Arbitrary Pattern Formation by Asynchronous, Anonymous, Oblivious Robots*

Paola Flocchini† Giuseppe Prencipe‡ Nicola Santoro§ Peter Widmayer¶

Abstract

From an engineering point of view, the problem of coordinating a set of autonomous, mobile robots for the purpose of cooperatively performing a task has been studied extensively over the past decade. In contrast, in this paper we aim at an understanding of the fundamental algorithmic limitations on what a set of autonomous mobile robots can or cannot achieve. We therefore study a hard task for a set of weak robots. The task is for the robots in the plane to form any arbitrary pattern that is given in advance. This task is fundamental in the sense that if the robots can form any pattern, they can agree on their respective roles in a subsequent, coordinated action. The robots are weak in several aspects. They are anonymous; they cannot explicitly communicate with each other, but only observe the positions of the others; they cannot remember the past; they operate in a very strong form of asynchronicity.

We show that the tasks that such a system of robots can perform depend strongly on their common agreement about their environment, i.e., the readings of their environment sensors. If the robots have no common agreement about their environment, they cannot form an arbitrary pattern. If each robot has a compass needle that indicates North (the robot world is a flat surface, and compass needles are parallel), then any odd number of robots can form an arbitrary pattern, but an even number cannot (in the worst case). If each robot has two independent compass needles, say North and East, then any set of robots can form any pattern.

1 Introduction

1.1 The Framework

The current trend in robotic research, both from engineering and behavioural viewpoints, has been to move away from the design and deployment of few, rather complex, usually expensive, application-specific robots. In fact, the interest has shifted towards the design and use of a large number of "generic" robots which are very simple, with very limited capabilities and, thus, relatively inexpensive, but capable, together, of performing rather complex tasks.

The advantages of such an approach are clear and many, including: reduced costs (due to simpler engineering and construction costs, faster development and deployment time, etc); ease of system expandability (just add a few more robots) which in turns allows for incremental and on-demand deployment (use only as few robots as you need and when you need them); simple

*Preliminary versions of some of these results have appeared in [13, 14].

†School of Information, Technology and Engineering, University of Ottawa - flocchin@site.uottawa.ca

‡Dipartimento di Informatica - Università di Pisa - prencipe@di.unipi.it

§School of Computer Science, Carleton University - santoro@scs.carleton.ca

¶Theoretische Informatik, ETH Zürich - widmayer@inf.ethz.ch

and affordable fault-tolerance capabilities (replace just the faulty robots); re-usability of the robots in different applications (reprogram the system to perform a different task).

Leading research activities in the engineering area include the Cellular Robotic System (CEBOT) of Kawaguchi et al. [18], the Swarm Intelligence of Beni et al. [4], the Self-Assembly Machine ("fructum") of Murata et al. [20], etc. In the AI community there has been a number of remarkable studies, e.g., on social interaction leading to group behavior by Matarić [19], on selfish behavior of cooperative robots in animal societies by Parker [21], on primitive animal behavior in pattern formation by Balch and Arkin [3], to cite just a few. An investigation with an algorithmic flavor has been undertaken within the AI community by Durfee [12], who argues in favor of limiting the knowledge that an intelligent robot must possess in order to be able to coordinate its behavior with others.

A group of mobile autonomous robots, each with very limited capabilities, can form (complex) patterns in the space it occupies. The basic research questions are which patterns can be formed, and how they can be formed. These questions have been studied mostly from an empirical point of view, with no actual proofs of correctness. Actually, many solutions do not terminate and they never form the desired pattern (the robots just converge towards it); such solutions are called "*convergence*".

We are interested in provably correct "algorithmic" solutions, which, if possible, form the pattern, and the conditions under which they exist. The underlying research goal is to understand what kind of basic capabilities a set of robots must have in order to accomplish a given task in a distributed fashion. By assuming the "weakest" robots, it is possible to analyze in greater detail the strengths and weaknesses of distributed control; furthermore, this approach allows to highlight the set of robots' capabilities that are necessary to accomplish a certain task.

This approach has been first employed in the investigations of Suzuki and Yamashita [26, 28], and with their collaborators [2, 25]; their algorithmic focus is a rarity in the mobile robots literature. They have given an elegant and systematic account on the algorithmics of pattern formation for robots, under several assumptions on the power of the individual robots. They consider rather weak robots, which are identical, without any central control, execute the same deterministic algorithm, and do not have any explicit communication mechanism. The life of a robot is a sequence of *cycles*; in each cycle, the robot observes the positions of the fellow robots, computes a destination point according to the algorithm, and moves towards such a point. It is however assumed that the robots actions (including movement) are atomic and instantaneous. Their work has inspired and motivated several other investigations, including those by Agmon and Peleg [1], Défago and Konagaya [10], as well as our own investigations [6, 15].

1.2 The Problem and Existing Solutions

In this paper, we concentrate on the particular coordination problem that requires the robots to form a specific but arbitrary geometric pattern, the Arbitrary Pattern Formation problem (shortly Apf problem); a pattern is a set of points (given by their Cartesian coordinates) in the plane and it is known initially by all robots in the system. As an instance of this problem, we might require the robots to place themselves on the circumference of a circle, with equal spacing between any two adjacent robots, just like kids in the kindergarten are sometimes requested to do. We do not prescribe the position of the circle in the world, and we do not prescribe the size of the circle, just because the robots do not have a notion of the world coordinate system's origin or unit of length. Initially, the robots are in arbitrary positions, with the only requirement that no two robots are in the same position, and that of course the number of points prescribed in the pattern and the number of robots are the same. The robots are said to *form the pattern*, if the

actual positions of the robots "coincide" with the points of the pattern, where the pattern may be *translated*, *rotated*, *scaled*, and *flipped* into its mirror position in each local coordinate system. This problem has been extensively investigated in the literature, mostly as an initial step that lets the robots proceed in the desired formation [3, 16] (just like a flock of birds or a troupe of soldiers); it is interesting algorithmically, because if the robots can form any pattern, they can agree on their respective roles in a subsequent, coordinated action. The APF problem includes as special cases many coordination problems, such as *leader election*: we just define a pattern with a uniquely distinguished point; whoever occupies that position becomes the leader. Thus, studying the solvability of the APF problem means to investigate what coordination problems can be solved, and under what conditions. The only means for the robots to coordinate is the observation of the others' positions; therefore, the only means for a robot to send information to some other robot is to move and let the others observe (reminiscent of bees in a bee dance).

This problem has been investigated in [28], where a complete characterization of the class of formable patterns has been provided for the class of autonomous and anonymous robots that, in addition to being capable of instantaneous and atomic actions, have an unbounded amount of memory (i.e., they are non-oblivious).

These two robots' capabilities, *instantaneous* and *atomic* actions, and *unbounded non-obliviousness* are rather powerful; furthermore, they are possibly unfeasible in real systems. This motivates our study of the APF problem by robots that do not have those capabilities.

1.3 Our Contribution

We investigate the solvability of the APF problem by a weaker class of robots than those of [26, 27, 28]. In fact, the robots we consider are *fully asynchronous*: any robots' action takes a finite but unpredictable amount of time; and they are *totally oblivious*: the robots do not remember results from any of the previous computations.

These two weaknesses have radical computational consequences. For instance, full asynchronicity implies that, since actions are not instantaneous, while a robot is computing the others might move; hence, by the time the computation ends, the resulting movement might not be "coherent" with the current configuration. It also implies that a robot can be seen by other robots *while* it is moving[1].

We give a characterization of what can and what cannot be achieved by this class of robots. We show that the patterns that can be formed depend strongly on the level of a priori agreement the robots have about the *orientation* and *direction* of the axes in their local coordinate systems.

First, we show that if the robots have *no agreement* on the direction and orientation of the axes, the APF problem is unsolvable; that is there are patterns that can not be formed, regardless of the algorithm, from some initial configurations of the robots. Here, agreement on the direction of the x axis means that all robots know and use the fact that all the lines identifying their individual x axes are parallel. Similarly, agreement on the orientation of an axis means that the positive side of that axis in the local coordinate systems coincides for all robots.

We then show that if the robots agree on the direction and the orientation of both axes (a situation we shall call *total agreement*), the pattern formation problem is always solvable and the proof is constructive. Note that agreement on the directions and orientations of both axes does not imply agreement on the origin or the unit of length.

[1]Note that this does not mean that the observing robot can distinguish a moving robot from a non moving one.

Finally, we study the case when the robots agree only on the direction and orientation of only one axis (a situation we shall call *partial agreement*). We show that, with partial agreement, the APF problem can be solved whenever the number of robots is odd, and that it is in general unsolvable when the number of robots is even. Also in this case, the proof of possibility result is constructive. When the system is populated by an even number of robots, as mentioned, not all patterns are formable; we fully characterize the patterns that can be achieved in this case, and provide an algorithm that allows the robots to do so.

The paper is organized as follows. In Section 2 the formal definition of the model under study is presented; furthermore, we review the state of the art with respect to the analysis of the limitations to pattern formation by autonomous mobile robots. In Section 3 we present and solve a preliminary problem that will be useful to present the main results of this paper. In Sections 4–7 we present the formal proof of the limitations and the algorithms for the problem. Finally, in Section 8 we draw some conclusions and present suggestions for further study.

2 Definition and Properties

2.1 The Model

We study the problem of coordinating a set of *autonomous*, mobile robots in the plane. The coordination mechanism must be totally *decentralized*, without any central control. The robots are *anonymous*, in the sense that a robot does not have an identity that it can use in a computation, and all robots execute the exact same algorithm. Each robot has its own, *local view* of the world. This view includes a local Cartesian coordinate system with origin, unit of length, and the *directions* of two coordinate axes, identified as x axis and y axis, together with their *orientations*, identified as the positive sides of the axes. The robots do not have a common understanding of the *handedness (chirality)* of the coordinate system that allows them to consistently infer the orientation of the y axis once the orientation of the x axis is given; instead, knowing North does not distinguish East from West. The robots observe the environment and move; this is their only means of communication and of expressing a decision that they have taken. The only thing that a robot can do is make a *step*, where a step is a sequence of three actions. First, the robot observes the positions of all other robots with respect to its local coordinate system. Each robot is viewed as a point, and therefore the *observation* returns a set of points to the observing robot. The robot cannot distinguish between its fellow robots; they all look identical. Second, the robot performs an arbitrary *local computation* according to its algorithm, based only on the *common knowledge* of the world (assumed e.g. to be stored in read-only-memory and to be read off from sensors of the environment) and the observed set of points. Since the robot does not memorize anything about the past, we call it *oblivious*. For simplicity, we assume that the algorithm is *deterministic*, but it will be obvious that all of our results hold for nondeterministic algorithms as well (randomization, however, makes things different). Third, as a result of the computation, the robot either stands still, or it moves (along any curve it likes). The *movement* is confined to some (potentially small) unpredictable, nonzero amount. Hence, the robot can only go towards its goal along a curve, but it cannot know a priori how far it will come in the current step. While it is on its continuous move, a robot may be seen an arbitrary number of times by other robots, even within one of its steps.

The robots are *fully asynchronous*: the amount of time spent in observation[2], in computation, in movement, and in inaction is finite but otherwise unpredictable. In particular, the robots

[2] i.e., activating the sensors and receiving their data.

do not (need to) have a common notion of time. Each robot makes steps at unpredictable time instants. The (global) time that passes between two successive steps of the same robot is finite; that is, any desired finite number of steps could have been made by any robot after some finite amount of time. In addition, we do not make any timing assumptions within a step: The time that passes after the robot has observed the positions of all others and before it starts moving is arbitrary, but finite. That is, the actual move of a robot may be based on a situation that lies arbitrarily far in the past, and therefore it may be totally different from the current situation. We feel that this assumption of *asynchronicity within a step* is important in a totally asynchronous environment, since we want to give each robot enough time to perform its local computation.

2.2 The Computational Cycle

The robots execute the same deterministic algorithm, which takes as input the observed positions of the robots within the visibility radius, and returns a destination point towards which the executing robot moves.

A robot is initially in a *waiting* state (*Wait*); asynchronously and independently from the other robots, it *observes* the environment in its area of visibility (*Look*); it *calculates* its destination point based only on the observed locations of the robots in its (*Compute*); it then *moves* towards that point (*Move*); after the move it goes back to a waiting state.

The sequence: *Wait - Look - Compute - Move* will be called a *computation cycle* (or briefly *cycle*) of a robot.

The operations performed by the robots in each state will be now described in more details.

1. **Wait** The robot is idle. A robot cannot stay infinitely idle (see Assumption A1 below).

2. **Look** The robot observes the world by activating its sensors which will return a snapshot of the positions of all other robots with respect to its local coordinate system (since robots are viewed as a point, their positions in the plane is just the set of their coordinates). This snapshot will be called the *view of the world*.

3. **Compute** The robot performs a *local computation* according to its deterministic, oblivious algorithm. The result of the computation is a destination point; if this point is the current location, the robot stays still (*null movement*),

4. **Move** The robot moves towards the computed destination; this operation can terminate before the robot has reached it[3]. The movement can not be infinite, nor infinitesimally small (see Assumption A2 below).

Note that we do not require the robots to be able to detect *multiplicity* (i.e. whether there is more than one robot on any of the observed points, included the position where the observing robot is. In the model, there are only two limiting assumptions about time and space. The first refers to the length of a computational cycle.

Assumption A1(Computational Cycle) *The amount of time required by a robot r to complete a computational cycle is neither infinite nor infinitesimally small.*

In particular, there exists a constant $\epsilon_r > 0$ such that, if the destination point is not reached during the cycle, the cycle will require at least ϵ_r time.

[3]e.g. because of limits to the robot's motorial autonomy.

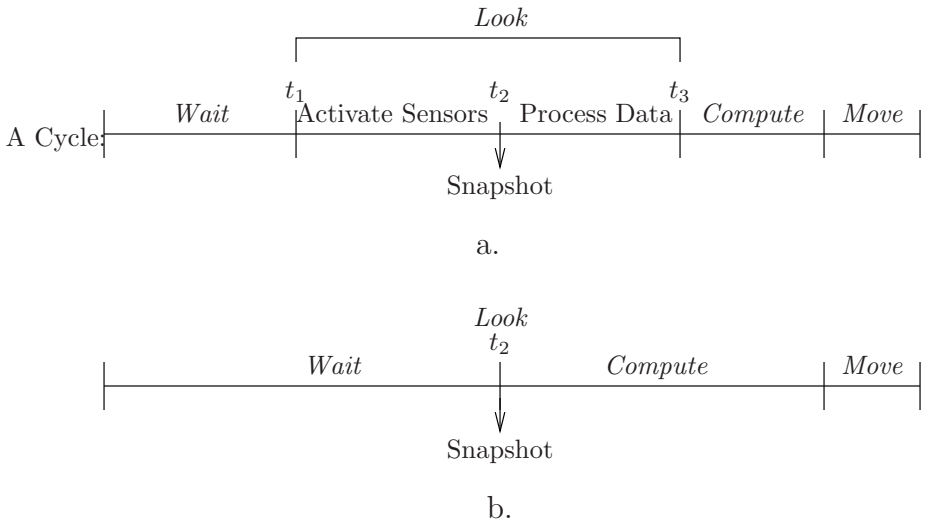

Figure 1: (a) An example of a cycle. In particular, the three parts that constitute the *Look* state (between time t_1 and t_3) are put in evidence. (b) For simplicity reasons, we can assume that the *Look* state is constituted only by the instantaneous snapshot.

As no other assumption on time exists, the resulting system is truly *asynchronous* and the duration of each activity (or inactivity) is unpredictable. As a result, the robots do not have a common notion of time, robots can be seen while moving, computations can be made based on obsolete observations.

The second assumption in the model refers to the distance traveled by a robot during a computational cycle.

Assumption A2 (Distance) *The distance traveled by a robot r in a move is neither infinite nor infinitesimally small.*

In particular, there exists an (arbitrarily small) constant $\delta_r > 0$, such that if the destination point is closer than δ_r, r will reach it; otherwise, r will move towards it of at least δ_r.

As no other assumptions on space exists, the distance traveled by a robot in a cycle is unpredictable. In the following, we shall use $\delta = \min_r \delta_r$.

Only one remark regarding the *Look* state. As already stated, the result of this state is a set of positions retrieved at one time instant, i.e. at the time when the snapshot of the world was done. That is, each *Look* can be split in three parts: in the first part the sensors are activated; in the second part the actual snapshot is performed; in the last part, the data captured by the sensors are sent away in order to be processed. For instance, referring to the cycle depicted in Figure 1.a, the first part of the *Look* is executed between time t_1 and t_2, the snapshot is done at time t_2, and the third part is executed between time t_2 and t_3. In the following, we will assume that the first and third part have null length. This is not a loss of generality: in fact, the first part can be thought to be part of the previous *Wait* state, and the third part of the following *Compute* state (as shown in Figure 1.b). Therefore, each *Look* coincides with the snapshot. According to this assumption, if r is executing a *Look* at time t, then its view of the world is the snapshot retrieved at t.

2.3 The Arbitrary Pattern Formation Problem

We study the problem of forming an arbitrary geometric pattern, where a pattern $\mathbb{P}$ is a set of points $p_0, \ldots, p_n$ (given by their Cartesian coordinates) in the plane. The pattern is known initially by all robots in the system. Initially, the robots are in arbitrary positions, with the only requirement that no two robots be in the same position, and that, of course, the number of points prescribed in the pattern and the number of robots are the same.

Let a *configuration (of the robots)* at time t, denoted by $\mathbb{D}_t$, be a set of robots' positions at time t, one position per robot, with no position occupied by more than one robot; in particular, $\mathbb{D}_0$ denotes the configuration of the robots at the beginning of the computation, at time t_0. Given a pattern $\mathbb{P}$ and a configuration $\mathbb{D}_f$, the robots are said to *have formed* $\mathbb{P}$ at time f, if there exists a transformation $\mathcal{T}$, where $\mathcal{T}$ can be an arbitrary sequence of *translation*, *rotation*, *scaling*, or *flipping* into mirror position, such that, $\mathcal{T}(\mathbb{D}_f) = \mathbb{P}$: in other words, the final positions of the robots must coincide with the points of the input pattern, where the formed pattern may be *translated*, *rotated*, *scaled*, and *flipped* into its mirror position with respect to the input pattern $\mathbb{P}$ in each local coordinate system. In this case, we say that $\mathbb{D}_f$ is a *final configuration* for $\mathbb{P}$ Given an arbitrary initial configuration and an arbitrary pattern $\mathbb{P}$, a *pattern formation algorithm* is a deterministic algorithm that brings the robots in the system to a final configuration for $\mathbb{P}$ in a finite number of cycles. We say that a pattern formation algorithm is *collision-free*, if, at any point in time t, there are no two robots that occupy the same position in the plane at t.

2.4 Basic Limitations and Relationships

Another problem that we will refer to in the following is the *leader election* problem: the robots in the system are said to elect a leader if, after a finite number of cycles, all the robots deterministically agree on (i.e., choose) the same robot l, called the leader. A deterministic algorithm that lets the robots in the system elect a leader in a finite number of cycles, given any initial configuration, is called a *leader election algorithm*.

The relationship between the pattern formation problem and the leader election problem, is stated in the following

Theorem 2.1. *If it is possible to solve the pattern formation problem for $n \geq 3$ robots, then the leader election problem is solvable too.*

Proof. Let $\mathcal{A}$ be a pattern formation algorithm. Let $\mathbb{P}$ be a pattern defined in the following way:

1. All the robots but one are evenly placed on the same line l; the distance between two adjacent robots is d; and
2. the last robot is on l, but the distance from its unique adjacent robot is $2d$.

After all the robots execute $\mathcal{A}$ to form $\mathbb{P}$, the unique robot that has only one neighbor, and whose distance from it is $2d$, is identified as the leader. □

We note that, since rotation is allowed, two robots always form the desired pattern. Therefore we will assume to have at least 3 robots in the system.

We will now show that in general, the leader election problem is deterministically unsolvable. In particular, the following lemma states its unsolvability under the assumptions of an even number of robots in the system.

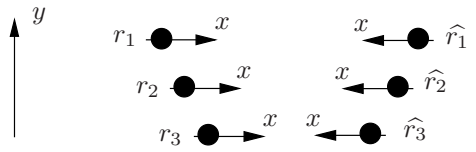

Figure 2: Specular configuration for proof of Theorem 2.2, where each r_i has the same view of the world of $\widehat{r_i}$, for $i = 1, 2, 3$.

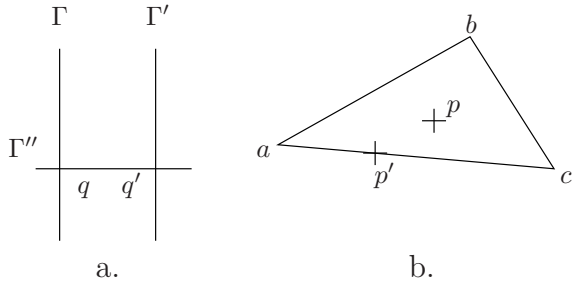

Figure 3: Notation used in the paper. (a) Horizontal distance between Γ and Γ′. (b) Given the triangle $\triangle(a, b, c)$, we have $p \in \triangle(a, b, c)$ and $p' \in \triangle(a, b, c)$.

Theorem 2.2. *There exists no deterministic algorithm that solves the leader election problem, when n is even.*

Proof. By contradiction, let $\mathcal{A}$ be a deterministic leader election algorithm. Consider an initial placement of the robots symmetric with respect to a vertical axis; i.e., each robot r has a *specular partner* $\widehat{r}$. In addition, let the local coordinate systems be specular with respect to the symmetry axis: the directions of the x axis of r and the x axis of $\widehat{r}$ are opposite; thus the view of the world is the same for r and $\widehat{r}$ (see Figure 2). In this setting, at time $t = 0$, both r and $\widehat{r}$ are in the same state; i.e., $\sigma(r, 0) = \sigma(\widehat{r}, 0)$. Consider now a semi-synchronous scheduler: robots are activated at discrete time instants; each robot is activated infinitely often; an active robot performs its operations instantaneously. Additionally, if a robot r is activated at time $t \geq 0$, the scheduler will activate at that time also $\widehat{r}$. As a consequence, if $\sigma(r, t) = \sigma(\widehat{r}, t)$, since the two robots execute the same protocol $\mathcal{A}$, their next state will still be the same: if r moves to d, $\widehat{r}$ moves to the point $\widehat{d}$ specular to d with respect to the symmetry axis. In other words, in this execution of protocol $\mathcal{A}$, $\sigma(r, t) = \sigma(\widehat{r}, t)$ for all $t \geq 0$. On the other hand, since $\mathcal{A}$ is an election protocol, it must exist a time $t' > 0$ such that a robot, say r' becomes leader. Since the leader is unique, $\sigma(r', t') \neq \sigma(r, t')$ for all $r \neq r'$, contraddicting the fact that $\sigma(r', t') = \sigma(\widehat{r'}, t')$. □

Corollary 2.1. *In a system with $n > 2$ anonymous robots that agree only on one axis direction and orientation, the pattern formation problem is unsolvable when n is even.*

Proof. It follows from Theorems 2.1 and 2.2. □

2.5 Notation

In this section, we introduce the notation that will be used through the paper. In general, r indicates a robot in the system, and $r.x(t)$ and $r.y(t)$ the coordinates of robot r at time t; r is

used also to represent the point in the plane occupied by that robot. In the following, when no ambiguity arises, the time reference will be omitted. Capital calligraphic letters (e.g. $\mathcal{Z}$) indicate regions; given a region $\mathcal{Z}$, we denote by $|\mathcal{Z}|_t$ the number of robots in that region at time t. In particular, $\mathcal{C}$ denotes a circle. Given a circle $\mathcal{C}$ with center c and radius *Rad*, and a robot $r(t)$, we will say that $r(t)$ *is on* $\mathcal{C}$ if $dist(r(t), c) = Rad$, where $dist(a, b)$ denotes the Euclidean distance between point a and b (i.e., r is on the circumference of $\mathcal{C}$); if $dist(r, c) < Rad$, we will say that r *is inside* $\mathcal{C}$.

Double lined letters (e.g., $\mathbb{Z}$) indicates sets of points, sets of robots' positions in the plane, and set of robots. In particular,

1. $\mathbb{W}(t)$ denotes the set of robots that are in *Wait* at time t;
2. $\mathbb{L}(t) = \mathbb{L}_\emptyset(t) \cup \mathbb{L}_+(t)$, is the set of robots that at time t are in state *Look*. The set $\mathbb{L}_\emptyset(t)$ contains those robots whose computation's result in their next *Compute* state is a *null movement*, while $\mathbb{L}_+(t)$ contains those robots whose computation's result in their next *Compute* is some destination point different from the position where the observing robot is (we say that they will execute a *real movement*).
3. $\mathbb{C}(t) = \mathbb{C}_\emptyset(t) \cup \mathbb{C}_+(t)$, is the set of all the robots that at time t are in state *Compute*. The set $\mathbb{C}_\emptyset(t)$ contains those robots whose computation's result is a *null movement*, while $\mathbb{C}_+(t)$ contains those robots whose computation's result is a *real movement*.
4. $\mathbb{M}(t) = \mathbb{M}_\emptyset(t) \cup \mathbb{M}_+(t)$ is the set of all the robots that at time t are executing a movement. The set $\mathbb{M}_\emptyset(t)$ contains the robots executing a *null movement* (they stay still); $\mathbb{M}_+(t)$ contains those executing a *real movement* (they are effectively moving towards a destination).

Finally, we define a particular set of robots, $\mathbb{I}(t)$, that will be useful in order to analyze the behavior of the robots while executing the algorithms studied in the following. Namely,

5. $\mathbb{I}(t) = \mathbb{W}(t) \cup \mathbb{L}(t) \cup \mathbb{C}_\emptyset(t) \cup \mathbb{M}_\emptyset(t)$,

which contains the *immobile* robots: those robots that at time t are not moving and, if computing, will not move in the current cycle. Lines, half-lines and segments will be denoted by capital greek letters (e.g., Ψ, Ξ); given a line Ψ, we denote by $|\Psi|$ the number of robots on that line. In particular, given two distinct points a and b, $[ab)$ denotes the half-line that starts in a and passes through b; $[ab]$ the segment between a and b. Moreover, given two distinct parallel lines Γ and Γ', and a line Γ'' orthogonal to Γ and Γ', we define the *horizontal distance* between Γ and Γ', denoted by $\overline{\Gamma\Gamma'}$, as the length of the segment $[qq']$, with $q = \Gamma \cap \Gamma''$ and $q' = \Gamma' \cap \Gamma''$ (see Figure 3.a). Furthermore, given a point p, we define the horizontal distance of p from Γ, denoted by $\overline{p\Gamma}$, as the horizontal distance between Γ and the line passing through p and parallel to Γ.

A *vertical stripe* is a region of the plane delimited by two distinct vertical *borders*. A border can be a vertical line, a vertical half-line, or a vertical segment, and it belongs to the vertical stripe it delimits. If a point p is between or on one of the two borders, then p is said to be *inside* the vertical stripe. We say that a point p is *strictly inside* a vertical stripe, if p is between the two borders, but not on one of them. If all the positions occupied by a robot r during a *Move* are always inside the stripe, we say that r moves inside it; similarly, if the positions are strictly inside the stripe, we say that r moves strictly inside it.

With $\triangle(a, b, c)$ we define the triangle having as vertices points a, b and c, and by $p \in \triangle(a, b, c)$ we denote that the point p is either inside or on one of the sides of the triangle (see Figure 3.b).

Finally, in the code of the algorithms

- **EndC**. denotes a robot that ends its *Compute* state;
- `destination`:= p; assign to the calling robot p as its destination for the next *Move*. If p is `null`, then the robot will not move.

3 A Preliminary Problem: "Go To Point"

Before discussing the solutions for the arbitrary pattern formation, we first introduce and solve a problem that will be useful in the following: the "Go To Point" problem (GTP).

3.1 The GTP Problem

Roughly speaking, consider an area of the plane that contains a set of robots, a set of targets, and a set of obstacles. We want a single robot to reach a target *safely*, i.e. avoiding collisions with its fellow robots and with the obstacles present in the environment.

More precisely, let $\mathcal{V}$ be a vertical stripe with borders Γ and Γ', and $\mathbb{FR}(\tilde{t})$ be the set of robots that at a given time $\tilde{t}$ are inside $\mathcal{V}$. Furthermore, let $\mathbb{O}$ be a finite set of static *obstacles* (points) in $\mathcal{V}$; and $\mathbb{FT}$ be a set of targets (points) inside $\mathcal{V}$, with $|\mathbb{FR}| \neq \emptyset$ and $|\mathbb{FT}| \neq \emptyset$, such that no point in $\mathbb{FT}$ is occupied by a robot in $\mathbb{FR}$ or by an obstacle in $\mathbb{O}$.

The GTP problem is defined as follows:

> One of the robots in $\mathbb{FR}$, say r, has to move towards one of the targets in $\mathbb{FT}$, say p, in such a way that r stays always inside $\mathcal{V}$ avoiding collisions; furthermore, all other robots — i.e. those in $\mathbb{FR} \setminus \{r\}$ — do not move until r reaches p,

subject to the following conditions:

C0. all robots in $\mathbb{FR}$ are in $\mathbb{I}(\tilde{t})$;

C1. there is total agreement on the coordinate system among the robots in $\mathbb{FR}$.

C2. for any $t' \geq \tilde{t}$ such that r is not on p at t', $|\mathbb{FR}(t')| = |\mathbb{FR}(\tilde{t})|$ (i.e., no *extra* robots enter the vertical stripe $\mathcal{V}$).

Without loss of generality, we will assume that Γ is to the left of Γ', according to the agreement assumed in C0.

An algorithm that solves this problem is `CloseToDestination`($\mathbb{FR}, \mathbb{FT}, \mathbb{O}, \Gamma, \Gamma'$), reported as Algorithm 1.

The idea behind `CloseToDestination()` is as follows. First it chooses the robot r in the set $\mathbb{FR}$ that has the minimum Euclidean distance from a point in the set $\mathbb{FT}$; in other words, it chooses the unique pair (r, p) such that

$$dist(r, p) = \min_{\substack{r' \in \mathbb{FR} \\ p' \in \mathbb{FT}}} dist(r', p'), \tag{1}$$

where ties are broken by choosing the lexicographically largest pair, according to the common coordinate system (Condition C1. above; this pair is returned by routine `Minimum()` on Line 1).

Algorithm 1 `CloseToDestination(`$\mathbb{FR}, \mathbb{FT}, \mathbb{O}, \Gamma, \Gamma'$`)`

(r, p) := `Minimum(`$\mathbb{FR}, \mathbb{FT}$`);`% Unique since ties are broken by lexicographic order%
If I Am Not r **Then** `destination:=null`; **EndC**.
Else %I Am r%
  If No Obstacle Is On The Line Passing Through r And p **Then**
5: `destination:=` p; **EndC**.
  Else
    $SafeT$:= Voronoi Cell Of p in $\mathbb{FT}$;
    $SafeR$:= Circle Centered In p With Radius $[p, r]$;
    $Safe := SafeT \cap SafeR$;
10: d := Point Strictly Inside $Safe$ Not Occupied By Any Obstacle, And Such That No Obstacle Is On $[dp]$;
    `destination:=` d; **EndC**.

Then, if no robot is on the line segment through r and p, the algorithm simply allows r to move towards p, while all other robots stay still (Lines 2 and 5). Otherwise, it finds an alternative path for r towards p so that the invariant that r is the robot in $\mathbb{FR}$ "closest" to a target in $\mathbb{FT}$ is maintained, and collisions avoided. In particular, the routine locates two *safe* regions of the plane.

The first one, $SafeT$, is the Voronoi cell of p in the set of all target points (Line 7). If r moves inside this region, it can not get closer to any other point in $\mathbb{FT}$ except for p; that is, if r moves inside $SafeT$, it will not change its target destination while moving.

The second one, $SafeR$, is the circle around center p with radius $[p, r]$ (Line 8). If r moves inside this region, no other robot in $\mathbb{FR}$ can get closer to p than r; that is, if r moves inside $SafeR$, it will be always the closest robot to p among the robots $\mathbb{FR}$ during its movement.

Algorithm `CloseToDestination()` moves r always strictly inside $SafeT \cap SafeR$: in this way, r stays the robot in $\mathbb{FR}$ "closest" to p (as long as it does not reach p), and collisions are avoided (Lines 9–11).

Some simple facts follow immediately.

Observation 3.1.

1. Since $SafeT$ and $SafeR$ are both convex, their intersection $Safe = SafeT \cap SafeR$ is convex. Furthermore, $Safe$ is never empty.

2. By construction, in $SafeR$ (circle centered in p and having radius $[pr]$) there are no robots in $\mathbb{FR} \setminus \{r\}$. Therefore, by definition of $SafeT$, in $Safe$ there are no robots from $\mathbb{FR} \setminus \{r\}$, and r is closest to p than to any other target in $\mathbb{FT} \setminus \{p\}$.

3. $p \in SafeR$ (as its center) and clearly it is in its Voronoi cell.

4. $r \in SafeT$ (by Equation (1)) and clearly it is in $SafeR$ (more precisely, on its boundary).

Furthermore, since there is only a finite number of obstacles

Observation 3.2. There exists a point d strictly inside $Safe$ such that there are no obstacles on $[r, d] \cup [d, p]$.

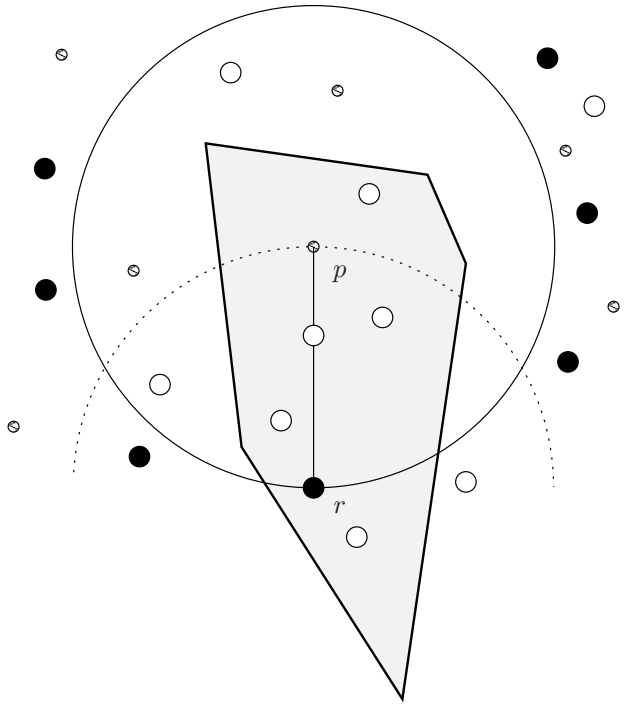

Figure 4: The circle centered in p and having radius $[p, r]$ is *SafeR*. The grey filled region is *SafeT*. The black circles are the robots in $\mathbb{FR}$, the white circles the obstacle, and the small filled circles the targets in $\mathbb{FT}$.

3.2 Correctness of Algorithm `CloseToDestination()`

We aim to show that if $(r(\tilde{t}), p(\tilde{t}))$ is the pair that satisfies Equation (1) at time $\tilde{t}$ with respect to a set of robots $\mathbb{FR}$ and a set of points in the plane $\mathbb{FT}$, then the execution of routine `CloseToDestination(`$\mathbb{FR}, \mathbb{FT}, \mathbb{O}, \Gamma, \Gamma'$`)` at time $\tilde{t}$ lets in a finite number of cycles r reach p, and no other robot in $\mathbb{FR}(\tilde{t})$ is allowed to move until this happens, and any collision is avoided.

The following theorem states that, if no obstacle is between r and p, then r reaches p in a finite number of cycles, while all other robots do not move.

Theorem 3.1. *Let conditions C0–C2 hold, and no obstacle be between r and p on $[pr]$ at time $\tilde{t}$. Then, in a finite number of cycles, say at time $t' > \tilde{t}$, r will reach p avoiding collisions. If $[pr] \subseteq \Gamma$ or $[pr] \subseteq \Gamma'$, then r moves always on one of the two borders of $\mathcal{V}$ between time $\tilde{t}$ and t'; otherwise, r moves always strictly inside $\mathcal{V}$ between time $\tilde{t}$ and t'. Furthermore, all robots in $\mathbb{FR}(\tilde{t}) \setminus \{r\}$ are in $\mathbb{I}(t)$, for all $\tilde{t} \leq t \leq t'$, and $r \in \mathbb{I}(t')$.*

Proof. According to `CloseToDestination()`, and by conditions C0–C2, r is the only robot in $\mathbb{FR}$ allowed to move at time $\tilde{t}$. Furthermore, r moves straight towards p, and since no robot is on its way, any collision is avoided. During this movement, it follows by Equation (1) and Observation 3.1 that r remains the closest robot to p. Furthermore, during its movement towards p, there is no other target in $\mathbb{FT}$ that can become closer to r than p (by definition of *SafeT*), and the theorem follows. □

The following theorem deals with the case in which there is an obstacle between r and p on $[pr]$. Also in this case, no robot is allowed to move until r reaches p, and r reaches p in a finite number of cycles avoiding collisions.

Theorem 3.2. *Let conditions C0–C2 hold, and an obstacle be between r and p on $[pr]$. Then, in a finite number of cycles, say at time $t' > \tilde{t}$, r will reach p, avoiding collisions and moving always strictly inside $\mathcal{V}$. Furthermore, all the robots in $\mathbb{FR}(\tilde{t}) \setminus \{r\}$ are in $\mathbb{I}(t)$, for all $\tilde{t} \leq t \leq t'$, and $r \in \mathbb{I}(t')$.*

Proof. In the hypotheses of the theorem, `CloseToDestination()` computes as destination point for r a point d inside *Safe* (Line 10). At time $t_m > \tilde{t}$, r starts moving towards d. By Observation 3.1, while r moves towards d, r is always inside *Safe*; hence, p remains the point in $\mathbb{FT}$ closest to r and r remains the robot in $\mathbb{FR}$ closest to p, and, by C0–C2, (r, p) is the only pair that satisfies Equation (1) (that is, r is the only robot in $\mathbb{FR}$ allowed to move by `CloseToDestination()` until it reaches d). In a finite number of cycles, r reaches d. When on d, r is still the only robot allowed to move by `CloseToDestination()`. By the way d has been chosen, no obstacle is on $[dp]$; therefore, by Theorem 3.1, the theorem follows. □

4 Total Agreement on Coordinate Systems

in this section we consider the case when there is total agreement on the directions and orientations of both axes; however this does not imply agreement on the origin and the unit of distance. We show that the robots can form an arbitrary given pattern.

First, each robot establishes the (lexicographic) total order of the points of the local pattern (Figure 5.a).

Second, each robot establishes the (lexicographic) total order of the robots' positions retrieved in the last *Look* (Figure 5.b). As we will see, this order will be the same for all robots.

Third, the first and second robots move to the positions matching the first and second pattern points. This movement can be performed in such a way that the order of the robots does not change (Figure 5.c and d). Once this is done, the first two robots' positions will determine the translation and scaling of the pattern (Figure 5.e).

Fourth, all the other robots go to points of the pattern. This can be done by moving the robots sequentially to the pattern's points. The sequence is chosen in such a way to guarantee that, after one robot has made even only a small move towards its destination, no other robot will move before that one has reached its destination (Figure 5.f).

We note that the final positions of the robots are not rotated w.r.t. the input positions; in other words the algorithm keeps the "orientation" given by the input pattern. Moreover, in this case Theorem 2.1 holds also for $n = 2$, since the rightmost and topmost robot in the system can always be identified as the leader.

The algorithm (called COMPLETE, and whose pseudo-code is reported in Algorithm 2) calls routines `Angle()`, `Sort()`, `GoIntoPosition()`, `FindFinalPositions()`, whose behavior is described in the following, as well as `CloseToDestination(())`, described in Section 3.

`Angle`(p, q) computes the clockwise angle between the horizontal axis passing through p and the segment $[pq]$.

`Sort()` gives a lexicographic order to all the positions of the robots in the system observed in the last *Look*, including the robot calling the procedure, say from left to right and from bottom to top; it returns the sorted sequence.

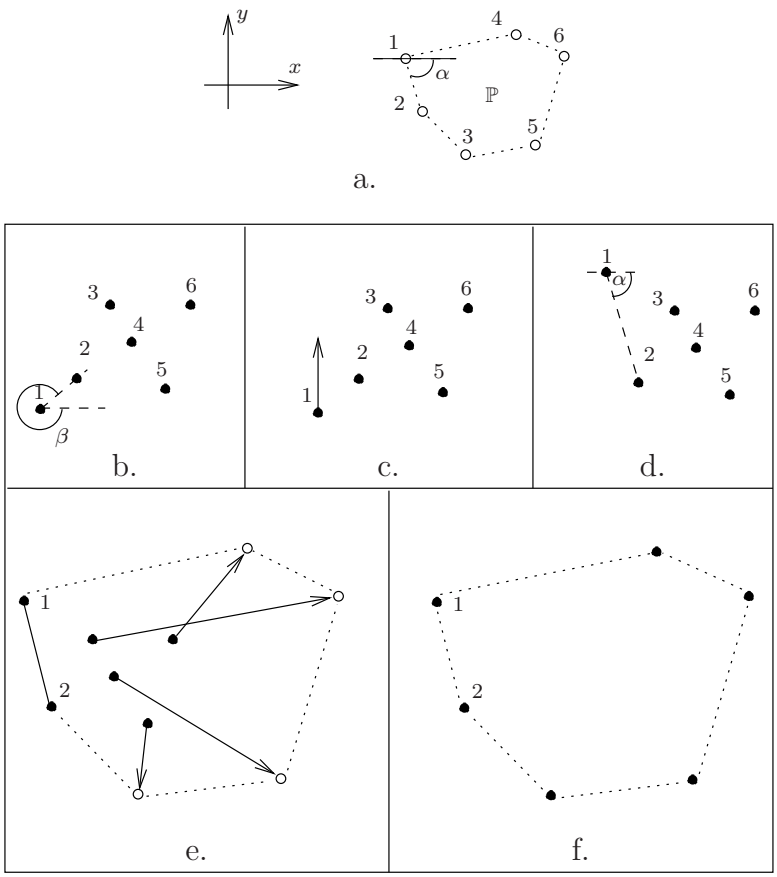

Figure 5: Algorithm 2. (a) The input pattern $\mathbb{P}$. The robots have total agreement on the local coordinate systems. The numbers represent the lexicographical ordering the robots give to the points of $\mathbb{P}$, and $\alpha = \texttt{Angle}(p_1, p_2)$. (b) The robots sort the robots' positions retrieved in the last *Look* state, and compute $\beta = \texttt{Angle}(r_1, r_2)$. (c) r_1 moves in such a way that $\texttt{Angle}(r_1, r_2) = \alpha$, according to routine `GoIntoPositions()`. (d) The relative positions of r_1 and r_2 are such that $\texttt{Angle}(r_1, r_2) = \alpha$. (e) At this point, all the robots can translate and scale the input pattern according to $[r_1 r_2]$. Then, all the robots, one at a time, reach the final positions of the pattern to form. (f) The final configuration.

Algorithm 2 COMPLETE

Input: An arbitrary pattern $\mathbb{P}$ described as a sequence of points $p_1, \ldots, p_n$, given in lexicographic order, with the ordering given left-right, bottom-up. There is total agreement on the coordinate system.

$\alpha :=$ `Angle`(p_1, p_2);
$SortedRobots :=$ `Sort()`;
$r_1 :=$ First robot in $SortedRobots$;
$r_2 :=$ Second robot in $SortedRobots$;
$r^* :=$ Last Robot in $SortedRobots$;
$\beta :=$ `Angle`(r_1, r_2);
Case I Am

- r_2:
  `destination`:= `null`; **EndC**.
- r_1:
  If $\alpha = \beta$ **Then** `destination`:= `null`; **EndC**.
  Else
  $p :=$ `GoIntoPosition`(r_1, r_2, α).
  `destination`:= p; **EndC**.
- Default: %I am neither r_1 nor r_2%
  If $\alpha = \beta$ **Then**
  $Unit := dist(r_1, r_2)$; %all the robots agree on a common unit distance%
  $FinalPositions :=$ `FindFinalPositions`$(r_1, r_2, Unit)$;
  $Ext :=$ Rightmost Point In $FinalPositions$;
  If I Am On One Of The $FinalPositions$ **Then**
  `destination`:= `null`; **EndC**.
  Else
  $FreeRobots :=$ {*Robots Not On One Of The FinalPositions*};
  $FreePoints :=$ {*FinalPositions With No Robots On Them*};
  $Obstacles :=$ {*Robots On One Of The FinalPositions*};
  $\Gamma :=$ Vertical Line Through r_2;
  $\Gamma^* :=$ Vertical Line Through r^*;
  $\Gamma_{Ext} :=$ Vertical Line Through Ext;
  $\Gamma' :=$ Rightmost Vertical Line Among Γ^* and Γ_{Ext};
  `CloseToDestination`$(FreeRobots, FreePoints, Obstacles, \Gamma, \Gamma')$.
  Else `destination`:= `null`; **EndC**.

`GoIntoPosition`(r_1, r_2, α) orders r_1 to move so as to achieve angle α with r_2 while staying lexicographically first. During this movement, all other robots do not move.

`FindFinalPositions`$(r_1, r_2, Unit)$ figures out the final positions of the robots according to the given pattern and to the positions of r_1 and r_2. In particular, p_1 is translated onto the position of r_1, and p_2 onto the position of r_2. The common scaling of the input pattern is defined by the common unit distance $Unit$.

4.1 Correctness of Algorithm Complete

Now, we are ready to show that Algorithm Complete lets the robots correctly form the input pattern. Given a configuration where the robots' positions are ordered according to routine `Sort()`, we call it an *agreement configuration* if the first two robots r_1 and r_2 are such that `Angle`$(r_1, r_2) = \alpha$, with α the angle computed in Line 1 of Algorithm Complete.

Lemma 4.1. *If the robots are not in a final configuration, then in a finite number of cycles, say at time t_α, and avoiding collisions, they will reach an agreement configuration. Furthermore, all the robots but r_1 are in $\mathbb{I}(t)$, for all $0 \leq t \leq t_\alpha$, and $r_1 \in \mathbb{I}(t_\alpha)$.*

Proof. Let r_1 and r_2 be the first two robots in the lexicographic order defined by `Sort()` in the initial configuration $\mathbb{D}_0$. If `Angle`$(r_1, r_2) = \alpha$, then the lemma clearly follows, and $t_\alpha = 0$. Otherwise, according to the algorithm, the only robot that is allowed to move at the beginning is r_1 (Lines 9, 12, and 28), and it executes `GoIntoPosition`(r_1, r_2, α). During this movements, r_1 stays lexicographically first; hence, as long as `Angle`$(r_1, r_2) \neq \alpha$, it is the only robot allowed to move, and all the others compute only *null movements*. In a finite number of cycles, say at time t_α, r_1 reaches a position such that `Angle`$(r_1, r_2) = \alpha$. Since until t_α no robot but r_1 is allowed to move, the lemma follows. □

We can now state that

Theorem 4.1. *Algorithm* Complete *lets the robots correctly form the input pattern $\mathbb{P}$ in a finite number of cycles, while avoiding collisions.*

Proof. According to the previous lemma, at time t_α the robots are in an agreement configuration. From now on, r_1 and r_2 never move again (Lines 9 and 11). At this point, the distance $Unit = dist(r_1, r_2)$ is used in routine `FindFinalPositions`$(r_1, r_2, Unit)$ to compute the positions the robots have to reach in order to correctly form the input pattern (Line 16). By definition of this routine, and since at t_α all the robots are in $\mathbb{I}(t_\alpha)$, from now on all the robots will agree on the set $FinalPositions$ (i.e., in subsequent computations, they will all compute the same set of points, up to the translations due to the different origins of the local coordinate systems). Moreover, by the way these positions have been computed, r_1 and r_2 are already on their final targets. Let Γ and Γ' be as defined in Lines 23 and 26 of the algorithm.

If a robot is on a final position, it never moves again (Line 19). Otherwise, it will call at every cycle Algorithm `CloseToDestination()` (Line 27). Let (r, p) be the pair that satisfies Equation (1) at time t_α with respect to the sets $FreeRobots$ and $FreePoints$ as defined in Lines 21 and 22 of the algorithm. With respect to Algorithm `CloseToDestination()`, the set $FreePoints$ corresponds to $\mathbb{FT}$, the set $FreeRobots$ to $\mathbb{FR}$, and the obstacles $\mathbb{O}$ are given by the set of robots that have already reached their final destination. Note that all the four conditions required by `CloseToDestination()` are met. In particular, the sets $\mathbb{FT}$, $\mathbb{FR}$, and $\mathbb{O}$ are inside the vertical stripe delimited by Γ and Γ'; $|\mathbb{FT}| = |\mathbb{FR}|$; at time t_α all robots are in $\mathbb{I}(t_\alpha)$; and all robots in $\mathbb{O}$ will never move again (Line 19 of Algorithm Complete).

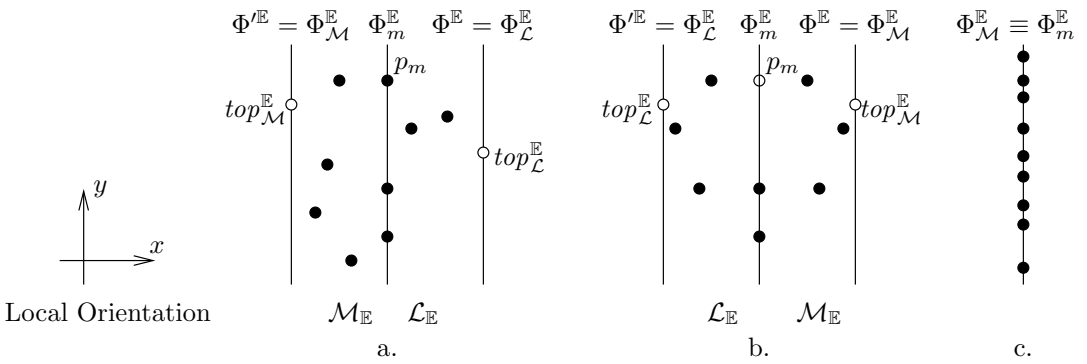

Figure 6: (a)–(b) Computing the *reference points* of a set $\mathbb{E}$. In (a) an *unbalanced* set of points. In (b) an example of a *balanced* set of points. In this case, $\Phi^{\mathbb{E}}_{\mathcal{M}}$ is chosen according to the local orientation depicted. Finally, in (c) a degenerated set of points is reported.

By Theorems 3.1 and 3.2, r reaches in a finite number of cycles p while avoiding collisions, say at $t' > t_\alpha$, while all the other robots compute only *null movements*. Furthermore, r does not trespass the region of the plane delimited by Γ and Γ'. By the way the input pattern has been scaled, no final positions can be on Γ below r_2. Furthermore, by the way the robots' positions have been sorted by `Sort()`, no robot can be on Γ below r_2. Hence r_1 and r_2 stay the first two lexicographical robots in the system. Moreover, at t' the cardinality of sets *FreeRobots* and *FreePoints* decreases by one, and r joins $\mathbb{O}$. Hence, since the number of points in the pattern equals the number of robots in the system, and by iterating the above argument, eventually each robot will occupy a pattern point position, while avoiding collisions. □

Result 1. *With total agreement on the local coordinate systems, a set of autonomous, anonymous, oblivious, mobile robots can form an arbitrary given pattern.*

5 Partial Agreement: The Odd Case

In this section, we deal with the case of the robots having partial agreement: they agree since the beginning only on the orientation of one axis, say y. As an aside, note that this case would trivially coincide with the first one, if the robots would have a common handedness (or sense of orientation, as Suzuki and Yamashita call it [26, 28]).

As stated in Corollary 2.1, this problem is unsolvable in general, since symmetric initial configuration can impede the formation of arbitrary patterns. We now show that for breaking the symmetry, it is sufficient that the number n of robots is odd. We make use of the fact that, since n is odd, either the robots are in a symmetric initial situation, in which there is a unique middle robot that will move in order to break the symmetry; or the initial situation is not symmetric, and this asymmetry can be used to identify an orientation of the x axis. We feel that this technique of symmetry breaking for mobile robots may have other applications, and hence it may be of independent interest. Also in this algorithm we do not rotate the final positions w.r.t the input pattern.

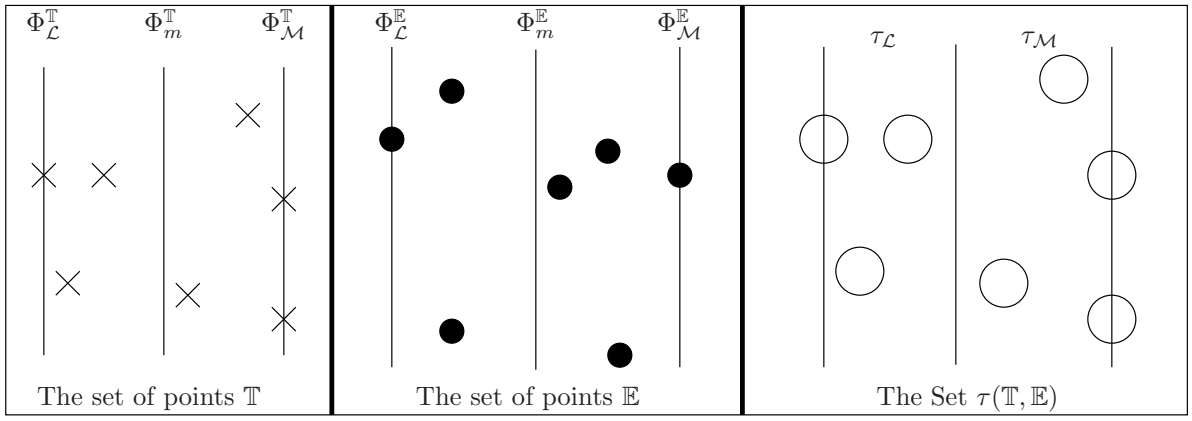

Figure 7: Given the set of points $\mathbb{T}$ and $\mathbb{E}$, the circles in (c) are the points in $\tau(\mathbb{T},\mathbb{E})$; in this case $\tau^{-1}(\cdot,\cdot)$ is empty.

5.1 Basic Definitions

Given a set of points $\mathbb{E}$, we say that it is *degenerated* if all points lie on the same vertical axis. Furthermore, we define some *references* related to $\mathbb{E}$ that will be used in the following. Consider the two vertical lines that are tangent to the convex hull of $\mathbb{E}$, and the median $\Phi^{\mathbb{E}}_m$: the vertical line that is in the middle between them. These three vertical lines delimit two regions (or *sides*): one to the left of $\Phi^{\mathbb{E}}_m$ and one to its right; for technical reasons, $\Phi^{\mathbb{E}}_m$ will not be considered part of either region. Let $\mathcal{M}_{\mathbb{E}}$ and $\mathcal{L}_{\mathbb{E}}$ denote the side in $\mathbb{E}$ with more and less points, respectively. If the two sides have the same number of points, then $\mathcal{M}_{\mathbb{E}}$ is the rightmost side. Moreover, $\Phi^{\mathbb{E}}_{\mathcal{M}}$ denotes the one of the two axes tangent to the convex hull of $\mathbb{E}$ that lies in $\mathcal{M}_{\mathbb{E}}$, and $\Phi^{\mathbb{E}}_{\mathcal{L}}$ the other (Figure 6.a). We say that a point is *strictly* inside $\mathcal{M}_{\mathbb{E}}$ (resp. $\mathcal{L}_{\mathbb{E}}$) if it belongs to $\mathcal{M}_{\mathbb{E}}$ (resp. $\mathcal{L}_{\mathbb{E}}$) but it is not on $\Phi^{\mathbb{E}}$ (resp. $\Phi'^{\mathbb{E}}$).

If $|\mathcal{M}_{\mathbb{E}}| \neq |\mathcal{L}_{\mathbb{E}}|$, we will say that $\mathbb{E}$ is *unbalanced* (see Figure 6.a); otherwise, we will call it *balanced* (see Figure 6.b).

Moreover, let $top^{\mathbb{E}}_{\mathcal{M}}$ and $top^{\mathbb{E}}_{\mathcal{L}}$ be the topmost points on $\Phi^{\mathbb{E}}_{\mathcal{M}}$ and on $\Phi^{\mathbb{E}}_{\mathcal{L}}$, respectively.

Given two sets of points $\mathbb{E}$ and $\mathbb{T}$, with $|\mathbb{E}| = |\mathbb{T}|$, we define the transformation $\tau(\mathbb{T},\mathbb{E})$ of $\mathbb{T}$ with respect to $\mathbb{E}$, as the set of points obtained by scaling $\mathbb{T}$ with respect to $\overline{\Phi^{\mathbb{E}}_{\mathcal{M}}, \Phi^{\mathbb{E}}_{\mathcal{L}}}$, and by translating and flipping $\mathbb{T}$ so that $\Phi^{\mathbb{T}}_m$ is mapped onto $\Phi^{\mathbb{E}}_m$, $top^{\mathbb{T}}_{\mathcal{M}}$ is mapped onto the topmost robot on $\Phi^{\mathbb{E}}_{\mathcal{M}}$, and $top^{\mathbb{T}}_{\mathcal{L}}$ is mapped onto the topmost robot on $\Phi^{\mathbb{E}}_{\mathcal{L}}$ (see the example depicted in Figure 7). If such a translation and flipping cannot be obtained, then $\tau(\mathbb{T},\mathbb{E}) = \emptyset$. By $\tau^{-1}(\mathbb{T},\mathbb{E})$, we denote the mirroring of $\tau(\mathbb{T},\mathbb{E})$ with respect to $\Phi^{\mathbb{E}}_m$. Moreover, if $\mathbb{E}$ is unbalanced, we denote by $\tau_{\mathcal{M}}$ and $\tau_{\mathcal{L}}$ the subset of $\tau(\mathbb{T},\mathbb{E})$ whose points are in $\mathcal{M}_{\mathbb{E}}$ and in $\mathcal{L}_{\mathbb{E}}$, respectively. Analogously, we define $\tau^{-1}_{\mathcal{M}}$ and $\tau^{-1}_{\mathcal{L}}$.

The configurations of robots observed during the *Look*s, as well as the input pattern $\mathbb{P}$, are set of points, on which all previous definitions apply. However, note that a configuration observed by a robot is expressed in terms of the local coordinate system of the observing robot. Therefore, we will denote by $\mathbb{E}[r]$ a set of points (coordinates) in the local coordinate system of r. Notice that, $\Phi^{\mathbb{E}}_m[r] = \Phi^{\mathbb{E}}_m[r']$, for all $r \neq r'$, regardless of the local coordinate systems. Furthermore, if $\mathbb{E}$ is unbalanced, then all robots agree on $\mathcal{M}_{\mathbb{E}}$, $\mathcal{L}_{\mathbb{E}}$, $\Phi^{\mathbb{E}}_{\mathcal{M}}$, $\Phi^{\mathbb{E}}_{\mathcal{L}}$, $top^{\mathbb{E}}_{\mathcal{M}}$, and $top^{\mathbb{E}}_{\mathcal{L}}$, regardless of the local orientation of the x axis.

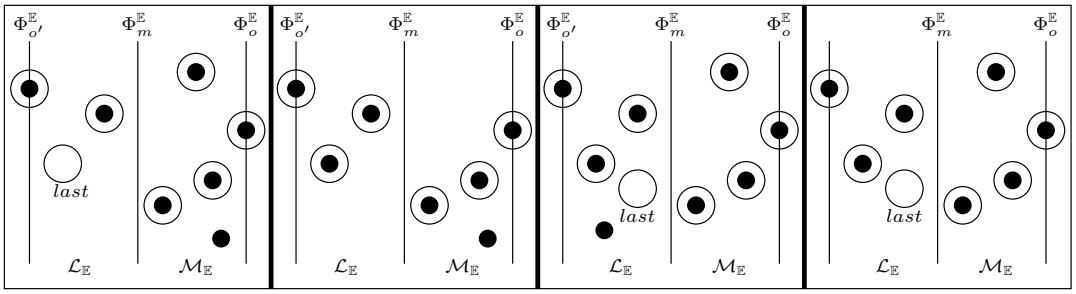

Figure 8: Definition 5.2: examples of possible *semi-final* configurations. The circles represent points in $\tau(\mathbb{P}, \mathbb{E})$.

Definition 5.1. A configurations of robots $\mathbb{E}$ is *final* for a given pattern $\mathbb{P}$ when either $\tau(\mathbb{P}, \mathbb{E}) = \mathbb{E}$ or $\tau^{-1}(\mathbb{P}, \mathbb{E}) = \mathbb{E}$.

A particular configuration of robots $\mathbb{E}$ that will be used in the following is the *semi-final* configuration (refer to the example depicted in Figure 8).

Definition 5.2. Let $\tau(\mathbb{P}, \mathbb{E}) \neq \emptyset$. A configuration of robots $\mathbb{E}$ is *semi-final* for a given pattern $\mathbb{P}$ if the following holds:

1. $\mathbb{E}$ is unbalanced; and,
2. one of the following holds:
   (a) All robots in $\mathcal{M}_{\mathbb{E}} \cup \mathcal{L}_{\mathbb{E}}$ except exactly one, r, occupy one distinct point in $\tau_{\mathcal{M}} \cup \tau_{\mathcal{L}}$; furthermore, $r \in \mathcal{M}_{\mathbb{E}}$ and among the points in $\tau_{\mathcal{M}} \cup \tau_{\mathcal{L}}$ exactly one is not occupied by any robots, call it *last*.
   (b) All robots in $\mathcal{M}_{\mathbb{E}} \cup \mathcal{L}_{\mathbb{E}}$ except exactly one, r, occupy one distinct point in $\tau_{\mathcal{M}} \cup \tau_{\mathcal{L}}$; furthermore, $r \in \mathcal{M}_{\mathbb{E}}$ and all points in $\tau_{\mathcal{M}} \cup \tau_{\mathcal{L}}$ are occupied by exactly one robot.
   (c) All robots in $\mathcal{M}_{\mathbb{E}} \cup \mathcal{L}_{\mathbb{E}}$ except exactly one, r, occupy one distinct point in $\tau_{\mathcal{M}} \cup \tau_{\mathcal{L}}$; furthermore, $r \in \mathcal{L}_{\mathbb{E}}$ and among the points in $\tau_{\mathcal{M}} \cup \tau_{\mathcal{L}}$ exactly one is not occupied by any robots, call it *last*; *last* is in $\mathcal{L}_{\mathbb{E}}$.
   (d) All robots in $\mathcal{M}_{\mathbb{E}} \cup \mathcal{L}_{\mathbb{E}}$ occupy one distinct point in $\tau_{\mathcal{M}} \cup \tau_{\mathcal{L}}$; furthermore, among the points in $\tau_{\mathcal{M}} \cup \tau_{\mathcal{L}}$ exactly one is not occupied by any robots, call it *last*; *last* is in $\mathcal{L}_{\mathbb{E}}$.

The previous definition implies that, in a *semi-final* configuration, there is only one point in $\tau_{\mathcal{L}} \cup \tau_{\mathcal{M}}$ not occupied by a robot; this point can clearly be in $\mathcal{M}_{\mathbb{E}}$, $\mathcal{L}_{\mathbb{E}}$, or on $\Phi^{\mathbb{E}}_m$. In cases (a), (c) and (d), we will call such a point *last*, where in case (b) *last* is always set to $\emptyset$. Examples of *semi-final* configurations are depicted in Figure 8.

In a *quasi-final* configuration $\mathbb{E}$, all robots except for those on $\Phi^{\mathbb{E}}_m$, are in a final configuration (see Figure 9).

Definition 5.3. A configuration $\mathbb{E}$ is *quasi-final* if either

1. each point in $\tau_{\mathcal{M}} \cup \tau_{\mathcal{L}}$ is occupied by exactly one robot, and $|\tau_{\mathcal{M}} \cup \tau_{\mathcal{L}}| = |\mathcal{M}_{\mathbb{E}}| + |\mathcal{L}_{\mathbb{E}}|$; or
2. each point in $\tau^{-1}_{\mathcal{M}} \cup \tau^{-1}_{\mathcal{L}}$ is occupied by exactly one robot, and $|\tau^{-1}_{\mathcal{M}} \cup \tau^{-1}_{\mathcal{L}}| = |\mathcal{M}_{\mathbb{E}}| + |\mathcal{L}_{\mathbb{E}}|$.

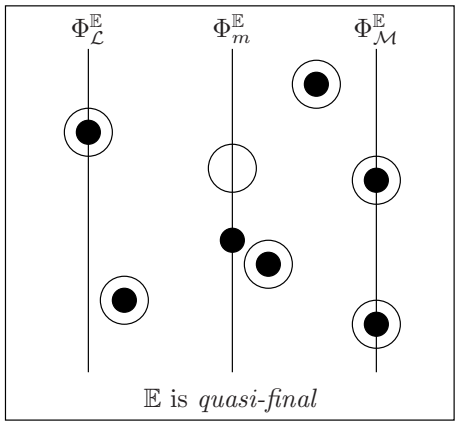

Figure 9: Definition 5.3: the circles represent points in $\tau(\mathbb{P}, \mathbb{E})$.

In the following, $\mathbb{D}_t$ will denote the configuration of the robots at time t; to simplify the notation, we will use $\Upsilon_m = \Phi^{\mathbb{P}}_m$, $\Upsilon^+ = \Phi^{\mathbb{P}}_{\mathcal{M}}$, $\Upsilon^- = \Phi^{\mathbb{P}}_{\mathcal{L}}$ to denote the references in $\mathbb{P}$, and $K_m = \Phi^{\mathbb{D}_t}_m$, $K^+ = \Phi^{\mathbb{D}_t}_{\mathcal{M}}$, $K^- = \Phi^{\mathbb{D}_t}_{\mathcal{L}}$ to denote the references in $\mathbb{D}_t$. When no ambiguity arises, the time reference will be omitted.

In order to solve the APF problem with an odd number of robots, we distinguish the two possible cases:

Case a. $\mathbb{P}$ is non-degenerated;

Case b. $\mathbb{P}$ is degenerated (see Figure 6.c).

We discuss the two cases separately.

5.2 Case a.: Non-degenerated pattern

In this section, we present Algorithm POND (whose pseudo-code is reported in Algorithms 3 and 4), that solves the APF problem with Partial agreement and an Odd number of robots that have to form a Non-Degenerated input pattern.

5.2.1 The POND Algorithm

The overall strategy of the algorithm is (1) first to bring the robots in a non-degenerated configuration. Then, (2) the robots are forced to form an unbalanced configuration in order to reach an agreement on the direction of the x axis. Once such a configuration has been reached, say at time t, the robots compute the set of *final positions* according to the transform $\tau(\mathbb{P}, \mathbb{D}_t)$: these are the points they have to reach in order to correctly form $\mathbb{P}$. Finally, (3) the robots reach these final positions.

Let us describe these three phases in more details (the pseudo-code of the routines used by the POND Algorithm is reported in Appendix A).

Non-degenerated configuration. First the references of $\mathbb{P}$ are computed. Then, the algorithm checks whether all the robots are on a single vertical axis, say Ξ. In this case, since by hypothesis $\Upsilon^+ \not\equiv \Upsilon_m$, the algorithm forces the second topmost robot on Ξ, say r, to move away from Ξ (recall that $n \geq 3$), so that a non-degenerated configuration (i.e., where the robots

Algorithm 3 POND– First Part

Input: An arbitrary pattern $\mathbb{P}$ described as a sequence of points $p_1, \ldots, p_n$, given in lexicographic order, such that $\Upsilon^+ \not\equiv \Upsilon_m$. The direction and orientation of the y axis is common knowledge.

$(\Upsilon^+, \Upsilon^-, \Upsilon_m) := (\Phi^{\mathbb{P}}_{\mathcal{M}}, \Phi^{\mathbb{P}}_{\mathcal{L}}, \Phi^{\mathbb{P}}_m)$;
$\mathbb{D}$:= Current Configuration Of The Robots;
$K_m := \Phi^{\mathbb{D}}_m$;
If $\mathbb{D}$ Is Final Configuration **Then STOP**.
If $\mathbb{D}$ Is *quasi-final* **Then** `Fix(`$\mathbb{D}, K_m$`)`; **EndC**.
$(SF, r, last)$:=`TestSF(`$\mathbb{D}$`)`;
If SF **Then**
  Case $(r, last)$
    - $(\neq \emptyset, \neq \emptyset)$
      If I Am r **Then**
        If *last* Is Not In The Side Where I Am **Then**
          `destination`:=Point On K_m With No Robots On $[rp]$; **EndC**.
        Else `CloseToDestination(`$\{top\}, \{last\}, K^+, K^-$`)`;**EndC**.
      Else `destination`:= `null`; **EndC**.
    - $(\neq \emptyset, = \emptyset)$
      If I Am r **Then**
        `destination`:=Point On K_m With No Robots On $[rp]$; **EndC**.
      Else `destination`:= `null`; **EndC**.
    - $(= \emptyset, \neq \emptyset)$
      top:= Topmost Robot On K_m;
      If I Am top **Then** `CloseToDestination(`$\{top\}, \{last\}, K^+, K^-$`)`;**EndC**.
      Else `destination`:= `null`; **EndC**.
Ξ := Vertical Axis With More Robots On It;
If $|\Xi| = n$ **Then** `SameVerticalAxis(`Ξ`)`; **EndC**.
If $|\Xi| = n - 1$ **Then**
  r :=Robot not on Ξ;
  If $dist(r, \Xi) \neq dist($`top(`Ξ`), bottom(`Ξ`)`$)$ **Then** `SameVerticalAxis2(`Ξ`)`; **EndC**.
If $\mathbb{D}$ Is Unbalanced **Then** $(K^+, K^-) := (\Phi^{\mathbb{D}}_{\mathcal{M}}, \Phi^{\mathbb{D}}_{\mathcal{L}})$;
Else `GetUnbalanced(`$\mathbb{D}$`)`**EndC**.
$(\mathcal{S}^+, \mathcal{S}^-)$:= Sides of $\mathbb{D}$ where K^+ and K^- lie, respectively;
$(top^+, top^-) := (top^{\mathbb{D}}_{\mathcal{M}}, top^{\mathbb{D}}_{\mathcal{L}})$;
If `Angle(`$top^{\mathbb{P}}_{\mathcal{M}}, top^{\mathbb{P}}_{\mathcal{L}}$`)`$\neq$ `Angle(`top^+, top^-`)` **Then** `FixOutermosts(`top^+, top^-`)`; **EndC**.
If I Am top^+ **Or** top^- **Then** `destination`:= `null`; **EndC**.
$FinalPositions$:= `FindFinalPositions(`$\mathbb{P}$`,`$\mathbb{D}$`)`;

Algorithm 4 POND– Second Part

If `AllOn()` **Then**
  $r :=$ Robot Strictly Inside $\mathcal{S}^+$;
  $p :=$`MoveInside`$\mathcal{S}^+($`(`$FinalPositions, \mathcal{S}^+, K^+, r$`)`;
  If I Am r **Then** `destination`$:= p$; **EndC**. **Else** `destination`$:=$ `null`; **EndC**.
39: $p_b :=$ Bottommost Of The $FinalPositions$ On K^+;
If There Are At Least Two Robots On K^+ **Then**
  $p :=$ Point On K^+ Below $p_b.y$;
  $r_b :=$ Bottommost Robot On K^+;
  If $r_b.y \geq p_b.y$ **Then**
44: **If** I Am r_b **Then** `destination`$:= p$; **EndC**. **Else** `destination`$:=$ `null`; **EndC**.
Else
  $r :=$ Robot Strictly Inside $\mathcal{S}^+$ With The Smallest Horizontal Distance From K^+ (ties are broken by choosing the topmost);
  $p :=$ Point On K^+ Below $p_b.y$ So That There Are No Robots On $[rp]$;
  If I Am r **Then** `destination`$:= p$; **EndC**. **Else** `destination`$:=$ `null`; **EndC**.
49: **If** I Am On One Of The $FinalPositions$ **Then** `destination`$:=$ `null`; **EndC**.
$FreeRobots\mathcal{S}^- :=$ {*Robots' positions In* $\mathcal{S}^-$ *Not In FinalPositions*};
$FreeRobots\mathcal{S}^+ :=$ {*Robots' positions In* $\mathcal{S}^+$ *Not In FinalPositions*};
If $FreeRobots\mathcal{S}^- \neq \emptyset$ **Then** `FromSidesToMedian(`$K^+, K^-, K_m, \mathcal{S}^-, FinalPositions$`)`; **EndC**.
If $FreeRobots\mathcal{S}^+ \setminus r_b \neq \emptyset$ **Then** `FromSidesToMedian(`$K^+, K^-, K_m, \mathcal{S}^+, FinalPositions$`)`; **EndC**.
54: $FreePoints\mathcal{S}^+ :=$ {*FinalPositions In* $\mathcal{S}^+$ *With No Robots On Them* };
$FreePoints\mathcal{S}^- :=$ {*FinalPositions In* $\mathcal{S}^-$ *With No Robots On Them* };
If $FreePoints\mathcal{S}^+ \neq \emptyset$ **Then** `FromMedianToSides(`$K^+, K^-, K_m, \mathcal{S}^+, FreePoints\mathcal{S}^+$`)`; **EndC**.
If $FreePoints\mathcal{S}^- \neq \emptyset$ **Then** `FromMedianToSides(`$K^+, K^-, K_m, \mathcal{S}^+, FreePoints\mathcal{S}^-$`)`; **EndC**.

occupy at least two distinct vertical axes) is reached (`SameVerticalAxis()` in Line 24). In particular, r moves to its local right of a distance equal to the distance between the topmost and the bottommost robot on Ξ; all the other robot are forced to not move until r reaches such a distance.

Unbalancing the configuration. At this point, the robots form a non-degenerated configuration $\mathbb{D}$, and the references for $\mathbb{D}$ can be computed (Line 28). Then the algorithm forces the robots to create an unbalanced configuration, so that an agreement on the direction of the x axis can be reached. This is achieved by routine `GetUnbalanced()` (Line 29). If $\mathbb{D}$ is balanced, the symmetry that derives from having the two sides with the same number of robots is broken as follows. First all the robots[4] in $\mathcal{M}_{\mathbb{D}}$ are moved on K^+ and all the robots in $\mathcal{L}_{\mathbb{D}}$ on K^-. After all the robots have performed these movements, since $\mathbb{D}$ is still balanced and the total number of robots is odd, there is an odd number of robots on K_m: the topmost robot on K_m, say top^*, is selected to move towards its (local) right, so that an unbalanced configuration can be achieved. This movement is performed carefully since, as soon as top^* leaves K_m and enters the side to

[4]Note that, since at this time the robots do not still have a common agreement on the direction of the x axis, for some robots $\mathcal{M}_{\mathbb{D}}$ and $\mathcal{L}_{\mathbb{D}}$ might be different. All of them, however, agree on K_m.

its right, the configuration will become unbalanced.

The fact that the configuration is unbalanced allows the robots to implicitly reach an agreement on the direction of the x axis; hence, on a *global coordinate system* (*GCS*): the common orientation of the x axis is given by mapping $\mathcal{M}_{\mathbb{P}}$ onto $\mathcal{M}_{\mathbb{D}}$.

Once the *GCS* has been established, the topmost robots on K^+ and on K^- (top^+ and top^-, respectively) move strictly on K^+ and on K^-, respectively, until they reach positions corresponding to the two topmost points on Υ^+ and Υ^- in $\mathbb{P}$ (routine `FixOutermosts()` in Line 32, reported in Appendix A.4). More precisely, their final positions will form the same angle as the topmost points in the local pattern (routine `Angle()` in `FixOutermosts()`). Once top^+ and top^- place themselves in the correct positions, they will never move again.

At this point, the set of final positions of the robots is computed (routine `FindFinalPositions()` in Line 34, reported in Appendix A.5) by transforming the pattern according to $\tau(\mathbb{P}, \mathbb{D})$ (note that $top^{\mathbb{P}}_{\mathcal{M}}$ and $top^{\mathbb{P}}_{\mathcal{L}}$ are mapped onto top^+ and top^-, respectively, that are already in their correct positions).

Now, all robots are ready to reach their final destinations. Notice, however, that at this point it might be possible that the unbalancing process is not completed yet; i.e., top^* is still moving towards its destination. Should this be the case, the other robots can detect it, and will not start their move until top^* stops. Let us describe in more detail how top^* performs its move.

Recall that, when top^* decides to move, all robots are on K^+, K^-, or on K_m. Robot top^* knows that, (1) as soon as it enters the side to its right, the configuration will become unbalanced; furthermore, since the algorithm is the same for all the robots, top^* also knows which robots will move as soon as the configuration will become unbalanced. In particular, it knows that (2) the two topmost robots on K^+ and K^- will move to reach the points corresponding to $top^{\mathbb{P}}_{\mathcal{M}}$ and $top^{\mathbb{P}}_{\mathcal{L}}$ (routine `FixOutermosts()` in Line 32). It also knows that, after such a move, (3) the input pattern will be scaled according to $\tau(\cdot, \cdot)$. Hence, top^* can compute the final positions in the plane that the robots must eventually reach in order to correctly form the input pattern (routine `FindFinalPositions()` in Line 34). Therefore, top^* can compute, before it leaves K_m, the set of final positions as returned by `FindFinalPositions()`. If at least one of these final positions is inside its (local) right side, then top^* chooses as its destination the closest among them, and moves there. Otherwise (i.e., there are no final positions in its right side), it moves towards K^+. During this move, top^+ and top^- might be moving to reach their final positions. All the other robots, however, detect that $|\mathcal{L}| = |\mathcal{M}| - 1$, and that all robots are either on K^+, K^-, or on K_m, except for one (top^*) that is inside $\mathcal{S}^+$ and not on one of the final positions. In this scenario, test `AllOn()` return *true* (Line 35). In particular, routine `AllOn()` (described in Appendix A.7) returns *true* when

1. $|\mathcal{S}^-| = |\mathcal{S}^+| - 1$,
2. all the robots but one, say r, are either on K^+, or on K^-, or on K_m, and
3. r is strictly inside $\mathcal{S}^+$ not on one of the final positions.

Therefore, Lines 35–38 force all robots but top^* to wait until top^* reaches a final position strictly inside $\mathcal{S}^+$ or K^+.

Once top^* reaches its destination, we say that the unbalancing process has been completed. Once the unbalancing is completed, the next step is to have a particular robot move onto K^+, below any robots and below[5] any of the final positions that are on this axis (Lines 39–48). This

[5]i.e., the y coordinate of the chosen robot must be smaller than the y coordinate of any robot and of any of the final positions on K^+.

will be used to ensure that the agreement on the direction of the x axis can be correctly kept until the very end. Until this process has been completed, no other robot is allowed to move.

Reaching the final destinations. From now on, all the robots that are on one of the final positions never move again (Line 49). Furthermore, the following four steps are executed:

First, the robots in $\mathcal{S}^-$ sequentially fill the final positions that are in $\mathcal{S}^-$ (routine `FromSidesToMedian()` called in Line 52, and reported in Appendix A.8). If there are more robots than available final positions, the "extra" robots are sequentially moved towards K_m, starting from the topmost robots that is closest to K_m.

Second, the robots in $\mathcal{S}^+$, except for the bottommost on K^+, sequentially fill the final positions in $\mathcal{S}^+$ (routine `FromSidesToMedian()` in Line 53). If there are more robots than available final positions, the "extra" robots are sequentially moved towards K_m, starting from the topmost robots that is closest to K_m.

Third, if there are still unfilled final positions in $\mathcal{S}^+$ (that is, there were not enough robots in $\mathcal{S}^+$ in the second step), the robots on K_m are sequentially moved in $\mathcal{S}^+$, starting from the topmost, to fill the final positions occupied by no robots (routine `FromMedianToSides()` called in Line 56, and reported in Appendix A.9).

Fourth, if there are still unfilled final positions in $\mathcal{S}^-$ (that is, there were not enough robots in $\mathcal{S}^-$ in the first step), the robots on K_m are sequentially moved in $\mathcal{S}^-$, starting from the topmost, to fill the final positions still available (routine `FromMedianToSides()` in Line 57).

At this point, all the robots not on K_m occupy the correct positions except one: the bottommost robot on K^+, say r. This scenario is captured by Lines 6–22 that test whether the current configuration is *semi-final*. In particular, test in Line 6 returns $(true, r, last)$. According to Definition 5.2, we distinguish the possible cases:

1. If $last \neq \emptyset$, with *last* is inside $\mathcal{S}^+$, then r goes there (Line 13). At this point, all the robots but those on K_m are in correct positions. In this case, the configuration is *quasi-final*, and routine `Fix()` (Line 5, reported in Appendix A.2) moves the robots on K_m so that they reach their final positions; hence, the pattern is formed.

2. If $last = \emptyset$ (i.e., there are no available final positions inside $\mathcal{S}^+$ and $\mathcal{S}^-$), r moves towards K_m (Line 17). Once it reaches the median axis, all the robots but those on K_m are in correct positions. Again, the configuration is *quasi-final*, and, by calling routine `Fix()`, the pattern is formed.

3. If $last \neq \emptyset$, with *last* inside $\mathcal{S}^-$, r first moves towards K_m (Line 12). Then, Lines 19–22 will move the topmost robot on K_m in $\mathcal{S}^-$ on the *last* unfilled final position. Once also this position becomes occupied, only the robots on K_m must be adjusted, i.e., the configuration is *quasi-final*: again, the pattern is formed by invoking routine `Fix()`.

The above description of the algorithm is in *global terms*, that is, it describes the execution as seen by an external observer. The protocol (Algorithms 3 and 4), however, is expressed in *local terms*, that is from the point of view of a robot (recall, they all execute the same protocol). Moreover, since the robots are oblivious, every time a robot starts a cycle (observing the current configuration $\mathbb{D}$ and executing the protocol accordingly) it will do so without any memory of

past observations and executions. Each robot must guess which step of the global execution is currently being performed and what is its own role in it.

Hence, the sequence of steps of the *global execution* have been structured in the local view (i.e., in the protocol), so that this obliviousness does not affect its correctness, as we will show. In particular, each robot checks if the observed configuration is final, or *quasi-final*, or *semi-final* before it considers other possible configurations.

An example that shown the overall behavior of Algorithm Pond is pictured in Figure 10.

5.2.2 Correctness of Algorithm Pond

In this section we show that Algorithm Pond solves the pattern formation problem for an arbitrary pattern, if $\Upsilon^+ \not\equiv \Upsilon_m$.

In the following we will say that the robots satisfy the *termination conditions* at time t, denoted by $\mathcal{TC}_t$, if the configuration of the robots at time t is either *final*, or *semi-final*, or *quasi-final*. Alternatively, we will say that $\mathcal{TC}_t = true$.

The following lemma shows that, if the initial configuration $\mathbb{D}_0$ is degenerated, then Algorithm Pond brings the robots in a non-degenerated configuration in a finite number of cycles.

Lemma 5.1. *In a finite number of cycles, at time $t_{dis} \geq t_0$, the robots are in a non-degenerated configuration. Furthermore, until time t_{dis} any collision is avoided, and at t_{dis} all the robot are in $\mathbb{I}(t_{dis})$.*

Proof. If $\mathbb{D}_{t_0}$ is non-degenerated, then the lemma trivially follows. Otherwise, in $\mathbb{D}_{t_0}$, all the robots lie on the same vertical axis, say Ξ. In this configuration, a robot can only call routine `SameVerticalAxis(`Ξ`)` (Line 5). According to this routine, the second topmost robot r on Ξ is the only robot allowed to move: it moves to a point p at horizontal distance d from Ξ, where d is the distance between the topmost and the bottommost robot on Ξ. Let t_1 be the first time when r leaves Ξ. At time t_1, $|\Xi| = n - 1$. If r is observed when it is not on Ξ, the second case of the routine holds: all the robots are forced to not move as long as r has not reached p. Therefore, until this happens, all the other robots compute only *null movements*, and any collision is avoided. Let $t_{dis} \geq t_1$ be the first time when r is at p (t_{dis} is finite, by Assumptions A1 and A2 of the model). Since at this time all the robots are in $\mathbb{I}(t_{dis})$, the lemma follows. □

By the previous lemma, at time t_{dis} the robots are on at least two distinct vertical axes; hence, $\Phi_{\mathcal{M}}^{\mathbb{D}_{t_{dis}}} \not\equiv \Phi_{\mathcal{L}}^{\mathbb{D}_{t_{dis}}}$. In the following, it will be shown that the two vertical axes tangent to the convex hull of $\mathbb{D}_{t_{dis}}$ will never change; i.e., there will be at least a robot on each of these axes that will never leave them: $top_{\mathcal{M}}^{\mathbb{D}_{t_{dis}}}$ (on $\Phi_{\mathcal{M}}^{\mathbb{D}_{t_{dis}}}$) and $top_{\mathcal{L}}^{\mathbb{D}_{t_{dis}}}$ (on $\Phi_{\mathcal{L}}^{\mathbb{D}_{t_{dis}}}$). Therefore, to simplify the notation, we will refer to them as follows:

1. If $\mathbb{D}_{t_{dis}}$ is balanced, let top^* be the topmost robot on $\Phi_m^{\mathbb{D}_{t_{dis}}}$ (since n is odd, an odd number of robots must be on the median axis), and $\mathbb{A} = \mathbb{D}_{t_{dis}}[top^*]$. Then, let $K^+ = \Phi_{\mathcal{M}}^{\mathbb{A}}$, $K^- = \Phi_{\mathcal{L}}^{\mathbb{A}}$, and $K_m = \Phi_m^{\mathbb{A}}$.

2. If $\mathbb{D}_{t_{dis}}$ is unbalanced, let $K^+ = \Phi_{\mathcal{M}}^{\mathbb{D}_{t_{dis}}}$, $K^- = \Phi_{\mathcal{L}}^{\mathbb{D}_{t_{dis}}}$, and $K_m = \Phi_m^{\mathbb{D}_{t_{dis}}}$ (in this case, all robots agree on which side has the most number of robots inside).

Let an *empty* configuration be a configuration where all robots are either on K^+, K^-, or on K_m.

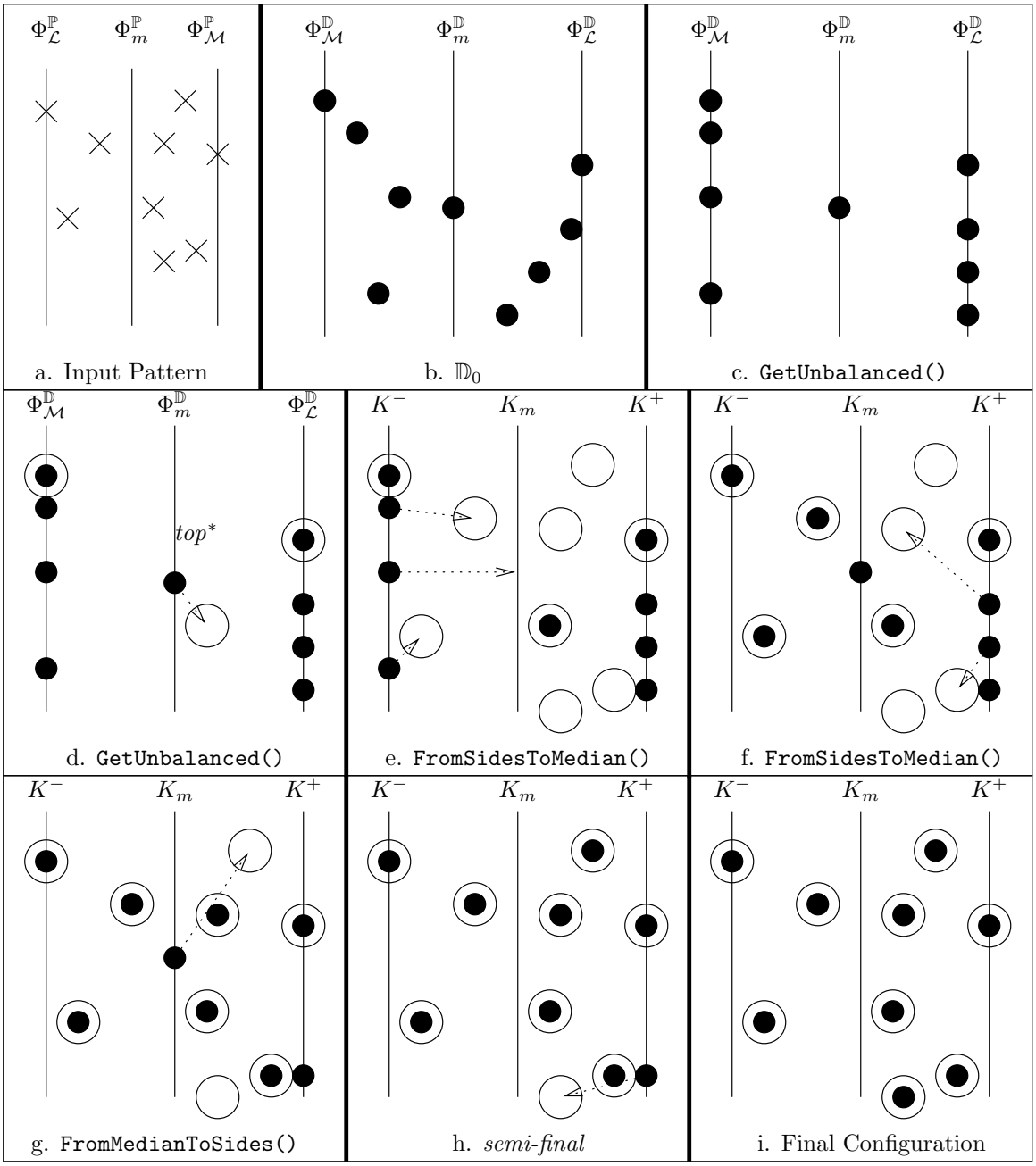

Figure 10: Main steps executed by Algorithm Pond. (a) The input pattern. (b) The initial configuration $\mathbb{D}_{t_0}$ is balanced. (c) Routine `GetUnbalanced()` places all the robots on K_m, K^+, and K^- (*sweeping* process). (d) Routine `GetUnbalanced()` unbalance the configurations by moving the topmost robots on K_m towards the closest of the final positions in its local right. (e) Routine `FromSidesToMedian()` in $\mathcal{S}^-$ moves all the free robots in $\mathcal{S}^-$ towards final positions in $\mathcal{S}^-$. The only "extra robot" in $\mathcal{S}^-$ is directed towards K_m. (f) Routine `FromSidesToMedian()` in $\mathcal{S}^+$. (g) Routine `FromMedianToSides()` in $\mathcal{S}^+$ moves the topmost robot on K_m towards a final position in $\mathcal{S}^+$. (h) The configuration is *semi-final*: r_b moves towards the *last* final positions in $\mathcal{S}^+$. (i) The final configuration.

Let p^+ and p^- be the two points on K^+ and K^- and above $top^{\mathbb{A}}_{\mathcal{M}}$ and $top^{\mathbb{A}}_{\mathcal{L}}$, respectively, such that[6] $\texttt{Angle}(p^+, p^-) = \texttt{Angle}(top^{\mathbb{P}}_{\mathcal{M}}, top^{\mathbb{P}}_{\mathcal{L}})$. Finally, let $\mathcal{S}^+$ (resp., $\mathcal{S}^-$) be the side where K^+ (resp., K^-) is, and $\mathbb{D}' = \mathbb{A} \setminus \{top^{\mathbb{A}}_{\mathcal{M}}, top^{\mathbb{A}}_{\mathcal{L}}\} \cup \{p^+, p^-\}$.

Lemma 5.2. *Let t_{dis} be as defined in Lemma 5.1, let $\mathcal{TC}_{t_{dis}}$ =false, and let $\mathbb{D}_{t_{dis}}$ be balanced. Then in a finite number of cycles, say at time $t_g > t_{dis}$, the robots either reach a* quasi-final *or a* final *configuration, or a configuration*

1. *that is unbalanced; and*
2. *where $top^{\mathbb{D}_{t_g}}_{\mathcal{M}}$ and $top^{\mathbb{D}_{t_g}}_{\mathcal{L}}$ are on p^+ and p^-, respectively; and*
3. *either*
   (a) *the configuration is empty, or*
   (b) *there is no robot strictly inside $\mathcal{L}_{\mathbb{D}_{t_g}}$, and only one robot, r^*, is inside $\mathbb{D}_{t_g}$, with r^* on one of the points in $\tau(\mathbb{P}, \mathbb{D}')$;*

Furthermore, at time t_g all robots are in $\mathbb{I}(t_g)$, and until this time any collision is avoided.

Proof. Since $\mathbb{D}_{t_{dis}}$ is balanced, the robots execute routine `GetUnbalanced()` (Line 29 of Algorithm POND). In particular, the only robots that are allowed to move are those not on $K_m \cup K^+ \cup K^-$, starting a *sweeping* process. Let us consider what happens in $\mathcal{S}^+$ (the same happens in the other side, $\mathcal{S}^-$). If the robots in this side are not all on K^+, then the robots strictly inside $\mathcal{S}^+$ move sequentially towards K^+ (Lines 10–14 of `GetUnbalanced()`). In particular, the topmost robot strictly inside $\mathcal{S}^+$ with the smallest horizontal distance from K^+, say r, is allowed to move, while all the others in $\mathcal{S}^+$ and those on K_m wait. According to this routine, r chooses to move to a position on K^+ not occupied by any robot and below $top^{\mathbb{A}}_{\mathcal{M}}$, say p. Since r is the closest to K^+ among all the robots strictly inside $\mathcal{S}^+$, there is no robot on $[rp]$; hence, during its movement towards p, any collision is avoided.

Applying iteratively the same argument to all the robots that are strictly inside both sides, we can conclude that in a finite number of cycles, say at time $t_e \geq t_{dis}$, all the robots will be either on K^+, or on K_m, or on K^-; i.e., $\mathbb{D}_{t_e}$ is empty. Furthermore, at this time all robots are in $\mathbb{I}(t_e)$.

Note that, for all $t_{dis} \leq t \leq t_e$, $top^{\mathbb{D}_t}_{\mathcal{M}} = top^{\mathbb{A}}_{\mathcal{M}}$ and $top^{\mathbb{D}_t}_{\mathcal{L}} = top^{\mathbb{A}}_{\mathcal{L}}$: this follows from the way the robots moved towards K^+ and K^- during the sweeping process. In fact, each *sweeped* robot in $\mathcal{S}^+$ (resp., $\mathcal{S}^-$) choose as destination a point below $top^{\mathbb{A}}_{\mathcal{M}}$ (resp., $top^{\mathbb{A}}_{\mathcal{L}}$). Furthermore, at time t_e, since no robot changed side (that is, no robot that at time t_{dis} was in $\mathcal{S}^+$ is now in the other side, and viceversa), the configuration is still balanced.

Since n is odd and no robot left K_m between time t_{dis} and t_e, an odd number of robots must lie on K_m at t_e. If $\mathbb{D}_{t_e}$ is *quasi-final* or *final*, the lemma follows with $t_g = t_e$.

Let $\mathcal{TC}_{t_e}$ =*false*. By Lines 5–10 of routine `GetUnbalanced()`, the topmost robot on K_m, top^*, is allowed to move, and until it does not leave K_m, all the other robots cannot move (Line 7 of routine `GetUnbalanced()`). top^* chooses as destination (by executing routine `ChooseDestination()`) a point p^* in the side to its (local) right. By definition of routines `ChooseDestination()` and `MoveInside`$\mathcal{S}^+$`()`, p^* is either one of the points in $\tau(\mathbb{P}, \mathbb{D}')$, or a point on K^+.

[6] Recall that routine $\texttt{Angle}(p, q)$ returns the convex angle between the horizontal axis through q and the segment $[p, q]$.

Let t^* be the first time top^* leaves K_m towards p^*, and $\mathbb{D}_{t^*}$ becomes unbalanced. As long as top^* does not reach p^* or K^+, Lines 35–38 force all the other robots, but two, to stay still: the only two robots allowed to move are $top^{\mathbb{A}}_{\mathcal{M}} = top^{\mathbb{D}_{t^*}}_{\mathcal{M}}$ and $top^{\mathbb{A}}_{\mathcal{L}} = top^{\mathbb{D}_{t^*}}_{\mathcal{L}}$; for brevity, call them top^+ and top^-, respectively. In fact, as long as $\texttt{Angle}(top^+, top^-) \neq \texttt{Angle}(top^{\mathbb{P}}_{\mathcal{M}}, top^{\mathbb{P}}_{\mathcal{L}})$, routine `FixOutermosts()` will move either top^+ or top^- upwards until

$$\texttt{Angle}(top^-, top^-) = \texttt{Angle}(top^{\mathbb{P}}_{\mathcal{M}}, top^{\mathbb{P}}_{\mathcal{L}}). \tag{2}$$

Note that, the two points on K^+ and K^- that satisfy Equation (2) are p^+ and p^-. Since $\mathcal{TC}_{t_e} = false$ and $\mathbb{D}_{t_e}$ is empty, at time t_e test `AllOn()` (Appendix A.7) at Line 35 returns *true*; hence Lines 35–38 are executed. Observe that

1. at time t^* there are no other robots strictly inside the sides; furthermore, if p^* is on K^+, by definition of $\texttt{MoveInside}\mathcal{S}^+\texttt{()}$ (in Appendix A.3), p^* is chosen with no robot on it, and below top^+;
2. top^+ and top^- only move upwards;
3. during the movements of top^*, top^+ and top^-, no other robots is allowed to move.

Thus, during these movements any collision is avoided. Therefore, in a finite number of cycles, at time t_g, top^* reaches either p^* or K^+, and top^+ and top^- will be on p^+ and p^-, respectively. Furthermore, at t_g all robots are in $\mathbb{I}(t_g)$, and the lemma follows. □

Corollary 5.1. *At time t_g, test* `AllOn()` *returns false.*

Lemma 5.3. *Let t_{dis} be as defined in Lemma 5.1 and at that time let $\mathcal{TC}_{t_{dis}} = false$. Then in a finite number of cycles, say at time $t_u \geq t_{dis}$, the robots reach an unbalanced configuration where $top^{\mathbb{D}_{t_u}}_{\mathcal{M}}$ and $top^{\mathbb{D}_{t_u}}_{\mathcal{L}}$ are on p^+ and p^-, respectively. Furthermore, until this time any collision is avoided, and at t_u test* `AllOn()` *returns false, and all robots are in $\mathbb{I}(t_u)$.*

Proof. If $\mathbb{D}_{t_{dis}}$ is balanced, the lemma trivially follows by previous Lemma 5.2 and Corollary 5.1.

If $\mathbb{D}_{t_{dis}}$ is unbalanced, the robots execute routine `FixOutermosts()` (Line 32). If at time t_{dis} $\texttt{Angle}(top^{\mathbb{D}_{t_{dis}}}_{\mathcal{M}}, top^{\mathbb{D}_{t_{dis}}}_{\mathcal{L}}) = \texttt{Angle}(top^{\mathbb{P}}_{\mathcal{M}}, top^{\mathbb{P}}_{\mathcal{L}})$, then the lemma trivially follows.

Otherwise, let us denote $top^+ = top^{\mathbb{D}_{t_{dis}}}_{\mathcal{M}}$ and $top^- = top^{\mathbb{D}_{t_{dis}}}_{\mathcal{L}}$. As long as top^+ and top^- are not on p^+ and p^-, respectively (i.e., $\texttt{Angle}(top^+, top^-) \neq \texttt{Angle}(top^{\mathbb{P}}_{\mathcal{M}}, top^{\mathbb{P}}_{\mathcal{L}})$), routine `FixOutermosts()` will move either top^+ or top^- upwards until $\texttt{Angle}(top^+, top^-) = \texttt{Angle}(top^{\mathbb{P}}_{\mathcal{M}}, top^{\mathbb{P}}_{\mathcal{L}})$. During this time, no other robot is allowed to move; hence no collision occurs.

In a finite number of cycles, say at time t_h, top^+ and top^- reach their positions, p^+ and p^-, respectively. At this time, if `AllOn()` returns *false*, the lemma clearly follows. Otherwise, all robots but one, r, are on $K^+ \equiv \Phi^{\mathbb{D}_{t_h}}_{\mathcal{M}}$, $K^- \equiv \Phi^{\mathbb{D}_{t_h}}_{\mathcal{L}}$, and on $K_m \equiv \Phi^{\mathbb{D}_{t_h}}_{m}$; furthermore, r is not on one of the points in $\tau(\mathbb{P}, \mathbb{D}_{t_h})$. In this scenario, r is the only one robot allowed to move (Lines 35–38). By definition of routine $\texttt{MoveInside}\mathcal{S}^+\texttt{()}$, r chooses as destination either one of the points in final positions, or a point on K^+. Since no other robot is strictly inside $\mathcal{S}^+$ and $\mathcal{S}^-$, this movement cannot cause any collision. Furthermore, in a finite number of cycles, say at time t_u, r reaches p and `AllOn()` returns *false*; hence the lemma follows. □

From now on, we will refer to *final positions* as the points returned by routine `FindFinalPositions()` when executed on $\mathbb{D}_{t_u}$. Moreover, top^+ and top^- will denote $top^{\mathbb{D}_{t_u}}_{\mathcal{M}}$ and $top^{\mathbb{D}_{t_u}}_{\mathcal{L}}$, respectively.

Observation 5.1. By the way the set final positions is computed, at time t_u, top^+ and top^- occupy two positions, p^+ and p^-, that are in final positions. Furthermore, all the points in final positions cannot be neither above top^+ on K^+, nor above top^- on K^-, nor outside the region of the plane delimited by K^+ and K^-. Moreover, $K^+ \equiv \Phi_{\mathcal{M}}^{\mathbb{D}_{t_u}}$ and $K^- \equiv \Phi_{\mathcal{L}}^{\mathbb{D}_{t_u}}$, i.e., K^+ and K^- have not changed since t_{dis}.

Let p_b be the bottommost point final positions that lies on K^+.

Lemma 5.4. *Let t_u be as defined in Lemma 5.3 and at that time let $\mathcal{TC}_{t_u}$ =false. In a finite number of cycles, at time $t_b \geq t_u$, the bottommost robot on K^+, say r_b, is such that $r_b.y < p_b.y$. Furthermore, until this time any collision is avoided, and at time t_b all the robots are in $\mathbb{I}(t_b)$, $\mathbb{D}_{t_b}$ is unbalanced, and* `AllOn()` *is* false.

Proof. Two cases can occur.

1. At time t_u there are at least two robots on K^+. Let r_b be the bottommost robot on K^+. If $r_b.y < p_b.y$, then the lemma trivially follows. Otherwise, r_b is forced to move to a position that is below p_b, say p (Lines 39–44 of Algorithm Pond); furthermore, as long as r_b does not reach this position, all the other robots can not move. Hence, in finite time, say at time $t_b > t_h$, r_b reaches p (note that it is possible that r_b goes through p_b; however, it will not stop there). Furthermore, by Lemma 5.3, `AllOn()` is *false* at time t_u, and all movements between time t_u and t_b occur vertically on K^+; hence, `AllOn()` will continue to be false in this time interval, and the lemma follows.

2. At time t_u there is only top^+ on K^+. Note that, by Lemma 5.3, $\mathbb{D}_{t_u}$ is unbalanced. Therefore, since $n \geq 3$, there must be at least two robots strictly inside $\mathcal{S}^+$, otherwise, $n = 3$ and the configuration would be *semi-final*, a contradiction.

   Among the robots strictly inside $\mathcal{S}^+$, the robot with the smallest horizontal distance from K^+ is chosen, r (ties are broken by choosing the topmost). The algorithm forces r to move towards K^+ (Lines 45–48 of Algorithm Pond). In particular, r chooses as destination a point p on K^+ below p_b such that there are no robots on $[rp]$. The absence of robots on the chosen trajectory is required in order to avoid collisions. Since r moves towards K^+, it is always chosen in Line 46 as the only robot to move towards K^+. In a finite number of cycles, r reaches its destination on K^+. At this time, if `AllOn()` is *false*, then the lemma follows.

   Otherwise, there is only one robot r' strictly inside $\mathcal{S}^+$, and r' is not on any of the final positions. By routine `MoveInside`$\mathcal{S}^+$`()` (invoked in Line 37), this robot is the only one allowed to move: it moves towards one of the final positions (if at least one of them is strictly inside $\mathcal{S}^+$), or towards K^+. Once r' reaches its destination, at time $t_b > t_u$, `AllOn()` becomes *false*. At this time, there are at least two robots on K^+ (r and top^+), with r below p_b, and the lemma follows.

□

Let $FreeP(\mathcal{S}^+, t)$ (resp. $FreeP(\mathcal{S}^-, t)$) be the subset of points in final positions that are in $\mathcal{S}^+$ (resp. $\mathcal{S}^-$) and with no robots on them, at time t; and $FreeR(\mathcal{S}^+, t)$ (resp. $FreeR(\mathcal{S}^-, t)$) be the set of robots in $\mathcal{S}^+$ (resp. $\mathcal{S}^-$) that do not occupy points in final positions at time t.

Lemma 5.5. *Let t_b be as defined in Lemma 5.4 and at that time let $\mathcal{TC}_{t_b}$ =false. In a finite number of cycles, at time $t_l \geq t_b$, $|FreeR(\mathcal{S}^-, t_l)| = 0$. Furthermore, between time t_b and t_l, r_b*

does not move, any collision is avoided, all the robots that are on a final position do not move, `AllOn()` *is* false, $\mathbb{D}_{t_l}$ *is unbalanced, and at time* t_l *all the robots are in* $\mathbb{I}(t_l)$.

Proof. By the previous lemma, at time t_b, there is a robot on K^+, r_b, below any robot and any final positions on K^+; furthermore, `AllOn()` is *false* and $\mathbb{D}_{t_b}$ unbalanced. Since by hypothesis $\mathcal{TC}_{t_b}$ =*false*, routine `FromSidesToMedian(`$K^+, K^-, K_m, \mathcal{S}^-$`)` is executed. We observe that,

Observation 5.2. As long as $FreeR(\mathcal{S}^-, \cdot) \neq \emptyset$, only robots in $FreeR(\mathcal{S}^-, \cdot)$ are allowed to move (by routine `FromSidesToMedian()`); in particular, r_b cannot move until $FreeR(\mathcal{S}^-, \cdot) = \emptyset$.

Let us assume that at time t_b, $FreeR(\mathcal{S}^-, t_b) \neq \emptyset$ (otherwise the lemma would trivially follow). According to the algorithm, all the robots in $FreeR(\mathcal{S}^-, t_b)$ execute Algorithm `CloseToDestination()` (Line 8 of `FromSidesToMedian()`; `CloseToDestination()` has been presented in Section 3), while all the others can compute only *null movements*. With respect to Algorithm `CloseToDestination()`, the set $FreeR(\mathcal{S}^-, t_b)$ corresponds to $\mathbb{FR}$, the set $FreeP(\mathcal{S}^-, t_b)$ to $\mathbb{FT}$, and the obstacles $\mathbb{O}$ are all the robots except those in $FreeR(\mathcal{S}^-, t_b)$. Let (r, p) be the pair that satisfies Equation (1) at time t_b, among all the robots in $FreeR(\mathcal{S}^-, t_b)$ and all the points in $FreeP(\mathcal{S}^-, t_b)$. First of all, note that all the four conditions required by `CloseToDestination()` are met. In particular, among $\mathbb{FR}$ there is a total agreement on the coordinate systems; by Lemma 5.4 all the robots are in $\mathbb{I}(t_b)$ at time t_b; by Observation 5.2 the obstacles do not move (in particular, no robot in $\mathcal{S}^+$ or on K_m can enter $\mathcal{S}^-$ as long as $\mathbb{FR} \neq \emptyset$). Hence, according to Theorems 3.1 and 3.2, r will reach p in a finite number of cycles. Once r reaches p, it becomes an obstacle and the cardinality of $\mathbb{FR}$ and $\mathbb{FT}$ decreases by one, and r joins $\mathbb{O}$. Therefore, since as long as $FreeR(\mathcal{S}^-, \cdot) \neq \emptyset$ routine `FromSidesToMedian()` will be executed, we can conclude that

(A) if $0 < |FreeR(\mathcal{S}^-, t_b)| \leq |FreeP(\mathcal{S}^-, t_b)|$ then in a finite number of cycles and avoiding collisions, $|FreeR(\mathcal{S}^-, \cdot)| = 0$; let $t_1 \geq t_b$ be the first time such that $|FreeR(\mathcal{S}^-, t_1)| = 0$.

(B) if $|FreeR(\mathcal{S}^-, t_b)| > |FreeP(\mathcal{S}^-, t_b)| > 0$), then in a finite number of cycles and avoiding collisions, $|FreeP(\mathcal{S}^-, \cdot)| = 0$; let $t_2 \geq t_b$ be the first time such that $|FreeP(\mathcal{S}^-, t_2)| = 0$. Note that, at this time $|FreeR(\mathcal{S}^-, t_2)| > 0$.

If Case (A) above applies, then the lemma clearly follows, with $t_l = t_1$. Otherwise (Case (B) above applies), at time t_2 the *extra* robots in $FreeR(\mathcal{S}^-, t_2)$ that are still in $\mathcal{S}^-$ are sequentially directed towards K_m by routine `ChooseOnMedian()`(Line 10 of `FromSidesToMedian()`).

Specifically, the topmost robot in $FreeR(\mathcal{S}^-, t_2)$ with smallest horizontal distance from K_m is allowed to move towards K_m. Its destination point on K_m is chosen by routine `ChooseOnMedian()`, using a strategy that avoids collisions (see Figure 14). During this movement, r remains the (closest to K_m) topmost robot in $FreeR(\mathcal{S}^-, t_2)$; hence it is the only one allowed to move until it reaches K_m. Furthermore, by Observation 5.2, the destination point on K_m computed by r can not be the destination point of another robot in $\mathcal{S}^+$ (i.e., it can not collide on K_m with a robot coming from $\mathcal{S}^+$). Therefore, r reaches K_m in a finite number of cycles, while avoiding collisions. By iterating this argument, all the *extra* robots in $\mathcal{S}^-$ will reach K_m in a finite number of cycles while avoiding collisions, say at time t_3. In conclusion, within a finite number of cycles, at time $t_l = t_3$, all the robots in $\mathcal{S}^-$ are on one of the points in final positions; that is, $|FreeR(\mathcal{S}^-, t_l)| = 0$, and the lemma follows. □

Lemma 5.6. *Let* t_l *be as defined in Lemma 5.5 and at that time let* $\mathcal{TC}_{t_l}$ =*false. In a finite number of cycles, at time* $t_r \geq t_l$, $|FreeR(\mathcal{S}^+, t_r)| = 1$. *Furthermore, between time* t_l *and* t_r, $|FreeR(\mathcal{S}^-, \cdot)| = 0$, r_b *does not move, and any collision is avoided; at time* t_r *all the robots are in* $\mathbb{I}(t_r)$, *and* `AllOn()` *is* false.

Proof. By Lemmas 5.4–5.5, at time t_l there is a robot on K^+, r_b, below any robot and any final position on K^+. Furthermore, at this time, all the robots in $\mathcal{S}^-$ are on a final position. In this scenario, routine `FromSidesToMedian(`$K^+, K^-, K_m, \mathcal{S}^+$`)` is executed. First, we observe that, routine `FromSidesToMedian()`, when executed on $\mathcal{S}^+$, does not consider r_b as one of the robots in $FreeRobots$. Moreover,

Observation 5.3. As long as $FreeR(\mathcal{S}^+, \cdot) \setminus \{r_b\} \neq \emptyset$, only robots in $FreeR(\mathcal{S}^+, \cdot)$ are allowed to move (by routine `FromSidesToMedian()`); in particular, r_b cannot move until $FreeR(\mathcal{S}^+, \cdot) \setminus \{r_b\} = \emptyset$.

Let us assume that at time t_l, $FreeR(\mathcal{S}^+, t_l) \setminus \{r_b\} \neq \emptyset$ (otherwise the lemma would trivially follow). According to the algorithm, all the robots in $FreeR(\mathcal{S}^+, t_l) \setminus \{r_b\}$ execute Algorithm `CloseToDestination()`, while all the others can compute only *null movements*. With respect to Algorithm `CloseToDestination()`, the set $FreeR(\mathcal{S}^+, t_l) \setminus \{r_b\}$ corresponds to $\mathbb{FR}$, the set $FreeP(\mathcal{S}^+, t_l)$ to $\mathbb{FT}$, and the obstacles $\mathbb{O}$ are all the robots inside $\mathcal{S}^+$ except those in $FreeR(\mathcal{S}^-, t_l) \setminus \{r_b\}$. Let (r, p) be the pair that satisfies Equation (1) at time t_l, among all the robots in $FreeR(\mathcal{S}^+, t_l) \setminus \{r_b\}$ and all the points in $FreeP(\mathcal{S}^+, t_l)$. First of all, note that all the four conditions required by `CloseToDestination()` are met. In particular, among $\mathbb{FR}$ there is a total agreement on the coordinate systems; by Lemma 5.5 all the robots are in $\mathbb{I}(t_l)$ at time t_l; by Observation 5.3 the obstacles do not move (in particular, no robot in $\mathcal{S}^-$ or on K_m can enter $\mathcal{S}^+$ as long as $\mathbb{FR} \neq \emptyset$). Let $t' > t_l$ be the first time r moves. We now distinguish two cases, depending on the value of `AllOn()` at this time.

1. `AllOn()` is *false* at time t'. According to Theorems 3.1 and 3.2, r will reach p in a finite number of cycles.

2. `AllOn()` is *true* at time t'. By Lemma 5.5 `AllOn()` is *false* at time t_l; furthermore, no robot moves between time t_l and time t', hence `AllOn()` is *false* between time t_l and t'. Since, by hypothesis, at time t' the movement of r makes `AllOn()` *true*, this implies that, between t_l and t', r is on K^+ and no robot is strictly inside $\mathcal{S}^+$. In this case, if r does not reach p in one cycle, it will execute Line 37 in the next one. The destination of r, chosen by routine `MoveInside`$\mathcal{S}^+$`()`, will still be p. Hence, in a finite number of cycles, r reaches p avoiding collisions (since no other robot was strictly inside $\mathcal{S}^+$ at t'), and `AllOn()` becomes *false*.

Once r reaches p, it becomes an obstacle and the cardinality of $\mathbb{FR}$ and $\mathbb{FT}$ decreases by one, and r joins $\mathbb{O}$. Therefore, since as long as $FreeR(\mathcal{S}^+, \cdot) \setminus \{r_b\} \neq \emptyset$ routine `FromSidesToMedian()` is executed, we can conclude that

(A) if $0 < |FreeR(\mathcal{S}^+, t_l) \setminus \{r_b\}| \leq |FreeP(\mathcal{S}^+, t_l)|$ then in a finite number of cycles and avoiding collisions, $|FreeR(\mathcal{S}^+, \cdot)| = 1$; let $t_1 \geq t_l$ be the first time such that $|FreeR(\mathcal{S}^+, t_1)| \setminus \{r_b\} = 0$.

(B) if $|FreeR(\mathcal{S}^+, t_l)| \setminus \{r_b\} > |FreeP(\mathcal{S}^-, t_l)| > 0)$, then in a finite number of cycles and avoiding collisions, $|FreeP(\mathcal{S}^+, \cdot)| = 0$; let $t_2 \geq t_l$ be the first time such that $|FreeP(\mathcal{S}^+, t_2)| = 0$. Note that, at this time $|FreeR(\mathcal{S}^+, t_2)| \setminus \{r_b\} > 0$.

In both cases, applying an argument similar to the one adopted in Lemma 5.5, and recalling that r_b is never allowed to move by routine `FromSidesToMedian(`$\cdot, \cdot, \cdot, \mathcal{S}^+$`)`, the lemma follows. □

Note that, at time t_r the only robots, if any, that might not be on final positions, in addition to r_b, are those on K_m.

Lemma 5.7. *Let t_r be as defined in Lemma 5.6 and at that time let $\mathcal{TC}_{t_r}$ =false. In a finite number of cycles, at time $t_s \geq t_r$, $|FreeP(\mathcal{S}^+, t_s)| \leq 1$. Furthermore, between time t_r and t_s, $|FreeR(\mathcal{S}^-, \cdot)| = 0$, r_b does not move,* `AllOn()` *is* false*, and any collision is avoided; at time t_s all the robots are in $\mathbb{I}(t_s)$, and $|FreeR(\mathcal{S}^+, t_s)| = 1$*

Proof. At time t_r, $FreeR(\mathcal{S}^-, t_r) = \emptyset$, and $|FreeR(\mathcal{S}^+, t_r)| = 1$ (r_b is in $\mathcal{S}^+$). In this scenario, test in Line 53 fails; hence, `FromMedianToSides`$(K^+, K^-, K_m, \mathcal{S}^+)$ is executed. If $|FreeP(t_r, \mathcal{S}^+)| \leq 1$, then the lemma trivially follows. Let $k = |FreeP(t_r, \mathcal{S}^+)| > 1$. Observe that at this time there must be at least $k-1$ free robots on K_m, (since there are no free robots in $\mathcal{S}^-$).

Only the topmost robot on K_m, say *top*, is allowed to move towards one of the points in $FreeP(\mathcal{S}^+, t_r)$ (Lines 3–5 of routine `FromMedianToSides()`). Since $|\mathcal{S}^+| \geq |\mathcal{S}^-| + 1$ (recall that r_b is in $\mathcal{S}^+$ at this time), when *top* leaves K_m (towards $\mathcal{S}^+$), $\mathcal{S}^+$ clearly stays the side with more robots inside; hence, the global agreement on the orientation of the x axis of *GCS* does not change. Moreover, when *top* enters $\mathcal{S}^+$, $|\mathcal{S}^+| \geq |\mathcal{S}^-| + 2$; hence, `AllOn()` will be *false*.

top moves according to routine `CloseToDestination`$(\{top\}, FreeP(t_r, \mathcal{S}^+), K^+, K^-)$ (Line 4 in routine `FromMedianToSides()`). With respect to Algorithm `CloseToDestination()`, the set $\{top\}$ corresponds to $\mathbb{FR}$, the set $FreeP(\mathcal{S}^+, t_r)$ to $\mathbb{FT}$, and the obstacles $\mathbb{O}$ are all the robots except *top*. Let (top, p) be the pair that satisfies Equation (1) at time t_r, among all the points in $FreeP(\mathcal{S}^+, t_r)$. First of all, note that all the four conditions required by `CloseToDestination()` are met. In particular, by Lemma 5.6 all the robots are in $\mathbb{I}(t_r)$ at time t_r, and at time t_r only *top* is allowed to move.

As soon as *top* moves towards p, then $FreeR(\mathcal{S}^+, \cdot) \setminus \{r_b\} = \{top\} \neq \emptyset$; therefore, routine `FromSidesToMedian`$(K^+, K^-, K_m, \mathcal{S}^+)$ in Line 53 still allows only *top* to keep moving towards p; this movement is once again controlled in routine `CloseToDestination`$(\{top\}, FreeP(t_r, \mathcal{S}^+), K^+, K^-)$. Furthermore, by Observation 5.3, the four conditions required by `CloseToDestination()` are met as long as *top* moves towards p. Hence, according to Theorems 3.1 and 3.2, *top* will reach p in a finite number of cycles. Moreover, as soon as *top* reaches p, say at time $t_1 > t_r$, the only robot in $\mathcal{S}^+$ that is not on one of the final positions is r_b; hence, $|FreeR(\mathcal{S}^+, t_1)| = 1$.

Therefore, if the number of robots on K_m and not on a final position at time t_r is equal to $|FreeP(\mathcal{S}^+, t_r)| - 1$, in a finite number of cycles, say at t_s, there are no more robots on K_m, and the configuration becomes *semi-final* (with the *last* available final position in $\mathcal{S}^+$), and $|FreeP(\mathcal{S}^+, t_s)| = 1$.

Otherwise (the number of robots on K_m and not on a final position at time t_r is greater than or equal to $|FreeP(\mathcal{S}^+, t_r)|$), by iterating the above argument, in a finite number of cycles, say at time t_s, $|FreeP(\mathcal{S}^+, t_s)| = 0$, $|FreeR(\mathcal{S}^+, t_s) = 1$.

Finally, between time t_r and t_s, routine `FromMedianToSides`$(\cdot, \cdot, \cdot, \mathcal{S}^+)$ allows to move only robots from K_m towards $\mathcal{S}^+$, and none of these robots is allowed to move inside $\mathcal{S}^-$; hence, since $|FreeR(\mathcal{S}^-, t_r)| = 0$ (Lemma 5.6), it follows that $|FreeR(\mathcal{S}^-, t)| = 0$, for all $t_r \leq t \leq t_s$, and the lemma follows. □

Lemma 5.8. *Let t_s be as defined in Lemma 5.7 and at that time let $\mathcal{TC}_{t_s}$ =false. In a finite number of cycles, at time $t_z \geq t_s$, $|FreeP(\mathcal{S}^-, t_z)| \leq 1$. Furthermore, between time t_s and t_z, $|FreeR(\mathcal{S}^+, \cdot)| = 1$, $|FreeP(\mathcal{S}^+, \cdot)| = 0$, r_b does not move,* `AllOn()` *is* false*, and any collision is avoided; at time t_z all the robots are in $\mathbb{I}(t_z)$ and $|FreeR(\mathcal{S}^-, t_z)| = 0$.*

Proof. By Lemma 5.7, $FreeR(\mathcal{S}^-, t_s) = \emptyset$ and $|FreeR(\mathcal{S}^+, t_s)| = 1$ (r_b is in $\mathcal{S}^+$); furthermore, since by hypothesis $\mathcal{TC}_{t_s}$ =*false*, it follows by previous lemma that $|FreeP(\mathcal{S}^+, t_s)| = 0$,

otherwise the configuration would be *semi-final*.

In this scenario, routine `FromMedianToSides`$(K^+, K^-, K_m, \mathcal{S}^-)$ is executed. If $|FreeP(\mathcal{S}^-, t_s)| \leq 1$, then the lemma trivially follows. Let $k = |FreeP(\mathcal{S}^-, t_s)| > 1$. Observe that there must be at least $k-1$ robots on K_m.

Only the topmost robot on K_m, say *top*, is allowed to move towards one of the points in $FreeP(\mathcal{S}^-, t_s)$ (Lines 3–5 of routine `FromMedianToSides()`). Since $|FreeP(\mathcal{S}^+, t_s)| = 0$, $|FreeR(\mathcal{S}^-, t_s)| = 0$, and r_b is in $\mathcal{S}^+$ as long as routine `FromMedianToSides`$(K^+, K^-, K_m, \mathcal{S}^-)$ is executed (i.e. as long as there are free positions in $\mathcal{S}^-$ and there is a robot on K_m), when *top* leaves K_m, $|\mathcal{S}^+| \geq |\mathcal{S}^-| + 1$; hence, $\mathcal{S}^+$ is still the side with more robots inside, and the global agreement on the orientation of the x axis of *GCS* does not change. Moreover, since *top* moves inside $\mathcal{S}^-$, `AllOn()` will be *false*.

By using arguments similar to the ones adopted in the proof of the previous lemma, we can conclude the following:

1. If the number of robots on K_m and not on a final position at time t_s is equal to $|FreeP(\mathcal{S}^-, t_s)| - 1$, in a finite number of cycles, say at time t_z, there are no more robots on K_m, and the configuration becomes *semi-final* (with the *last* available final position in $\mathcal{S}^-$), and $FreeP(\mathcal{S}^-, t_z) = 1$.

2. Otherwise (the number of robots on K_m and not on a final position at time t_s is greater than or equal to $|FreeP(\mathcal{S}^-, t_s)|$), in a finite number of cycles, say at time t_z, $|FreeP(\mathcal{S}^-, t_z)| = 0$.

Furthermore, $|FreeR(\mathcal{S}^+, t)| = 1$ and $|FreeP(\mathcal{S}^+, t)| = 0$, for all $t_s \leq t \leq t_z$, `AllOn()` is *false*, and the lemma follows. □

From the proofs of Lemmas 5.7 and 5.8, we can state the following

Corollary 5.2. *Let t_z be as defined in Lemma 5.8. $\mathbb{D}_{t_z}$ is* semi-final.

Finally, we deal with the terminal cases; i.e., when the current configuration is either *semi-final*, or *quasi-final*.

Lemma 5.9. *Let $\mathbb{D}_{t_{sf}}$ be a* semi-final *configuration of the robots at a given time $t_{sf} \geq t_0$ such that all the robots are in $\mathbb{I}(t_{sf})$. In a finite number of cycles, at time t_{qf}, the configuration becomes* quasi-final *or final avoiding any collisions. Furthermore, at time t_{qf} all the robots are in $\mathbb{I}(t_{qf})$.*

Proof. First note that $\mathbb{D}_{t_{sf}}$ is neither *final* nor *quasi-final*. Therefore, `TestSF`$(\mathbb{D}_{t_{sf}})$ returns $= (true, r, last)$, and Lines 7–22 are executed. By Definition 5.2 of *semi-final* configuration, $\mathbb{D}_{t_{sf}}$is non-degenerated and unbalanced. We distinguish the possible cases, according to the definition of *semi-final*.

1. $(r \neq \emptyset, last \neq \emptyset)$. We distinguish the two possible cases.

   (a) If r and *last* are in the same side, then Algorithm `CloseToDestination()` is called. According to this algorithm, r moves towards *last*. Furthermore, during this movement, no other robot is allowed to move (note that any other robot is either on a final position or on K_m). By Theorems 3.1 and 3.2, in a finite numebr of cycles, say at time $t_{qf} > t_{sf}$, r reaches *last*.

(b) If r and *last* are not in the same side, then Line 12 allows r to move towards K_m, while any other robot is forced to not move. Therefore, since during this movement the configuration stays *semi-final*, in a finite number of cycles r reaches K_m. At this time, the configuration is still *semi-final*, but now the *semi-final* test in Line 6 returns $(true, \emptyset, last)$, and Case 3. below applies.

2. $(r \neq \emptyset, last = \emptyset)$. By Lines 16–18 of Algorithm POND, r is the only robot allowed to move, and its destination is a point on K_m. While r approaches K_m the configuration stays *semi-final*; hence, in a final number of cycles, say at time $t_{qf} > t_{sf}$, r reaches K_m.

3. $(r = \emptyset, last \neq \emptyset)$. In this case, Lines 20–22 allows *top*, the topmost robot on K_m, to move towards *last*. Let t be the first time when *top* leaves K_m towards *last*. During its movement, the configuration clearly remains *semi-final*, but now the *semi-final* test in Line 6 returns $(true, r \neq \emptyset, last)$. If *top* does not reach *last* in one cycle, then Lines 10–14 are executed at its next cycle; in particular, *top* and *last* lie in the same side, and previous Case 1.a above applies.

In all cases, $\mathbb{D}_{t_{qf}}$ is either *final* or *quasi-final*; furthermore, between time t_{sf} and t_{qf} no other robot moves besides r, and at time t_{qf} all the robots are in $\mathbb{I}(t_{qf})$, and the lemma follows. □

Before proceeding with the overall sequence of lemmas, let us establish a useful property of routine `MoveCarefully(`$\mathbb{R}, \mathbb{T}$`)` (invoked by `Fix()`) described in Appendix A.2.

Property 5.1. *Let $\mathbb{R}$ and $\mathbb{T}$ be a set of robots and targets, respectively, that at are all on the same vertical axis K at a given time t; furthermore, let $|\mathbb{R}| = |\mathbb{T}|$. If after t no other robot enters on K, then* `MoveCarefully(`$\mathbb{R}, \mathbb{T}$`)` *let in a finite number of cycles each robot in $\mathbb{R}$ to reach one of the targets in $\mathbb{T}$. Furthermore, until that time any collision is avoided, and at time t' all robots in $\mathbb{R}$ are in $\mathbb{I}(t')$.*

Proof. First, routine `MoveCarefully()` sorts topdown both input sets; let *SortedR* and *SortedP* be the result of the sorting on robots and targets, respectively.

Then, i-th robot in *SortedR*, r_i, is assigned as target the i-th target in *SortedP*, p_i. However, according to the routine, r_i will not start moving as long as there is a robot j, with $j > i$ on $[r_i, p_i]$.

Let us define the *waiting graph* $WG = (V, E)$ as follows. The nodes in this graph are the robots in *SortedR*, and there is an edge between r_i and r_j if r_j is on r_i's way; that is, $r_j \in [r_i, p_i]$.

First note that, if $(r_i, r_j) \in E$, then $(r_j, r_i) \notin E$. In fact, let us assume that r_i is above r_j; i.e., $i < j$. The edge $(r_i, r_j) \in E$ implies that r_i is waiting to reach the point p_i that is below r_j; since the points are sorted, p_j must be below p_i, and $(r_j, r_i) \notin E$ (a symmetrical argument applies if $i > j$.

In order to show that each robot reaches its assigned target in a finite number of cycles, it sufficient to prove that WG contains no cycles. By contradiction, let us assume there is a cycle C in WG, and, without loss of generality, let $C = r_i \to r_j \to \cdots \to r_k \to r_z \to r_k$, with $i < j < \cdots < k < (k+1) < \cdots < z$. The presence of the edges $r_k \to r_{k+1}$ and $r_z \to r_k$, with $k < z$, implies that r_k is waiting to reach point p_k below r_{k+1}, and that r_z is waiting to reach point p_z above p_k. This is a contradiction, since p_z clearly is not the z-th point in *SortedP*. □

The following lemma states that, if a configuration is *quasi-final*, then the arbitrary pattern formation problem is solved in a finite number of cycles.

Lemma 5.10. *Let $\mathbb{D}_{t_{qf}}$ be a non-degenerated* quasi-final *configuration of the robots at a given time $t_{qf} \geq t_0$ such that all the robots are in $\mathbb{I}(t_{qf})$. In a finite number of cycles, at time t_f, the configuration becomes* final *avoiding any collisions. Furthermore, at time t_f all the robots are in $\mathbb{I}(t_{qf})$.*

Proof. By definition of *quasi-final* configuration (Definition 5.3), $|K_m| = |\Upsilon_m|$.

In such a configuration, all robots inside $\mathcal{S}^+$ and $\mathcal{S}^-$ occupy a position of $\tau(\mathbb{P}, \mathbb{D}_{t_{qf}}) \cup \tau^{-1}(\mathbb{P}, \mathbb{D}_{t_{qf}})$.

Let us define

$$Targets = \begin{cases} \tau(\mathbb{P}, \mathbb{D}_{t_{qf}}) & \text{if } \tau(\mathbb{P}, \mathbb{D}_{t_{qf}}) \neq \emptyset \\ \tau^{-1}(\mathbb{P}, \mathbb{D}_{t_{qf}}) & \text{otherwise} \end{cases}$$

Let $Targets'$ be the subset of $Targets$ containing the points that are on K_m.

At time t_{qf}, the robots are in a *quasi-final* configuration, and routine `Fix()` is invoked. This routine first computes the set of final positions and of robots on K_m, and then calls `MoveCarefully()`. According to this routine, the robots on K_m are the only ones allowed to move. Furthermore, these robots and the final positions on K_m are sorted topdown (routine `Sort()`), and the i-th robot in this ordering chooses as its destination the i-th point in $Targets'$. By previous Property 5.1, in a finite number of cycles a final configuration is reached, and the lemma follows. □

From Lemmas 5.1–5.10, we conclude:

Theorem 5.1. *With Algorithm* Pond*, the robots correctly form the input pattern $\mathbb{P}$.*

Result 2. *When one axis direction and orientation is commonly agreed upon, an odd number of autonomous, anonymous, oblivious, mobile robots can form any arbitrary given pattern. An even number of robots cannot form any arbitrary given pattern.*

5.3 Case b.: Degenerated Pattern

If $\Upsilon^+ \equiv \Upsilon_m$, the points in the pattern lie all on the same vertical line. In this section, we present Algorithm Pod (whose pseudo-code is reported in Algorithm 5), that solves the Apf problem with Partial agreement and an Odd number of robots that have to form a Degenerated pattern. The idea is briefly described in the following. First, the robots reach an unbalanced configuration, by executing routine `GetUnbalanced2()` (Appendix B.2). This routine is very similar to `GetUnbalanced()` defined in Algorithm Pond; the only difference is in Line 5. Let $\mathcal{S}^+$ and $\mathcal{S}^-$ be the sides with more and less robots in such configuration, respectively. Moreover, let K^+ and K^- the two vertical axis tangent the unbalanced configuration, and that lie in $\mathcal{S}^+$ and $\mathcal{S}^-$, respectively.

Then, all the robots inside $\mathcal{S}^+$ move sequentially to K^+ (from the topmost with the smallest horizontal distance from K^+, routine `Towards(`$\mathcal{S}^+$`,`K^+`)`). After this, all the robots on K_m move sequentially (from the topmost to the bottommost, `Towards(`K_m, K^+`)`) to K^+; and finally all the robots inside $\mathcal{S}^-$ move sequentially to K^+ (`Towards(`$\mathcal{S}^-$`,`K^+`)`).

Eventually, all the robots are on the same vertical axis, K_m: we call *quasi-final* such a configuration. At this point, routine `LastFix(`$\mathbb{D}, K_m$`)` is called, that uses a strategy similar to `Fix()` of Algorithm Pond to place all the robots in their correct positions.

`FindFinalPositions2()` computes the set of points that the robots have to reach in order to correctly form the input pattern, after they are all on the same vertical axis K. In particular, let

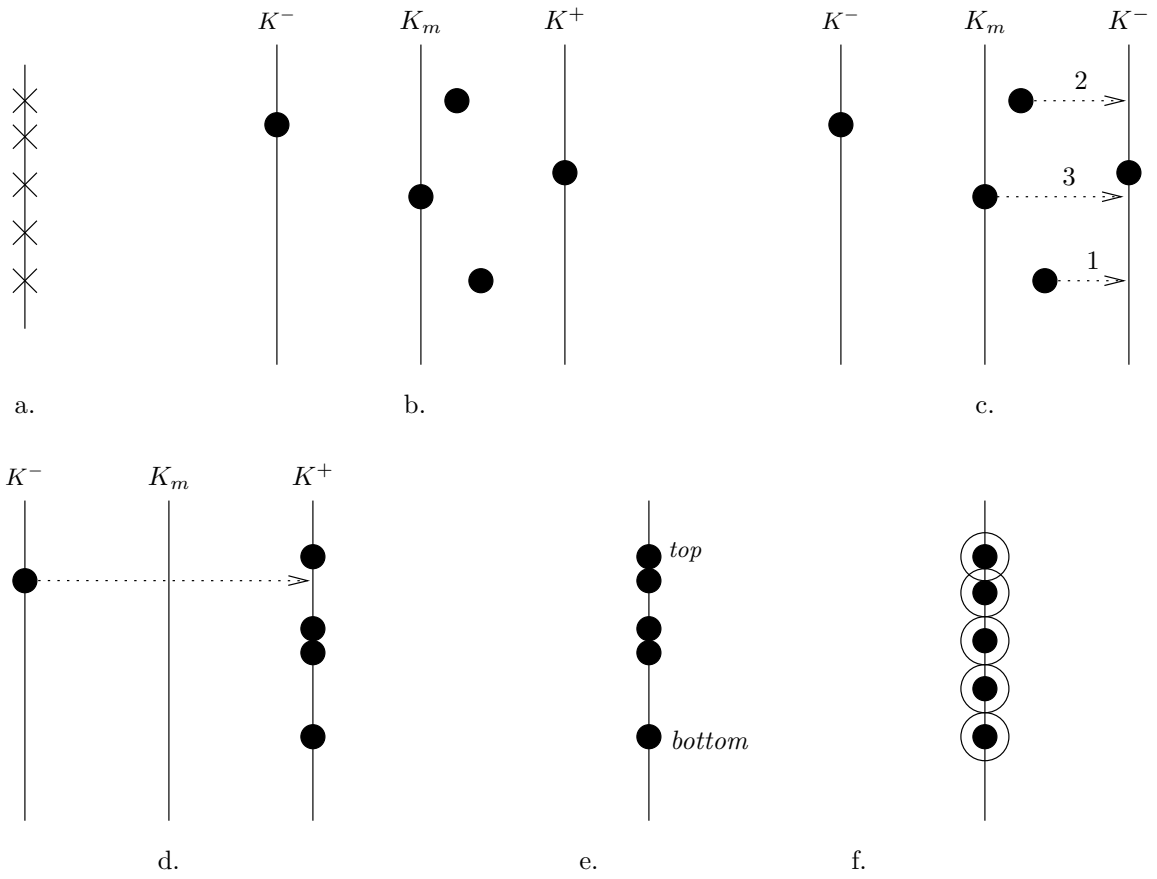

Figure 11: An example of the behavior of Algorithm 5. (a) The input patterns. (b) The initial configuration $\mathbb{D}_0$. (c) `Towards(`$\mathcal{S}^+$`,`K^+`)` and `Towards(`K_m, K^+`)`. (d) `Towards(`$\mathcal{S}^-$`,`K^+`)`. (e) All the robots on the same vertical axis. (f) The final configuration.

Algorithm 5 POD

Input: An arbitrary pattern $\mathbb{P}$ described as a sequence of points $p_1, \ldots, p_n$, given in lexicographic order, such that $\Upsilon^+ \equiv \Upsilon_m$. The direction and orientation of the y axis is common knowledge.

$(\Upsilon^+, \Upsilon^-, \Upsilon_m) := (\Phi^{\mathbb{P}}_{\mathcal{M}}, \Phi^{\mathbb{P}}_{\mathcal{L}}, \Phi^{\mathbb{P}}_m)$;
$\mathbb{D}$:= Current Configuration Of The Robots;
If $\mathbb{D}$ Is Final Configuration **Then STOP**.
$K_m := \Phi^{\mathbb{D}}_m$;
5: **If** $\mathbb{D}$ is *quasi-final* **Then** `LastFix(`$\mathbb{D}, K_m$`)`; **EndC**.
If $\mathbb{D}$ Is Unbalanced **Then** $(K^+, K^-) := (\Phi^{\mathbb{D}}_{\mathcal{M}}, \Phi^{\mathbb{D}}_{\mathcal{L}})$;
Else `GetUnbalanced2(`$\mathbb{D}$`)` **EndC**.
$(\mathcal{S}^+, \mathcal{S}^-)$:= Sides of $\mathbb{D}$ where K^+ and K^- lie, respectively;
FinalPositions :=`FindFinalPositions2(`$\mathbb{P}, K^+$`)`;
10: **If** `AllOn()` **Then**
r := Robot Strictly Inside $\mathcal{S}^+$;
p := Point on K^+ With No Robots On It And Below The Topmost On K^+;
If I Am r **Then** `destination`:= p; **EndC**. **Else** `destination`:= `null`; **EndC**.
$\mathbb{SI}$:= {Robots Strictly Inside $\mathcal{S}^+$ };
15: **If** $|\mathbb{SI}| \neq 0$ **Then** `Towards(`$\mathcal{S}^+, K^+$`)`; **EndC**.
$\mathbb{SI}$:= {Robots on K_m};
If $|\mathbb{SI}| \neq 0$ **Then** `Towards(`K_m, K^+`)`; **EndC**.
$\mathbb{SI}$:= {Robots Strictly Inside $\mathcal{S}^-$ };
If $|\mathbb{SI}| \neq 0$ **Then** `Towards(`$\mathcal{S}^-, K^+$`)`; **EndC**.

h be the distance between the topmost (say $top^{\mathbb{P}}$) and the bottommost (say $bottom^{\mathbb{P}}$) robot on $\mathbb{P}$. Then, we define the transform $\tau(\mathbb{P}, K)$ that returns the set of points obtained by scaling $\mathbb{P}$ so that h is equal to distance between *top* (the topmost robot on K) and *bottom* (the bottommost robot on K), and translating it so that $top^{\mathbb{P}}$ is mapped onto *top* and $bottom^{\mathbb{P}}$ onto *bottom*. `FindFinalPositions2()` returns such set of points.

Routines `Towards()` and `LastFix()` are reported in detail in Appendix B.

Lemma 5.11. *Let $\mathbb{D}_{t_0}$ be non-degenerated. Then in a finite number of cycles, say at time $t_u \geq t_0$, the robots reach an unbalanced empty configuration. Furthermore, until this time any collision is avoided, and all robots are in $\mathbb{I}(t_u)$.*

Proof. If $\mathbb{D}_{t_0}$ is unbalanced, the lemma trivially follows. Let $\mathbb{D}_{t_0}$ be balanced. This implies that the robots execute routine `GetUnbalanced2()` (Line 7 of Algorithm POD). In particular, the only robots that are allowed to move are those not on $K_m \cup K^+ \cup K^-$, starting the *sweeping* process analyzed in Lemma 5.2. Hence, in a finite number of cycles, say at time $t_e \geq t_0$, all the robots will be either on K^+, or on K_m, or on K^-; i.e., $\mathbb{D}_{t_e}$ is empty (that is, all robots lie on K^+, K^-, or on K_m). Furthermore, at this time all robots are in $\mathbb{I}(t_e)$ and $\mathbb{D}_{t_e}$ is still balanced (and not final).

Since n is odd and no robot left K_m between time t_0 and t_e, an odd number of robots must lie on K_m at t_e.

By Lines 4–7 of routine `GetUnbalanced2()`, the topmost robot on K_m, top^*, is allowed to move, and until it does not leave K_m, all the other robots cannot move (Line 7 of routine `GetUnbalanced()2`). top^* chooses as destination a point p^* on K^+ below the topmost robot on

K^+.

Let t^* be the first time top^* leaves K_m towards p^*, and $\mathbb{D}_{t^*}$ becomes unbalanced.

Since $\mathbb{D}_{t_e}$ was empty, at time t^* test `AllOn()` at Line 10 returns *true*; hence Lines 11–13 are executed: top^* is the only robot allowed to move, and it keeps moving towards p^* on K^+. Thus, during the movements of top^* any collision is avoided. Therefore, in a finite number of cycles, at time t_u, top^* reaches p^* on K^+. Furthermore, at t_u all robots are in $\mathbb{I}(t_u)$, and the lemma follows. □

Lemma 5.12. *Let $\mathbb{D}_t$, $t \geq 0$, be a configuration where* `AllOn()` *is* true*, and such that all robots are in $\mathbb{I}(t)$. Then, in a finite number of cycles, at time t_z, (1)* `AllOn()` *returns false; (2) $|\mathcal{S}^+| = |\mathcal{S}^-| + 1$; and (3) $\mathbb{D}_{t_z}$ is empty. Furthermore, at t_z all robots are in $\mathbb{I}(t_z)$.*

Proof. If `AllOn()` is *true* at time t, then at this time $|\mathcal{S}^+| = |\mathcal{S}^-| + 1$, there are no robots strictly inside $\mathcal{S}^-$, and only one robots, r, is strictly inside $\mathcal{S}^+$. By definition of $\tau(\mathbb{P}, K^+)$, clearly r is not on one of the points in $\tau(\mathbb{P}, K^+)$ (these points are all on K^+).

Therefore, Lines 11–13 are executed: r is the only robot allowed to move, and it moves towards K^+, on a point between the topmost and the bottomost robot on K^+. During this movement, no other robot is allowed to move. Hence, in a finite number of cycles, r reaches K^+, and the lemma follows. □

Lemma 5.13. *Let $\mathbb{D}_{t_0}$ be non-degenerated. Then in a finite number of cycles, say at time $t_s \geq t_0$, the robots reach a* quasi-final *configuration. Furthermore, until this time any collision is avoided, and all robots are in $\mathbb{I}(t_s)$.*

Proof. First, in a finite number of cycles, at time t_a, the robots reach a configuration that is unbalanced and where `AllOn()` returns *false*. In fact, if $\mathbb{D}_{t_0}$ is balanced, this follows by Lemmas 5.11 and 5.12, with $t_a = t_u$. If $\mathbb{D}_{t_0}$ is unbalanced and `AllOn()` returns *true*, then, by Lemma 5.12, $t_a = t_z$.

At time t_a, the following happens:

1. First, routine `Towards(`$\mathcal{S}^+, K^+$`)` moves sequentially the robots strictly inside $\mathcal{S}^+$ towards K^+, in such a way that any collision is avoided. Therefore, in a finite number of cycles, there are no robots strictly inside $\mathcal{S}^+$.

   Note that, if at time t_a there are no robots strictly inside $\mathcal{S}^-$, $|\mathcal{S}^+| = |\mathcal{S}^-| + 1$, and there are at least two robots strictly inside $\mathcal{S}^+$, when the second last robot strictly inside $\mathcal{S}^+$ reaches K^+, then `AllOn()` returns *true*. However, by Lemma 5.12, the last robot strictly inside $\mathcal{S}^+$ will reach K^+ in a finite number of cycles, say at t^*; at that time `AllOn()` returns *false*. Moreover, at t^* there are no more robots strictly inside $\mathcal{S}^+$, and at this time $|\mathcal{S}^+| = |\mathcal{S}^-| + 1$. From now on (as described in the following), robots on K_m and inside $\mathcal{S}^-$ will enter $\mathcal{S}^+$; therefore, after t^*, $|\mathcal{S}^+| > |\mathcal{S}^-| + 1$, and `AllOn()` will never return *true* again.

2. Second, routine `Towards(`K_m, K^+`)` moves sequentially the robots on K_m towards K^+, in such a way that any collision is avoided. Therefore, in a finite number of cycles, $|K_m| = 0$. Note that, during these moves, `Towards(`$\mathcal{S}^+$`,`K^+`)` will be invoked, and `AllOn()` returns always *false*, since the number of robots in $\mathcal{S}^+$ increases.

3. Finally, routine `Towards(`$\mathcal{S}^-, K^+$`)` moves sequentially the robots strictly inside $\mathcal{S}^-$ towards K^+, in such a way that any collision is avoided. Note that, after a robot in $\mathcal{S}^-$ reaches K_m and K^+, routines `Towards(`$\mathcal{S}^+$`,`K^+`)`and `Towards(`K_m, K^+`)` can be possibly be invoked.

Therefore, in a finite number of cycles, there are no robots strictly inside $\mathcal{S}^-$. furthermore, until then, `AllOn()` returns always *false*, since the number of robots in $\mathcal{S}^+$ increases.

When the last robot on K^- moves towards K^+, both K^- and K_m change their positions. However, since the robots on K^+ do not move, the agreement on $\mathcal{S}^+$ and $\mathcal{S}^-$ does not change. Therefore, in a finite number of cycles, all the robots are on the same vertical axis, and the lemma follows. □

Lemma 5.14. *Let $\mathbb{D}_t$ be* quasi-final*, $t \geq 0$, and let all robots be in $\mathbb{I}(t)$. Then in a finite number of cycles, say at time $t_f > t$, the robots reach a* final *configuration. Furthermore, until this time any collision is avoided, and all robots are in $\mathbb{I}(t_f)$.*

Proof. In this scenario, routine `LastFix(`$\mathbb{D}, K$`)` is executed, and by Property 5.1 in a finite number of cycles a final configuration is reached, and the lemma follows. □

By previous lemmas, we can state the following

Theorem 5.2. *If $\Upsilon^+ \equiv \Upsilon_m$, then in a finite number of cycles the robots are in a final configuration, by executing Algorithm* Pod*.*

6 Partial Agreement: The Even Case

6.1 Characterization

We know from Section 2.4 that an arbitrary pattern can not be formed by an even number of robots (Corollary 2.1). In this section, we are interested in determining which class of patterns, if any, can be formed in this case. From now on, we will assume that the robots in the system have common agreement on the direction and orientation of only the y axis[7], and that the number n of robots in the system is even.

We say that $\mathbb{P}$ is a *symmetric pattern* if it has at least one axis of symmetry Λ; that is, for each $p \in \mathbb{P}$ there exists exactly another point $p' \in \mathbb{P}$ such that p and p' are symmetric with respect to Λ (see Figure 12.b, c and d).

The proof of the unsolvability result of Theorem 2.2 is useful to better understand what kind of patterns can not be formed, hence what kind of pattern formation algorithms can not be designed. In fact, the ability to form a particular type of patterns would imply the ability to elect a robot in the system as the leader. Formally,

Theorem 6.1. *If an algorithm $\mathcal{A}$ lets the robots form (a.) an asymmetric pattern, or (b.) a symmetric pattern that has all its axes of symmetry passing through some vertex, then $\mathcal{A}$ is a leader election algorithm.*

Proof.

Part a. Let $\mathcal{A}$ be an algorithm that lets the robots form an asymmetric pattern $\mathbb{P}$ of n points. Let $\mathbb{D}_f$ be the final configuration after they execute the algorithm, starting from an arbitrary initial configuration. Moreover, let Γ and Γ' be respectively the vertical axes passing through the outermost robots in $\mathbb{D}_f$, and let Γ_m be the vertical axis equidistant from Γ and Γ'. Γ_m splits the plane in two regions, $\mathcal{S}$ and $\mathcal{S}'$. If some robots are on Γ_m, the topmost one on Γ_m can be elected as a leader, and the theorem follows. If no robot is on Γ_m, we can distinguish two cases:

[7]This implies that there is common agreement also on the direction of the x axis, but not on its orientation.

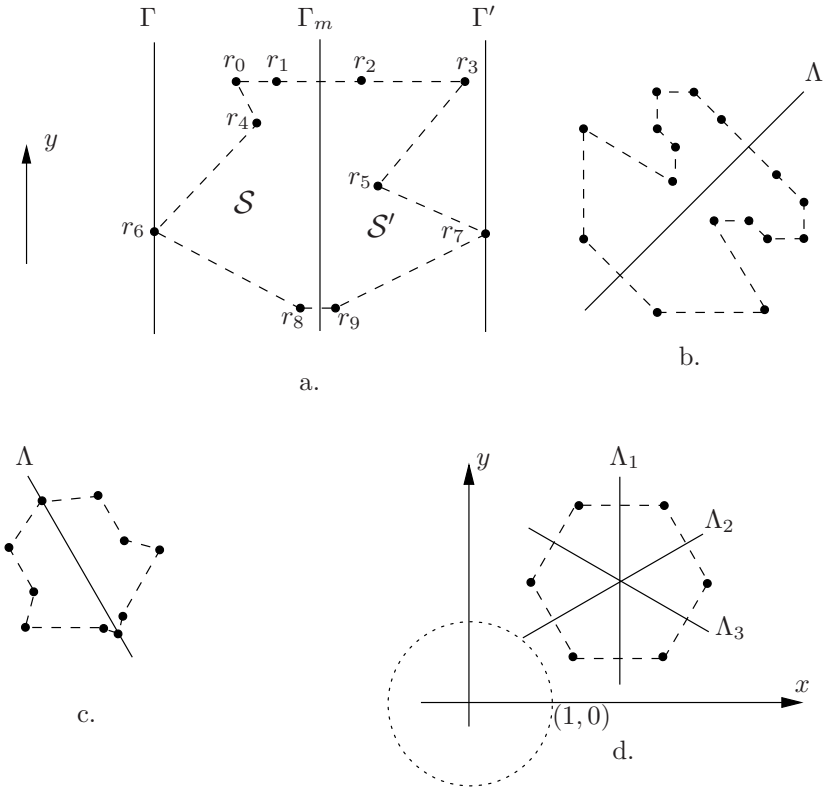

Figure 12: (a) An unachievable asymmetric pattern. In this example, the sorted sequence of pairs of robots from the proof of Theorem 6.1 is the following: (r_1, r_2), $(r_0, \emptyset)$, $(r_3, \emptyset)$, $(r_4, \emptyset)$, $(r_5, \emptyset)$, (r_6, r_7), (r_8, r_9). In this case r_0 would be elected as the leader. (b) An achievable pattern with one axis of symmetry not passing through any vertex. (c) An unachievable pattern. (d) An achievable pattern that has three axes of symmetry not passing through any vertex. Note that this pattern has also axes of symmetry passing through vertices. In this case, the routine `Choose(`$\mathbb{P}$`)` of Algorithm 6 would choose the axis Λ_2.

1. $|\mathcal{S}| \neq |\mathcal{S}'|$. In this case, the robots can agree on the most populated region as the positive side of x; hence, starting from any initial configuration, it is possible to elect a leader (e.g., the topmost rightmost one), and the theorem follows.
2. $|\mathcal{S}| = |\mathcal{S}'|$. In this case, for each robot $r_i \in \mathcal{S}$, we build a pair (r_i, x), $x \in \mathcal{S}' \cup \{*\}$, where $* \notin \mathcal{S}$ denotes a special symbol, as follows. Let $r.y$ indicates the height of robot r. If there exists $r_j \in \mathcal{S}'$ such that $r_i.y = r_j.y$ and $\overline{r_i\Gamma_m} = \overline{r_j\Gamma_m}$, then $x = r_j$; otherwise $x = *$. Analogously, we build pairs for each $r_j \in \mathcal{S}'$. Given that (r_i, r_j) is defined if and only if (r_j, r_i) is defined, we can sort all the pairs in descending order, with respect to the height and the horizontal distance of the robots from Γ_m. Namely, (e.g., see in Figure 12.a):

$$\begin{array}{lll}
(r_i,*) > (r_j,*) & \Leftrightarrow & r_i.y > r_j.y \vee (r_i.y = r_j.y \wedge \overline{r_i\Gamma} < \overline{r_j\Gamma}) \\
(r_i,*) > (r_j,r_h) & \Leftrightarrow & r_i.y > r_j.y \vee (r_i.y = r_j.y \wedge \overline{r_i\Gamma} < \overline{r_j\Gamma}) \\
(r_i,r_j) > (r_h,*) & \Leftrightarrow & r_i.y > r_h.y \vee (r_i.y = r_h.y \wedge \overline{r_i\Gamma} < \overline{r_h\Gamma}) \\
(r_i,r_j) > (r_h,r_k) & \Leftrightarrow & r_i.y > r_h.y \vee (r_i.y = r_h.y \wedge \overline{r_i\Gamma} < \overline{r_h\Gamma})
\end{array}$$

We observe that the set of obtained pairs is independent from the orientation of the x axis in the local coordinate systems of the robots; moreover, since $\mathbb{D}_f$ is asymmetric w.r.t Γ by hypothesis, there must exist at least a pair with an $*$. It follows that we can elect as a leader the robot in the first pair that has $*$ as an element, and the theorem follows.

Part b. Let $\mathcal{A}$ be an algorithm that lets the robots form a symmetric pattern $\mathbb{P}$ that has all its axes of symmetry passing through some vertex in $\mathbb{P}$, starting from any arbitrary initial configuration. After the robots run $\mathcal{A}$, they are in a final configuration $\mathbb{D}_f$ whose positions correspond to the vertices of $\mathbb{P}$ (up to scaling and rotation); hence, $\mathbb{D}_f$ must be symmetric with all its axes of symmetry passing through some vertex (robot's position). We distinguish two cases.

1. $\mathbb{D}_f$ is not symmetric with respect to any line Γ' parallel to y (e.g., see Figure 12.c). In this case, the same argument of Part a. can be used to conclude that a leader can be elected, and the theorem follows.
2. $\mathbb{D}_f$ is symmetric with respect to a Γ' parallel to y (notice that such a vertical axis is unique). Since by hypothesis Γ' must pass through a vertex, a leader can be elected (e.g., the topmost robot on Γ'), and the theorem follows. □

From Theorem 2.2 and Theorem 6.1, it follows that

Corollary 6.1. *There exists no pattern formation algorithm that lets the robots in the system form (a.) an asymmetric pattern, or (b.) a symmetric pattern that has all its axes of symmetry passing through some vertex.*

Let us call $\mathfrak{T}$ the class containing all the arbitrary patterns, and $\mathfrak{P} \subset \mathfrak{T}$ the class containing only patterns with at least one axis of symmetry not passing through any vertex (e.g., see Figures 12.b and 12.d); let us call *empty* such an axis. Corollary 6.1 states that if $\mathbb{P} \in \mathfrak{T} \setminus \mathfrak{P}$, then $\mathbb{P}$ can not be in general formed; hence, according to Part (b.), the only patterns that might be formed are symmetric ones with at least one *empty axis*. In the following, we prove that all these patterns can actually be formed. In particular, we present an algorithm that lets the robots form exactly these kind of patterns, if local rotation of the pattern is allowed.

6.2 Basic Definitions

By the results shown in the previous section, it follows that $\mathbb{P}$ is symmetric; therefore, $top^{\mathbb{P}}_{\mathcal{M}}$ and $top^{\mathbb{P}}_{\mathcal{L}}$ are at the same height. Moreover, by Corollary 6.1, the input pattern can not be a vertical line.

Following the notation introduced in Section 5, given a set of points $\mathbb{E}$, we will denote by $\mathcal{S}$ and $\mathcal{S}^{'}$ the two sides of $\mathbb{E}$, by Γ, Γ' the two vertical axis tangent to the convex hull of $\mathbb{E}$, and by *top*and *top*' the two topmost robots on Γ and Γ', respectively.

Finally, given a configuration $\mathbb{D}$, $\tau(\mathbb{P}, \mathbb{D})$ is defined as in Section 5.1.

6.3 The Algorithm

In this section, we present Algorithm Pen (whose pseudo-code is reported in Algorithm 6), that solves the Apf problem with Partial agreement and an Even Number of robots. In particular, it lets the robots form symmetric patterns with at least one *empty axis*.

The overall strategy is as follows. First, the robots compute locally an *empty axis* of the input pattern $\mathbb{P}$, say Λ, and then rotate $\mathbb{P}$ so that Λ is parallel to the common understanding of the orientation of y. Then they place themselves in a non-degenerated configuration. Finally, half of the robots goes in $\mathcal{S}$ and half of them goes in $\mathcal{S}'$, placing themselves on the final positions (points in $\tau(\mathbb{P}, \cdot)$). The two sides of the patterns are formed in parallel and independently of each other.

In particular, if the robots at the beginning lie all on the same vertical line, the algorithm forces them to place themselves in a non-degenerated configuration (routine `SameVerticalAxis()` in Line 7, as defined in Algorithm Pond). Then, the topmost robot on Γ, *top*, and the topmost robot on Γ', *top*$^{'}$, move so that they place themselves to the same height. At this point, the set of final positions can be computed (Line19), by using $\tau(\mathbb{P}, \cdot)$.

Now, the robots move to reach a balanced configuration, with each side containing half of the robots. The balancing is obtained as follows.

- In the side that has more than $n/2$ robot (if any), the robots are moved sequentially (starting from the topmost with the smallest horizontal distance from Γ_m) towards Γ_m, using a path that avoids collisions, until there are exactly $n/2$ robots in that side.
- In a side that has less than or equal to $n/2$ robots, the robots are moved towards the final positions in that side; the movement are controlled by Algorithm `CloseToDestination()`.
- The robots that are on Γ_m wait until $|\mathcal{S}| \leq n/2$ and $|\mathcal{S}'| \leq n/2$, and all the robots in the two sides are on a final position. At this point, sequentially (from the topmost) they move towards the final positions still available in the two sides, by executing Algorithm `CloseToDestination()`. In fact, by the way the input pattern has been rotated in Line 2, no final positions can be on Γ_m.

Algorithm Pen calls routines `Choose(`$\mathbb{P}$`)` and `Rotate(`$\mathbb{P}, S$`)`. In particular, the first one locally chooses an *empty axis* of symmetry in the input pattern $\mathbb{P}$; since this is a local operation, and $\mathbb{P}$ is the same for all the robots, every robot can be made to choose the same axis of symmetry[8].

[8]For instance, starting from the point $(1, 0)$ on the unit circle centered in the origin of the local coordinate system, they can choose the first *empty axis* that is hit moving counterclockwise (according to the local orientation of the x axis), after having translated all the *empty axes* in such a way that they pass through the origin. In the example depicted in Figure 12.d, axis Λ_2 would be chosen.

Algorithm 6 PEN

Input: An arbitrary pattern $\mathbb{P}$ described as a sequence of points $p_1, \ldots, p_n$, given in lexicographic order. $\mathbb{P}$ is symmetric and has at least one *empty axis*. The direction and orientation of the y axis is common agreement.

$\Lambda :=$ `Choose(`$\mathbb{P}$`)`;
$\mathbb{P}_R :=$ `Rotate(`$\mathbb{P}, \Lambda$`)`;
$(\Upsilon^+, \Upsilon^-, \Upsilon_m) := (\Phi^{\mathbb{P}_R}_{\mathcal{M}}, \Phi^{\mathbb{P}_R}_{\mathcal{L}}, \Phi^{\mathbb{P}_R}_{m})$;
$\mathbb{D} :=$ Current Configuration Of The Robots;
If $\mathbb{D}$ Is A Final Configuration **Then STOP**.
$\Xi :=$ Vertical Axis With More Robots On It;
If $|\Xi| = n$ **Then** `SameVerticalAxis(`Ξ`)`; **EndC**.
If $|\Xi| = n - 1$ **Then**
    $r :=$Robot not on Ξ;
    If $dist(r, \Xi) \neq dist($`top(`Ξ`), bottom(`Ξ`))` **Then** `SameVerticalAxis2(`Ξ`)`; **EndC**.
$(\Gamma, \Gamma', \Gamma_m) := (\Phi^{\mathbb{D}}_{\mathcal{M}}, \Phi^{\mathbb{D}}_{\mathcal{L}}, \Phi^{\mathbb{D}}_{m})$;
If I Am Not On Γ_m **Then** $\mathcal{MS} :=$ Side In $\mathbb{D}$ Where I Lie;
Else $\mathcal{MS} := \Gamma_m$;
$top :=$ Topmost Robot On Γ;
$top' :=$ Topmost Robot On Γ';
If `Angle(`$top^{\mathbb{P}_R}_{\mathcal{M}}, top^{\mathbb{P}_R}_{\mathcal{L}}$`)`$\neq$ `Angle(`top, top'`)` **Then**
    `FixOutermosts(`top, top'`)`; **EndC**.
If I Am top **Or** top' **Then** `destination:= null`; **EndC**.
$FinalPositions :=$ `FindFinalPositions(`$\mathbb{D}, \mathbb{P}_R$`)`;
If $\mathcal{MS} = \Gamma_m$ **Then**
    If All The Robots In The Two Sides Of $\mathbb{D}$ Are On $FinalPositions$ **Then**
        $top^* :=$ Topmost Robot On Γ_m;
        If I Am top^* **Then**
            $FreePoints :=$ {$FinalPositions$ *With No Robots On Them*};
            `CloseToDestination(`{top^*}$, FreePoints, \Gamma, \Gamma'$`)`; **EndC**.
        Else `destination:= null`; **EndC**.
    Else `destination:= null`; **EndC**.
If $|\mathcal{MS}| > n/2$ **Then**
    $r :=$ Topmost Robot In $\mathcal{MS}$ With The Smallest Horizontal Distance From Γ_m;
    $p :=$ Point On Γ_m Such That There Is No Robot On $[rp]$;
    If I Am r **Then** `destination:=` p; **EndC**. **Else** `destination:= null`; **EndC**.
If I Am On One Of The $FinalPositions$ **Then** `destination:= null`; **EndC**.
If $|\mathcal{MS}| \leq n/2$ **Then**
    $FreePoints :=$ {$FinalPositions$ *In* $\mathcal{MS}$ *With No Robots On Them*};
    $FreeRobots :=$ {*Robots in* $\mathcal{MS}$ *Not On* $FinalPositions$};
    $\Gamma^* :=$ Axis Among Γ And Γ' In $\mathcal{MS}$;
    `CloseToDestination(`$FreeRobots, FreePoints, \Gamma_m, \Gamma^*$`)`; **EndC**.

`Rotate`$(\mathbb{P}, S)$ locally rotates $\mathbb{P}$ in such a way that the axis of symmetry Λ chosen by `Choose`$(\mathbb{P})$ becomes parallel to the y axis. The rotation is (locally) performed clockwise, and its result is stored in $\mathbb{P}_R$. Then, all the reference points for the pattern are computed on the rotated pattern $\mathbb{P}_R$.

All the other routines called by the algorithm are the same as in Algorithm Pond.

6.4 Correctness of Algorithm Pen

Let $\mathbb{P}_R$ be the result of the local rotation of the input pattern in Line 2. By using the same proof adopted in Lemma 5.1, at time $t_{dis} \geq 0$, the robots are in a non-degenerated configuration. Let $\Gamma = \Phi_{\mathcal{M}}^{\mathbb{D}_{t_{dis}}}$, $\Gamma' = \Phi_{\mathcal{L}}^{\mathbb{D}_{t_{dis}}}$, $\Gamma_m = \Phi_m^{\mathbb{D}_{t_{dis}}}$. Furthermore, let *top* and *top*$'$ be the topmost robots on Γ and Γ', respectively, at time t_{dis}.

Lemma 6.1. *Let the robots be in a non-final configuration at time t_{dis}. Then, in a finite number of cycles, at time $t_h \geq t_{dis}$, top and top$'$ place themselves at the same height (so that* `Angle`$(top, top')=$ `Angle`$(top_{\mathcal{M}}^{\mathbb{P}}, top_{\mathcal{L}}^{\mathbb{P}})$*). Furthermore, until this time any collision is avoided, and at time t_h all the robots are in $\mathbb{I}(t_h)$.*

Proof. First note that, since $\mathbb{P}_R$ is symmetric with respect to Λ, and Λ is parallel to y after the rotation performed in Line 2, $top_{\mathcal{M}}^{\mathbb{P}}$ and $top_{\mathcal{L}}^{\mathbb{P}}$ are at the same height.

If at time t_{dis} *top* and *top*$'$ are at the same height, then the lemma trivially follows. Otherwise, as long as `Angle`$(top, top')\neq$ `Angle`$(top_{\mathcal{M}}^{\mathbb{P}}, top_{\mathcal{L}}^{\mathbb{P}})$, routine `FixOutermosts()` will move either *top* or *top*$'$ topwards until they are at the same height. During this time no other robot can move; hence, no collision occurs, and the lemma follows. □

Let *FinalPositions* be the set of points as returned by routine `FindFinalPositions`$(\mathbb{D}_{t_h}, \mathbb{P}_R)$. Furthermore, let $\mathcal{S}$ and $\mathcal{S}'$ be the two sides determined by Γ_m. Observe that

Observation 6.1. By definition of $\mathbb{P}_R$ and Λ, none of the final positions can be on Γ_m.

Lemma 6.2. *Let t_h be as defined in Lemma 6.1, and let the configuration at this time be not final. In a finite number of cycles, at time t_{qf}, $|\mathcal{S}| \leq n/2$, $|\mathcal{S}'| \leq n/2$, and all the robots in the two side are on final positions. Furthermore, between time t_h and time t_{qf} no collisions occur, and at time t_{qf} all the robots are in $\mathbb{I}(t_{qf})$.*

Proof. If $|\mathcal{S}| \leq n/2$, $|\mathcal{S}'| \leq n/2$, and all the robots in the two side are on final positions, then the lemma trivially follows. Otherwise, let us consider what happens in $\mathcal{S}$ (the same happens symmetrically in $\mathcal{S}'$). We distinguish two cases, according to the cardinality of $\mathcal{S}$.

$|\mathcal{S}| \leq n/2$. First observe that,

> **Observation 6.2.** As long as the robots inside $\mathcal{S}$ are not all on final positions, all the robots on Γ_m do not move (Line 27).

The robots inside $\mathcal{S}$ move towards the final positions in $\mathcal{S}$ by executing Algorithm `CloseToDestination()`. With respect to this algorithm, the set of final positions in $\mathcal{S}$ not occupied by any robot corresponds to $\mathbb{FT}$; the set of robots in $\mathcal{S}$ not on any of the final positions corresponds to $\mathbb{FR}$; and the obstacles $\mathbb{O}$ are all the robots not in $\mathbb{FR}$ and in $\mathcal{S}$. Let (r, p) be the pair that satisfies Equation (1) at time t_h, among all the robots in $\mathbb{FR}$ and all the points in $\mathbb{FT}$. First of all, note that all the four conditions required by

`CloseToDestination()` are met. In particular, among $\mathbb{FR}$ there is a total agreement on the coordinate systems, with the x axis oriented from Γ_m to Γ^*, with Γ^* as defined in Line 36; by Lemma 6.1 all the robots are in $\mathbb{I}(t_h)$ at time t_h; by Observation 6.1, Observation 6.2, and since the robots in $\mathcal{S}'$ can move at most up to Γ_m (Line 37), no robot can enter $\mathcal{S}$ as long as $\mathbb{FR} \neq \emptyset$. Hence, according to Theorems 3.1 and 3.2, r will reach p in a finite number of cycles. Therefore, iterating the above argument, we can conclude that in a finite number of cycles, at time $t' \geq t_h$, all the robots inside $\mathcal{S}$ reach a final position.

$|\mathcal{S}| > n/2$. In this case, all the robots on Γ_m do not move (Lines 28–31). The robots inside $\mathcal{S}$ move sequentially towards Γ_m. In particular, let r be the topmost robots with the smallest horizontal distance from Γ_m. r chooses as its destination a point p on Γ_m such that there are no robots on the segment $[rp]$; in this way any collision with other robots is avoided. Note that, since $|\mathcal{S}'| \leq n/2$, by what observed in the previous case, no robot from $\mathcal{S}'$ can move on Γ_m; hence, no collisions can occur with robots coming from the other side. This process continues as long as $|\mathcal{S}| > n/2$. As soon as $|\mathcal{S}| = n/2$, the previous case applies and, in a finite number of cycles, at time $t'' > t_h$, all the robots inside $\mathcal{S}$ are on final positions.

In conclusion, in a finite number of cycles, at time $t_{qf} = \max\{t', t''\}$, all the robots in both sides are on final positions, and the lemma follows. □

Lemma 6.3. *Let t_{qf} be as defined in Lemma 6.2. In a finite number of cycles, at time $t_f \geq t_{qf}$, the robots reach a final configuration avoiding collisions. Furthermore, at time t_f all the robots are in $\mathbb{I}(t_f)$.*

Proof. If $\mathbb{D}_{t_{qf}}$ is final, then the lemma trivially follows. Otherwise, by Observation 6.1, all the robots on Γ_m must leave Γ_m to reach the final positions still available in the two sides of $\mathbb{D}_{t_{qf}}$. The robots on Γ_m are the only ones allowed to move, using Algorithm `CloseToDestination`$(\{top^*\}, FreePoints, \Gamma, \Gamma')$. According to this algorithm, $\{top^*\}$ corresponds to $\mathbb{FT}$; the set $FreePoints$ of final positions still available corresponds to $\mathbb{FT}$; and the obstacles $\mathbb{O}$ are all the robots not in $\mathbb{FR}$. Let (top^*, p) be the pair that satisfies Equation (1) at time t_{qf}, among all the robots in $\mathbb{FR}$ and all the points in $\mathbb{FT}$. First of all, note that all the four conditions required by `CloseToDestination()` are met. In particular, by Lemma 6.2, the cardinality of both sides must be $\leq n/2$, and all the robots in the two sides are on final positions and do not move. Furthermore, all the robots on Γ_m different from top^* (if any) cannot move by Line 26. If top^* does not reach p in one cycle, then `CloseToDestination()` (Line 37) is executed again. Hence, according to Theorems 3.1 and 3.2, top^* will reach p in a finite number of cycles. Therefore, by iterating the above argument, we can conclude that in a finite number of cycles, at time $t_f > t_{qf}$, the robots reach a final configuration, and the lemma follows. □

Theorem 6.2. *Algorithm* Pen *is a collision-free pattern formation algorithm for $\mathbb{P} \in \mathfrak{P}$.*

Result 3. *An even number of autonomous, anonymous, oblivious, mobile robots that agree on the direction and orientation of y axis, can form a pattern $\mathbb{P}$ if and only if $\mathbb{P} \in \mathfrak{P}$.*

6.5 Remarks on Rotation

In Section 6.1 we provided a characterization of the class of patterns that can be formed by an even number of anonymous robots, provided they have agreement on the direction and orientation of the y axis. This characterization assumes that the robots can locally *rotate* the

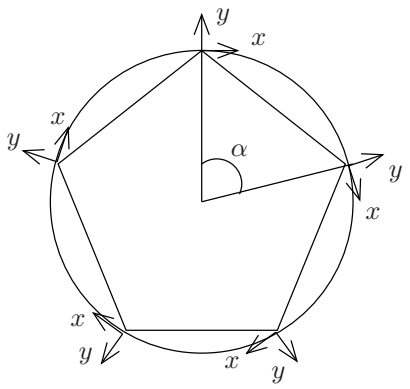

Figure 13: Theorem 7.1: the unbreakable symmetry of a 5-gon.

input pattern. Should the robots be incapable to perform such an operation, the characterization is different; not surprisingly, the class of achievable patterns is smaller. Let $\mathfrak{P}' \subset \mathfrak{P}$ be the class of symmetric patterns with at least one *empty axis*, and with no *empty axis* parallel to y.

Theorem 6.3. *There exists no pattern formation algorithm that lets the robots form a symmetric pattern $\mathbb{P} \in \mathfrak{P}'$ without allowing local rotation of the input pattern.*

Proof. By contradiction, let $\mathcal{A}$ be an algorithm that, starting from an arbitrary initial configuration, lets the robots form a pattern $\mathbb{P} \in \mathfrak{P}'$ without rotation. Let $\mathbb{D}_f$ be the final configuration of the robots for $\mathbb{P}$ after they execute $\mathcal{A}$. Since no local rotation of the pattern is allowed, $\mathbb{D}_f$ is symmetric with no *empty axis* parallel to y. Let $\Gamma = \Phi^{\mathbb{D}_f}$, $\Gamma' = \Phi'^{\mathbb{D}_f}$, and $\Gamma_m = \Phi_m^{\mathbb{D}_f}$. If $\Gamma_m \equiv \Gamma \equiv \Gamma'$, then all the robot are on Γ_m, hence a leader can be elected (e.g., the topmost robot on Γ_m), contradicting Theorem 2.2. Otherwise, if $\mathbb{D}_f$ is symmetric with respect to Γ_m, then there must be at least one robot on Γ_m (by hypothesis, $\mathbb{D}_f$ has no *empty axis* parallel to y); hence, the topmost of these robots can be elected as leader, contradicting Theorem 2.2. Therefore, $\mathbb{D}_f$ is not symmetric with respect to Γ_m; also in this case, a leader can be elected (e.g., following an approach similar to the one used in the proof of Theorem 6.1.a), thus contradicting again Theorem 2.2. □

As a concluding remark, we note that skipping the operation `Rotate(ℙ)` (at Line 2 in Algorithm Pen), we have a pattern formation algorithm that does not make use of local rotation and correctly allows the formation of a symmetric pattern that has at least one *empty axis* that is parallel to y. Hence, we can state the following

Result 4. *An even number of autonomous, anonymous, oblivious, mobile robots that agree on the direction and orientation of y axis, can form a pattern $\mathbb{P}$ if and only if $\mathbb{P} \in \mathfrak{P} \setminus \mathfrak{P}'$, if no local rotation of $\mathbb{P}$ is allowed.*

7 No Agreement

We will now show that giving up the total agreement on the coordinate system leads to the inability of the system to form arbitrary patterns.

Theorem 7.1. *Without a total agreement on the coordinate system, a set of autonomous, anonymous, oblivious, mobile robots cannot form an arbitrary given pattern.*

Proof. By contradiction, let $\mathcal{A}$ be a deterministic algorithm for solving the pattern formation problem without a total agreement on the coordinate system. We show that there are input patterns, initial configurations of the robots, and a scheduling of the actions of the robots, such that the robots never can form the input patterns. Consider any pattern different from a regular n-gon or a single point, and let the initial positions be such that the robots form a regular n-gon. Let $\alpha = 360°/n$ be the characteristic angle of the n-gon, and let the local coordinate system of each robot be rotated by α with respect to its neighbor on the polygon (see Figure 13). In this situation, all the robots have the same view of the world. Now, for any move that any one robot can make in its local coordinate system by executing algorithm $\mathcal{A}$, we know that each robot can make the same move in its local coordinate system. If all of them move in the exact same way at the same time (i.e., they move according to a synchronous schedule), they again end up in a regular n-gon or a single point. Therefore, by letting all the robots move at the same time in the same way, we always proceed from one regular n-gon or single point to the next. Hence, the desired pattern cannot be formed. □

8 Discussion

We have shown that from an algorithmic point of view, only the most fundamental aspects of mobile robot coordination are being understood. In other papers, we have proposed two algorithms for the point formation problem for oblivious robots; the first one does not need any common knowledge [6], and the second one works with limited visibility, when two axes are known [15].

There is a wealth of further questions that suggest themselves. First, we have shown that an arbitrary pattern cannot always be formed; it is interesting to understand in more detail which patterns or classes of patterns can be formed under which conditions, because this indicates which types of agreement can be reached, and therefore which types of tasks can be performed. Second, in contrast with other researchers who have looked at modeling natural behaviors, our robots perform quite a complex computation in each step; it is interesting to understand in more detail the tradeoff between computation complexity and knowledge of the world. Third, the operating conditions of our robots have been quite restricted; it is interesting to look at more relaxed models, where for instance robots have a bounded amount of memory at their disposition, or they have a spatial extent, they collide as they move, or their camera rotates slowly when taking a picture, so that a robot may never see the world as it was at any time instant. Slightly faulty snapshots, a limited range of visibility [2], obstacles that limit the visibility and that moving robots must avoid or push aside, as well as robots that appear and disappear from the scene clearly suggest that the algorithmic nature of distributed coordination of autonomous, mobile robots merits further investigation. Recently some issues related to inaccurate sensing, faults, and inconsistent compasses have been addressed for the convergence problem [7, 8, 9, 23]. For a recent survey on algorithms for autonomous mobile robots see [22].

Acknowledgements

This work has been partially supported by the Swiss National Science Foundation, and the Natural Science and Engineering Research Council of Canada.

References

[1] N. Agmon and D. Peleg. Fault-tolerant Gathering Algorithms for Autonomous Mobile Robots. In *Proc. of the* 15*th annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 1070–1078, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[2] H. Ando, I. Suzuki, and M. Yamashita. Formation and Agreement Problems for Synchronous Mobile Robots with Limited Visibility. *Proc. IEEE Int. Symp. on Intelligent Control*, pages 453–460, August 1995.

[3] T. Balch and R. C. Arkin. Motor Schema-based Formation Control for Multiagent Robot Teams. *ICMAS*, pages 10–16, 1995.

[4] G. Beni and S. Hackwood. Coherent Swarm Motion Under Distributed Control. In *Proc. DARS'92*, pages 39–52, 1992.

[5] Q. Chen and J. Y. S. Luh. Coordination and Control of a Group of Small Mobile Robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2315–2320, 1994. San Diego, CA.

[6] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving The Gathering Problem. In 30^{th} *International Colloquium on Automata, Languages and Programming (ICALP '03)*, 2003.

[7] R. Cohen and D. Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. In 12^{th} *European Symposium on Algorithms (ESA '04)*, pages 228–239, 2004.

[8] R. Cohen and D. Peleg. Convergence for Autonomous Robots with Inaccurate Sensors and Movements. In 23^{th} *Annual Symposium on Theoretical Aspects of Computer Science (STACS '06)*, pages 549–560, 2006.

[9] X. Défago, M. Gradinariu, S. Messika, and P. Raipin-Parvédy. Fault-tolerant and Self-stabilizing Mobile Robots Gathering. In 20^{th} *Intl. Symp. on Distributed Computing (DISC '06)*, pages 46–60, 2006.

[10] X. Défago and A. Konagaya. Circle Formation for Oblivious Anonymous Mobile Robots with No Common Sense of Orientation. In *Workshop on Principles of Mobile Computing*, pages 97–104, 2002.

[11] J. Desai, J. Ostrowski, and V. Kumar. Control of Formations for Multiple Robots. *IEEE International Conference on Robotics and Automation*, May 1998.

[12] E. H. Durfee. Blissful Ignorance: Knowing Just Enough to Coordinate Well. *ICMAS*, pages 406–413, 1995.

[13] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots. In 10^{th} *International Symposium on Algorithm and Computation (ISAAC '99)*, pages 93–102, 1999.

[14] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Pattern Formation by Autonomous Mobile Robots Without Chirality. In 18^{th} *Int. Colloquium on Structural Information and Communication Complexity (SIROCCO '01)*, pages 147–162, 2001.

[15] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Asynchronous Robots with Limited Visibility. *Theoretical Computer Science*, 337(1-3):147–168, 2005.

[16] V. Gervasi and G. Prencipe. Coordination Without Communication: The Case of The Flocking Problem. *Discrete Applied Mathematics*, 143:203–223, 2003.

[17] J. Halpern and Y. Moses. Knowledge and Common Knowledge in a Distributed Environment. In *Proceedings of the* 3^{rd} *ACM Symposium on Principles of Distributed Computing*, pages 50–61, 1984.

[18] Y. Kawauchi and M. Inaba and. T. Fukuda. A Principle of Decision Making of Cellular Robotic System (cebot). In *Proc. IEEE Conf. on Robotics and Automation*, pages 833–838, 1993.

[19] M. J. Mataric. Designing Emergent Behaviors: From Local Interactions to Collective Intelligence. *From Animals to Animats 2: Int. Conf. on Simulation of Adaptive Behavior*, pages 423–441, 1993.

[20] S. Murata, H. Kurokawa, and S. Kokaji. Self-Assembling Machine. In *Proc. IEEE Conf. on Robotics and Automation*, pages 441–448, 1994.

[21] L. E. Parker. Adaptive Action Selection for Cooperative Agent Teams. *Proc. Second Int'l. Conf. on Simulation of Adaptive Behavior*, pages 442–450, 1992.

[22] G. Prencipe and N. Santoro. Distributed Algorithms for Autonomous Mobile Robots. In 5^{th} *IFIP Int. Conference on Theoretical Computer Science (TCS'06)*, 2006.

[23] S. Souissi, X. Défago, and M. Yamashita. Gathering Asynchronous Mobile Robots with Inaccurate Ccompasses. In 10^{th} *Intl. Conf. on Principles of Distributed Systems (OPODIS '06)*, pages 333–349, 2006.

[24] K. Sugihara and I. Suzuki. Distributed Motion Coordination of Multiple Mobile Robots. *Proc. 5th IEEE Int'l. Symp. Intelligent Control*, pages 138–143, 1990.

[25] K. Sugihara and I. Suzuki. Distributed Algorithms for Formation of Geometric Patterns with Many Mobile Robots. *Journal of Robotics Systems*, 13:127–139, 1996.

[26] I. Suzuki and M. Yamashita. Formation and Agreement Problems for Anonymous Mobile Robots. *Proceedings of the 31st Annual Conference on Communication, Control and Computing, University of Illinois, Urbana, IL*, pages 93–102, 1993.

[27] I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots - Formation and Agreement Problems. *Proc. Third Colloq. on Struc. Information and Communication Complexity (SIROCCO)*, pages 313–330, 1996.

[28] I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *Siam J. Comput.*, 28(4):1347–1363, 1999.

[29] P. K. C. Wang. Navigation Strategies for Multiple Autonomous Mobile Robots Moving in Formation. *Journal of Robotic Systems*, 8(2):177–195, 1991.

Appendix

A Routines Used in Algorithm POND

A.1 SameVerticalAxis()

```
SameVerticalAxis(Ξ)
    d := dist(top(Ξ), bottom(Ξ));
    If I Am The Second Topmost Robot On Ξ Then
        p := Point To My Right At Horizontal Distance d From Ξ;
        destination:= p; EndC.
5:  Else
        destination:= null; EndC.

SameVerticalAxis2(Ξ)
    r := Robot Not On Ξ;
    d := dist(top(Ξ), bottom(Ξ));
    If I Am r Then
        p := Closest Point To Me At Horizontal Distance d From Ξ;
5:      destination:= p; EndC.
    Else
        destination:= null; EndC.
```

Since by hypothesis $\Upsilon^+ \not\equiv \Upsilon_m$, this routine handles the case when all the robots are at the beginning on the same vertical line: in fact, in the final configuration the robots must lie on three distinct vertical lines. The distance d between the topmost (returned by `top(Ξ)`) and the bottommost (returned by `bottom(Ξ)`) robot on Ξ is computed by `dist(top(Ξ),bottom(Ξ))`; if there are exactly n robots on Ξ, the second topmost robot r on Ξ moves to its right until it is at an horizontal distance d from Ξ (note that, while r is moving, the case $|\Xi| = n - 1$ forces all the other robots to stay still until r is at distance d from Ξ).

A.2 Fix()

```
Fix(𝔻, K_m)
    If τ(ℙ, 𝔻) ≠ ∅ Then Targets := τ(ℙ, 𝔻) Else Targets := τ^{-1}(ℙ, 𝔻)
    Targets' := Points In Targets That Are On K_m;
    RobotsOnK_m := Robots In 𝔻 That Are On K_m;
    MoveCarefully(RobotsOnK_m, Targets').
```

`MoveCarefully()` moves the calling robot r so that any collision is avoided; that is, if r "sees" a robot r' on its way, it stops before r'.

```
MoveCarefully(ℝ, 𝕋)
    SortedR := Sort(ℝ);
    SortedP := Sort(𝕋);
    Me:= My Current Position;
    If I Am The i-th Robot In SortedR Then p := i-th Point In SortedP;
5:  If Some Robot Is On [Me, p] Then destination:= null; EndC.
```

Else `destination`:= p; **EndC**.

This routine is called in Line 7 of Algorithm POND, when $\mathbb{D}$ is a *quasi-final* configuration.

Routine `Sort(`*Input*`)`, given as input a set of points that all lie on the same vertical axis, sorts them from the topmost to the bottommost.

A.3 `GetUnbalanced()`

`GetUnbalanced(`$\mathbb{D}$`)`

Input: A balanced configuration $\mathbb{D}$

$(K^+, K^-, K_m) := (\Phi^{\mathbb{D}}_{\mathcal{M}}, \Phi^{\mathbb{D}}_{\mathcal{L}}, \Phi^{\mathbb{D}}_{m})$;
If All The Robots Are Either On $K^+ \cup K^- \cup K_m$ **Then**
  top^* := Topmost Robot On K_m;
  If I Am top^* **Then**
5: p^* := `ChooseDestination(`$\mathbb{D}$`,`top^*`)`;
    `destination`:= p^*; **EndC**..
  Else `destination`:= `null`; **EndC**.
Else
  If I Am On $K^+ \cup K^- \cup K_m$ **Then** `destination`:= `null`; **EndC**.
10: $\mathcal{MS}$:= Side Where I Am;
  K^* := Vertical Axis Among K^+ And K^- That Is In $\mathcal{MS}$;
  r := Topmost Robot In $\mathcal{MS}$ With Smallest Horizontal Distance From K^*;
  p := Position On K^* Occupied By No Robot And Below The Topmost On K^*;
  If I Am r **Then** `destination`:= p; **EndC**. **Else** `destination`:= `null`; **EndC**.

The aim of this routine is to unbalance the current configuration $\mathbb{D}$ as follows. It first *sweeps* the two sides of the configuration by placing all the robots either on K_m, or on K^+, or on K^- (Lines 11-17). At this point, since no robot changed side during these movements (Lines 13–14), the configuration is not unbalanced yet; therefore, the topmost robots on K_m moves towards a point chosen by routine `ChooseDestination()` in order to unbalance the configuration. Note that top^* is the only robot that executes `ChooseDestination()`.

`ChooseDestination(`$\mathbb{D}$`,`top^*`)`
$\mathcal{S}^+$:= Side To My Right;
$\mathcal{S}^-$:= Side To My Left;
$(K^+, K^-, K_m) := (\Phi^{\mathbb{D}}_{\mathcal{M}}, \Phi^{\mathbb{D}}_{\mathcal{L}}, \Phi^{\mathbb{D}}_{m})$;
(top^+, top^-) := Topmost Robots On K^+ and K^-, Respectively;
5: (p^+, p^-) := Positions on K^+ and K^- Where top^+ And top^- Will Be After `FixOutermosts()` Is Executed;
$\mathbb{D}' := \mathbb{D} \setminus \{top^+, top^-\} \cup \{p^+, p^-\}$;
$FinalPositions$:= `FindFinalPositions(`$\mathbb{D}'$, $\mathbb{P}$`)`;
p^* :=`MoveInside`$\mathcal{S}^+$`(`$FinalPositions, \mathcal{S}^+, K^+, top^*$`)`;
Return p^*.

In `ChooseDestination()`, top^* chooses a trajectory so that it avoids collisions with any other robots. Furthermore, it chooses it in such a way to avoid also any of the $FinalPositions$ that will be computed in next Line 34 of Algorithm POND. In fact, since top^* knows in which side it is going to move, and it also knows how the two topmost robots on K^+ and K^- will be

moved by routine `FixOutermosts()` in order to reach their final positions, it knows already (in advance) the way in which $\mathbb{P}$ will be scaled and translated in routine `FindFinalPositions()`.

`MoveInside`$\mathcal{S}^+$`()` is responsible to find a destination p^* for top^* that avoids any collisions. Note that, when this routine is called, all the robots are either on K_m, or on K^+, or on K^-.

`MoveInside`$\mathcal{S}^+$`(`$FinalPositions, \mathcal{S}^+, K^+, r$`)`

If There Is At Least One Of The $FinalPositions$ Inside $\mathcal{S}^+$ **Then**
  $p :=$ Point in $FinalPositions$ Closest To r;
Else
  $top^+ :=$ Topmost Robot On K^+;
  $p :=$ Point On K^+ With No Robot On It And Below top^+;
Return p.

A.4 `FixOutermosts()`

`FixOutermosts(`top^+, top^-`)`

$\alpha :=$ `Angle(`$top^{\mathbb{P}}_{\mathcal{M}}, top^{\mathbb{P}}_{\mathcal{L}}$`)`;
$\beta :=$ `Angle(`top^+, top^-`)`;
If $top^{\mathbb{P}}_{\mathcal{L}}.y < top^{\mathbb{P}}_{\mathcal{M}}.y$ **Then**
  If $top^-.y \leq top^+.y$ **Then**
5: **If** $\alpha \geq \beta$ **Then**
      $p :=$ Point On K^+ Such That $\alpha = \beta$;
      If I Am top^+ **Then** `destination`$:= p$; **EndC**. **Else** `destination`$:=$ `null`; **EndC**.
    Else
      $p :=$ Point On K^- Such That $\alpha = \beta$;
10:   **If** I Am top^- **Then** `destination`$:= p$; **EndC**. **Else** `destination`$:=$ `null`; **EndC**.
  Else
    $p :=$ Point On K^+ Such That $p.y = top^-.y$;
    If I Am top^+ **Then** `destination`$:= p$; **EndC**. **Else** `destination`$:=$ `null`; **EndC**.
If $top^{\mathbb{P}}_{\mathcal{L}}.y = top^{\mathbb{P}}_{\mathcal{M}}.y$ **Then**
15: **If** $top^-.y < top^+.y$ **Then**
    $p :=$ Point On K^- Such That $\alpha = \beta$;
    If I Am top^- **Then** `destination`$:= p$; **EndC**. **Else** `destination`$:=$ `null`; **EndC**.
  Else
    $p :=$ Point On K^+ Such That $\alpha = \beta$;
20: **If** I Am top^+ **Then** `destination`$:= p$; **EndC**. **Else** `destination`$:=$ `null`; **EndC**.
If $top^{\mathbb{P}}_{\mathcal{L}}.y > top^{\mathbb{P}}_{\mathcal{M}}.y$ **Then**
  If $top^-.y \geq top^+.y$ **Then**
    If $\alpha < \beta$ **Then**
      $p :=$ Point On K^+ Such That $\alpha = \beta$;
25:   **If** I Am top^+ **Then** `destination`$:= p$; **EndC**. **Else** `destination`$:=$ `null`; **EndC**.
    Else
      $p :=$ Point On K^- Such That $\alpha = \beta$;

If I Am top^- **Then** `destination`:= p; **EndC**. **Else** `destination`:= `null`; **EndC**.

Else

30: $p :=$ Point On K^- Such That $p.y = top^+.y$;
If I Am top^- **Then** `destination`:= p; **EndC**. **Else** `destination`:= `null`; **EndC**.

`FixOutermosts()` places top^+ and top^- computed in Line 9 of Algorithm POND so that their relative angle is the same as the relative angle between $top^{\mathbb{P}}_{\mathcal{M}}$ and $top^{\mathbb{P}}_{\mathcal{L}}$. In particular, routine `Angle`(p, q) returns the convex angle between the horizontal axis through q and the segment $[p, q]$. Note that the movements of top^+ and top^- happens strictly vertically on K^+ and K^-, respectively, and always upwards; hence, any collisions is avoided.

A.5 `FindFinalPositions()`

`FindFinalPositions`$(\mathbb{P}, \mathbb{D})$ returns the set of points $\tau(\mathbb{P}, \mathbb{D})$.

Note that this routine is called in Line 12 of Algorithm POND, that is when the current configuration of the robots $\mathbb{D}$ is already unbalanced.

A.6 `TestSF()`

`TestSF`$(\mathbb{D})$ tests whether configuration $\mathbb{D}$ is *semi-final* (see Definition 5.2). In particular it returns the triple $(SF, r, last)$, where $SF =$*true* if $\mathbb{D}$ is *semi-final*. Following Definition 5.2, we have that

1. if $r \neq \emptyset$, then r is the only robot (not on the median axis of the configuration) not on one of the $FinalPositions$;
2. if $r = \emptyset$, then all robots in the two sides of the configuration occupy one of the $FinalPositions$;
3. if $last \neq \emptyset$, then $last$ is the only final position that is inside on of the two sides of the configuration and with no robot on it;
4. if $last = \emptyset$, then the only final positions with no robots on them lie on the median axis of the configuration.

A.7 `AllOn()`

Routine `AllOn()` returns *true* for a configuration $\mathbb{D}$, if

1. $|\mathcal{L}_{\mathbb{D}}| = |\mathcal{M}_{\mathbb{D}}| - 1$,
2. all the robots but one, say r, are either on K^+, or on K^-, or on K_m, and
3. r is strictly inside $\mathcal{M}_{\mathbb{D}}$ not on one of the final positions.

A.8 `FromSidesToMedian()`

`FromSidesToMedian`$(K^+, K^-, K_m, Side, FinalPositions)$

$FreePoints :=$ {*FinalPositions In Side With No Robots On Them* };
$FreeRobots :=$ {*Robots' positions In Side Not On FinalPositions*};

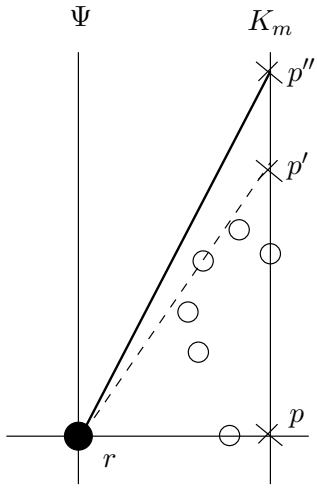

Figure 14: Routine `ChooseOnMedian()` determines the destination point for r on K_m. The white circles represent robots' positions. The thick line is the path followed by r to reach K_m.

If $Side = \mathcal{S}^+$ **Then**
  $r_b :=$ Bottommost Robot On K^+;
5: $FreeRobots := FreeRobots \setminus \{r_b\}$;
If I Am In $FreeRobots$ **Then**
  If $FreePoints \neq \emptyset$ **Then**
    `CloseToDestination(`$FreeRobots, FreePoints, K^+, K^-$`)`.
  Else %I am in *FreeRobots* but there are no *FreePoints* in *Side*%
10: `ChooseOnMedian(`$FreeRobots, K_m$`)`;
Else `destination:= null`; **EndC**.

This routine acts in two steps: first, all the robots in *Side* that are not on any of the *FinalPositions* (as computed in Line 9 of Algorithm POND), reach sequentially the *FinalPositions* in *Side* that are not occupied by any robot (Lines 7–10). Then, when there are no more free positions in *Side*, the robots in *Side* not on one of the *FinalPositions* (if any) are sequentially directed towards Γ_m by calling routine `ChooseOnMedian()` (Lines 11–12).

`ChooseOnMedian(`$FreeRobots, K_m$`)`
Me:= My Current Position;
If I Am The Topmost And Closest To K_m In $FreeRobots$ **Then**
  $p :=$ Intersection Between K_m And Horizontal Line Passing Through *Me*;
  If No Robot Is On The Line Passing Through *Me* And p **Then** `destination:=` p; **EndC**.
  $\Psi :=$ Vertical Line Passing Through *Me*;
  $\mathcal{V} :=$ Region Of The Plane Delimited By Ψ and K_m;
  $\mathcal{H} :=$ Half Plane Above Line Through r and p;
  $\mathcal{R} := (\mathcal{V} \cap \mathcal{H}) \setminus \Psi$;
  $Avoid := \{Positions\ In\ \mathcal{R}\ Occupied\ By\ Robots\}$;
  $Intersections := \emptyset$;
  For All $p' \in Avoid$ **Do**
    $\Psi' :=$ Line Passing Through *Me* And p';
    $Intersections := Intersections \cup \{\ Intersection\ Between\ K_m\ And\ \Psi'\}$;

End For
p' := Topmost Point In $Intersections$;
p'' := Point On K_m Above p' At Distance $\eta > 0$.
`destination`:= p''; **EndC**.
Else `destination`:= `null`; **EndC**.

In other words, $\mathcal{R}$ is the region of the plane above $[rp]$, and delimited by Ψ and K_m; all the points on Ψ are not included in $\mathcal{R}$. `ChooseOnMedian()` allows the calling robot to choose a path that goes above all the robots that are inside $\mathcal{R}$, maintaining the invariant to remain the (closest to K_m) topmost robot in $FreeRobots$. η is an arbitrary positive constant.

A.9 `FromMedianToSides()`

FromMedianToSides(K^+, K^-, K_m,*Side*,$FreePoints$)
top := Topmost Robot On K_m;
$FreePoints$:= {*FinalPositions In Side With No Robots On Them* };
If I Am *top* **Then**
`CloseToDestination`($\{top\}, FreePoints, K^+, K^-$).
Else `destination`:= `null`; **EndC**.

This routine moves sequentially the robots on K_m (from the topmost to the bottommost) towards $FinalPositions$ in *Side* with no robots on them.

B Routines Used in Algorithm POD

B.1 `LastFix()`

`LastFix`($\mathbb{D}, K$)
(*top*, *bottom*) := Topmost And Bottommost Robot On K, Respectively;
$Targets := \tau(\mathbb{P}, \mathbb{D},)$;
`MoveCarefully`($\mathbb{D}, Targets$).

where routine `MoveCarefully()` is as described in Appendix A.

B.2 `GetUnbalanced2()`

`GetUnbalanced`($\mathbb{D}$)
Input: A balanced configuration $\mathbb{D}$

$(K^+, K^-, K_m) := (\Phi^{\mathbb{D}}_{\mathcal{M}}, \Phi^{\mathbb{D}}_{\mathcal{L}}, \Phi^{\mathbb{D}}_{m})$;
If All The Robots Are Either On $K^+ \cup K^- \cup K_m$ **Then**
top^* := Topmost Robot On K_m;
If I Am top^* **Then**
p^* := Point On K^+ With No Robot On It And Below The Topmost On K^+;
`destination`:= p^*; **EndC**..
Else `destination`:= `null`; **EndC**.
Else
If I Am On $K^+ \cup K^- \cup K_m$ **Then** `destination`:= `null`; **EndC**.

10: $\mathcal{MS}$:= Side Where I Am;
K^* := Vertical Axis Among K^+ And K^- That Is In $\mathcal{MS}$;
r := Topmost Robot In $\mathcal{MS}$ With Smallest Horizontal Distance From K^*;
p := Position On K^* Occupied By No Robot;
If I Am r **Then** `destination`:= p; **EndC**. **Else** `destination`:= `null`; **EndC**.

B.3 `Towards()`

`Towards(`*Side*`,`K^+`)`

$\mathbb{SI}$:= {Robots Strictly Inside *Side* };
r := Topmost Robot In $\mathbb{SI}$ With The Smallest Horizontal Distance From K^+;
If I Am r **Then**
p := Point On K^+ So That No Robot Is On $[rp]$;
5: `destination`:= p; **EndC**.
Else `destination`:= `null`; **EndC**.

Note that in Line 9 of Algorithm POD, this routine has as argument K_m: in this case, we consider K_m as a region constituted by only one vertical line. Therefore, $|\mathbb{SI}| = 0$ (test in Line 2 of the routine) if and only if $|K_m| = 0$, and r computed in Line 3 is simply the topmost robot on K_m.