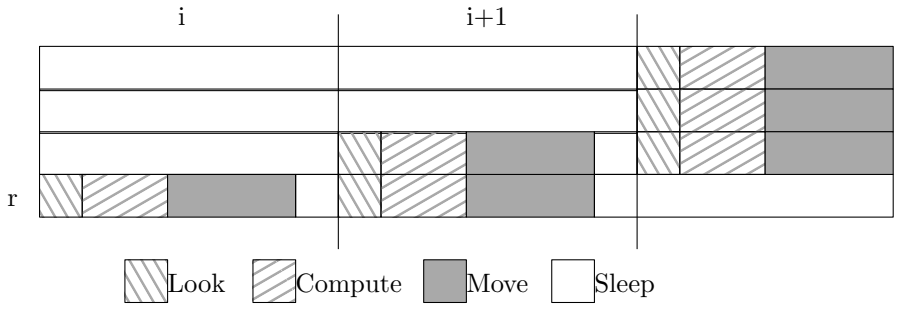


Figure 5.1: An example of a schedule that would NOT be allowed by StricterSSync. Robot r and the two rounds i and $i+1$ are shown.

the FCOMM robots need to be able to get information about the color of their own light. To do this, we propose the procedure FCOMMTOLUMI(A). The main difference with the procedure used under FSYNC is that under FSYNC the robots have a guarantee that every robot operates at the same time. Therefore, in this new procedure, the robots need to synchronize in some way.

5.1 Procedure

This procedure FCOMMTOLUMI(A) takes a deterministic algorithm A designed for LUMI robots under STRICTERSSYNC and simulates it using FCOMM robots under STRICTERSSYNC. We will assume that during the execution of A there will be no multiplicities. Initially, let us also assume all robots have the same chirality and no robot is present at the center of gravity. These last two assumptions will later be lifted, the assumption about no multiplicities will not be lifted. Now a circular ordering on the robots can be defined as follows. Calculate the center of gravity and order the locations of the robots around it in an anticlockwise manner. In the case that two robots are collinear with the center of gravity, we call them *coradial*. When coradial robots are present, the order can be defined by letting a robot closer to the center of gravity precede a robot further away. This order implies that any robot r has two neighbouring robots. One of them is the predecessor $pred(r)$ and one the successor $succ(r)$. This information will be used by FCOMMTOLUMI. The robots will calculate their predecessor and successor and transmit the color of their neighbours' lights back. Because there is a constant number of neighbours, each robot only has to transmit a constant amount of extra information.

Recall that the FCOMM robots need information about the color of their own light to simulate LUMI robots. Let us call the light used by algorithm A the *lumi-light*. The way in which FCOMMTOLUMI gives a robot access to the information about the color of its own light, is by letting a robot do nothing until another robot has seen the color and has communicated it back. To this purpose, various extra lights are added to each FCOMM robot. Recall that this is equivalent to a robot having only a single light. For clarity of presentation, we will speak of each robot having multiple lights. Each FCOMM robot has the following lights.

lumi-light This light is the light originally used by algorithm A designed for LUMI robots. Both its purpose and colors are the same as in algorithm A. The number of colors that this light can be is still the same, let us call this number c.

copy-lumi-light Every robot has two of these lights, corresponding to the two neighbours defined by the circular ordering. The purpose is to communicate the color of the *lumi-light* to the neighbours. It can choose from the same set of colors as the *lumi-light*. If a robot knows that every single *copy-lumi-light* has been set properly, the *copy-lumi-lights* of neighbours can be used to deduce the color of the robot's own *lumi-light* in the following way.

Let $x.$*copy-lumi-lights* be the set of colors of the *copy-lumi-lights* of robot x and let $x.$*lumi-light* be the single element set containing the color of the *lumi-light* of robot x. Now if r takes the set of *copy-lumi-lights* of its predecessor, it can find its own color by removing the color of the *lumi-light* of the other neighbour of its predecessor.

$$r.color = pred(r).\text{copy-lumi-lights} \setminus pred(pred(r)).\text{lumi-light}$$

Where "\" is the difference between two sets. An example of this calculation is shown in Figure 5.2.

In the case there are only 2 robots, this calculation refers to the color of the robot's own *lumi-light*. Take in this case $pred(pred(r)).$*lumi-light* to be the empty set. The rest of the procedure will be exactly the same, regardless of the amount of robots.

state-light This light is used to synchronize the robots in FCOMMTOLUMI. It can be one of 5 states: *Copy*, *DoneCopy*, *Trying*, *Finished*, and *Reset*. Initially it is set to *Copy*.

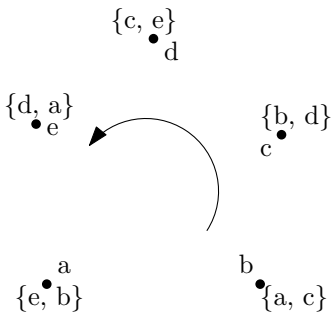


Figure 5.2: An example of how a robot can calculate its own color light. The circular ordering is visible with an arrow. For each of the 5 robots the color of the *lumi-light* is listed (a, b, c, d, or e), as well as the two colors of the *copy-lumi-lights* in order (first predecessor, then successor). For the robot with color of the *lumi-light* b, the calculation is as follows: $\{e, b\} \setminus e = b$.

execute-light The purpose of this light is to ensure fairness of the simulated schedule. Without it, we could not guarantee that at every point in time, every robot will be able to do a step in LUMI algorithm A eventually. Its set of colors consists of *NotExecuted*, *Executed* and *AllExecuted*. It is initially set to *NotExecuted*.

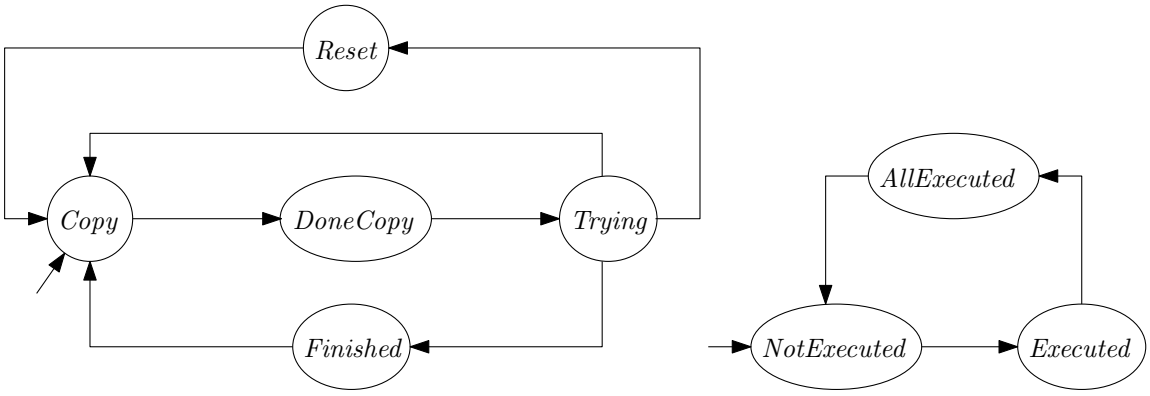
copy-execute-light A robot cannot see the status of its own *execute-light*. To let every robot still have access to this information, the same approach is used as for the *lumi-light*. Every robot has two *copy-execute-lights*, corresponding to the two neighbours of a robot. Now if the colors are correctly copied, information about a robot's own *execute-light* can be deduced from neighbouring *copy-execute-lights* in the same way as *copy-lumi-lights* are used to find information about a robot's *lumi-light*.

Together, this means that every robot has a total of $1 + 2 + 1 + 1 + 2 = 7$ lights, with a total number of $c \cdot c^2 \cdot 5 \cdot 3 \cdot 3^2 = 135c^3$ possible color combinations. This is a constant number as long as c is constant and does not depend on the number of robots. The robots do not know in which state (color of *state-light*) they currently are. They have to deduce this from the *state-light* of the other robots. An overview of the state transitions is shown in Figure 5.3. The transitions possible are shown in Table 5.1. The first column lists the states that a robot observes. The second column lists a possible extra requirement on the *execute-light*. Together these two columns define which transition to take. The third column is only added as reference and states the possible states a robot can be in when it decides to take the corresponding transition. The fourth column defines which new state a robot taking that transition will go to. The last column lists a possible extra action the robot will do. This can be setting the *execute-light*, doing a copy procedure which will be explained later, or executing a step in A. Take for example transition (12). A robot wakes up and the multiset of observed state lights contains only *Reset* and *Trying*. This robot will go to *Reset* and set its *execute-light* to *NotExecuted*.

The simulation can be subdivided into phases. In the first phase the robots copy the lights of their neighbours by using the *copy-lumi-lights* and *copy-execute-lights*. In the second phase some robots execute a step in LUMI algorithm A. This is now possible because they can calculate the color of their own *lumi-light* by looking at their neighbours. In the third phase the robots go back to their starting states such that they can start this cycle over again.

This is the normal behaviour of the algorithm and we will call it an *execute-cycle*. Every one of these *execute-cycles* starts and ends with the robots in the following states. Either all n robots are in state *Copy*, or $n - 1$ robots are in state *Copy* and the remaining robot is in either state *Finished* or *Trying*.

At the start of the first phase, at some point a robot will wake up and see all other robots in state *Copy*. It knows that lights need to be copied and it will go to *DoneCopy*, copying the lights of its neighbours. This corresponds to transition (1) in Table 5.1. The copying is done by first



(a) Transitions between states.

(b) Transitions between execute states.

Figure 5.3: Transitions for the states and execution states. Robots might skip a state and immediately go to the next in some situations. Moreover, robots do not know their own state and have to deduce this information from the other robots.

calculating the predecessor and successor. Next, the two *copy-lumi-lights* are set to the same color as the *lumi-light* of the corresponding neighbour. The two *copy-execute-lights* are set to the same color as the *execute-light* of the corresponding neighbour.

Now as long as robots wake up and still see another robot in state *Copy*, they will do this copy procedure (transition (2)). This goes on until at some point a robot wakes up and sees all other robots with the *state-light* set to *DoneCopy*. To indicate to all other robots that every robot has copied the lights of its neighbours, this robot goes to *Trying* (transition (3)). However, it could be that this robot itself was still in *Copy*. Therefore it still has to do the copy procedure. Now all robots see at least one other robot with state *Trying*, thus knowing they already copied the lights and also go to state *Trying* (transition (4)). The second phase will start as soon as either all robots are in state *Trying*, or $n-1$ robots are in state *Trying* and 1 robot is in state *DoneCopy*.

In the second phase, some robots will execute a step in algorithm A. At some point at the start of this phase, one or more robots will wake up and see every other robots in state *Trying*. These robots will now execute a step in A (transition (5)). They have all the information to do so, because no *lumi-light* has changed since all robots copied it at the end of the first phase. Therefore, the robot can interpret the color of its own light based on the lights of its neighbours. These robots go to *Finished*. This phase ends with some subset of the robots in state *Finished*, and the other robots in state *Trying*.

In the third phase, the robots try to reach the starting configuration of the first phase again. All robots that wake up see at least one robot in *Finished*, except in one specific situation. If in the second phase only one robot activated, this is the only robot in *Finished* and the other robots are all in *Trying*. If this robot would now activate again, it would see only other robots in state *Trying* and it would try to exeecute again. This would cause problems, as the lights are not correctly copied at this point. However, it is not allowed for this robot to be active again under STRICTERSSYNC. Therefore, all robots that wake up now see at least one other robot in state *Finished*. These robots go back to *Copy* (transition (9)). Next, all robots see robots both in state *Trying* and *Copy*. This situation informs them that some robots have activated and that they need to go back to *Copy* (transitions (10) and (11)). This goes on until at one point either all n robots are in *Copy*, or $n-1$ robots are in *Copy* and the last robot is still in *Finished* or *Trying*. This is the starting situation and the first phase will start again.

These three phases would make sure that the robots will continually be able to perform some steps in the algorithm A. However, it could be the case that every second phase, the same robots activate and perform a step in A. This would not make for a fair schedule. To combat this, the *execute-light* is used. If a robot executes a step in A in the second phase, it sets its *execute-light*

to *Executed*, signalling it has already executed a step in A (transition (5)). This information is copied in the first phase, so that next time the robots are all in state *Trying*, every robot knows the color of its own *execute-light*. It will only execute a step in A if it has not already done so. Otherwise it will just stay in *Trying* and let another robot go (transition (6)). This ensures that robots all get a turn to execute a step in A.

At one point every robot has executed a step in A and every *execute-light* will be set to *Executed*. To reset these lights a special cycle is used: the *reset-cycle*. The first phase is the same as for an *execute-cycle* as the robots do not know yet that all robots have executed. As soon as they reach the start of the second phase, the robots recognize that they all have executed and the robots first set the *execute-light* to *AllExecuted* (transition (7)). At some point some robots will wake up and see that all other robots have set their *execute-light* to the new color. These robots will go to *Reset* and reset their own *execute-light* to *NotExecuted* (transition (8)). Now they wait in *Reset* until every robot recognizes this and also goes to *Reset*, resetting their lights (transition (12)). In the end, all robots (except possibly one) are in *Reset* and have reset their light. Now the robots that wake up see every other robot in *Reset* and they will go to *Copy* (transition (14)). These robots still need to reset their light, as we do not know if they were the last robot still in *Trying*. After this, robots that wake up will see both robots in *Copy* and in *Reset* and they will also go to *Copy* (transition (13)). At one point all robots are back in *Copy* while all having their *execute-light* reset. It could be the case that there is still one robot in *Reset*, but as soon as that robot activates, it will start an *execute-cycle* again by going to *DoneCopy*.

Every *Look-Compute-Move* cycle in LUMI algorithm A corresponds to either an *execute-cycle*, or a *reset-cycle* followed by an *execute-cycle* in the FCOMM algorithm.

	Observed *state-light*	Own *execute-light*/ observed *execute-light*	Inferred own color	New State	Action
(1)	*Copy*		*Copy*, *DoneCopy*, *Finished*, *Reset*, *Trying*	*DoneCopy*	Copy neighbour lights
(2)	*Copy*, *DoneCopy*		*Copy*, *DoneCopy*	*DoneCopy*	Copy neighbour lights
(3)	*DoneCopy*		*Copy*, *DoneCopy*	*Trying*	Copy neighbour lights
(4)	*DoneCopy*, *Trying*		*DoneCopy*, *Trying*	*Trying*	
(5)	*Trying*	*NotExecuted*/ does not matter	*DoneCopy*, *Trying*	*Finished*	Execute light to *Executed* and execute a step in A
(6)	*Trying*	*Executed*, *AllExecuted*/ > 1 *NotExecuted*	*DoneCopy*, *Trying*	*Trying*	
(7)	*Trying*	*Executed*, *AllExecuted*/ no *NotExecuted*, not all *AllExecuted*	*DoneCopy*, *Trying*	*Trying*	Execute light to *AllExecuted*
(8)	*Trying*	all robots in *Trying* have *AllExecuted*	*Trying*	*Reset*	Execute light to *NotExecuted*
(9)	*Trying*, *Finished*		*Trying*, *Finished*	*Copy*	
(10)	*Copy*, *Trying*, *Finished*		*Copy*, *Trying*, *Finished*	*Copy*	
(11)	*Copy*, *Trying*		*Copy*, *Trying*	*Copy*	
(12)	*Reset*, *Trying*		*Reset*, *Trying*	*Reset*	Execute light to *NotExecuted*
(13)	*Copy*, *Reset*		*Copy*, *Reset*, *Trying*	*Copy*	Execute light to *NotExecuted*
(14)	*Reset*		*Copy*, *Reset*, *Trying*	*Copy*	Execute light to *NotExecuted*

Table 5.1: A list of all transitions possible. The first column consists of the set of *state-light* colors that a robot observes. The second column differentiates between some transitions based on the *execute-light*. It first states the color of the robots own *execute-light*, which can be inferred from its neighbours and then the observed *execute-light* of other robots. The third column is not used in the algorithm, but is only added as reference. It states in which states the robot could be before executing the transition. The fourth column is the new state after the transition. The last column indicates a possible extra action that occurs while performing the transition. This could be copying the colors of the neighbouring *lumi-light* and *execute-light* or setting its own *execute-light* to a different color.

5.2 Correctness

To prove correctness of FCOMMTOLUMI, we will first prove that every *execute-cycle* corresponds to some activation of LUMI robots executing A under SSYNC. To do this, we take two teams of robots. The first, R_{LUMI}, performs algorithm A under SSYNC. The second team, R_{FCOMM}, performs FCOMMTOLUMI(A) under STRICTERSSYNC. Both teams start in the exact same starting configuration C_0. The lights of the R_{FCOMM} robots form a valid initial state for the *execute-cycle*, i.e. all n robots are in state *Copy*, or $n-1$ robots are in state *Copy* and the remaining robot is in either state *Finished* or *Trying*. Every robot in R_{FCOMM} has a counterpart in R_{LUMI}, which is the robot on the same location. This creates pairs of robots. Every pair of robots starts with the same color for the *lumi-light*. Now R_{FCOMM} executes the three phases of an *execute-cycle* such that these robots end up in some configuration C_1.

Lemma 5.2. After execution of phase one of the *execute-cycle*, the *lumi-light* of every robot will have the same color as its R_{LUMI} counterpart, every R_{FCOMM} robot has done the copy procedure, and they are all in state *Trying*, except possibly one robot, which could still be in state *DoneCopy*.

Proof. During phase one of the *execute-cycle* the *lumi-light* of the robots will stay the same. Because the light has the same color as its R_{LUMI} counterpart at the start, the colors will still be the same at the end of phase one. Every robot, except possibly one, will go from *Copy* to *DoneCopy*, and do the copy procedure. The single robot that may skip *DoneCopy* and go to *Trying* immediately, will still do the copy procedure. Therefore every robot has set all its copy lights correctly. Every robot still in *DoneCopy* will take the transition to *Trying* such that in the end, all robots except possibly one are in *Trying*. If not all of them are in *Trying*, the last robot will still be in *DoneCopy*. □

Lemma 5.3. After execution of phase two of the *execute-cycle*, there exists a valid SSYNC activation for R_{LUMI} such that after this activation, they form C_1 and the color of the light of every robot in R_{LUMI} is the same as the color of the *lumi-light* of its R_{FCOMM} counterpart.

Proof. At the start of the second phase, all copy lights (*copy-lumi-lights* and *copy-execute-lights*) are set correctly, because in the first phase all robots have executed the copy procedure and the *lumi-light* and *execute-light* have not changed color since. Therefore each robot can calculate the color of its own *lumi-light* and its own *execute-light*. Let S be the subset of R_{FCOMM} robots that activate now and go to *Finished*. The R_{FCOMM} robots form C_1 after this activation and have possibly a different color of their *lumi-light*. Both the forming of C_1 and the changing of the *lumi-light* is determined by algorithm A. At the start of this phase, the positions of all R_{FCOMM} robots are equal to the positions of the R_{LUMI} robots. Furthermore, the *lumi-light* of every R_{FCOMM} robot has the same color as the light of its R_{LUMI} counterpart. Therefore, if we take S' to be the counterparts of S in R_{LUMI} and let these robots activate, they will calculate the same target positions and same changes of light. This is true because both sets of robots calculate according to A and the input is the same. If robots in S got halted during their movement, we also halt their counterpart in S' at the same position. This interruption is valid, because the interruption of the robot in S is valid and its counterpart in S' will have travelled the same distance. This activation of S' together with those interruptions is a valid SSYNC activation. Moreover, both teams of robots now form C_1 and the colors of the *lumi-light* of every R_{FCOMM} robot is the same as the color of the light of its counterpart in R_{LUMI}. □

Lemma 5.4. After execution of phase three of the *execute-cycle*, the R_{FCOMM} robots still form C_1, have not changed their *lumi-light* and are now either all in state *Copy*, or all in state *Copy* except one, which could be in *Finished* or *Trying*.

Proof. In the third phase, neither the color of the *lumi-light* nor the position of any R_{FCOMM} robot changes. Therefore they will stay the same, assuming they were the same at the beginning of the phase. Some robots will be in *Trying* and some will be in *Finished*. The robots that activate will take a transition to *Copy*. Now every robot that activates takes a transition to *Copy*, until all of

them are in *Copy*, or all of the robots are in *Copy* with the last one still being in either *Finished* or *Trying*. □

Lemma 5.5. For every *execute-cycle* of R_{FCOMM}, there exists a valid SSYNC activation for R_{LUMI} such that after this activation, both teams of robots form the same configuration and the color of the light of every robot in R_{LUMI} is the same as the color of the *lumi-light* of its R_{FCOMM} counterpart.

Proof. Lemma 5.2 shows that the R_{FCOMM} robots can reach the start of the second phase and that at that point all robots have done the copy procedure. The location of the robots nor the color of the *lumi-light* has changed. Lemma 5.3 shows that there is a valid SSYNC activation for R_{LUMI} that will result in the R_{LUMI} robots creating the same configuration as the R_{FCOMM} robots. Moreover, the color of the *lumi-light* of every robot in R_{FCOMM} is the same as the color of its counterpart in R_{LUMI}. Lastly, Lemma 5.4 proves the robots can reach the starting state of the next *execute-cycle*. Together, these three lemmas show that each *execute-cycle* of the R_{FCOMM} robots corresponds to a valid SSYNC activation for the R_{LUMI} robots. □

Lemma 5.5 also holds if the FCOMM robots perform a *reset-cycle* first and then an *execute-cycle*, because a *reset-cycle* does not change the location or the *lumi-light* of a robot.

Lemma 5.6. Between two *reset-cycle*s, every robot executes a step in A exactly once.

Proof. First we will prove that a robot performs at most one step in A in between *reset-cycle*s. For this purpose, assume that a robot r executes a step in A for the second time since the last *reset-cycle*. This means that the color of its *execute-light* is *NotExecuted* (discovered by looking at its neighbours). However, the first time r executed a step in A, the *execute-light* was set to *Executed*. Only an execution in A or a *reset-cycle* change the color of the *execute-light* of the robots. Therefore either r has not yet executed a step in A or a *reset-cycle* has happened since the last execution. Both contradict the assumption and therefore every robot executes at most 1 step in A in between two *reset-cycle*s.

To prove that some robot r performs at least one execution step in A in between *reset-cycle*s, assume that a new *reset-cycle* will happen while r has not executed a step in A yet. Because a new *reset-cycle* will happen, the *execute-light* of r is set to *Executed*. At the end of the previous *reset-cycle* the *execute-light* of r was *NotExecuted*. Therefore, the *execute-light* of r must have changed. However, the only time the *execute-light* of r could have changed was if it executed a step in A, contradicting the assumption.

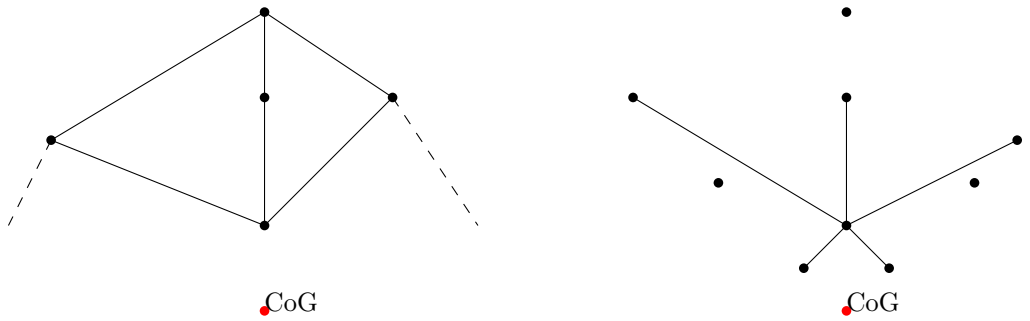
Because every robot performs at least 1 and at most 1 step in A in between two *reset-cycle*s, we can conclude that every robot executes a step in A exactly once between two *reset-cycle*s. □

Because every robot executes a step in A exactly once in between two *reset-cycles*, this simulation creates a fair LUMI schedule.

Lemma 5.7. LUMI robots under SSYNC can be simulated using FCOMM robots under the STRICTERSSYNC scheduler using $135c^3$ colors, when there are no multiplicities, there is never a robot at the center of gravity and the robots have the same chirality.

Proof. If the conversion from any FCOMM schedule to a LUMI schedule by FCOMMTOLUMI(A) gives a correct and fair schedule, this lemma holds. Every *execute-cycle* or *reset-cycle* followed by an *execute-cycle* performed by the FCOMM robots corresponds to a valid activation for the LUMI robots as stated by Lemma 5.5. Using this as an induction step and the fact that the robots start in the same location as the base case, it follows by induction that the conversion gives a correct schedule. Lemma 5.6 shows every robot activates exactly once in between two *reset-cycles*. Because there are an infinite number of *reset-cycles*, this shows the created schedule is fair. □

We will now lift both the assumption about a robot at the center of gravity and robots having the same chirality. To solve the problem when there is a robot at the center of gravity, we simply add one more *copy-lumi-light* to all robots and say that this central robot is a neighbour of every other robot, while the central robot itself does not have any neighbours. If there is no central



(a) Neighbouring relationships for coradial robots.

(b) Maximum number of neighbours is 5 when there are three coradial sets of robots next to each other. Possibly 6 with a robot at the CoG.

Figure 5.4: Neighbouring relations when there are coradial robots.

robot, this extra copy light will be set to a special color *Not used* during the copy procedure. Adding this light adds an extra factor $c + 1$ to the number of colors needed per robot. We also add an extra *copy-execute-light* to every robot, adding a factor $3 + 1 = 4$ to the number of colors per robot.

The solution would also still work when the chirality is not the same for every robot. There will be no notion of order, but every robot will still have 2 neighbours (3 when a robot is at the center of gravity). Robots will still be able to find out the color of their own *lumi-light* by doing the same calculation, but including both neighbours. For coradial robots, we let the outermost robots have the same neighbours as it would have if there was no coradiallity plus the one that it is coradial with, see Figure 5.4. This leads to a constant maximum of 6 neighbours. Again, if that many neighbours do not exist, the lights will be set to *Not used*. Every robot can now still deduce its color if the copy-lights are set correctly, while only using a constant number of lights, each with a constant number of colors. Each *copy-lumi-light* now can have one of $c + 1$ colors. Again we do the same for the *copy-execute-lights*, adding a factor 4 per neighbour. This results in the total number of colors being $c \cdot (c + 1)^6 \cdot 5 \cdot 3 \cdot 4^6 = 61440c(c + 1)^6$.

Naturally, LUMI $\geq$ FCOMM for STRICTERSSYNC schedules. We have also shown that LUMI^S robots can be simulated using FCOMM robots under the STRICTERSSYNC scheduler. We will now extend FCOMMTOLUMI even further to let the FCOMM robots simulate LUMI robots under STRICTERSSYNC. The only time a schedule will be created that does not adhere to STRICTERSSYNC is in the following scenario. Let r be the last robot that has not executed a step in A since the last *reset-cycle*. Now r will be the only robot that executes a step in A in the second phase. After this cycle finishes, the next cycle will be a *reset-cycle* and all robots will reset their *execute-light*. In the next *execute-cycle*, r can execute a step in A again, violating STRICTERSSYNC.

To avoid this situation, we add a new light to every robot, the *last-executed-light*. This light can be either *On* or *Off*. The robots need to know the status of their own *last-executed-light*. To do apply for this, we use the same solution as before. We add one *copy-last-executed-light* to each robot, which will copy the status of the *last-executed-light* of its neighbours. If any neighbours have their *last-executed-light* to *On*, the *copy-last-executed-light* will also be set to *On*. This copying will be done in the first phase of every cycle, as with all other copying lights. Now a robot r will set its *last-executed-light* to *On* whenever the above described scenario happens: Every robot has its *execute-light* set to *Executed*, except for r. Robot r will now execute, go to *Finished*. It will furthermore set its *last-executed-light* to *On*. The *last-executed-light* will be set to *Off* whenever r wants to go to *Copy* and its neighbours indicate the the *last-executed-light* of r is set to *On*.

Now whenever r was the only robot to execute before the *reset-cycle*, r will know this in the next cycle and will not execute a step in A. All robots will then go back to *Copy*, finishing the cycle. When r goes back to *Copy*, it will set its *last-executed-light* back to *Off*. This will be copied by the other robots and r will be able to execute again next cycle.

Adding these 2 extra lights to each robot, each with 2 colors, adds a factor of 4 to the number of colors. This brings the number of colors to $245760c(c+1)^6$. This number could possibly be lowered, because a robot always has at least two neighbours. This would eliminate the need for an extra color *Not used* for 4 of the *copy-lumi-lights* and *copy-execute-lights* per robot. Furthermore, not all combinations of lights are valid. For example, a robot will never be in state *DoneCopy*, while having its *execute-light* set to *AllExecuted*. However, this will be left for future work. As a result, we have the following.

Lemma 5.8. LUMI robots under STRICTERSSYNC can be simulated using FCOMM robots under STRICTERSSYNC using $245760c(c+1)^6$ colors, when there are no multiplicities.

Using Lemma 5.8 and the fact that the opposite holds by definition under any scheduler: LUMI $\geq$ FCOMM, we have the following.

Theorem 5.9. LUMI $\equiv$ FCOMM *for non-rigid robots under* STRICTERSSYNC*, when there are no multiplicities.*

Note that in FSYNC, LUMI $\equiv$ FCOMM and in SSYNC LUMI $>$ FCOMM. STRICTERSSYNC is a scheduler with weaker restrictions than FSYNC but stronger restrictions than SSYNC for which the two models are still computationally equivalent. Although we did not prove it here, we think that STRICTERSSYNC is the weakest scheduler for which this equivalence holds.

Chapter 6

Discussion

In this thesis we have investigated the computational relations between the four models Oblot, FState, FComm, and Lumi under the two synchronous schedulers FSync and SSync. We did this for both non-rigid robots as well as for rigid robots and for robots that have a common chirality as well as robots that lack such common notion of orientation. For these relations, there are still a lot of open problems. The first is how models under FSync relate to models under SSync, as we were not able to show these relations for non-rigid robots or for robots without a common chirality. Furthermore, when considering a relation between two models under the same scheduler, we only considered them both to be either rigid or non-rigid. It would be interesting to see the relations when we compare one model with rigid robots to another with non-rigid robots. The same holds for a common chirality. Lastly, we only considered models under the two synchronous schedulers SSync and FSync. The relations concerning models under the asynchronous scheduler ASync, where robots do not activate in rounds and there is no common notion of time, have not yet been investigated.

Furthermore, we have shown a scheduler with weaker restrictions than FSync but stronger restrictions than SSync for which the two models Lumi and FComm are computationally equivalent. Note that for the proof of this equivalence, we assumed that there are no multiplicities. It is not trivial to adapt this simulation procedure to one that is able to handle multiplicities and this is left for future work. This stands in contrast to some of the other relations between models, for which we used problems that do include multiplicities. Moreover, it is still an open problem if the proposed new scheduler is the weakest scheduler for which this equivalence between Lumi and FComm holds. Lastly, in the simulation procedure the focus was not on the minimum number of colors. Therefore, it could very well be possible that a procedure with a lower number of colors is viable. This holds for all algorithms presented in this thesis.

Bibliography

[1] Hideki Ando, Yoshinobu Oasa, Ichiro Suzuki, and Masafumi Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 15(5):818–828, 1999. 2

[2] Subhash Bhagat, S Gan Chaudhuri, and Krishnendu Mukhopadhyaya. Fault-tolerant gathering of asynchronous oblivious mobile robots under one-axis agreement. *Journal of Discrete Algorithms*, 36:50–62, 2016. 2

[3] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012. 2

[4] Reuven Cohen and David Peleg. Convergence of autonomous mobile robots with inaccurate sensors and movements. *SIAM Journal on Computing*, 38(1):276–302, 2008. 2

[5] Andreas Cord-Landwehr, Bastian Degener, Matthias Fischer, Martina Hüllmann, Barbara Kempkes, Alexander Klaas, Peter Kling, Sven Kurras, Marcus Märtens, Friedhelm Meyer Auf der Heide, et al. Collisionless gathering of robots with an extent. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 178–189. Springer, 2011. 1

[6] Jurek Czyzowicz, Leszek Gasieniec, and Andrzej Pelc. Gathering few fat mobile robots in the plane. *Theoretical Computer Science*, 410(6-7):481–499, 2009. 1

[7] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. The power of lights: synchronizing asynchronous robots using visible bits. In *2012 IEEE 32nd International Conference on Distributed Computing Systems*, pages 506–515. IEEE, 2012. 2

[8] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. Autonomous mobile robots with lights. *Theoretical Computer Science*, 609:171–184, 2016. 2, 6, 16

[9] Xavier Défago, Maria Gradinariu, Stéphane Messika, and Philippe Raipin-Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering. In *International Symposium on Distributed Computing*, pages 46–60. Springer, 2006. 1, 2

[10] Xavier Défago and Samia Souissi. Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity. *Theoretical Computer Science*, 396(1-3):97–112, 2008. 2

[11] Mattia D'Emidio, Daniele Frigioni, and Alfredo Navarra. Synchronous robots vs asynchronous lights-enhanced robots on graphs. *Electr. Notes Theor. Comput. Sci.*, 322:169–180, 2016. 2

[12] Giuseppe A Di Luna, Paola Flocchini, Nicola Santoro, and Giovanni Viglietta. Turingmobile: a turing machine of oblivious mobile robots with limited visibility and its applications. *arXiv preprint arXiv:1709.08800*, 2017. 2

[13] Yoann Dieudonné and Franck Petit. Swing words to make circle formation quiescent. In *International Colloquium on Structural Information and Communication Complexity*, pages 166–179. Springer, 2007. 2

[14] Yoann Dieudonné and Franck Petit. Squaring the circle with weak mobile robots. In *International Symposium on Algorithms and Computation*, pages 354–365. Springer, 2008. 2

[15] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Self-deployment of mobile sensors on a ring. *Theoretical Computer Science*, 402(1):67–80, 2008. 2

[16] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Distributed computing by mobile robots: Solving the uniform circle formation problem. In *International Conference on Principles of Distributed Systems*, pages 217–232. Springer, 2014. 2

[17] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Distributed computing by mobile robots: uniform circle formation. *Distributed Computing*, 30(6):413–457, 2017. 2

[18] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *International Symposium on Algorithms and Computation*, pages 93–102. Springer, 1999. 1

[19] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1-3):147–168, 2005. 2

[20] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1-3):412–447, 2008. 2

[21] Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Masafumi Yamashita. Rendezvous with constant memory. *Theoretical Computer Science*, 621:57–72, 2016. 2, 11

[22] Paola Flocchini, Nicola Santoro, and Koichi Wada. On memory, communication, and synchronous schedulers when moving and computing. In *23rd International Conference on Principles of Distributed Systems (OPODIS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. 2, 3, 11, 17, 19

[23] Nao Fujinaga, Yukiko Yamauchi, Hirotaka Ono, Shuji Kijima, and Masafumi Yamashita. Pattern formation by oblivious asynchronous mobile robots. *SIAM Journal on Computing*, 44(3):740–785, 2015. 2

[24] Adam Heriban, Xavier Défago, and Sébastien Tixeuil. Optimally gathering two robots. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, pages 1–10, 2018. 2

[25] Taisuke Izumi, Samia Souissi, Yoshiaki Katayama, Nobuhiro Inuzuka, Xavier Défago, Koichi Wada, and Masafumi Yamashita. The gathering problem for two oblivious robots with unreliable compasses. *SIAM Journal on Computing*, 41(1):26–46, 2012. 2

[26] Branislav Katreniak. Biangular circle formation by asynchronous mobile robots. In *International Colloquium on Structural Information and Communication Complexity*, pages 185–199. Springer, 2005. 2

[27] Marcello Mamino and Giovanni Viglietta. Square formation by asynchronous oblivious robots. *arXiv preprint arXiv:1605.06093*, 2016. 2

[28] Giuseppe Prencipe. Impossibility of gathering by a set of autonomous mobile robots. *Theoretical Computer Science*, 384(2-3):222–231, 2007. 2

[29] Samia Souissi, Xavier Défago, and Masafumi Yamashita. Using eventually consistent compasses to gather memory-less mobile robots with limited visibility. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(1):1–27, 2009. 2

[30] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots—formation and agreement problems. In *Proc. 3rd International Colloquium on Structural Information and Communication Complexity (SIROCCO'96)*, pages 313–330, 1994. 1

[31] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999. 1, 2, 11

[32] Giovanni Viglietta. Rendezvous of two robots with visible bits. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 291–306. Springer, 2013. 2

[33] Masafumi Yamashita and Ichiro Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theoretical Computer Science*, 411(26-28):2433–2453, 2010. 2