ED130
École doctorale Informatique, Télécommunications et Électronique
(Paris)

THÈSE

DE L'UNIVERSITÉ PIERRE ET MARIE CURIE

SPÉCIALITÉ : INFORMATIQUE

par Laure MILLET

pour obtenir le grade de **DOCTEUR**

Vérification et synthèse d'algorithmes de robots

Soutenue le date

Devant la commission d'examen formée de :

Béatrice Bérard	*Directrice*	UPMC LIP6
Amal El Fallah Seghrouchni	*Examinatrice*	UPMC LIP6
Paola Flocchini	*Rapporteure*	Ottawa University
Stephan Merz	*Rapporteur*	INRIA Nancy & LORIA
Laure Petrucci	*Examinatrice*	Paris13 LIPN
Maria Potop-Butucaru	*Directrice*	UPMC LIP6
Nathalie Sznajder	*Encadrante*	UPMC LIP6
Xavier Urbain	*Examinateur*	LRI
Josef Widder	*Examinateur*	TU Wien

UPMC

DOCTORAL SCHOOL EDITE

Ecole doctorale Informatique, Télécommunications et Électronique (Paris)

PHD THESIS

to obtain the title of

PhD of the University of Paris 6

Specialty : Computer Science

Defended by Laure Millet

Verification and synthesis of robot protocols

defended on date

Jury:

Béatrice Bérard	*Advisor*	UPMC LIP6
Amal El Fallah Seghrouchni	*Examiner*	UPMC LIP6
Paola Flocchini	*Reviewer*	Ottawa university
Stephan Merz	*Reviewer*	INRIA Nancy & LORIA
Laure Petrucci	*Examiner*	Paris13 LIPN
Maria Potop-Butucaru	*Advisor*	UPMC LIP6
Nathalie Sznajder	*Advisor*	UPMC LIP6
Xavier Urbain	*Examiner*	LRI
Josef Widder	*Examiner*	TU Wien

Contents

List of Figures

List of Tables

Chapter 1
Introduction

The tasks which may be performed by autonomous robots are increasingly numerous and complex, both in our everyday life or for industrial use. Many investigations have been focusing recently on autonomous mobile robots which must cooperate to achieve a common complex task that they cannot perform alone. Surveys can be found for instance in [Pre13; PRT11; FPS12]. Possible applications for such multi-robot systems include environmental monitoring, map construction, urban search and rescue, surface cleaning, surrounding or surveillance of risky areas, exploration of unknown environments, etc. For example, such robots can patrol the docks in a harbor, as depicted in Figure 1.1. They must coordinate to keep

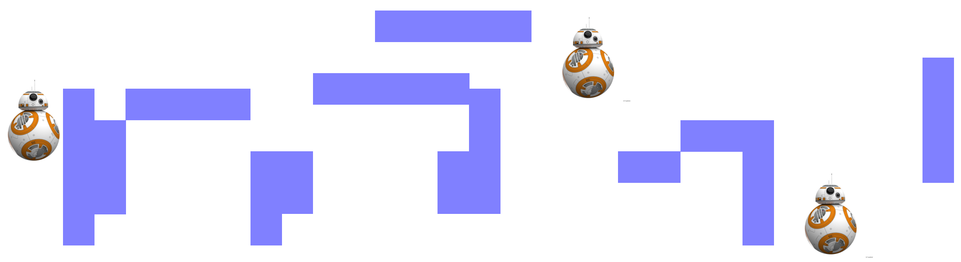


Figure 1.1: Robot example

away thieves from the cargo containers, by verifying regularly that no container has been opened. After chasing a thieve outside the port, they have to drive back near containers that need the most to be checked.

Such a cooperating team is called a distributed system [Lyn96a; Tel01]. A system is distributed when it is composed of a set of autonomous computation entities endowed with communication abilities in order to solve a common task. Traditionally, the entities have been assumed to be stationary and to communicate with each other thanks to message passing. The robot model we study in this thesis [SY99; FPS12] differs from the classical one in two aspects: the entities are mobile and they do not communicate via message passing. Moreover, these mobile entities may be endowed with limited capabilities. The concern in mobile robots research is to understand what kind of basic capabilities are needed for the robot team in order to accomplish a given task, in the absence of any central coordinating authority, and at what cost. Several tasks have been investigated both in the plane or in a discrete environment. Since these assigned task may be critical, it must be ensured that the robots accomplish them correctly, in spite of their limitations.

So far, robots networks have been studied empirically and most of the results have been validated only by handmade proofs. These proofs are hard to write and read, and reasoning on complex systems is both cumbersome and error-prone. Automated proofs on an abstract mathematical model of the system could remove the fastidious task of inspecting the system behavior to ensure its correctness with respect to a certain specification, *i.e.*, the definition of what the system is expected to do, described by a set of properties. Such automatic techniques are called formal verification techniques. Several approaches exist:

Test Test is probably the most frequently used method. In a first phase, execution scenarios whose results are known in advance have to be developed. The system is then executed in accordance with such a scenario and one can check if the output is as expected. The presence of such tests does not guarantee the correctness of system behaviors outside the cases tested. Moreover, the design of adequate tests is more difficult for concurrent or distributed systems.

Proofs In this approach, the system is expressed as a set of logic formulae, that need to be combined to construct new lemmas to deduct or invalidate the property to check. The implementation of axioms is generally not automatic: an expert user must interact to properly orient the proof construction. Besides, when a proof fails, the diagnostic is known to be difficult. Various proof assistants have emerged since the 60's: PVS[1], Coq[2] and Isabelle/HOL[3] are among the most widely used.

Model checking The system and the properties to be verified must be first formalized, possibly abstracting away irrelevant details. An algorithm, which depends on the classes of models for the system and the properties, is then applied to check whether the properties are satisfied by the model. One advantage of this technique is its ability to extract behavior invalidating properties, thanks to its completeness. Thus, it can be used at the design phase to validate prototypes. A drawback of this method is the so called combinatorial explosion: exploring all executions in a model of large size is

[1]http://pvs.csl.sri.com
[2]https://coq.inria.fr/
[3]https://isabelle.in.tum.de

time and memory consuming. The model checkers UPPAAL[4] and SPIN[5] are amongst the most well known.

Each of these approaches has its strengths and drawbacks: While tests can be easy to implement, they cannot be applied in the design phase and the generation of exhaustive tests is a difficult problem; Proofs are difficult to implement since they require the presence of an expert, but they are exhaustive, applicable at the design phase, and they can handle systems where some variables are not specified (like for instance the number of processes), called parameterized systems; Finally, the model checking approach is a comprehensive and automatic one but limited by the model size. Moreover, the problem is undecidable in general for parameterized systems [AK86].

While the correctness of an algorithm can be verified at the design phase by model checking and assisted proof, another formal technique called synthesis aims at automatically generating a protocol from its specification. An advantage of this method is that the generated protocol is correct by design. This question raised early interest [CE81; MW84] and actually goes back to Church [Chu63; BL69]. It is even more difficult when the program to generate is intended to work as an open system, maintaining an on-going interaction with a (partially) unknown environment. Given a specification, if there exists a program such that its behavior satisfies this specification, regardless of the environment behavior, then it must be automatically built. Otherwise, it must be proved that the problem cannot be solved.

It is known since [BL69] that a successful approach consists in viewing the synthesis problem as a *game* between the system and the environment. The system and its environment are considered as opposite players, the winning condition being the specification the system should fulfill. Then, the classical problem in game theory of determining winning strategies for the players is equivalent to find how the system should act in any situation, in order to always satisfy its specification. However, this problem is also undecidable in general for parameterized systems [PR90].

In this work, our aim is to investigate how formal methods can be applied in the context of mobile robot algorithms. We would like to bring out the benefits of these methods compared to traditional approaches.

In the next sections, we present the robot model considered in this manuscript, as well as existing work on the two main problems studied here: exploration and gathering by mobile robots. Then, we give a more detailed description of the formal methods devoted to the verification of mobile robots protocols.

[4]http://www.uppaal.org

[5]http://spinroot.com/spin/whatispin.html

1.1 Mobile robots

In this thesis, our interest is on a theoretic model [SY99; FPS12], where it is possible to express that robots with limited capabilities cooperate to achieve a common objective. In this model, each robot behavior is an infinite sequence of cycles where each cycle is divided into three phases: The Look phase, the Compute phase and the Move phase. More precisely:

Look The robot observes its environment ; the result of this operation is a snapshot of the positions of all robots within its radius of visibility with respect to its own coordinate system.

Compute The robot executes the algorithm (the same for all robots), using the snapshot of the Look operation as input, the result is a destination point given relatively to its coordinate system.

Move The robot moves toward the computed destination; if the destination is the current location, the robot stays still, performing a null movement.

This model called ATOM, also referred in the literature as the SYm model, was proposed by Suzuki and Yamashita [SY99]. It was then refined by Prencipe [Pre00] into a more realistic version called the CORDA model. These models differ in their degrees of atomicity:

- In the historical model, ATOM [SY99], some non-empty subset of robots executes the three phases synchronously and atomically. This gives rise to two variants: Fsync, for the fully-synchronous model where all robots are scheduled at each cycle, and Ssync, for the semi-synchronous model, where a strict subset of robots can be scheduled.

  In the semi-synchronous (Ssync) model, one or more robots are activated and obtain the same snapshot at each cycle; based on that snapshot, they compute and perform their move. As a consequence, no robot will ever be observed while moving, and the understanding of the universe by the active robots is always consistent. In this case, the system behavior corresponds to executing all operations instantaneously (all Look/Compute operations immediately followed by all Move operations). The fully synchronous (Fsync) model is a particular case of Ssync, since in each cycle, all robots are activated.

- The second model, CORDA [Pre00], also called Async, is a more realistic variant: In this less constrained model, each robot is activated asynchronously and independently from the other robots. Furthermore, the duration of each phase as well as the time between successive phases in the same cycle are finite but unknown. As a result, computations can be based on totally obsolete observations, taken arbitrarily far in the past. Another consequence is that robots can be seen while moving, creating further inconsistencies in robot views.

A particular variant, between Async and Ssync, has been considered in [LMA07]. In this limited form of asynchrony, called partial Async, the time spent by a robot in the Look, and Compute phases is bounded by a globally predefined amount, while the time spent in the Move phase is bounded by a locally predefined quantity (not necessarily the same for each robot).

Note that in term of executions, Fsync is included in Ssync, and Ssync is included in Async.

The ability of a team to achieve an assigned task depends mainly on the capabilities of its robots: the more powerful they are, the more easily the task is solved.

1.1.1 Robot weaknesses

We now detail the minimal assumptions usually made on these robots. The robots are identical and anonymous, they execute the same algorithm and they cannot be distinguished using their appearances, but they can have different computing speeds and different moving speeds (in the Async case). Robots may have identities but neither them nor the other robots have knowledge or access to these identities.

The robots are oblivious *i.e.*, they have no memory of their past actions. This capability implies that any state can be considered as initial. Hence, robot algorithms will have self-stabilization properties: A self-stabilizing distributed protocol ensures that a correct behavior can be recovered in a finite time without any external or manual help.

Robots have neither a common sense of direction, nor a common handedness (chirality). Each robot has its own unit of length, and a local compass defining his own local Cartesian coordinate system. This local coordinate system is self-centric, *i.e.*, the origin is the robot position. Moreover, the local coordinate system of oblivious robots may completely change during the robot life. However, it remains invariable during a cycle.

The robots are silent: there is no communication by message passing. They communicate by observing other robots positions, and taking a decision accordingly. In other words, the only mean for a robot to send information to some other robot is to move and let the others observe.

1.1.2 Robot capabilities

To execute their Look-Compute-Move cycles, the robots are endowed with sensing, computing and moving capabilities. These capabilities depend on the environment that can be continuous or discrete. Two cases have been studied in the literature:

- The continuous euclidean space [SY99; FPS12], in which the robots entities move on a plane,
- The discrete universe [KMP06; Flo+13], in which space is represented by a graph, where nodes correspond to the possible locations and edges the routes for a robot from one location to another.

The discrete representation is motivated by practical aspects with respect to the unreliability of sensing devices used by the robots as well as inaccuracy of their motorization [Cle+08]. A discrete setting permits to ignore these features and simplify the design of robot models by reasoning on finite structures. However, it is more sensitive to the size of constants, which may significantly increase the number of symmetric configurations when the underlying graph is also symmetric (*e.g.* a ring) and thus the size of correctness proofs [DSN11a; Kam+11; Kam+12].

The sensing capability

Robots are endowed with visibility sensors providing the locations of other robots. The obtained location is either fine grained (which means that it has been obtained with some degree of accuracy) or coarse grained (robots can only be observed at some specific discrete locations, each location being adjacent to one another). In the first case, the literature mostly refers to the continuous space model, while in the latter case, it is the discrete one.

Robots are dimensionless and thus their visibility cannot be obstructed: if three robots r_1, r_2, and r_3 are aligned, with r_2 in the middle, r_1 can see r_3. Moreover, robots may share the same position: this is called a *multiplicity point* or a *tower* [FPS12]. The ability for a robot to detect multiplicity is crucial to achieve some particular tasks. We distinguish weak and strong multiplicity detection.

- The weak multiplicity detector detects whether there is zero, one or more than one robot at a particular location.
- The strong multiplicity detector senses the exact number of robots at a particular location.

This sensor may be local or global: in the local setting a robot only detects multiplicity at its current position, while in the global setting every multiplicity of all positions is known.

A third characteristic of robot sensing capability is their visibility radius. It can be infinite *i.e.*, a robot is able to sense the position of all other robots, or finite *i.e.*, there exists a bound (expressed as a distance) beyond which a robot cannot sense anything. Note that the sensing is defined in the robot's own coordinate system.

The computing capability

As in classical distributed systems, robots are assumed to be able to perform any finite sequence of computing steps in negligible time. Since robots are oblivious, volatile memory is used to perform computing tasks in a single Look-Compute-Move cycle, but the memory content is erased at the end of each cycle. The computation takes as input the observation made in the Look phase and gives the robot a move. When two robots are on the same location or are symmetric, they should be given the same move.

The moving capability

Robots may move only to the location provided by the computing phase of the current cycle. In some instances, due to symmetry, the computed location may be ambiguous: it then corresponds to a non deterministic move, which can be resolved by a scheduler. In the discrete space model, a robot may move only to a location that is adjacent to its current location. In the continuous space model, a robot moves toward its computed destination.

Scheduling

When several processes execute concurrently, scheduling plays an important role: Schedulers are abstractions used to characterize the degree of asynchrony in the robot network [Déf+06; FPS12]. The most general scheduler is the unconditionally fair scheduler, that activates every robot infinitely often, in order to avoid starvation, but some robots may be activated arbitrarily more than other. The t-bounded scheduler ensures that, between any two activations of a given robot, every other robot is activated at most t times. So the ratio between the fastest and the slowest robot is at most t.

There are other types of fairness which depend on the scheduling of actions. An action that can be performed in the current state of a robot is called *enabled.* The *Strongly Fair* scheduler ensures that every action that is infinitely often enabled should be executed infinitely often, the *Weakly fair* scheduler ensures that every action that is continuously enabled from some point on should be executed infinitely often.

1.1.3 Other variants of robots

In the literature, weaker robots than those described above have been studied.

Limited visibility

Limited visibility has been studied when robots are myopic [And+99; Flo+05; GP11; Dat+13]. A myopic robot cannot see the nodes located beyond a certain fixed distance. The strongest myopia corresponds to when a robot can only see robots located at its own and at neighboring locations. Note that the weaker myopia is when the myopia distance is the diameter of the graph, this corresponds to an infinite visibility.

An other way to describe limited visibility is by considering an environment where the line of sight of a robot is obstructed by the closest robot on that line. This is typically assumed in (and is the main motivation for) the study of robots that are solid (i.e., with a physical dimension) [CM15; CGP09; HPT14].

Faulty robots

Usually robots are assumed to operate without failures (those robots are called correct). Yet, some unexpected behaviors may occur. In the worst case, robots are Byzantine, meaning that they can behave arbitrarily. A less serious fault is the crash fault, where a robot unexpectedly stops moving forever. Fault-tolerant algorithms for gathering were studied in [AP06; Déf+06; Cou+15b].

In this thesis, we focus on the discrete universe and study two main problems for asynchronous, identical, and oblivious robots: The gathering problem [KMP06; KKN08], where robots must all reach a common given location, and the exploration problem [Flo+13; DPT13], where the robots must visit all locations. In the next section, we informally describe existing algorithms for these two problems.

1.2 Gathering and exploration

One of the benchmarking problems for mobile robots is *gathering* [KKM10] (also known as the *Rendez-Vous* problem). This problem was the first studied in the literature. Regardless of their initial positions, robots have to move in such a way that they are eventually located on the same position, not known beforehand, and remain there thereafter. Similarly to the Consensus problem in conventional distributed systems, where all entities must agree on a same value, gathering has a simple definition but the existence of a solution greatly depends on the synchrony of the system.

The gathering problem draws its significance from the fact that it permits to obtain a common coordinate system. If the robots can gather at a single point, then they can agree to use that point as the origin of the common coordinate system. In the sequel we discuss the results obtained in the plane and in the discrete environment. A survey discussing how varying assumptions influence feasibility and complexity of gathering under different environments is presented in [Pel11].

1.2.1 Gathering in the plane

The gathering study begins with Suzuki and Yamashita [SY99] who propose a gathering algorithm for non-oblivious robots in a continuous Euclidean space, with a synchronous model. They proved that gathering cannot be solved with two oblivious robots: All configurations are symmetric and may lead to robots endlessly swapping their positions. However, Défago *et al.* [Déf+06] propose probabilistic algorithms in the ATOM model without any additional assumption. These algorithms permit to gather two robots with a fair scheduler, and any number of robots with a t-bounded scheduler.

Even if there is no deterministic algorithm for the gathering of two robots, the problem can be solved for three or more robots if there are no towers in the initial configuration: at each step, either the robots remain symmetric and they eventually reach the same location, or the symmetry is broken and this is used to move one robot at a time. Suzuki and Yamashita [SY99] propose an algorithm that exploits the properties of the center of gravity (sometimes called the center of mass, the barycenter, or the average) of the team.

When robots are asynchronous there might be various oscillatory effects on the computing of the center of gravity, preventing robots from moving towards each other and possibly even causing them to diverge and stay away from each other in certain scenarios.

Prencipe [Pre05] studied the problem of gathering in both synchronous and asynchronous models. He proved that the problem cannot be solved in Async without additional assumptions. The idea of this impossibility result is that from any initial configuration and for any algorithm with $k \geq 3$ robots, there exists an asynchronous scheduling producing a configuration where the set of robots are gathered into two different locations, reducing the problem to gathering with two robots. Moreover, Défago *et al.* [Déf+06] prove that, without additional assumption, there is no deterministic algorithm for the Async gathering even under a t-bounded scheduler. They propose a probabilistic algorithm to gather $k \geq 2$ robots under a t-bounded scheduler.

Therefore, to solve the gathering problem in Async, some additional assumptions must be made. Cieliebak [CP02; Cie+03] proposes to add weak global multiplicity detectors to the robots, or to use non-oblivious robots [Cie04]. In [CP02], some algorithms are presented for 3 or 4 robots, that handle all initial configurations without tower. The algorithm in [Cie+03] permits to gather 5 or more robots. The general idea on which these algorithms are based is to let the robots reach a configuration where there is exactly one location ℓ in the plane with multiplicity greater than 1. When such a configuration is reached, all the robots move towards ℓ avoiding collisions (i.e., ℓ remains the only point with a tower).

The gathering problem was also studied in a system where the robots have limited visibility [FPS12]. Ando *et al.* [And+99] present an algorithm allowing the convergence (towards a common point, without ever reaching it) in the Ssync model. In Ssync, robots with chirality and eventually consistent compasses, can gather [SDY09].
In the asynchronous model, the problem becomes solvable if the robots have a common coordinate system (agreement on axes and directions of a common coordinate system, but not necessarily on the origin nor on the unit distance) [Flo+05], then they can gather without multiplicity detectors even when the visibility is limited.

Fault-tolerant algorithms for gathering were studied in [AP06; Déf+06]. For the gathering problem, in a crash-prone system, there is no deterministic algorithm under a fair t-bounded scheduler, and there is no probabilistic algorithm under a fair scheduler [Déf+06]. However, the author proposes a probabilistic algorithm that solves the gathering under a t-bounded scheduler. When only non-faulty robots must gather, there is neither a probabilistic nor a deterministic algorithm that solves the gathering problem, for $k \geq 3$ robots when more than 2 faults happened, under a fair scheduler. But with only one fault Agmong *et al.* [AP06] present an algorithm in the asynchronous model under a fair scheduler.

For Byzantine faults, in Ssync it is impossible to gather all non-byzantines robots, even in the presence of a single Byzantine fault. In Fsync an algorithm is provided for gathering all non-byzantines robots with up to f faults, when the number of robots k is at least 3 and when $k \geq 3f + 1$. In [Déf+06] a probabilistic algorithm that solves the gathering of all non-byzantines robots is proposed. This algorithm requires a t-bounded scheduler and robots endowed with strong global multiplicity detectors.

1.2.2 Gathering in the discrete environment

In the discrete environment, gathering has been studied in various environments: trees [FP08], grids [DAn+12], rings [KMP06; KKN08], and general graphs [Des+06]. In the sequel we are only interested in the ring topology: The ring network is especially intricate since its regular structure induces a number of possible symmetric situations, from which the limited abilities of robots make it difficult to escape. In order to work with the most challenging environment, the ring is unoriented and anonymous (neither nodes nor links of the ring have any labels). Initially, some nodes of the ring are occupied by robots and there is at most one robot in each node.

In the discrete setting and more particularly in the ring, we use informal notions of symmetric and periodic organizations of robots, precise definitions are given in the next chapter.

Example 1. *In the ring configuration depicted in Figure 1.2, a white node represents a free location, and a grey node a location occupied by a robot. The first ring configuration is symmetric and the rightmost one is periodic i.e., invariant by non-trivial rotation.*

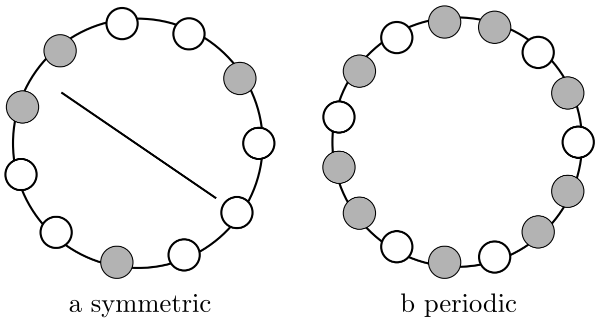


Figure 1.2: Particular configurations

The gathering problem on a ring was first investigated by Klasing *et al.* [KMP06]. They prove that gathering 2 robots is impossible on any ring, and gathering any number $k > 2$ robots is impossible without additional assumptions. Proofs of these impossibility results are similar to those in the plane. A more specific impossibility result is that even if robots are endowed with multiplicity detectors, gathering is impossible for a particular class of symmetric configurations. In the sequel we only discuss results given in the Async model.

When robots are endowed with weak global multiplicity detectors, the proposed protocols either exploit the symmetries [KKN08] or try to avoid them, and break them when encountered [KMP06]. In [KMP06] the protocol handles only an odd number of robots and starts from any tower-free configuration that is not periodic. In [KKN08], the authors provide an algorithm for gathering 18 or more robots on the ring, from any initial configuration not concerned by the impossibility results. For these initial configurations and less than 18 robots, a number of ring gathering algorithms have been proposed in the literature [DSN11b; DSN12; DAn+13; SN13]. They apply to various cases according to the size of the ring, the number of robots and the subclass of initial configurations. When multiplicity detection is available a unified strategy was proposed in [DSN12].

In [Izu+10; Kam+11; Kam+12] the authors achieve similar results with weak local multiplicity (robots are not able to see nodes that contain multiple robots unless it is their current node). The algorithm by Izumi *et al.* [Izu+10] assumes that initial configurations are neither symmetric nor periodic, and the number of robots is less than half the number of nodes. For an odd number of robots, Kamei *et al.* [Kam+11] propose an algorithm that also works from initial symmetric configurations. For an even number of robots on an odd-sized ring, Kamei *et al.* [Kam+12] propose an algorithm for non periodic initial configurations. An algorithm that achieves the gathering for any initial configuration where there is no impossibility result is presented in [DSN14]: it uses existing algorithms as subroutines for the basic solvable cases with 4 or 6 robots from [Kor10] and [DSN11a] respectively.

1.2.3 Exploration

Exploration is the process by which every location of a *discrete* environment is visited by at least one robot. The proposed algorithms mostly try to obtain a common sense of orientation: the intuition is to arrange the robots in a particular shape, which will allow them to have a common sense of direction, and then to define a direction to successfully explore their environment. The problem of exploration has several variants, among them the exploration with stop and the perpetual exploration problem. Both problems are unsolvable if the initial configuration is periodic [Flo+13].

The study of the exploration begins with the exploration with stop [Flo+13]: Robots achieve an exploration with stop if regardless of their initial location, the robots reach a configuration in which they all remain idle and each node has been visited by a robot. The difficulty of this task arises from the fact that robots need to stop after all locations have been explored. It requires robots to "remember" how much of the graph was explored: Since they have no persistent memory, this means that they must be able to distinguish between various stages of the exploration process.

Exploration with stop has been studied for paths [Flo+11], trees [Flo+10], grids [Dev+12], rings [Flo+13] and general graphs [Cha+10]. We focus again here on ring topologies, where the problem was studied only for robots endowed with strong global multiplicity sensors. Flocchini is the first to study this problem in a ring [Flo+13], she presents an algorithm that permits the exploration with stop for any $k \geq 17$ robots starting from any configuration without multiplicity and where the size of the ring and the number of robots are coprime. The idea is that if n and k are coprime then no periodic configuration can happen. Devisme *et al.* [DPT13] prove that there is no exploration protocol (even probabilistic) of an n-node ring with three robots for every $n > 3$. Moreover, there exists no deterministic protocol that can explore an even sized ring with $k \leq 4$ robots [LPT10]: 5 robots are necessary and sufficient when the size of the ring is even, and 5 robots are sufficient when the size of the ring is odd. In [LPT10] the authors also propose an Async algorithm for 5 robots in a n-node ring where n is coprime with five. The proposed algorithm is optimal in the number of robot moves.

Probabilistic algorithms for the exploration with stop have been studied by Devismes *et al.* [DPT13]. This work shows that four identical probabilistic robots are necessary and sufficient in the Ssync model, also removing the coprime constraint between the number of robots and the size of the ring. Their proof is constructive: they present a probabilistic protocol for four robots to explore any ring of size at least four, and show that there cannot exist any protocol with three robots that do the same.

The case where robots with limited visibility explore an n-size ring has been studied by Datta *et al.* [Dat+13]. When robots have a visibility of 1, the exploration problem is not solvable with deterministic algorithms in both Ssync and Async. Even in Fsync, the exploration problem cannot be solved with less than 5 robots when the ring size is more than 6. When they have a visibility of 3, no exploration is possible with less than 5 robots and a ring of size at least 13 in both Ssync and Async. In these conditions where the visibility is limited, several algorithms are proposed:

- Two solutions in the Fsync model, when the visibility is 1: with a minimal number of robots for $3 \leq n \leq 6$ and for $n > 6$.
- Two solutions in the Async model, when the visibility is 2: with 7 robots (for $n > 7$) and 9 robots (for $n > 2k + 1$).
- Two solutions in the Async model, when the visibility is 3, with 5 and 7 robots.

The more difficult problem of exclusive perpetual exploration has been studied recently: Robots achieve an exclusive perpetual exploration if regardless of their initial (tower-free) location, each node has been visited by a robot infinitely often and no multiplicity point appears. The latter happens as soon as a moving robot collides with another robot, moving or stationary. In this context, collisions are considered as undesirable events (with possibly negative consequences), and thus to be avoided. This is expressed by the *exclusivity property*, which states that any node must be occupied by at most one robot.

Exclusive perpetual exploration has been studied in rings [Bli+10; DAn+13], grids [Bon+11; Bal+08b], toruses [Dev+14], trees [BBN12] and general graphs [Bal+08a; BBN13]. Blin *et al.* [Bli+10] investigate both the minimal and the maximal number of robots that are necessary and sufficient to solve the exclusive perpetual exploration problem. They also propose algorithms for these two cases:

- On the minimal side, they prove that 3 robots are necessary and sufficient, provided that the size of the ring is at least 10, and show that no protocol with 3 robots can exclusively perpetually explore a ring of size less than 10.
- On the maximal side, they prove that $k = n - 5$ robots are necessary and sufficient to exclusively perpetually explore a ring of size n when n is co-prime with k.

A more generic algorithm has been proposed in [DAn+13]: starting from any tower-free configuration that is neither periodic nor symmetric, a ring of size at least 10 can be perpetually explored by at least 5 robots. The algorithm does not cover the case of 5 robots exploring a ring of size 10. Combination of the results in [Bli+10; DAn+13], leaves open the exclusive perpetual exploration of a ring of general size n by 4 robots.

All the algorithms described above have only been given handmade proofs, some of them rather sketchy. In the next section, we present cases where automated proofs were provided.

1.3 Formal Methods for robot algorithms

Formal methods require mathematical representations of the system and its specification, given as a set of properties. A distributed system is often described as a global transition system [Tel01; Lyn96b], obtained by a composition of models of its sub-systems. Properties can be classified into various types, among them the well known safety and liveness properties. Safety properties informally require that "something bad will never happen", like absence of deadlock. Invariants form an important subclass of safety properties, expressing that "something is true at every step in every execution". Liveness properties require that "something good eventually happens", for example there will be no starvation. Every possible specification can be written as the conjunction of safety and liveness properties [AS85].

In this section we present model checking, proof and synthesis, as well as related work for the use of these approaches in the context of mobile robot protocols.

1.3.1 Model checking

A model checker takes as input a model M, often in the form of a transition system, describing all possible executions of the system, and the property to be checked, expressed as a logic formula φ. It answers whether the model satisfies or not the formula. When the property is not satisfied, the model checker returns a counterexample, *i.e.*, an execution of the model invalidating the property. This counterexample is useful to find errors in complex systems. This is an advantage of model checking compared to the other formal methods, such as theorem proving, which can disprove a property but without systematically providing such a counterexample.

The automata approach for model checking was introduced by Vardi *et al.* [lincs 86] to provide a unified and extensible framework, initially applied to a class of logic formulas called LTL (described later in more details). This approach splits the verification process of an LTL formula into three operations:

- The language $\mathcal{L}(M)$ associated with M represents all possible executions of M. The formula φ is translated into an automaton $\mathcal{A}_{\neg\varphi}$ whose language, $\mathcal{L}(A_{\neg\varphi})$, is the set of all executions invalidating φ.
- Automata M and $\mathcal{A}_{\neg\varphi}$ are synchronized to obtain an automaton $M \times \mathcal{A}_{\neg\varphi}$ whose language $\mathcal{L}(M \times \mathcal{A}_{\neg\varphi}) = \mathcal{L}(M) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$, is the set of executions of M invalidating φ.
- Finally the model checker performs an emptiness check on this product. The model M satisfies φ if and only if $\mathcal{L}(M \times \mathcal{A}_{\neg\varphi}) = \emptyset$. If the emptiness check succeeds, it means that no execution invalidates φ, hence the property corresponding to φ is satisfied by M. Otherwise, an execution of M invalidating φ is produced as a counterexample.

The drawback of this method is the so-called combinatorial explosion problem [Val96] caused by the large size of the product $M \times \mathcal{A}_{\neg\varphi}$. The construction of $A_{\neg\varphi}$ is exponential in the size of φ. Moreover, starting from a system of concurrent processes, the automaton M of the system has a size exponential in the number of processes. Consequently, in the automata-theoretic approach the synchronous product is often too large for the emptiness check to be performed in a reasonable execution time and memory.

Distributed systems are naturally structured as a combination of components, among which several exhibit similar behaviors. Such components are said to be symmetric, and knowing the behavior of one such component is often sufficient to know the behavior of the combination. More formally, the symmetries of the system define an equivalence relation over its states. This relation can be used to produce a reduced state space, where at least one state per equivalence class is kept. If exactly one representative state per class is kept, then maximal reduction is achieved. The definition of symmetries guarantees that the reduced state space preserves properties, if the symmetries are respected [Cla+96; ES96]. This reduction is usually exponentially smaller than the original state space, thus reducing the execution time and memory of the verification process.

To our knowledge, in the context of mobile robots operating in discrete space, only one previous attempt, by Devismes *et al.* [Dev+12] investigates the possibility of automated verification of mobile robots protocols. They use LUSTRE [Hal+91] to describe and verify the problem of exploration with stop of a 3×3 grid by 3 robots in the Ssync model. They consider particular configurations with a tower of 2 robots and a single robot, where only the single robot wishes to move. For this case, they verify the invariant: *visited nodes* ≤ 4.

1.3.2 Proofs

In mechanical proof assistants, a user can express data, programs, theorems and proofs. Skeptical proof assistants provide an additional guarantee by checking mechanically the soundness of a proof after it has been interactively developed. They have been successfully employed for various tasks such as the formalization of programming language semantics [Ler09], certification of an OS kernel [Kle+10], verification of cryptographic protocols [Alm+12], etc. During the last twenty years, the use of tool-assisted verification has extended to the validation of distributed systems.

In the mobile robot model described above, mechanical proof assistants provided the certification of impossibility results regarding oblivious and anonymous mobile robots [Cou+15b], even in presence of byzantine behaviors [Aug+13]. A certified proof of the impossibility result from [SY99] is proposed in [Cou+15b], establishing that gathering is impossible with 2 robots. Courtieu *et al.* also provide a more general impossibility result: Gathering with an even number of robots, when any two robots are possibly initially at the same exact location is impossible.

1.3.3 Synthesis

Going one step further, it is interesting to not only verify or prove some existing algorithms, but to automatically generate an algorithm correct by construction, as done in the synthesis techniques. This problem takes as input a specification of a system interacting with an environment, and asks whether there exists a program satisfying this specification, regardless of the behavior of the environment. When the answer is positive, the program must be effectively built. A negative answer gives a proof that there is always a way for the environment to prevent the system from reaching its objective.

Let φ be the specification that the system must ensure, and let E be a model describing the environment. The synthesis problem asks whether there exists a program P such that $P \times E$ satisfies φ. The behavior of the system thus created must match exactly all behaviors eligible by the specification.

It may seem at first that the model actually needed is the one of *distributed games*, in which each robot represents a distinct player, all of them cooperating against a hostile environment. In distributed games, existence of a winning strategy for the team of players is undecidable [PR79]. However, the fact that robots are able to see their environment, and thus to always know the configuration of the system, allows us to stay in the framework of 2-player games, and to encode the set of robots as a single player. Of course, the strategy obtained will be centralized, but we will design the game in order to obtain only strategies that can be distributed among anonymous, memoryless robots without chirality.

To our knowledge, in the context of mobile robots operating in discrete space, only one previous attempt [Bon+12] investigates the possibility of automated synthesis of mobile robots protocols. The work considers the exclusive perpetual exploration by k robots of n-sized rings in the Ssync model. The approach is brute force: it mechanically generates all *unambiguous* protocols (those that do *not* have symmetric configurations), regardless of the problem to solve, and then check whether the protocol achieves gathering.

Following these attempts, we demonstrate in this work a larger use of model checking and synthesis in the context of robot protocols. Our approach differs from the previous ones, since our model is general enough to handle all atomicity models, and to accommodate various protocols. Previous works only handle the synchronous models, and apply on specific algorithms.

1.4 Contributions

In Chapter 2 we provide a formal model for a network of mobile robots operating under the three execution hypotheses described above, namely Fsync, Ssync and Async. We describe the logic LTL (Linear Temporal Logic) used to specify the requirements corresponding to robot tasks. Finally, we discuss implementation issues.

The rest of the manuscript presents our contributions and is divided into two parts. In Part I we formally verify two known protocols for variants of the ring exploration in an asynchronous setting: exploration with stop from [Flo+13] and exclusive perpetual exploration from [Bli+10]. Both protocols were given as informal descriptions in the original papers. This leads us either to a formal proof of correctness of the analyzed algorithms for particular instances or to a counterexample that shows a subtle flaw in the algorithm.

- In Chapter 3, we study the case of exploration with stop, and more particularly the protocol from [Flo+13]. This protocol was manually proved correct when the number of robots is $k > 17$, and n (the ring size) and k are coprime. As the necessity of this bound was not proved in the original paper, our methodology demonstrates that for many instances of k and n

not covered in the original paper, the protocol is still correct. We offer some conjecture for cases with $k \leq 17$.

- Then, in Chapter 4, we study the case of the perpetual exclusive exploration protocol [Bli+10]. In this case, we produce a counterexample in the asynchronous setting, where safety is violated. We correct the original protocol and verify the new one via model checking for several instances of n. Additionally, we prove the correctness of the protocol for any size of ring with an inductive approach.

In Part II, we show how automated synthesis can be applied to generate correct robot protocols, in the discrete space model. As a case study, we consider the problem of gathering.

- In Chapter 5, we propose an encoding of the gathering problem in a synchronous execution model as a reachability game, the players being the robot algorithm on one side and the scheduling adversary (that can also dynamically decide robot chirality at every activation) on the other side. Our encoding is general enough to encompass classical execution models for robots evolving on ring-shaped networks, including (and contrary to the existing ad-hoc solution [Bon+12]) when several robots are located at the same node and when symmetric situations occur. This allows us to automatically generate an *optimal* distributed algorithm, in the Fsync model, for three robots evolving on a fixed size ring. Our optimality criterion refers to the number of robot moves that are necessary to actually achieve gathering.

- In Chapter 6, we consider the asynchronous model. We first show how finding an algorithm for gathering asynchronous robots can be seen as an extension of a two player games, namely games with partial information. In these games, contrary to the previous ones, players have an incomplete view of the system. In order to fight the combinatorial explosion due to the asynchronous model, we propose a recursive algorithm that permits to obtain a gathering protocol in this setting, thanks to synchronous synthesis combined with model checking.

Chapter 2

Models and Notations

This chapter introduces the definitions and notations used in the rest of the manuscript. We first recall some definitions on graphs and automata that are useful to model the system, as well as definitions related to the Linear Temporal Logic, used to express specifications of the system. We then describe the robot model and its transcription into a product of automata. Finally, we discuss implementation issues and we introduce some mechanisms that permit to reduce the size of our model.

2.1 Background

Even if several different models have been used to represent distributed systems [Lyn96b; Tel01; BK08], we choose to use a general and simple one that permits to represent each processus of the system, and its own specificities. In particular the system will have global variables; these variables can be critical resources, or be used as communication channels or simply be shared data. Each process of the system is represented by a finite automaton which can handle these global variables. These automata once combined together represent the system.

If A is a finite alphabet, A^* is the set of finite sequences of elements of A (also called *words*), and A^ω is the set of infinite words, with ε the empty word. We note $A^+ = A^* \setminus \{\varepsilon\}$, and $A^\infty = A^* \cup A^\omega$. For a word $w \in A^\infty$, we denote its *length* by $|w|$, with $|w| = +\infty$ for any $w \in A^\omega$. For two words $w = a_1 \cdots a_k \in A^*$, $w' = a'_1 \cdots \in A^\infty$, we define the *concatenation* of w and w' by the word denoted by $w \cdot w' = a_1 \cdots a_k a'_1 \cdots$. We usually omit the dot symbol and simply write ww'. If $L \subseteq A^*$ and $L' \subseteq A^\infty$, we define $L \cdot L' = \{w \cdot w' \mid w \in L, w' \in L'\}$.

We denote by $\mathbb{N}$ the set of natural numbers and for any $n \in \mathbb{N}$, we use arithmetic modulo n, with operations written as $+_n$ (and $-_n$).

2.1.1 Directed Graphs and automata

An directed graph is set of nodes connected by directed edges:

Definition 1 (Directed graph)**.** *A directed graph is a pair $G = (V, E)$ where V is a set of vertices and $E \subseteq V \times V$.*

In a directed edge $e = (v_1, v_2) \in E$, v_2 is said to be a direct successor of v_1, and v_1 is said to be a direct predecessor of v_2. In a directed graph, a path $\pi = v_0 v_1 \cdots \in V^\infty$ is a finite or infinite sequence of vertices such that for all $0 < i < |\pi|$, $(v_{i-1}, v_i) \in E$.

We now recall the classical definition of finite automata. A finite automaton can be seen as a graph with an initial vertex and where edges are equipped with labels.

Definition 2 (Finite automaton)**.** *A finite automaton M is a tuple (S, s_0, A, T) where S is a finite set of states, $s_0 \in S$ is the initial state, A is a finite alphabet of actions and $T \subseteq S \times (A \cup \{\varepsilon\}) \times S$ is a finite set of transitions.*

A state describes the process at a certain step of its behavior, and a transition indicates an action that may involve a state change. A transition (s, a, s'), written $s \xrightarrow{a} s'$, represents a transition of the automaton from state s to state s' by executing the action a. The empty word ε is used as a label to represent an unobservable (or internal) action.

An execution of M is a sequence of transitions (s_0, a_1, s_1), (s_1, a_2, s_2), ... beginning in the initial state s_0, written $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots$.

For the synchronized product, we introduce a new symbol $-$, denoting the absence of action for a component. This label implies that the state of the corresponding component does not change and should not be confused with a non observable action labeled by ε.

Definition 3 (Product of automata)**.** *Let $M_1 = (S_1, s_{1,0}, A_1, T_1)$ and $M_2 = (S_2, s_{2,0}, A_2, T_2)$ be two finite automata and let A be an alphabet. A partial synchronization function is a mapping $f : (A_1 \cup \{\varepsilon, -\}) \times (A_2 \cup \{\varepsilon, -\}) \to A \cup \{\varepsilon\}$ such that $f(\varepsilon, \varepsilon) = f(\varepsilon, -) = f(-, \varepsilon) = \varepsilon$, and $f(-, -)$ is undefined.*

The product $M = (S, s_0, A, T) = M_1 \otimes_f M_2$ is defined as follows:

- *$S = S_1 \times S_2$ is the cartesian product of S_1 and S_2, with $s_0 = (s_{1,0}, s_{2,0})$ the initial state,*
- *the set T of transitions contains the transition $(s_1, s_2) \xrightarrow{c} (s'_1, s'_2)$ iff*
  - *there is $(a, b) \in (A_1 \cup \{\varepsilon\}) \times (A_2 \cup \{\varepsilon\})$ on which f is defined with $c = f(a, b)$, and $s_1 \xrightarrow{a} s'_1 \in T_1$, $s_2 \xrightarrow{b} s'_2 \in T_2$,*
  - *or there is $a \in A_1 \cup \{\varepsilon\}$ such that $f(a, -)$ is defined with $c = f(a, -)$, $s_1 \xrightarrow{a} s'_1 \in T_1$, and $s'_2 = s_2$,*
  - *or there is $b \in A_2 \cup \{\varepsilon\}$ such that $f(-, b)$ is defined with $c = f(-, b)$, $s'_1 = s_1$, and $s_2 \xrightarrow{b} s'_2 \in T_2$.*

This definition can be easily extended to a set of n automata M_1, ..., M_n with a n-ary synchronization function.

2.1.2 Synchronous and asynchronous execution models

The two extremal cases of synchronized product correspond respectively to totally synchronous and totally asynchronous systems. A synchronous system consists of synchronized rounds of message exchanges and/or computations. Hence the synchronization is the most constrained and is defined by $f(a,b) = c$ if and only if there is $(a,b) \in (A_1 \cup \{\varepsilon\}) \times (A_2 \cup \{\varepsilon\})$ such that $s_1 \xrightarrow{a} s_1' \in T_1$ and $s_2 \xrightarrow{b} s_2' \in T_2$.

In a totally asynchronous system the processes can interleave their steps in an arbitrary order. It represents the difference between processes speed. The corresponding synchronization function is simply defined on pairs $(a,-)$ or $(-,b)$, when $a \in A_1 \cup \{\varepsilon\}$, and $b \in A_2 \cup \{\varepsilon\}$ and $s_1 \xrightarrow{a} s_1' \in T_1$, $s_2 \xrightarrow{b} s_2' \in T_2$, by $f(a,-) = a$, and $f(-,b) = b$.

2.1.3 Communication

In our model robots are devoid of any mean of direct communication, the only way of them to communicate is made thanks to their observations. Robot observation can be seen as a communication by share variables: when a robot observes the positions of other robots, it reads some informations about the other robots and by moving it writes a change of its own information.

Process actions now can be divided into three groups: internal events, reading some variables or writing some variables. The system is an automaton $M = (S, s_0, A, T)$ obtained by the product of the process automata. For a system of p processes, the system states are of the form $s = (s_1, s_2, \ldots, s_p, V)$, where s_i is the local state of process i, and V is a variable valuation. For the initial state s_0 initial valuation for all variables is given.

A transition of the system is of the form $(s_1, \ldots, s_p, V) \xrightarrow{a} (s_1', \ldots, s_p', V')$ where $a = f(a_1, \ldots, a_p)$ is simply denoted by the tuple $(a_1, \ldots, a_p)$ and defined for $a_i \in A_i \cup \{\varepsilon, -\}$, $1 \le i \le p$, if and only if for any (i,j) such that $i \neq j$, a_i and a_j do not access the same variables.

2.2 LTL specifications

2.2.1 Notations

Given a set $\mathcal{P}$ of atomic propositions, the temporal logic LTL (for Linear Temporal Logic) is a specification language interpreted on infinite sequences over $2^{\mathcal{P}}$. The LTL formulae are defined by the following grammar:

$$\varphi ::= \; p \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2$$

where $p \in \mathcal{P}$, $\vee$ is the boolean disjunction, $\neg$ is the negation and X (next) and U (until) are temporal operators described below. Moreover two temporal operators $\Diamond$ and $\Box$ are usually defined from U by: $\Diamond\varphi = \mathit{true}\mathsf{U}\varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$. The formula $\Diamond\varphi$ states that φ holds *eventually*, and $\Box\varphi$ is satisfied *iff* φ holds *forever* from now on. Temporal and boolean operators can be nested. For instance $\Diamond\Box\varphi$ expresses that from some position in the future φ always holds, and $\Box\Diamond\varphi$ states that φ is satisfied infinitely often.

For $w \in (2^{\mathcal{P}})^\omega$, written $w = w_1 w_2, \ldots$ with $w_i \in 2^{\mathcal{P}}$, we note $w, i \vDash \varphi$ when φ is satisfied at position i of w (*i.e.*, from w_i on). The satisfaction relation is defined inductively by the rules given in Table 2.1.

$w, i \vDash p$	*iff*	$p \in \mathcal{L}(s_i)$
$w, i \vDash \neg\varphi$	*iff*	$w, i \nvDash \varphi$
$w, i \vDash \varphi_1 \vee \varphi_2$	*iff*	$w, i \vDash \varphi_1$ or $w, i \vDash \varphi_2$
$w, i \vDash \mathsf{X}\varphi$	*iff*	$w, i+1 \vDash \varphi$
$w, i \vDash \varphi_1 \mathsf{U} \varphi_2$	*iff*	$\exists j \geq i \mid w, j \vDash \varphi_2$ and $\forall i \leq k < j,\ w, k \vDash \varphi_1$

Table 2.1: LTL satisfaction relation.

To interpret LTL formulae on executions of an automaton $M = (S, s_0, A, T)$, the transition relation is assumed to be *non blocking*: for each $s \in S$, there is at least one transition starting from s. Moreover, a labeling function $\mathcal{L} : S \to 2^{\mathcal{P}}$ is added to M. This function maps each state of M to a set of atomic propositions that hold in this state. With execution $e : s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots$ of M, we associate the infinite word $w = \mathcal{L}(s_0)\mathcal{L}(s_1), \ldots$. For formula φ, we note $e, i \vDash \varphi$ if $w, i \vDash \varphi$.

Definition 4. *An automaton M, with labeling $\mathcal{L}$, satisfies φ if for each execution e of M, $e, 0 \vDash \varphi$.*

Given an automaton M that represents all possible behaviors of a system and an LTL formula φ describing a requirement on the system, LTL model-checking answers the question whether $M \models \varphi$ or not. When the answer is negative, a counter-example can be exhibited.

2.2.2 Fairness

For our purpose, correctness of the algorithms must be satisfied with a fair scheduler. Therefore, only fair executions of the system are considered. To express fairness properties in LTL, we consider two propositions associated with a process p: $\mathit{enabled}_p$ describes a state of process p where at least one action is enabled and $\mathit{executed}_p$ corresponds to a state where p is scheduled.

Strong Fairness For all processes p: $(\Box\Diamond \mathit{enabled}_p) \implies (\Box\Diamond \mathit{executed}_p)$.

Weak Fairness For all processes p: $(\Diamond\Box \mathit{enabled}_p) \implies (\Box\Diamond \mathit{executed}_p)$.

Unconditional fairness For all processes p: $\Box\Diamond \mathit{executed}_p$.

In order to verify a property φ only on fair executions of the system modeled by M, we in fact verify $M \models (\mathit{fair} \implies \varphi)$. In the sequel fairness means unconditional fairness.

2.3 Model for the robot algorithms

In the sequel we are interested in systems where all robots execute the same algorithm [FPS12], hence the behavior of each of them can be described by the same finite automaton. We first explain how this automaton describe the robot behavior. We then introduce a scheduler model that permits to obtain different execution models by representing the synchronization function by automata. The system is finally obtained by synchronizing robots with the scheduler, and we discuss implementation of the model with respect to the input language of specific tools.

2.3.1 Robot Modeling

Robots operate in *Look*, *Compute*, and *Move cycles* that can be seen in the automaton of Figure 2.1.

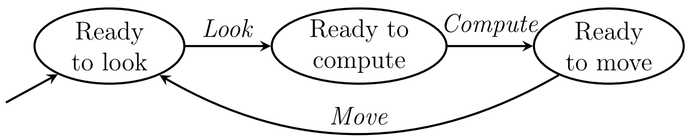


Figure 2.1: A generic automaton for the robot behavior.

To start a cycle, a robot takes a snapshot of its environment which is represented by the *Look* transition. Then, it computes its future location, represented by the *Compute* transition. Finally the robot moves according to its previous computation, this effective movement is represented by the *Move* transition.

In the sequel we work on the discrete environment represented by a graph, where nodes represent locations, and edges represent the possibility for a robot to move from one location to the other

The algorithm is implemented in the *Compute* transition, hence the "Ready to move" state is divided into as many parts as there are possible movements according to the protocol under study and the graph topology.

Note that the original model abstracts the precise time constraints (like the computational power or the locomotion speed of robots) and keeps only sequences of instantaneous actions, assuming that each robot completes each cycle in finite time. The model can be reduced by combining the *Look* and *Compute* phases to obtain the *LC* phase. This is simply done by merging the two states "Ready to look" and "Ready to compute" into a single state "Ready to Look-Compute".

To ensure the progress of the protocols, an implicit *fairness* assumption states that all robots must be infinitely often scheduled, which is expressed in LTL by:

$$\mathit{Fairness}: \bigwedge_{i=1}^{k} \Box\Diamond(RM_i) \ \wedge \ \bigwedge_{i=1}^{k} \Box\Diamond(RLC_i)$$

where RM_i (respectively RLC_i) is the label corresponding to one of the states "Ready to Move" (resp. to the state "Ready to Look-Compute") of robot r_i.

2.3.2 Scheduler Modeling

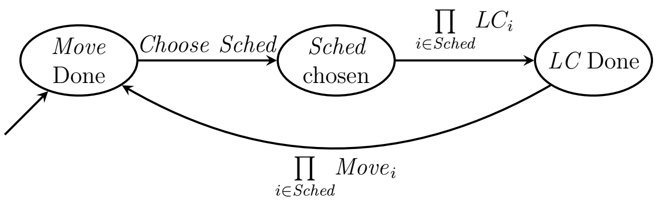
The scheduler organizes robot movements to obtain all possible behaviors with respect to the execution model, which depends on the synchronization hypotheses and defines the particular synchronization function. As the robots it is modeled by a finite automaton, one for each variant of the execution model, but unlike robots that have the same behavior regardless of the model, the scheduler is parameterized by the number of robots. By synchronizing one of these schedulers with robot automata, we obtain an automaton that represents the global behavior of robots in the chosen model.

To describe these scheduler models, we consider a set $Rob = \{r_1, \ldots, r_k\}$ of k robots. We denote by LC_i (respectively $Move_i$), the *LC* (resp. *Move*) phase of robot r_i. Note that LC_i and $Move_i$ are actually sets of possible actions in the corresponding phases. For a subset $Sched \subseteq Rob$, we denote the synchronization of all LC_i (respectively $Move_i$) actions of all robots in *Sched* by:
$\prod_{r_i \in Sched} LC_i$ (respectively $\prod_{r_i \in Sched} Move_i$).

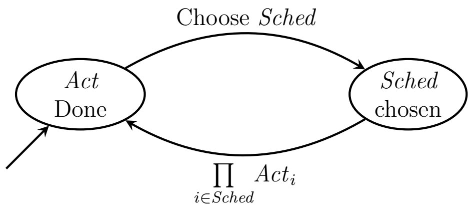
The particular execution models are: Fsync (Fully Synchronous), Ssync (Semi Synchronous) or Async (Asynchronoous). We describe each of them in the sequel.

In the Ssync model, a non-empty subset of robots is scheduled for execution at every phase, by choosing first a subset *Sched* of, and operations are executed synchronously. In this case, the automaton is a cycle, where a set $Sched \subseteq Rob$ is first chosen. In this cycle the *LC* and *Move* phases are synchronized for this set of robots. A generic automaton for Ssync is described in Figure 2.2a. Actually, the "*Sched* chosen" state has to be divided into 2^k states, where k is the number of robots, in order to represent all possible sets *Sched*.

The Fsync model is a particular case of the Ssync model, where all robots are scheduled for execution at every phase, and operate synchronously thereafter: In each global cycle, $Sched = Rob$, hence all global cycles are identical.



a) Ssync model.



b) Async model.

Figure 2.2: Scheduler automata.

The scheduler for the Async model is the one representing a totally asynchronous synchronization function. Any finite delay may elapse between LC and *Move* phases: During each phase a set *Sched* is chosen, and all robots in this set execute an action: the action Act_i is either in LC_i or in $Move_i$ depending on the current state of robot r_i. Hence, a robot can move according to an outdated observation. The automaton for this scheduler is depicted in Figure 2.2b.

2.3.3 System Modeling

A configuration c of the system describes robot positions on the graph. It is a mapping $c : Rob \to Pos$ associating with each robot r its position $c(r) \in Pos$. Hence, in a graph of n nodes with k robots there are n^k possible configurations. Let $\mathcal{C}$ be the set of all possible configurations of the system.

The model of the system is an automaton $M = (S, s_0, A, T)$ obtained by the synchronized product of k robot automata and all the possible configurations, as defined above (Section 2.1.1), where the scheduler is used to define the synchronization function. The alphabet of actions is $A = \prod_{r_i \in Rob} A_i$, with $A_i = LC_i \cup Move_i$ for each robot r_i. In the resulting automaton, states, also called system states in the sequel, are of the form $s = (s_1, \dots, s_k, c)$ where s_i is the local state of robot r_i, and c the configuration. An initial state is of the form $s_0 = (s_{1,0}, \dots, s_{k,0}, c)$ where $s_{i,0}$ is the initial local state of robot r_i and $c \in \mathcal{C}$ is an arbitrary configuration.

A transition of the system is labeled by a tuple $a = (a_1, \dots, a_k)$, where $a_i \in A_i \cup \{\varepsilon, -\}$ for all $1 \le i \le k$ and $(s_1, \dots, s_k, c) \xrightarrow{a} (s'_1, \dots, s'_k, c')$ if and only if for all i, $s_i \xrightarrow{a_i} s'_i$ and c' is obtained from c by updating the positions of all robots such that $a_i \in Move_i$. To represent the scheduling, we denote by $\prod_{r_i \in Sched} Act_i$ the action $(a_1, \ldots, a_k)$ such that $a_i = -$ if $r_i \notin Sched$ and $a_i \in LC_i \cup Move_i \cup \{\varepsilon\}$ otherwise.

2.3.4 Implementation issues

For our verification purpose, we consider two model-checkers: DiVinE [Bar+13] and ITS-tools [Col+13]. We chose these model-checkers for their ability to deal with large models and formulae, by using parallel computations for the first one or a symbolic approach for the second one. Moreover, both tools provide several metrics such as the number of states and transitions and they can handle the same input files. In particular, the original modeling language of DiVinE is DVE, which is also interpreted by ITS-tools. A DVE system is composed of processes, that are automata where transitions can be guarded by a *condition* (or *guard*) that determines if the transition can be fired. Therefore, the transcription of algorithms in DVE is straightforward.

In general, protocols in Fsync, Ssync, or Async models are described as a set of guarded actions. A guard of a robot algorithm is a guard on a LC transition. Transitions have so-called *effects* that actually are assignments to local or global variables. These correspond to the actions of a guarded-action algorithm. When two transitions can be fired, one of them is chosen nondeterministically.

Although the DiVinE language has a large expressive power, we had to deal with an important restriction: The DVE language cannot synchronize more than two automata. Therefore, we implement synchronized actions using a sequential order such that look/compute actions (LC_i) are executed first, and the move actions ($Move_i$) afterward.

More formally, we obtain the following system: $M' = (S', S_0, A, T')$ where S' is defined similarly to S, with the addition of a labeling of states (explained below), to indicate if the state is a transient or a steady state. The transition relation is defined as follows: Any transition $s \xrightarrow{a} s'$ in M is replaced in M' by a sequence of transitions, where all intermediate states are labeled as transient, while s and s' are steady states. More precisely, we note $\hat{a}_i = (-, \ldots, -, a_i, -, \ldots, -)$ the tuple of actions where only the robot r_i executes $a_i \in A_i$. An action $a = \prod_{r_i \in Sched} Act_i$ is executed as the sequence of actions $\hat{\ell}_1, \ldots, \hat{\ell}_k, \hat{m}_1, \ldots, \hat{m}_k$ where $\ell_i \in LC_i$ if $Act_i \in LC_i$ and $-$ otherwise, and similarly, $m_i \in Move_i$ if $Act_i \in Move_i$ and $-$ otherwise. Note that for each i, $\hat{\ell}_i$ and $\hat{m}_i$ are either $(-, \ldots, -)$ (containing only $-$, which corresponds to no action from any robot), or belong to $\{\hat{Act_i} \mid r_i \in Sched\}$.

Let $Exec(M)$ and $Exec(M')$ be respectively the set of executions of M and M'. We denote by $cf(e)$ the sequence of configurations in $e \in Exec(M)$ and by $cfs(e')$ the sequence of configurations of the steady states in $e' \in Exec(M')$. This notation is extended to the set of executions of M and M' by:

$$cf(Exec(M)) = \{cf(e), e \in Exec(M)\},$$

$$cfs(Exec(M')) = \{cfs(e), e \in Exec(M')\}.$$

We say that two executions $e \in Exec(M)$ and $e' \in Exec(M')$ are equivalent if $cf(e) = cfs(e')$.

Definition 5 (Model equivalence)**.** *The models M and M' are equivalent if*

$$cf(Exec(M)) = cfs(Exec(M')).$$

The following theorem states that our implementation is equivalent to the abstract asynchronous model (see Figure 2.2b).

Theorem 1. *The DiVinE implementation is equivalent to the abstract Async model.*

Proof. Let M be the abstract Async model and M' the model obtained from M as described above. Since M represents the most general behavior and contains all possible executions of the system, we clearly have $cfs(Exec(M')) \subseteq cf(Exec(M))$. To obtain the converse inclusion, we must prove that for each execution $e \in Exec(M)$ we can find an execution $e' \in Exec(M')$ such that e and e' are equivalent. This amounts to prove that M' simulates M for a simulation relation linking a state of M with the corresponding steady state of M', as well as with all the consecutive transient states.

Let $e \in Exec(M)$. With any transition $t : s \xrightarrow{a} s'$ in e, with $a = (a_1, \ldots, a_k)$, we associate the execution e_t in M' defined above by:

$$s \xrightarrow{\hat{\ell}_1} s_1 \ldots \xrightarrow{\hat{\ell}_k} s_k \xrightarrow{\hat{m}_1} s'_1 \ldots \xrightarrow{\hat{m}_k} s'_k$$

with all look actions before all move actions. We now define the execution $e' \in M'$ by replacing all transitions t in e by e_t. We must now prove that e and e' are equivalent.

For this, we show that each transition $t : s \xrightarrow{a} s'$ is equivalent to e_t by examining the ordering of actions. We say that two actions $\hat{a}_i$ and $\hat{a}_j$ commute, if for any system state p, if $r \xrightarrow{a_i} p_1 \xrightarrow{a_j} p'$, there exists p'_1 such that $p \xrightarrow{a_j} p'_1 \xrightarrow{a_i} p'$. This expresses the fact that the state reached is independent of the order of actions $\hat{a}_i$ and $\hat{a}_j$. Clearly, any two LC actions commute since they only modify the local state of the robot they belong to, and only depend on the current configuration that is not updated by LC_i. Similarly, any two *Move* actions on different robots commute, since they successively update the positions of robots i and j in c. Moreover, from the definition of M, all actions $\hat{a}_i$ being simultaneous, the LC actions must observe the initial configuration c in the initial steady state s. Therefore, since all LC actions appear before the move actions in e_t, this (sequential) execution is equivalent to the (simultaneous) version t. Combining all transitions in e', we obtain that e' and e are equivalent, which concludes the proof. □

2.4 Methodology for the ring

We denote by $Pos = \{0, \ldots, n-1\} \subseteq \mathbb{N}$ the set of positions on a ring of size n. Note that node $i +_n 1$ is the successor of node i in the clockwise direction.

2.4.1 Configurations

Since robots and nodes are anonymous, we gather the different configurations in an equivalence class such that only relative positions of the robots are taken into account. In the sequel we denote by $\circ$ the composition of applications, and by id the identity function.

Let π and $\overline{\pi}$ be the permutations of Pos defined as follows:

- $\pi(i) = i +_n 1$ for $i \in Pos$, $\pi^0 = id$ and $\pi^{m+1} = \pi \circ \pi^m$ for any $m \in \mathbb{N}$.
- $\overline{\pi}(i) = n -_n i$ for $i \in Pos$.

The first permutation corresponds to a one step shift in the clockwise direction, while the second one is an axial symmetry with respect to the diameter of the ring containing node 0. Hence $\overline{\pi}^2 = \pi^n = id$, so that $\overline{\pi}^{-1} = \overline{\pi}$, and we denote by π^{-m} the inverse of π^m.

Definition 6 (Configuration equivalence)**.** *The equivalence relation $\approx$ on the set Pos^{Rob} of configurations is defined for configurations c and c' by: $c \approx c'$ if there exist an integer m and some permutation β of Rob such that $c' = \pi^m \circ c \circ \beta$ or $c' = \overline{\pi} \circ \pi^m \circ c \circ \beta$.*

Example 2. *All configurations in Figure 2.3 are equivalent.*
Configuration a: $a(r_1) = 2$, $a(r_2) = 4$, $a(r_3) = 7$, $a(r_4) = 4$ is equivalent to configuration b: $b(r_1) = 4$, $b(r_2) = 6$, $b(r_3) = 9$, $b(r_4) = 6$ since $b = \pi^{-2} \circ a$.
Configuration b is equivalent to configuration c: $c(r_1) = 8$, $c(r_2) = 6$, $c(r_3) = 3$, $c(r_4) = 6$. We have $c = \overline{\pi} \circ b$ and $b = \pi^{-2} \circ a$, thus $c = \overline{\pi} \circ \pi^{-2} \circ a$.
Configuration c is equivalent to configuration d: $d(r_1) = 3$, $d(r_2) = 6$, $d(r_3) = 8$, $d(r_4) = 6$. Let β be the permutation defined by: $\beta(r_1) = r_3$, $\beta(r_2) = r_2$, $\beta(r_3) = r_1$, and $\beta(r_4) = r_4$. We have $d = a \circ \beta$, hence we have $d = \overline{\pi} \circ b \circ \beta$ and $d = \overline{\pi} \circ \pi^{-2} \circ a \circ \beta$.

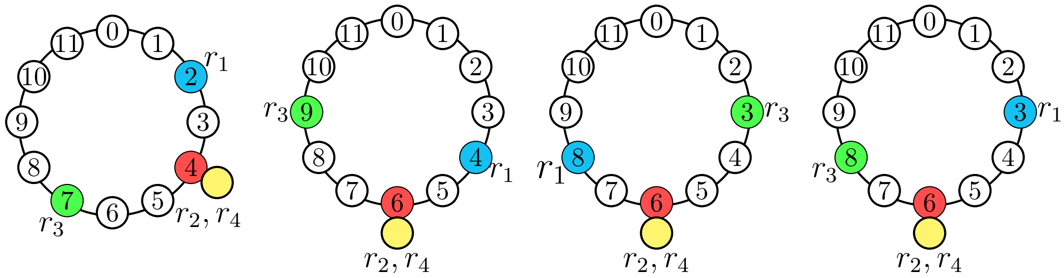


Figure 2.3: Equivalent configurations

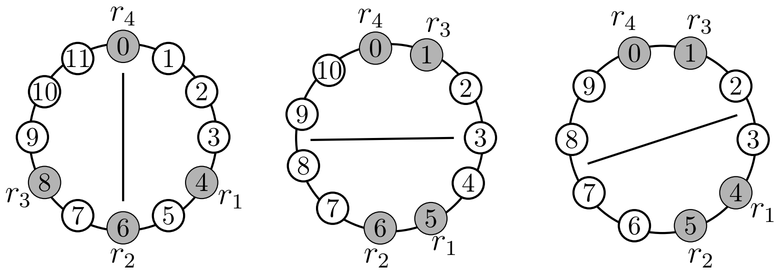
A configuration is *symmetrical* if there exists an axis of symmetry, that maps single robots into single robots, multiplicities into multiplicities, and empty nodes into empty nodes.

Definition 7 (Symmetries). *Configurations c and c' are* symmetrical*, written c sym c', if there are some m, m' and β such that $c' = \pi^m \circ \overline{\pi} \circ \pi^{-m'} \circ c \circ \beta$. Configuration c is* symmetrical *if c sym c.*

A symmetric configuration can be edge-edge, node-edge or node-node symmetrical if the axis goes through two edges, through one node and one edge, or through two nodes, respectively.

Example 3. *Configurations in Figure 2.5 are symmetric.*
In configuration $a : a(r_1) = 3$, $a(r_2) = 6$, and $a(r_3) = 9$, the axis of symmetry is the diameter that goes through the node 0, and the node 6. Let β be the permutation defined by: $\beta(r_1) = r_3$, $\beta(r_3) = r_1$, and $\beta(r_2) = r_2$. The configuration a is symmetrical since $a = \pi^6 \circ \overline{\pi} \circ \pi^{-6} \circ a \circ \beta$.
The configuration b is symmetrical with an axis of symmetry that goes through the node 3 and the edge $8-9$, since $b = \pi^3 \circ \overline{\pi} \circ \pi^{-3} \circ b \circ \beta$, where β is the robot permutation defined by: $\beta(r_1) = r_3$, $\beta(r_3) = r_1$, $\beta(r_2) = r_4$, and $\beta(r_4) = r_2$.
The configuration c is symmetrical with an axis of symmetry that goes through the edges $2-3$ and $7-8$, since $c = \pi^1 \circ \overline{\pi} \circ \pi^{-2} \circ c \circ \beta$, where β is the robot permutation defined by: $\beta(r_1) = r_3$, $\beta(r_3) = r_1$, $\beta(r_2) = r_4$, and $\beta(r_4) = r_2$.

When a configuration present several axes of symmetry, the configuration is periodic: it means that the configuration is invariant by non-trivial rotation. A configuration can be periodic without axis of symmetry, moreover we call a configuration that is neither periodic nor symmetrical nor contains a tower a *rigid* configuration. Note that a symmetric configuration is not periodic if and only if

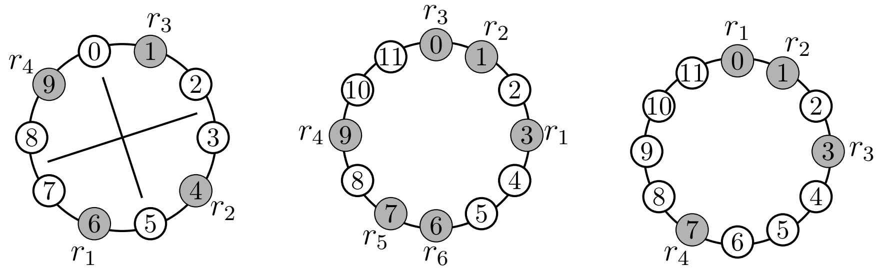


a : node-node b : node-edge c : edge-edge

Figure 2.4: Symmetrical configurations

it has exactly one axis of symmetry, and that a periodic configuration only occurs when$k$ is a multiple of n.

Example 4. *The configuration a is* periodic *due to several axes of symmetry. The configuration b is periodic but does not contain any axis of symmetry. The configuration c is rigid.*



a : periodical b : periodical c : rigid

Figure 2.5: Other types of configurations

In a configuration, each robot can observe the entire ring, from its own position. The robots take a snapshot of their environment to compute their future movement. Since this snapshot represents the graph, it depends on the ring topology.

2.4.2 Observations

Since the robots have no chirality, they can not distinguish the clockwise and the counter-clockwise direction. To describe robot observations we use the consecutive numbers of free nodes seen in a direction: We define $\mathcal{F} = \{(f_1, \cdots, f_k) \mid \Sigma_{i=1}^{k} f_i = n-k,\ f_i \in \{-1, 0, \cdots, n-1\}\}$. A k-tuple in $\mathcal{F}$ represents a possible observation of a robot in some direction relative to some configuration, where the f_is are the number of free nodes between consecutive robots in this direction. For $F = (f_1, \cdots, f_k) \in \mathcal{F}$, we set $\tilde{F} = (f_k, \cdots, f_1)$ the observation in the opposite direction to F. When two consecutive robots occupy adjacent nodes, $f_i = 0$, and when these two robots occupy the same node, $f_i = -1$.

Definition 8 (Observations). *The set of observations is* $Obs = \{\{F, \tilde{F}\}|\ F \in \mathcal{F}\}$.

Note that when $F = \tilde{F}$, the corresponding observation is a singleton. For an observation $o = \{(f_1, \cdots, f_k), (f_k, \cdots, f_1)\}$ in *Obs*, we define the *canonical configuration* by: $c_o(r_1) = 0$, $c_o(r_2) = f_1 +_n 1, \cdots, c_o(r_k) = \Sigma_{i=1}^{k} f_i +_n k$. Then o is the observation of r_1 in c_o, also written $obs(r_1, c_o)$. Note that o is also the observation of r_1 in the configuration $\overline{\pi} \circ c_o$ or in all configurations $c_o \circ \beta$ for any permutation β of *Rob* such that $\beta(r_1) = r_1$ and in all configurations $\pi^m \circ c_o$ for any m.

Example 5. *We illustrate this notion with examples from Figure 2.3 and 2.5:*
In Figure 2.3:

- $obs(r_1, a) = \{(1, -1, 2, 6), (6, 2, -1, 1)\} = obs(r_1, b) = obs(r_1, c)$,
- $obs(r_2, a) = \{(-1, 2, 6, 1), (1, 6, 2, -1)\} = obs(r_2, b) = obs(r_2, c) = obs(r_2, d)$,
- $obs(r_3, a) = \{(6, 1, -1, 2), (2, -1, 1, 6)\} = obs(r_3, b) = obs(r_3, c)$,
- $obs(r_4, a) = \{(2, 6, 1, -1), (-1, 1, 6, 2)\} = obs(r_4, b) = obs(r_4, c) = obs(r_4, d)$.

As depicted in Figure 2.5 when the configuration is symmetrical with respect to a given axis, two "corresponding" robots on both sides of the axis have the same observation. For simplification purpose we say that these robots are symmetrical. Moreover if there is a single robot on the axis then its observation is a singleton.

- $obs(r_1, a) = \{(2, 2, 4), (4, 2, 2)\} = obs(r_3, a)$,
- $obs(r_1, b) = obs(r_3, b)$ *and* $obs(r_2, b) = obs(r_4, b)$,
- $obs(r_2, a) = \{(2, 4, 2)\}$.

$obs(r_1, (a)) = \{(1, -1, 2, 6), (6, 2, -1, 1)\} = obs(r_3, (d))$,

We define on $\mathcal{F}$ the following relations:

- The rotation relation $\circlearrowleft \subseteq \mathcal{F} \times \mathcal{F}$ defined by: for all F, $F' \in \mathcal{F}$, $F \circlearrowleft F'$ if and only if $F = (f_i, f_{i+_k 1}, \cdots, f_{i+_k k-1})$ and $F' = (f_{i+_k 1}, f_{i+_k 2}, \cdots f_{i+_k k})$.

- The mirror relation $\sim \subseteq \mathcal{F} \times \mathcal{F}$ defined by: for all F, $F' \in \mathcal{F}$, $F \sim F'$ if and only if $F' = \tilde{F}$.

Combining the rotation relation and the mirror relation as $(\circlearrowright \cup \sim)^*$ produces an equivalence relation on $\mathcal{F}$:

Definition 9 (Equivalence on $\mathcal{F}$)**.** *The equivalence relation* $\equiv \subseteq \mathcal{F} \times \mathcal{F}$ *is defined by* $\equiv \stackrel{\text{def}}{=} (\circlearrowright \cup \sim)^*$.

We overload the relations $\circlearrowright$ and $\equiv$ on *Obs*. Let the rotation relation on *Obs*: $\circlearrowright \subseteq Obs \times Obs$ be defined for two observations o and o' by $o \circlearrowright o'$ if $o = \{F, \tilde{F}\}$, $o' = \{F', \tilde{F'}\}$, and $F \circlearrowright F'$. Since *Obs* is closed by symmetry, the equivalence relation $\equiv$ on *Obs* can be reduced to the reflexive and transitive closure $\circlearrowright^*$ of $\circlearrowright$ on *Obs*.

We define a mapping $rep : Obs/\!\equiv \to Obs$, associating with each equivalence class a unique representative: Writing $[o]_\equiv = \{\{F_1, \tilde{F_1}\}, \ldots, \{F_h, \tilde{F_h}\}\}$, we choose F as the minimal k-tuple (for the lexicographical order) in the subset $\{F_1, \tilde{F_1}, \ldots, F_h, \tilde{F_h}\}$ of $\mathcal{F}$. Then $rep([o]_\equiv) = \{F, \tilde{F}\}$.

The link between relation $\approx$ on configurations and $\equiv$ is given by the following proposition:

Proposition 2. *For two configurations* $c, c' \in \mathcal{C}$, $c \approx c'$ *if and only if there exist* $r, r' \in Rob$ *such that* $obs(r, c) \equiv obs(r', c')$.

Proof. Let c and c' be two configurations in $\mathcal{C}$.
We first show that if $c \approx c'$ then there exist $r, r' \in Rob$ such that $obs(r, c) \equiv obs(r', c')$:

- if $c' = \pi \circ c$ then $\forall r \in Rob$, $obs(r, c) = obs(r, c')$. By induction we easily show that for $m \in \mathbb{N}$, if $c' = \pi^m \circ c$ then $\forall r \in Rob$ $obs(r, c) = obs(r, c')$,
- if $c' = \overline{\pi} \circ c$ then $\forall r \in Rob$, $obs(r, c) = obs(r, c')$,
- if $c' = c \circ \beta$ for some robot permutation β, then $\forall r \in Rob$, $obs(r, c) = obs(\beta(r), c')$.

This implies the desired result.

Conversely, we then show that if $\exists r, r' \in Rob$ and $c, c' \in \mathcal{C}$ such that $obs(r, c) \equiv obs(r', c')$, then $c \approx c'$. If $obs(r, c) \equiv obs(r', c')$, *i.e.*, $obs(r, c) \circlearrowright^* obs(r', c')$, then there exists a permutation β and $m \in \mathbb{N}$ such that $c' = \pi^m \circ c \circ \beta$ where $r' = \beta(r)$. $\square$

Recall that robot decisions depend on what they have seen. Most of the time robots on the same tower have different observations (*i.e.* if they are not on an axis of symmetry), since they must have a similar behavior we introduce the notion of view.

2.4.3 Views

In a given configuration, a robot view depends on its observation. With the following definition of view, two robots on the same node have the same view.

Definition 10 (View)**.** *The view of robot r on a given configuration c is defined from $obs(r, c) = \{(f_1, \cdots, f_k), (f_k, \cdots, f_1)\}$ by $view(r, c) = \{(f_i, \ldots, f_j), (f_j, \ldots, f_i)\}$, where $i < j$ are respectively the smallest and greatest index such that $f_i \neq -1$ and $f_j \neq -1$. We denote by $\mathcal{V}$ the set of all views.*

Example 6.

Let c be the configuration of Figure 2.6:

$$c(r_1) = 1, c(r_2) = 1, c(r_3) = 4, c(r_4) = 8, c(r_5) = 8.$$

Observations of robot r_4 and robot r_5 are:

- $obs(r_4, c) = \{(-1, 2, -1, 2, 3), (3, 2, -1, 2, -1)\}$,
- $obs(r_5, c) = \{(2, -1, 2, 3, -1), (-1, 3, 2, -1, 2)\}$.

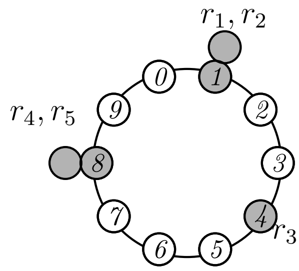


Figure 2.6: View

and their views are $view(r_4, c) = view(r_5, c) = \{(2, -1, 2, 3), (3, 2, -1, 2)\}$.

Note that if a robot does not belong to a tower then its view is equal to its observation. For instance in figure 2.6, we have $obs(r_3, c) = view(r_3, c)$. Moreover when a tower is on an axis of symmetry, the views of the robots on this tower are singletons. Hence, using views instead of observations, the view of any robot on an axis of symmetry is a singleton: such robots are called *disoriented* because they cannot distinguish one direction from the other.

Ordering the tuples of the form $(f_i, \ldots, f_j)$ by lexicographical order, we can obtain from a view of a robot r the maximal and the minimal view, denoted respectively by $view^{\max}\_r$ and $view^{\min}\_r$.

A view can also be described more succinctly by what we call an F-R-T sequence: an alternating sequence of symbols F, R and T indexed by natural numbers: F_x stands for x consecutive free nodes, R_x for x consecutive nodes, each one occupied by a single robot, and T_x for a tower of x robots.

We represent a class of configurations thanks to the minimal view, written as a F-R-T sequence. In the sequel, when describing a configuration (class), we simply give the corresponding F-R-T sequence.

Example 7. *View and F-R-T sequence.*
In Figure 2.6 $obs(r_3, c) = \{(3, -1, 2, -1, 2), (2, -1, 2, -1, 3)\}$ is the observation of the robot r_3, hence in this case

- $view^{\max\_r_3} = (3, -1, 2, -1, 2) = (R_1, F_3, T_2, F_2, T_2, F_2)$,
- $view^{\min\_r_3} = (2, -1, 2, -1, 3) = (R_1, F_2, T_2, F_2, T_2, F_3)$.

The configuration class can be represented by: $(T_2, F_2, T_2, F_2, R_1, F_3)$

2.4.4 Robot movements

The possible movements along edges also depend on the graph shape. On a ring there are only three possibilities: stay idle or move in the clockwise or anti-clockwise direction. The state "Ready to Move" (depicted in Figure 2.1) is then divided into three states $r.Front$, $r.Back$ and $r.Idle$. When a robot r is in state $r.Front$, it means that it will shift to its neighboring node in the direction given by $view^{\max\_r}$. Symmetrically, the robot in state $r.Back$ will go in the opposite direction. We define $\overline{r.Front} = r.Back$, $\overline{r.Back} = r.Front$, $\overline{r.Idle} = r.Idle$, and $\overline{RLC} = RLC$.

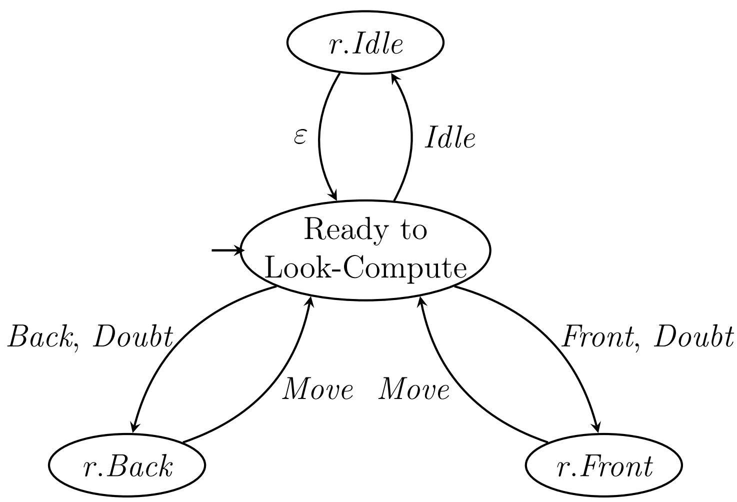


Figure 2.7: Automaton of robot r on a ring.

These movements, determined by the algorithm, are described by the LC actions in the set $\Delta = \{Front, Back, Doubt, Idle\}$. We define $\overline{Front} = Back$, $\overline{Back} = Front$, $\overline{Idle} = Idle$ and $\overline{Doubt} = Doubt$.

For a given robot r, the choice of an action depends on its own view of the configuration. If the robot chooses not to move, its action is *Idle*. If $view^{\max\_r} \neq view^{\min\_r}$, the robot can choose between the two directions, producing actions *Front* or *Back*. Otherwise the action is *Doubt*, corresponding to a non deterministic choice between *Front* and *Back*. This is depicted in Figure 2.7 which describes a robot automaton for the case of the ring.

This automaton will later be refined again, according to the algorithm executed by the robots. In particular, guards depending on the views will be associated with the actions, in order to implement a choice between them.

The equivalence between configurations can be extended to global states as follows. We consider system states as pairs (s, c), where the robot state s is a mapping from *Rob* to $\{Front, Back, Idle, RLC\}$.

Definition 11. *Two system states (s, c) and (s', c') are equivalent if*

- $c \approx c'$
- *if $c' = \pi^m \circ \overline{\pi} \circ c \circ \beta$ or $c' = \pi^m \circ c \circ \beta$ for some m and some robot permutation β, then for for any robot r, $s'(r) = s(\beta(r))$.*

Example 8. *This example illustrates the notion of equivalence of system states thanks to the system states (a, s_a) and (b, s_b) depicted in Figure 2.8. In this figure the robot states associated with the configurations a and b are respectively given by s_a, s_b defined by: $s_a(r_3) = Front$, $s_a(r_1) = Back = s_a(r_5)$, $s_a(r_2) = RLC = s_a(r_4) = s_a(r_6)$, and $s_b(r_6) = Front$, $s_b(r_2) = Front = s_b(r_3)$, $s_b(r_1) = RLC = s_b(r_4) = s_b(r_5)$. The two configurations a and b are equivalent since there is robot permutation β such that $a = \overline{\pi} \circ \pi^2 \circ b \circ \beta$ where $\beta$is defined by: $\beta(r_1) = r_6$, $\beta(r_2) = r_1$, $\beta(r_3) = r_5$, $\beta(r_4) = r_4$, $\beta(r_5) = r_2$, $\beta(r_6) = r_3$. The two system states are equivalent since $a \approx b$ and $s_a(r_1) = s_b(\beta(r_2)) = Back$, $s_a(r_2) = s_b(\beta(r_5)) = RLC$, $s_a(r_3) = s_b(\beta(r_6)) = Front$, $s_a(r_4) = s_b(\beta(r_4)) = RLC$, $s_a(r_5) = s_b(\beta(r_3)) = Back$, and $s_a(r_6) = s_b(\beta(r_1)) = RLC$.*

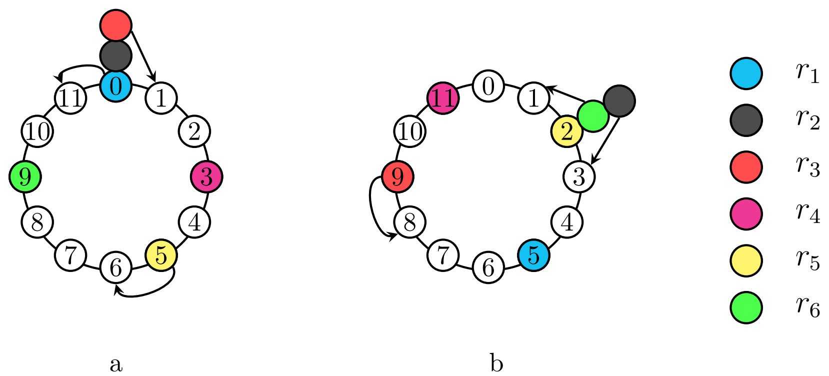


Figure 2.8: System configurations equivalence

The two system states (a, s_a) and (b, s_b) are equivalent, their configurations are in the class described by $T_3F_2R_1F_1R_1F_3R_1F_2$ and the states satisfy:

- *Among the robots of the tower one wants to move Front, one is in his RLC phase and the last one wants to move Back.*

- *The robot neighbor to 3 free nodes and 1 free node wants to move in the direction of the 3 free nodes.*
- *All other robots are in their RLC phase.*

2.4.5 Robot algorithms

The movements described above are defined by the algorithms. Recall that a robot algorithm must ensure that:

- Two symmetrical robots plan the same move,
- Robots on the same tower plan the same move,
- Disoriented robots choices are either to stay idle or to move in an arbitrary direction (with the *Doubt* action).

By definition symmetrical robots have the same observation thus the same view, robots on the same tower have different observations but the same views. Thus, an algorithm $\mathcal{S}$ can be given by a function that suggests a movement to a robot, according to its view. Such a function $\partial_{\mathcal{S}} : \mathcal{V} \to \Delta$ is called a *decision function* and defined as follows.

Definition 12 (decision function). *A* decision function *is a function $\partial : \mathcal{V} \to \Delta$ such that, for $view \in \mathcal{V}$, if $|view| = 1$, then $\partial(view) \in \{Idle, Doubt\}$ and if $\partial(view) = Doubt$ then $|view| = 1$.*

This definition states that a disoriented robot r in a configuration c, *i.e.*, $|view(r, c)| = 1$ (see Example 6), cannot decide between *Front* and *Back*. Its possible moves are either *Idle* or *Doubt*. Conversely if the robot is not disoriented it has to decide, hence its movement is not *Doubt*. An element of ∂ of the form $view \to action$ is called a *rule*.

To implement a given algorithm, we will need a pre-processing phase to express the corresponding rules in terms of guarded actions of the form $predicate \to action$, where the predicate is evaluated on the robot view. In the implementation, these rules are then translated into clockwise or anti-clockwise moves, according to the current configuration.

Once the system is modeled with robots implementing some protocol to be verified, the requirements are expressed in LTL, and model-checking is applied.

The next two chapters are devoted to case studies for the problems of ring exploration with stop and perpetual exclusive ring exploration, with [Flo+13] and [Bli+10] respectively, as representative protocols of these two classes.

Part I

Verification of mobile robot protocols: the exploration case

Chapter 3

Flocchini Algorithm

The problem of exploration with stop on a ring was first defined in [Flo+13]. It is proved there that the problem cannot be solved by a deterministic algorithm when the number k of robots divides the ring size n.

We consider the algorithm from Flocchini *et al.* that solves the problem for $k \geq 17$, with k and n coprime. The original paper only contains an informal description of the algorithm, Thus, our first contribution is to formally express the algorithm in order to remove ambiguities. We end by the verification results.

3.1 Specification of the exploration with stop

For any ring and any initial configuration where robots are located on different vertices, a protocol solves the problem of exploration with stop if within finite time and regardless of the initial positions of the robots, it guarantees the following two properties:

(i) *Exploration*: Each node of the ring is visited by at least one robot, and

(ii) *Termination*: Eventually, the robots reach a configuration where they all remain idle (their LC_i action leads to $r_i.Idle$).

Note that this last property requires robots to "remember" how much of the ring has been explored *i.e.*, these oblivious robots must be able to distinguish between various stages of the exploration process simply by their current view.

These two properties can be expressed in LTL (see Section 2.2) as follows:

- *Exploration*: $\bigwedge_{i=1}^{n} \Diamond \bigvee_{j=1}^{k} \big(c(r_j) = i\big)$.
- *Termination*: $\bigwedge_{j=1}^{k} \Diamond\Box\big(\neg r_j.Front \wedge \neg r_j.Back\big)$.

Definition 13. *A protocol solves the problem of ring exploration with stop if from any initial configuration, the following formula holds:*

$$Fairness \Rightarrow (Exploration \wedge Termination)$$

3.2 Formalisation of the algorithm

In this algorithm [Flo+13], a set of k identical robots explore an unoriented ring of n anonymous (*i.e.*, identical) nodes, with $n > 0$ and $k > 0$. Initially there is at most one robot in each node, Thus, $k \leq n$. Since the case where $n = k$ is trivial, we assume from now on that $k < n$. Moreover, n and k must be co-prime. The algorithm is divided into three phases, the *Set-Up* phase, the *Tower-Creation* phase and the *Exploration* phase. In the Set-Up phase, all robots are gathered in one group or two groups of the same size. In the second phase, the goal is to create one or two towers per block according to the parity of the blocks. The last phase is the exploration of the ring.

In order to express the algorithm as a decision function (and then in a guarded action language), some definitions and notations are introduced, related to a given configuration, and examples from the rings of Figure 3.1 are given along these definitions.

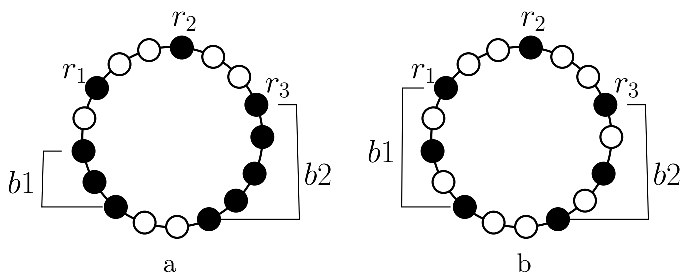


Figure 3.1: Illustration of the definitions.

- The *interdistance* d is the minimum distance between all pairs of distinct robots in the configuration, where distance is counted in number of edges. Hence, an interdistance $d = 0$ corresponds to the presence of (at least) two robots on a same node.

  For the configuration (a) of Figure 3.1, the interdistance is 1, while it is 2 for (b).

- A *block* is a maximal set of at least 2 robots that are located every d nodes (where d is the interdistance of the configuration). Maximality means that (at least) the $d+1$ adjacent nodes of a block in both directions are free. Note that a block contains at most $d - 1$ consecutive free nodes. We denote by *Blocks* the set of blocks (recall that *Rob* is the set of robots).

- The *size* of a block b, denoted by $b.size$, is the number of robots in this block.

  In configuration (a), there are two blocks of size 3 and 5, while there are two blocks of size 3 in (b).

- *Between*(b_1, b_2) is a pair of natural numbers counting the number of blocks between two blocks b_1 and b_2 (one integer for each direction). In both (a) and (b), *Between*$(b1, b2)$ is equal to $(0, 0)$.

We also define the following predicates:

Isolated(r)**:** this predicate is true if r is an isolated robot. A robot is *isolated* if it is not part of a block, that is, if in both directions its $d+1$ adjacent nodes are free.

In (a), Isolated$(r_1) = \mathit{true}$, Isolated$(r_2) = \mathit{true}$ and in (b), Isolated$(r_2) = \mathit{true}$

Border(r, b)**:** this predicate is true if robot r is a border of the block b. A robot r is a *border* of a block if it is one of the extremal robot that forms this block.

For example Border$(r_1, b1)$ is false in (a) and true in (b).

Neighbor(x, y)**:** this predicate is true if x and y are neighbors, x and y being either robots or blocks. Two robots, two blocks or a robot and a block are *neighbors* if there exists at least one direction such that only free nodes exist between them.

Neighbor(r_1, r_2) and Neighbor$(r_2, b2)$ are true in both (a) and (b), but the predicate Neighbor$(r_2, b1)$ is false in (a) and true in (b).

Leading(x)**:** this predicate is true if x is a leading block or a leading robot. A robot is leading if its view is minimal (among the different $\mathit{view}^{\min}{}_{-}{}^{r}$). Such a robot is called a *leader*. A block b is *leading* if it has a border robot which is a leader.

In the example, only Leading(r_3) is true, Hence, also Leading$(b2)$, while in (b), both Leading(r_1) and Leading(r_3) are true.

Finally, the distance $\mathit{dist}(x, y)$ between two neighbors x and y in $\mathit{Blocks} \cup \mathit{Rob}$ is the minimum length of a path between them containing only free nodes.

We now formally describe each phase of the algorithm as performed by each robot.

3.2.1 The Set-Up Phase

The aim of this phase is to gather robots on particular configurations called the *Set-Up final configurations*, defined by: $d = 1$, there is no isolated robots, and each block is a leading block. It can be proved that in such a case, either all robots are in the same block or the robots are divided into two blocks of the same size, since otherwise k and n are not co-prime anymore.

Starting from a tower-free configuration, absence of tower will be maintained throughout the Set-Up phase. This is expressed by the predicate: Set-Up := $d > 0$.

There are four types of configurations, namely A, B, C, D, that form a partition of all possible tower-free configurations. Configurations of type A contain isolated robots and configurations of type B, C or D contain only blocks of robots. Configurations of type D are the Set-Up final configurations, configurations of type C are similar(there is no isolated robots, and each block is a leading block) to these configurations but with an interdistance $d \geq 2$. All the remaining configurations without isolated robot are configurations of type B.

For each of these configuration types, we introduce some notations, sets and predicates to define the protocols executed by the robots.

Configurations of Type A. In these configurations with at least one isolated robot, the protocol is as follows: an isolated robot that is the nearest to the biggest blocks adjacent to any isolated robot, moves toward these biggest blocks (with *Doubt* in case of equality).

These movements are made in order to remove isolated robots by increasing the size of their biggest and nearest neighbor blocks. Hence, after a finite number of transitions, the configuration must be of type B, C or D, with the same interdistance (recall that the interdistance is denoted by d) as the starting configuration of type A.

We define:

$S = \max\{b.size \mid b \in Blocks \text{ s.t. } \exists r \in Rob : \text{Neighbor}(r, b) \wedge \text{Isolated}(r)\}$

$$\text{Move}(r, b) = \begin{cases} r.Doubt & \text{if } view^{\max}\_^{r} = view^{\min}\_^{r} \\ r.Back & \text{if } view^{\max}\_^{r} \neq view^{\min}\_^{r} \text{ and} \\ & \quad dist(r, b) > n - dist(r, b) - (b.size - 1) \times d \\ r.Front & \text{otherwise} \end{cases}$$

We also need the following predicates and sets:

Type$A$:= Set-Up $\wedge \exists r \in Rob : \text{Isolated}(r)$

$\text{NearS}(r, b) := \text{Isolated}(r) \wedge \text{Neighbor}(r, b) \wedge b.size = S$

$\text{Closest} = \{(r, b) \in Rob \times Blocks \mid \text{NearS}(r, b) \wedge \forall (r', b') \in Rob \times Blocks : \text{NearS}(r', b') \Rightarrow dist(r, b) \leq dist(r', b')\}$

The guarded action in type A for a robot r is thus:

$$[\text{Type}A \wedge \exists b \in \mathit{Blocks} : (r, b) \in \text{Closest}] \rightarrow \text{Move}(r, b).$$

Example 9. *This action is illustrated in Figure 3.2, where the isolated robot r is at distance 9 of b_1 (which is of maximal size) and 3 of b_2. Since the two views of r are different and $dist(r, b_1) > n - dist(r, b_1) - (b_1.size - 1) \times d$ holds with $n = 21$ and $d = 2$, its move must be r.Back, in the opposite direction to its minimal view, as indicated by the arrow on Figure 3.2.*

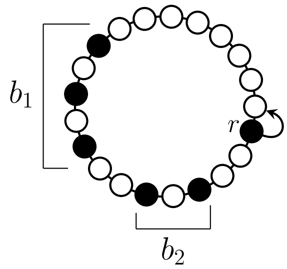


Figure 3.2: Movement in a type A configuration.

Configurations of types C or D. For type C or type D configurations, there is no isolated robot and each block is a leading block. They are defined by the following predicates:

TypeCD := Set-Up $\wedge$ $\neg$Type$A \wedge$ $\forall b \in \mathit{Blocks}$: Leading(b)
Type$C$:= $d \geq 2 \wedge$ TypeCD
Type$D$:= $d = 1 \wedge$ TypeCD

It can be seen that all C and D configurations are symmetric. In a configuration of type C, all blocks are leading, with a minimal view for all leader robots, and no isolated robot. Hence, there are two leaders in each block. The aim of the protocol here is to reduce the interdistance. Hence, the two leaders of a block will move inside their block, so that from a C configuration with interdistance d, a configuration of type A with interdistance $d - 1$ will be reached. The protocol executed by robot r in this case is:

$$[\text{Type}C \wedge \text{Leading}(r)] \rightarrow r.\mathit{Front},$$

meaning that the robot moves in the direction of its minimal view.

From a configuration of type D (Set-Up final) the *Tower-Creation* phase begins. Then, denoting by Tower-Creation the predicate satisfied in the *Tower-Creation* phase, we set:

$$\text{Type}D \rightarrow \text{Tower-Creation}.$$

Configurations of type B. When the current configuration is neither a type A configuration nor a type C or D configuration, then it is a type B configuration, which is by far the most complicated part of the algorithm:

$$\mathrm{Type}B := \textit{Set-Up} \wedge \neg(\mathrm{Type}A \vee \mathrm{Type}CD).$$

Configurations of type B are divided into the two types $B1$ and $B2$: if all blocks have the same size then the configuration is of type $B1$, otherwise it is of type $B2$.

In a configuration of type B, the aim of the protocol is to reduce the number of blocks. This is done according to the following cases which partition the B type:

- If the configuration is asymmetric, of type $B1$, then after a finite number of transitions, the configuration is of type $B2$, with the same interdistance, and there is one block less.
- If the configuration is symmetric, of type $B1$, with blocks of size 2 then after a finite number of transitions, the configuration is of type C or D, with the same interdistance.
- If the configuration is symmetric, of type $B1$, with blocks of size ≥ 3, then after a finite number of transitions, the configuration is of type $B2$, C or D, with the same interdistance, and there are fewer blocks.
- If the configuration is of type $B2$, then after a finite number of transitions, the configuration is of type B, C or D, with the same interdistance, and strictly fewer blocks.

Before presenting the formal rules of the algorithm for the configurations of type $B1$, we define:

$$\mathrm{Type}B1 := \mathrm{Type}B \wedge \forall b, b' \in \textit{Blocks} : b.\textit{size} = b'.\textit{size}$$

and the following predicates:

$\mathrm{symRobots}(r, r') := \textit{view}^{\max\_r} = \textit{view}^{\max\_r'} \wedge r \neq r'$

$\mathrm{symBlocks}(b, b') :=\ b \neq b' \wedge \exists (r_1, r_2) \in \textit{Rob}^2 :$
$\quad \mathrm{Border}(r_1, b) \wedge \mathrm{Border}(r_2, b') \wedge\ \mathrm{symRobots}(r_1, r_2)$

and the sets:

$\mathrm{L} = \{r \in \textit{Rob} \mid \mathrm{Leading}(r)\}$

$\mathrm{SymR} =\ \{(r_1, r_2) \in \textit{Rob}^2 \mid \mathrm{symRobots}(r_1, r_2) \wedge \exists b \in \textit{Blocks} : \mathrm{Border}(r_1, b)\}$

$\mathrm{SymB} =\ \{(b_1, b_2) \in \textit{Blocks}^2 \mid \mathrm{symBlocks}(b_1, b_2) \wedge \exists (x_1, x_2) \in \mathbb{N}^2 :$
$\quad \textit{Between}(b_1, b_2) = (x_1, x_2) \wedge\ x_1 \geq 3\ \wedge\ x_2 \geq 3\}.$

For subsets *robs* of $\textit{Rob}^2$, and *Bs* of *Blocks*:

$\text{NearPair}(robs) = \{(r_1, r_2) \in robs \mid \neg\text{Neighbor}(r_1, r_2) \wedge$
$dist(r_1, r_2) = \min\{dist(r, r'), (r, r') \in robs\}\}$
$\text{minView}(Bs) = \{r \in Rob \mid \exists b \in Bs, \exists r' \in Rob \text{Border}(r, b) \wedge \text{Border}(r', b) \wedge$
$view^{\min\_r} < view^{\min\_r'}\}$
The guarded actions in type $B1$ for a robot r are:

- $[\text{Type}B1 \wedge L = \{r\}] \rightarrow \; r.Back$
- $[\text{Type}B1 \wedge |L| = 2 \wedge \exists b \in Blocks:$
  $b.size = 2 \wedge \text{Border}(r, b) \wedge r \in \text{minView(SymB)}] \rightarrow \; r.Back$
- $[\text{Type}B1 \wedge |L| = 2 \wedge \exists b \in Blocks:$
  $b.size \neq 2 \wedge \text{Border}(r, b) \wedge r \in \text{NearPair(SymR)}] \rightarrow \; r.Back$

To explain how the algorithm works for the type $B2$, we define:

$$\text{Type}B2 := \text{Type}B \;\wedge \neg\text{Type}B1$$

and the variables:
$\text{m} = \min\{b.size \mid b \in Blocks\}$
$\text{M} = \max\{b_1.size \mid b_1 \in Blocks \text{ s.t. } \exists b_2 \in Blocks\text{: } \text{Neighbor}(b_1, b_2) \wedge b_2.size = \text{m}\}$
$\text{dmin} = \min\{dist(b_1, b_2) \mid (b_1, b_2) \in Blocks^2 \text{ s.t. } \text{Neighbor}(b_1, b_2) \wedge b_2.size = \text{m} \wedge b_1.size = \text{M}\}$
For a subset rob of Rob:
$\text{MaxV}(rob) = \max\{view^{\max\_r} \mid r \in rob\}$
$T = \{r \in Rob \mid \exists (b_1, b_2) \in Blocks^2:$
$\text{Border}(r, b_1) \wedge b_1.size = \text{m} \wedge \text{Neighbor}(r, b_2) \wedge b_2.size = \text{M} \wedge dist(b_1, b_2) = \text{dmin}\}$
The guarded action for a robot r is:

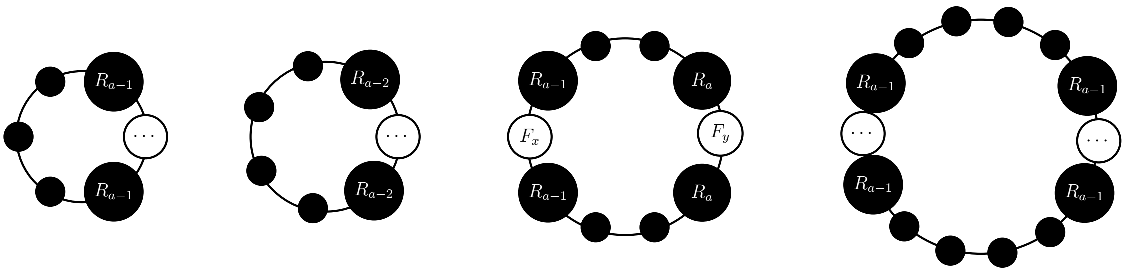
$$[\text{Type}B2 \wedge r \in T \wedge view^{\max\_r} = \text{MaxV}(T) \wedge \exists b \in Blocks:$$
$$(\text{Neighbor}(r, b) \wedge b.size = \text{M} \wedge dist(r, b) = \text{dmin})] \rightarrow \; r.Back$$

3.2.2 The Tower-Creation Phase

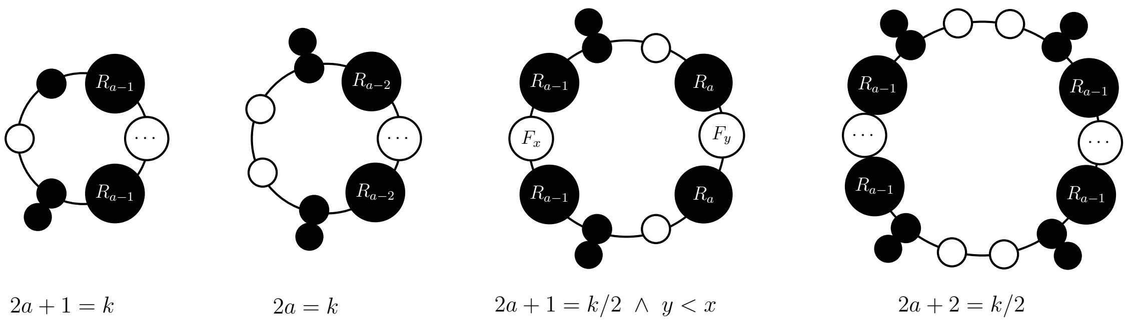
The aim of this phase is to form towers from the Set-Up final configurations. The configurations Thus, obtained are called tower-completed and are composed of one block or two symmetric blocks.

Informally, for each odd block one tower is formed by the central robot moving to its neighboring node containing the robot with the larger view. For each even block two towers are formed by the two central robots moving to their other neighbors.

The corresponding rules are described in Table 3.1, where scheduled robots stay idle in all cases not covered. The view is given by an *F*-*R*-*T* sequence as



a Set-Up final configurations.



b Tower-Completed configurations.

Figure 3.3: Tower-Creation phase from Set-Up final configurations.

described in Definition 10. Figure 3.3 illustrates the process of tower creation from the possible Set-Up final configurations. Each configuration in Figure 3.3a will produce the one just below in Figure 3.3b. Big circles contain a set of adjacent nodes which are all free or all occupied. A big black node R_x represents x adjacent occupied nodes, a big white node F_x represents x adjacent free nodes, and a white node containing dots represents a positive number of free nodes.

Tower-Creation Phase:					
Rule::	Condition	$\wedge$	$view(r,c)$	$\rightarrow$	Move
$TC1_0$::	$2a+1=k$	$\wedge$	(R_{a+1}, F_x, R_a)	$\rightarrow$	$r.Doubt$
$TC2_0$::	$2a=k$	$\wedge$	(R_{a+1}, F_x, R_{a-1})	$\rightarrow$	$r.Back$
$TC2_1$::	$2a=k$	$\wedge$	$(R_a, F_x, R_{a-2}, T_2, F_1)$	$\rightarrow$	$r.Front$
$TC3_0$::	$2a+1=k,\ y<x$	$\wedge$	$(R_{a+1}, F_y, R_{k/2}, F_x, R_a)$	$\rightarrow$	$r.Back$
$TC3_1$::	$2a+1=k,\ y<x$	$\wedge$	$(R_{a+1}, F_y, R_a, F_1, T_2, R_{a-1}, F_x, R_a)$	$\rightarrow$	$r.Back$
$TC4_0$::	$k/2=2a+2$	$\wedge$	$(R_{a+2}, F_x, R_{k/2}, F_y, R_a)$	$\rightarrow$	$r.Back$
$TC4_{11}$::	$k/2=2a+2$	$\wedge$	$(R_{a+1}, F_x, R_{k/2}, F_y, R_{a-1}, T, F_1)$	$\rightarrow$	$r.Front$
$TC4_{12}$::	$k/2=2a+2$	$\wedge$	$(R_{a+2}, F_x, R_{a-1}, F_1, T_2, R_{a-1}, F_y, R_a)$	$\rightarrow$	$r.Back$
$TC4_{13}$::	$k/2=2a+2$	$\wedge$	$(R_{a+2}, F_x, R_{a-1}, T_2, F_1, R_{a+1}, F_y, R_a)$	$\rightarrow$	$r.Back$
$TC4_{21}$::	$k/2=2a+2$	$\wedge$	$(R_{a+2}, F_x, R_{a-1}, T_2, F_2, T_2, R_{a-1}, F_y, R_a)$	$\rightarrow$	$r.Back$
$TC4_{22}$::	$k/2=2a+2$	$\wedge$	$(R_{a+1}, F_x, R_{a+1}, F_1, T, R_{a-1}, F_y, R_{a-1}, T, F_1)$	$\rightarrow$	$r.Front$
$TC4_{23}$::	$k/2=2a+2$	$\wedge$	$(R_{a+1}, F_x, R_{a-1}, T, F_1, R_{a+1}, F_y, R_{a-1}, T, F_1)$	$\rightarrow$	$r.Front$
$TC4_3$::	$k/2=2a+2$	$\wedge$	$(R_{a+1}, F_x, R_{a-1}, T, F_2, T, R_{a-1}, F_y, R_{a-1}T, F_1)$	$\rightarrow$	$r.Front$

Exploration Phase:					
Rule::	Condition	$\wedge$	$view(r,c)$	$\rightarrow$	Move
E_1::	$2a+1=k,\ x\geq 1$	$\wedge$	$(R_1, F_x, R_a, F_1, T_2, R_{a-2}, F_y)$	$\rightarrow$	$r.Front$
E_2::	$2a=k,\ x>0,\ z<(n-k+2)/2$	$\wedge$	$(R_1, F_x, R_1, F_y, R_{a-3}, T_2, F_2, T_2, R_{a-3}, F_z)$	$\rightarrow$	$r.Front$
E_{31}::	$2a+1=k/2,\ g<(g+b+c)/2$	$\wedge$	$(R_1, F_b, R_1, F_c, R_{a-2}, T_2, F_1, R_{a-1}, F_d,$ $R_1, F_e, R_1, F_f, R_{a-1}, F_1, T_2, R_{a-2}, F_g)$	$\rightarrow$	$r.Front$
E_{32}::	$2a+1=k/2,\ d<(d+e+f)/2$	$\wedge$	$(R_1, F_e, R_1, F_f, R_{a-1}, F_1, T_2, R_{a-2}, F_g, R_1,$ $F_b, R_1, F_c, R_{a-2}, T_2, F_1, R_{a-1}, F_d)$	$\rightarrow$	$r.Front$
E_4::	$2a+2=k/2,\ g<(g+b+c)/2$	$\wedge$	$(R_1, F_b, R_1, F_c, R_{a-2}, T_2, F_2, T_2, R_{a-2}, F_d,$ $R_1, F_e, R_1, F_f, R_{a-2}, T_2, F_2, T_2, R_{a-2}, F_g)$	$\rightarrow$	$r.Front$

Table 3.1: Rules of the Tower-Creation and Exploration phases for a robot r.

There are four cases:

- If there is only one block of odd size then the Set-Up final configuration looks like the first one of Figure 3.3a. A unique robot can move according to rule $TC1_0$, which produces the configuration just below in Figure 3.3b (or the symmetric one with the tower in the upper half instead of the lower half).
- If there is only one block of even size (second column), then two robots will move according to rule $TC2_0$. If one has moved before the other one could take a snapshot of the configuration then its movement is given by rule $TC2_1$.
- If there are two symmetric blocks of odd size (third case), then two robots will move according to rule $TC3_0$. If one has moved before the other one could take a snapshot of the configuration then its movement is obtained by rule $TC3_1$. In this case, the two free segments F_x and F_y have different sizes, and the robots move in the direction opposite to the shortest one.
- If there are two blocks of even size (rightmost column) then robots move according to rule $TC4_0$. In this case also, the two free segments have different sizes. If one tower is formed, the other robot movement is given by rules $TC4_{11}$, $TC4_{12}$ and $TC4_{13}$ according to the robot view. If two of them have moved, and two towers are formed, then the moving robots compute their movements according to one of the rules in $\{TC4_{21}, TC4_{22}, TC4_{23}\}$, depending of which robots have moved before. And when all robots but one have moved, the last one moves according to rule $TC4_3$.

3.2.3 The Exploration Phase

The exploration phase is the last phase of the algorithm. It starts from tower-completed configurations and is described in the second part of Table 3.1 (where again robots stay idle for non covered cases). No new towers are created during this phase.

Note that the empty nodes adjacent to towers have already been explored, so the segments of empty nodes between the blocks are the only ones possibly not yet explored. Each of these segments is explored in the current phase by one or two robots closest to the segment.

When k is odd, the configuration starting the exploration phase is made of two blocks, one of them containing a tower (leftmost configuration of Figure 3.3b). The explorer is the robot at the border of the block with the tower, the tower being the other border of the block. Its destination is the neighbor free node toward the block that does not contain the tower. The algorithm for the moving robot is given by rule E_1.

When k is even, there are as many explorers as blocks from the tower-completed configuration. An explorer is a robot at a border of a block, and which is adjacent to an empty segment not visited. Their destinations are their adjacent node towards the center of the empty segment. The explorers keep being isolated robots until they either are neighbors in the middle of the segment (when the empty segment is even) or they form another tower (when the empty segment has odd size). The corresponding rules of the algorithm are: E_2, E_{31}, E_{32} and E_4.

3.3 Results: Bounds refined

Using model-checking tools on this formal representation of the algorithm, we show that it satisfies the exploration and termination properties under fairness hypothesis (see Section 3.1). Hence, the algorithm is correct for all tested instances of k and n that satisfy the constraints given in the original paper: n, k are co-prime and $n, k \geq 17$.

k	n	Time	Mem (kB)
17	18	00: 00: 04	60 984
18	19	00: 00: 04	66 256
17	19	00: 25: 29	1 622 180
19	20	00: 00: 08	88 168
18	20	00: 12: 10	2 130 824
17	20	08: 08: 00	22 045 016
20	21	00: 00: 08	100 136
19	21	01: 08: 12	3 632 488
18	21	03: 00: 52	9 427 620
17	21	18: 40: 07	55 287 000
21	22	00: 00: 12	123 920
20	22	01: 58: 27	5 913 880
19	22	08: 25: 22	30 243 392
18	22	20: 32: 45	100 327 682

Table 3.2: Set-Up phase model-checking.

Since the most complex phase of the algorithm is the Set-Up phase, we present in Table 3.2 the verification results (time and memory) for the restriction to this particular phase, model-checking the property: every run reaches a configuration satisfying the Type$D$ predicate (corresponding to a Set-Up final configuration). The state space explosion occurring during the model-checking can be seen on these results.

k	**n**	**States**	**Transitions**	**Mem (kB)**
5	6	147	436	163 600
5	7	500	1 410	171 084
5	8	2 786	10 596	183 840
5	9	5 533	18 746	207 788
5	10	5 123 204	25 755 007	668 396
5	11	7 827	23 898	299 980
5	12	13 996	61 822	380 244
5	13	17 149	82 902	491 708
5	14	30 680	157 829	637 840
5	15	19 784 312	130 057 237	2 667 850
5	16	12 418	73 688	1 081 736
5	17	33 004	207 642	1 401 280
5	18	10165	66 120	1 790 644
7	8	680	1860	171 396
7	9	2 764	7 576	201 096
7	10	3 022	9 220	270 676
7	11	16 471	56 390	437 876
7	12	18 347	42 448	754 680
7	13	20 272	83 706	1 352 120
10	11	839	1942	190 884
10	12	3 834	8 868	460 750
10	13	7 924	23 731	756 000
10	14	8 357	27 524	2 135 987

Table 3.3: Model-checking small instances.

We also tested the algorithm for some small instances not covered by the original setting. Interestingly, our methodology permits to refine the correctness bounds for these cases. The performances can be seen in Table 3.3, where the algorithm satisfies the correctness property for all values of n and k appearing in the table.

From these experiments, we conjecture that the algorithm is correct for $n \leq 18$ in the following cases even when n and k are not co-prime, as long as the initial configuration is not periodic (where not periodic means that there is at most one symmetry axis in the ring and that it is not invariant by non-trivial rotation):

- When k is even the algorithm is correct as long as $n < k + \lceil k/2 \rceil$ and $10 \leq k < 17$.

- When k is odd the algorithm is correct if $5 \leq k < 17$.

Unfortunately, the combinatorial explosion made the verification exceed reasonable time for some cases. For instance, the computation was stopped for $k = 7$ and $n = 14$ after 1 day.

We outline here the number of states and the number of transitions in order to show that the memory and the time used increase as the number of transitions and states of the system. Moreover, when k and n are not co-prime these numbers explode, due to the complexity of the algorithm to ensure the exploration when there are symmetries.

We now recall the problem of perpetual exclusive ring exploration, and present the verification results for the *Min-Algorithm* [Bli+10].

CHAPTER 4

Min-Algorithm

Note that the same arguments as in [Flo+13] apply to obtain impossibility when the number k of robots divides the size n of the ring. For this algorithm, model checking tools are used to exhibit a counter-example. After identifying the rule producing this counter-example, we correct the algorithm and establish the correctness of the new version by model checking small instances and providing an inductive proof.

4.1 Specification of the perpetual exploration without collision

For any ring and any initial configuration where each node is occupied by at most one robot, an algorithm solves the perpetual exclusive exploration problem if it guarantees the following two properties:

(i) *Exclusivity*: There is at most one robot on any vertex and two robots never traverse the same edge at the same time in opposite directions.

(ii) *Liveness*: Each robot visits each node infinitely often.

These properties can be expressed in LTL (see Section 2.2) as follows: the *Exclusivity* property is the conjunction of the *No_ collision* and the *No_ switch* properties below:

- *No_ collision*: $\Box\Big(\bigwedge_{1\leq j<h}^{k} c(r_j) \neq c(r_h)\Big)$
- *No_ switch*: $\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{k}\bigwedge_{h=1}^{k} \neg \Diamond\Big(c(r_j) = i \wedge c(r_h) = i+1 \wedge r_j.\mathit{Front} \wedge r_h.\mathit{Back}\Big)$

The *No_ collision* property states that there is always at most one robot on each node, while the *No_ switch* property states that two neighbor robots cannot exchange their position by moving in opposite directions along an edge: one of them moves *Front* while the other moves *Back*. Note that the *No_ collision* property implies the *No_ switch* property in the asynchronous model, since one of the possible executions that form a tower is obtained when two neighbors want to switch their positions, and their moves are executed asynchronously.

In order to express that each robot visits all vertices infinitely often, we use the *Live* property:

$$Live: \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k} \Box\Diamond\big(c(r_j) = i\big).$$

The *Liveness* property needs the fairness assumption. Hence, it can be expressed by:

$$Liveness: Fairness \Rightarrow Live.$$

4.2 The algorithm

The *Min-Algorithm* from [Bli+10] is designed to ensure that 3 robots always exclusively and perpetually explore any ring of size $n \geq 10$ where n is not a multiple of 3. It is based on a classification of the set of tower-free configurations. The *Min-Algorithm* operates in two phases: the *Convergence* phase and the *Legitimate* phase. In the *Convergence* phase system states converge towards so-called *Legitimate states*. In the *Legitimate* phase the system cycles between its legitimate states, performing the exploration.

In Definitions 14 and 15 below, each set of configurations is an equivalence class of configurations, given by an *F-R-T* sequence, according to Definition 10.

Definition 14. Legitimate configurations *are defined for $n \geq 10$ by $\mathbb{L}^n = L1^n \cup L2^n \cup L3^n$ where:*

- $L1^n = (R_2, F_2, R_1, F_{n-5})$
- $L2^n = (R_1, F_1, R_1, F_{n-6}, R_1, F_2)$
- $L3^n = (R_1, F_3, R_2, F_{n-6})$

All other (tower free) configurations are called *non legitimate configurations.* We denote by $\mathbb{NL}^n$ the set of non legitimate configurations and we also partition $\mathbb{NL}^n$ according to the number of consecutive robots. This leads to the five sets A^n, B^n, C^n, D^n, E^n defined below.

Definition 15. Non legitimate configurations *are defined for $n \geq 10$, when n and 3 are co-prime, by $\mathbb{NL}^n = A^n \cup B^n \cup C^n \cup D^n \cup E^n$ as follows.*

When one robot is at the same distance of the two others, either if no robots are neighbors or if two robots are neighbors, then it is a B^n configuration:

$$B^n = \{(R_1, F_x, R_1, F_y, R_1, F_x) \mid x > 0 \wedge x \neq y \wedge n = 2x + y + 3\}.$$

Otherwise:

- *If no robots are neighbors, it is a C^n configuration:*

  $$C^n = \{(R_1, F_x, R_1, F_y, R_1, F_z) \mid 0 < x < z < y \wedge (x, z) \neq (1, 2) \wedge n = x+y+z+3\}$$

  Note that the case $(x, z) = (1, 2)$ corresponds to a $L2^n$ configuration. Hence, C^n configurations only appear when $n \geq 11$.

- *If only two robots are neighbors, if the minimal distance between these two robots and the last one is equal to 2 or 3, then it is a $L1^n$ or a $L3^n$ configuration. Otherwise:*

  - *If the minimal distance is equal to 1, then it is an E^n configuration: $E^n = (R_1, F_1, R_2, F_{n-4})$.*
  - *Otherwise the distance is larger than 3 and then it is an A^n configuration: $A^n = \{(R_1, F_x, R_2, F_z) \mid 4 \leq x < z \wedge n = x + z + 3\}$. Note that this type of configuration only exists when $n > 12$, since n and $k = 3$ must be co-prime.*

- *If the three robots are neighbors then the configuration is a D^n configuration: $D^n = (R_3, F_{n-3})$.*

From the disjunction of cases in Definitions 14 and 15 above, and observing that no two sets of configurations overlap, we have:

Proposition 3. *The sets of configurations: A^n, B^n, C^n, D^n, E^n, $L1^n$, $L2^n$, $L3^n$, form a partition of the set of all tower-free configurations of a ring of size $n \geq 10$, when n and $k = 3$ are co-prime.*

We now detail the two phases, described in Table 4.1. In the *Legitimate* phase the idea is to authorize, by exploiting the asymmetry of the network, a single robot to move at each step. Rule $RL1$ authorizes only the robot which is the farthest from the isolated robot to move. This robot goes to the only free neighboring node. Rule $RL2$ authorizes the robot which is the nearest to the other robots to move in order to minimize the distance between him and his nearest neighbor. Rule $RL3$ authorizes the isolated robot to come closer to the other robots. After the execution of n rounds (each one of the three robots has moved n times), all robots have explored the ring once.

Legitimate Phase:					
Rule::	Condition	$\wedge$	$view(r,c)$	$\to$	Move
$RL1$::			(R_2, F_2, R_1, F_{n-5})	$\to$	$r.Back$
$RL2$::			$(R_1, F_1, R_1, F_{n-6}, R_1, F_2)$	$\to$	$r.Front$
$RL3$::			(R_1, F_3, R_2, F_{n-6})	$\to$	$r.Front$

Convergence Phase:					
Rule::	Condition	$\wedge$	$view(r,c)$	$\to$	Move
$RC1$::	$4 \leq x < z$	$\wedge$	(R_1, F_x, R_2, F_z)	$\to$	$r.Front$
$RC2$::	$x \neq y$, $x > 0$	$\wedge$	$(R_1, F_x, R_1, F_y, R_1, F_x)$	$\to$	$r.Doubt$
$RC3$::	$0 < x < z < y \wedge (x,z) \neq (1,2)$	$\wedge$	$(R_1, F_x, R_1, F_y, R_1, F_z)$	$\to$	$r.Front$
$RC4$::			(R_3, F_{n-3})	$\to$	$r.Back$
$RC5$::			(R_1, F_1, R_2, F_{n-4})	$\to$	$r.Back$

Table 4.1: Rules of *Min-Algorithm* [Bli+10] for a robot r with $n \geq 10$.

The *Convergence* phase brings non-legitimate configurations into legitimate ones. The main point of this algorithm is to break possible symmetries and to converge to a pattern that allows the execution of one of the RL rules. Rule $RC1$ (resp. $RC2$, $RC3$, $RC4$ and $RC5$) is only applied for configurations in A^n (resp. B^n, C^n, D^n, E^n). Rule $RC1$ is applied in order to reduce the distance between the isolated robot and the two other robots. Rule $RC2$ is applied when a robot is at equal distance from the two other robots. This robot will break the symmetry by a shift of one position in any direction. Rule $RC3$ is applied when robots are scattered on the ring at distances $x < z < y$. The robot authorized to move is the one that is adjacent to the free spaces of size x and z. This robot will move such that the free space x is reduced by 1. The idea behind this movement is to create a block of robots. Rule $RC4$ captures the situation when the three robots are neighbors. In this case, due to the symmetry the two robots on the border can move. Rule $RC5$ is applied when the isolated robot is too close (at distance 1) to the block of robots. In this case it will move away from the block.

The specification for this algorithm is refined as follows:

(a) The *No_ collision* and *No_ switch* properties are satisfied.

(b) From any non-legitimate configuration a legitimate configuration is reached.

(c) The exploration is performed by cycling within the legitimate configurations (ensuring the *Liveness* property).

4.3 Results: A counter example

Recall that the setting of the *Min-Algorithm* features 3 robots in a ring of size $n \geq 10$ where n is not a multiple of 3. Thus, we first construct a model for this protocol and its properties in the model checker DiVinE [Bar+13]. Then we verify this algorithm for the smallest possible ring of size 10, for all models (Fsync, Ssync and Async). These results are presented in Table 4.2, with number of states, transitions, memory used, and time spent.

States	Transitions	Mem(kB)	Model	?
256 315	737 810	248 668	Fsync	ok
407 175	881 437	248 840	Ssync	ok
3 429 715	13 218 742	1 269 432	Async	col

Table 4.2: Model-checking of *Min-Algorithm* in the 3 models for the smallest ring

More importantly, our results show that the algorithm does not satisfy the *Exclusivity* property in the Async model. A counter-example is automatically generated, exhibiting a sequence of transitions leading to a collision (a tower), Hence, a violation of the *Exclusivity* property. It is presented in details in Figure 4.1, with a sequence of configurations obtained by successive robot moves. In each configuration a computation is represented by an arrow, which is dotted when the computation is made from an outdated snapshot.

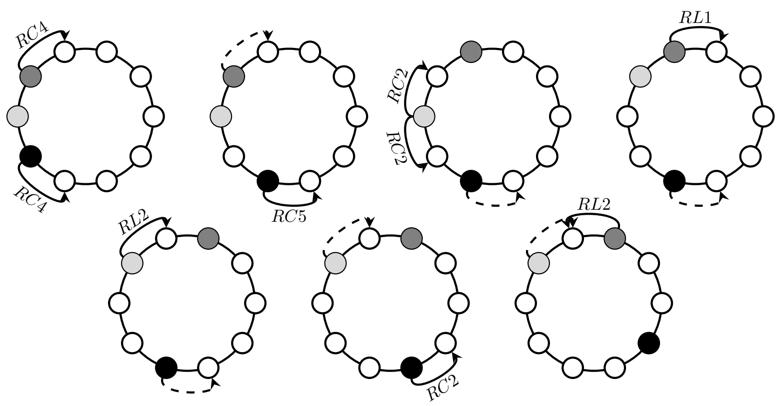


Figure 4.1: Counter-example.

In the starting configuration after the LC phase of all robots, the gray one and the black one have decided to move according to the $RC4$ rule, and the light gray one to stay idle. The black robot moves, which produces the second configuration. Before the gray robot could move, the black one performs its LC

phase and according to the $RC5$ rules, it chooses to roll away from the two other robots. The gray robot moves from the decision taken previously and the third configuration is reached. From this configuration the light gray robot performs its LC phase and chooses to move in any direction (as the configuration has a symmetry axe passing through this robot). The scheduler makes it move toward the gray one. From the fourth configuration Thus, obtained, the gray robot had to move according to rule $RL1$ after its LC phase. This movement permits to obtain the fifth configuration, from where the light gray robot chooses to move according to rule $RL2$. We obtain the sixth configuration thanks to the movement of the black robot (movement that he had chosen in the second configuration). From this configuration the black robot chooses to move according to rule $RC4$, and the move is performed. In the last configuration the gray robot performs its LC phase and according to the $RL2$ rule, it chooses to move toward the light gray one, on the same node where the light gray one had chosen to go in the fifth configuration. From there if these two robots move, they collide.

In this counter-example, we can see that the collision is due to the fact that there is always one movement that can be made from an outdated snapshot, Hence, we need to stop these movements. We now present a correction of the algorithm referred to as *Min-Algorithm-Corrected*. The change concerns the convergence phase, the legitimate phase being unchanged. More precisely, only rule $RC5$ is modified to avoid collisions induced by the previous rules, when movements computed on obsolete observations are taken into account. The new $RC5$ rule is:

$$RC5 \quad :: \quad (R_2, F_1, R_1, F_{n-4}) \quad \rightarrow \quad r.Back$$

Note that the moving robot has changed with respect to the old rule. If this new rule is applied in the counter-example, then from the second configuration, no movements from outdated snapshots can be made any more since the $RC5$ rule requires a configuration where the light gray and the black robots have stayed idle.

n	**States**	**Transitions**	**Mem(kB)**	**Time**
10	1 581 961	6 090 209	1 416 880	00: 06: 45
11	1 926 385	7 421 315	1 568 748	00: 09: 09
13	2 716 637	10 476 317	2 252 600	00: 20: 46
14	3 162 409	12 307 905	2 560 724	00: 26: 54
16	4 155 385	16 041 365	2 772 188	00: 36: 22

Table 4.3: Model-checking of the *Min-Algorithm-Corrected*

Verification results (where correctness is obtained) are given in Table 4.3 for several instances of n. All results show a limited blow up, due to the fact that when the number $k = 3$ of robots is fixed, the total number of configurations (and of states) is of order n^3.

Since this protocol is parameterized by the ring size n, model-checking does not permit to verify whether it is valid for all values of n. Therefore, while automated verification was used to prove the required properties for small values of n, we provide an inductive proof to obtain the correctness for arbitrary values of n.

4.4 Correctness of the new algorithm

We first prove point *(c)*: the exploration is performed by cycling within the legitimate configurations and point *(b)*: from all non-legitimate configurations a legitimate configuration is reached.

Definition 16. *We note* $[Type^n(x, y, z),\ \varphi(x, y, z)]$ *the set of configurations such that Type is the type of the configuration,* x*,* y *and* z *correspond to the number of free nodes that isolate each robot from the other, and* $\varphi(x, y, z)$ *is an additional constraint restricting the scope of values for* x, y, z*.*

Recall that $n = x + y + z + 3$ remains constant, with $n \geq 10$. Constraints defining the type itself are omitted, for instance, $[B^n(x, y, x) \mid x \neq y \wedge\ x > 0 \wedge n = y + 2x + 3]$ is simply denoted by $[B^n(x, y, x)]$.

Definition 17. *The tuple* $(s_x, s_y, s_z, [Type^n(x, y, z), \varphi(x, y, z)])$ *denotes the set*

$$\{(s_x, s_y, s_z, c) \mid c \in [Type^n(x, y, z), \varphi(x, y, z)]\}$$

of system states, where s_x *(respectively* s_y*,* s_z*) is the local state of the robot positioned before the* x *(respectively* y*,* z*) free nodes.*
For $w \in \{x, y, z\}$*, state* s_w *belongs to Front, Back, RLC. For the sake of readability, we do not represent Idle states, hence only scheduler choices about robots that can move are seen.*

For a set P *of system states, we denote by* $\mathcal{C}(P)$ *the set of configurations of* P *and by* $\mathcal{R}(P)$ *the set of rules of the algorithm that can be applied on* P*. For a rule* $R \in \mathcal{R}(P)$*, we define:*

$$succ_R(P) = \{s' \mid s \xrightarrow{R} s' \text{ for some } s \in P\}$$

the set of states produced by applying R *to states of* P*.*

An abstracted view of the algorithm is shown in Figures 4.2 and 4.3 using these notations. Gray states are initial states, more particularly the light gray ones are legitimate states. Each *Move* transition is guarded by a condition between brackets and corresponds to the choice of the scheduler to let all robots move. In the *Move*$_{any}$ transition, the scheduler lets only a single robot move.

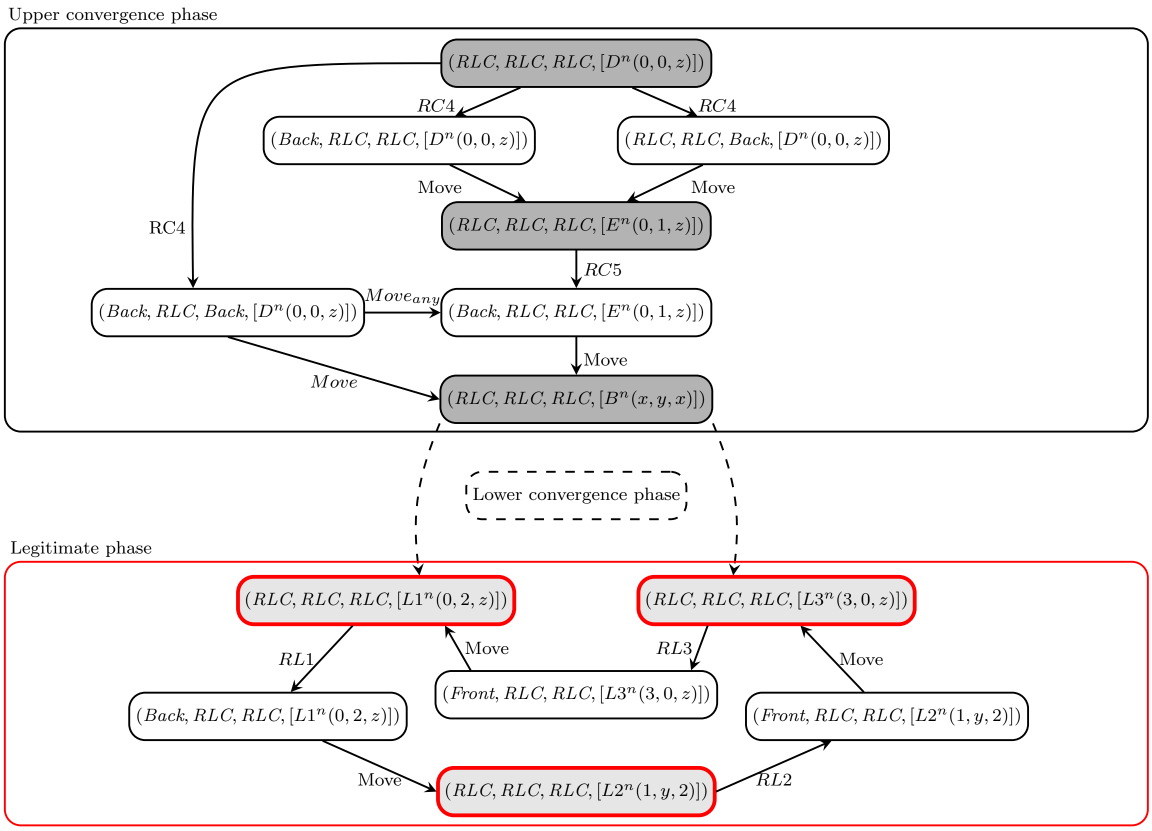


Figure 4.2: Graph of Min-algorithm.

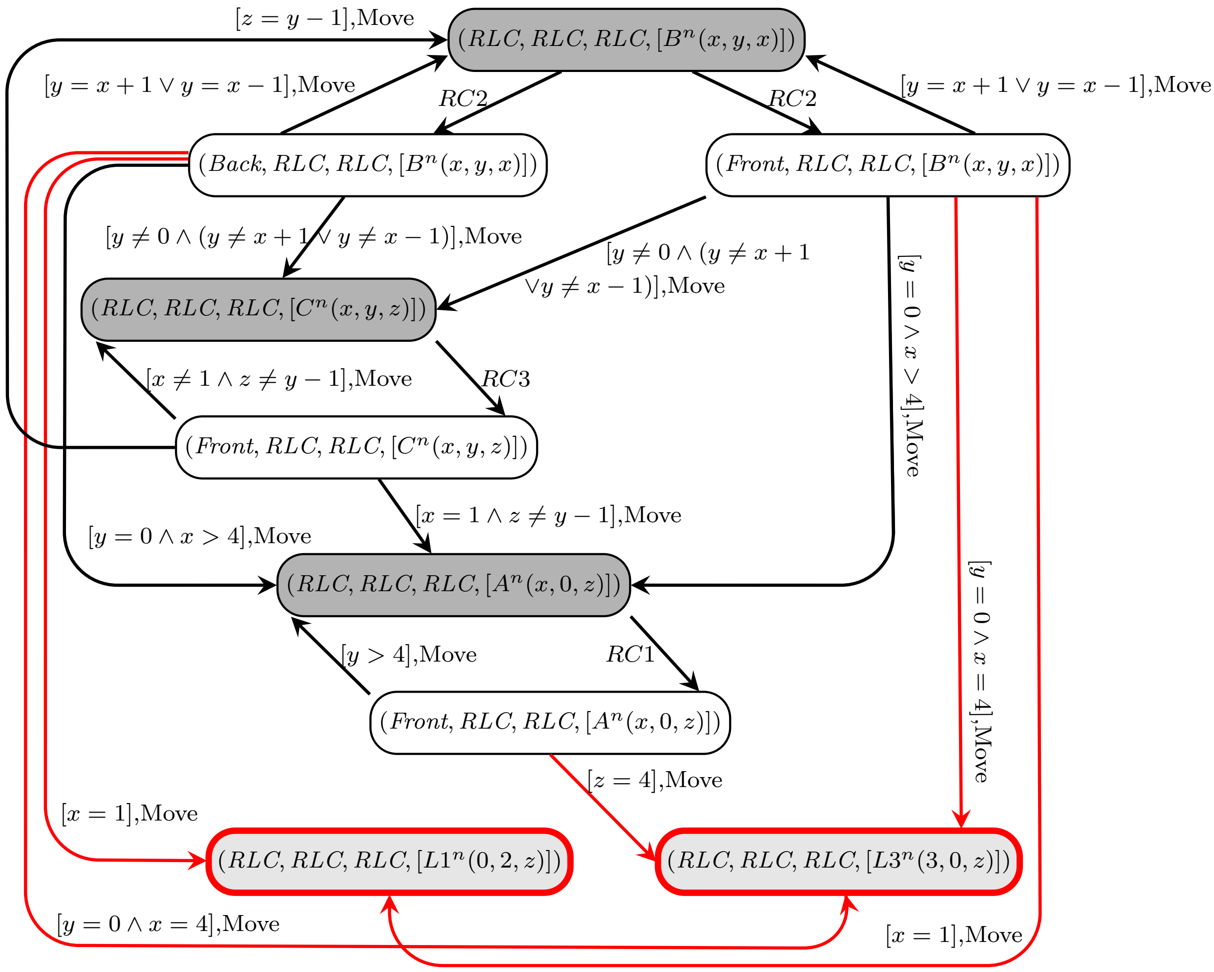


Figure 4.3: Lower convergence phase.

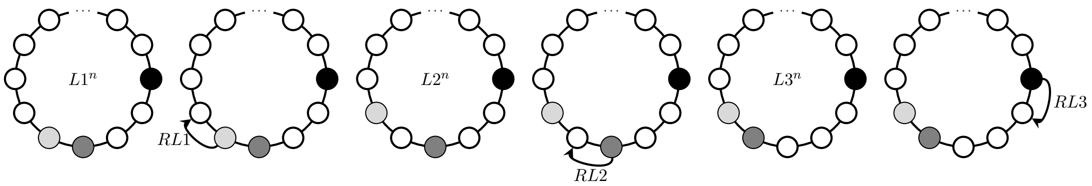


Figure 4.4: A step for the exploration.

Exploration from legitimate configurations

We prove the following theorem:

Theorem 4. *From any legitimate configuration the ring (of size $n \geq 10$, co-prime with 3) is perpetually explored.*

The result holds if from a legitimate configuration $(L1, L2, L3)$ only legitimate configurations are reached, and if from any legitimate configuration, an identical configuration is reached, where all positions have been shifted p times to the same direction, for any $p \in \mathbb{N}$. In particular, when $p > 0$ is a multiple of n, all robots have visited all nodes. These two properties are expressed by the following LTL formulas:

1. $\Box\,(\mathbb{L} \Rightarrow \Box\,\mathbb{L})$

2. $\forall i, k = 1, 2, 3,\ \forall j \in \{0, 1, \ldots, n-1\},\ \forall p \in \mathbb{N},$
   $\Box(Lk \wedge r[j] = r_i \Rightarrow \Diamond(Lk \wedge r[j+p] = r_i))$

where Lk is the predicate indicating that the configuration belongs to the corresponding set and $r[j] = r_i$ is the binary predicate giving the absolute position j for robot r_i.

By construction and for all $n \leq 10$, the first formula is satisfied since the only possible moves from $L1$, $L2$ and $L3$ for scheduled robots not staying idle are:

$succ_{RL1}\ ((RLC, RLC, RLC, [L1^n(0, 2, n-5)])) = (Back, RLC, RLC, [L1^n(0, 2, n-5)]),$
$succ_{Move}((Back, RLC, RLC, [L1^n(0, 2, n-5)])) = (RLC, RLC, RLC, [L2^n(1, 2, n-6)]),$

$succ_{RL2}\ ((RLC, RLC, RLC, [L2^n(1, 2, n-6)])) = (RLC, Front, RLC, [L2^n(1, 2, n-6)]),$
$succ_{Move}((RLC, Front, RLC, [L2^n(1, 2, n-6)])) = (RLC, RLC, RLC, [L3^n(0, 3, n-6)]),$

$succ_{RL3}\ ((RLC, RLC, RLC, [L3^n(0, 3, n-6)])) = (RLC, RLC, Front, [L3^n(0, 3, n-6)]),$
$succ_{Move}((RLC, RLC, Front, [L3^n(0, 3, n-6)])) = (RLC, RLC, RLC, [L1^n(0, 2, n-5)]).$

As mentioned previously, the second formula ensures the perpetual exploration. The proof is an easy induction over p, for an arbitrary size n.

The base case for $p = 1$: $(Lk \wedge r[j] = r_i) \implies \Diamond(Lk \wedge r[j+1] = r_i)$ results from chaining the three moves described above, as illustrated in Figure 4.4. For the induction step, assume that the property holds for p. This implies:

$$(Lk \wedge r[j] = r_i) \implies \Diamond(Lk \wedge r[j+p] = r_i).$$

Setting $j' = j + p$ and using the base case $(Lk \wedge r[j] = r_i) \implies \Diamond(Lk \wedge r[j+1] = r_i)$, we obtain: $(Lk \wedge r[j'] = r_i) \Rightarrow \Diamond(Lk \wedge r[j'+1] = r_i)$.
Hence, $(Lk \wedge r[j] = r_i) \Rightarrow \Diamond(Lk \wedge r[j+p+1] = r_i)$ and the property holds for $p+1$.

Convergence from illegitimate configurations

Theorem 5. *From any non legitimate configuration, a legitimate configuration is eventually reached (for a ring of size $n \geq 10$, co-prime with 3).*

To establish the convergence result, we associate with any subset P of system states a tree $\mathcal{T}(P)$ rooted in P, with nodes the subsets of states obtained by applying the rules of the algorithm. Reaching a set of successors in $\mathbb{L}$ without pending moves results in a leaf. More precisely:

Definition 18. *Given the set $\mathcal{R}$ of rules of the* Min-Algorithm*, let P_0 be a subset of system states. The tree $\mathcal{T}(P_0)$ has P_0 as root and for each node P:*

- *If $\mathcal{C}(P) \subseteq \mathbb{L}$ and for any $s \in P$, $w \in \{x, y, z\}$, $s_w \notin \{Front, Back\}$, then the node has no successor.*
- *Otherwise the node P has a successor $succ_R(P)$ for each $R \in \mathcal{R}(P)$.*

We now prove that for any set of states P such that $\mathcal{C}(P)$ is contained in one of the non legitimate configurations types, the tree $\mathcal{T}(P)$ is finite. This yields the desired convergence proof. If for some P, the tree $\mathcal{T}(P)$ is infinite, then there exists an infinite sequence of rules (on an infinite path of this tree) such that for all successor sets P' of P along this sequence, either $\mathcal{C}(P') \not\subseteq \mathbb{L}$ or there is some $s \in P'$ such that $s_w \in \{Front, Back\}$ for some $w \in \{x, y, z\}$, meaning that the corresponding robot has a pending move.

To prove this result, we exhaustively verify the property for all types A^n, B^n, C^n, D^n or E^n, by inductive proofs, in Lemmas 6 to 10 (where we assume $n \geq 10$ and n co-prime with 3). Note that these lemmas must be proved in the order A, C, B, E and D. Since $\mathbb{NL}^n = A^n \cup B^n \cup C^n \cup D^n \cup E^n$ the result follows.

Lemma 6. *The tree $\mathcal{T}(P)$ is finite for $P = (RLC, RLC, RLC, [A^n(x, 0, z), 4 \leq x < z])$.*

Proof. The idea of the proof is as follows: recall that from an A^n configuration (R_1, F_x, R_2, F_z) with $4 \leq x < z$, written $[A^n(x, 0, z), 4 \leq x < z]$, only one movement is feasible, leading to an $[L3^n(3, 0, z)]$ configuration if $x = 4$ and to an $[A^n(x-1, 0, z+1)]$ configuration otherwise. Hence, the number of free nodes in the x part decreases until an $L3$ configuration is reached.

We first prove the property for an arbitrary z when $x = 4$ (base case). Then we prove the induction step on x.

Base-case: For $P = (RLC, RLC, RLC, [A^n(4, 0, z)])$, with any $z \geq 6$, the tree $\mathcal{T}(P)$ is finite with the moves:

$$\begin{array}{c}
\boxed{(RLC, RLC, RLC, [A^n(4, 0, z)])} \\
\downarrow RC1 \\
\boxed{(Front, RLC, RLC, [A^n(4, 0, z)])} \\
\downarrow Move \\
\boxed{(RLC, RLC, RLC, [L3^n(3, 0, z + 1)])}
\end{array}$$

Induction step: Assume that the tree with root $P = (RLC, RLC, RLC, [A^n(x, 0, z)])$, for any $z > x$ is finite. For $x + 1 < z$, the moves are:

$$\begin{array}{c}
\boxed{(RLC, RLC, RLC, [A^n(x + 1, 0, z)])} \\
\downarrow RC1 \\
\boxed{(Front, RLC, RLC, [A^n(x + 1, 0, z)])} \\
\downarrow Move \\
\boxed{(RLC, RLC, RLC, [A^n(x, 0, z + 1])}
\end{array}$$

they lead to $(RLC, RLC, RLC, [A^n(x, 0, z + 1])$ for which the tree is finite from the induction hypothesis. This ensures the desired result. $\square$

Lemma 7. *The tree $\mathcal{T}(P)$ is finite for*
$P = (RLC, RLC, RLC, [C^n(x, y, z), 0 < x < z < y])$ *with* $(x, z) \neq (1, 2)$.

Proof. We first fix parameter x and show that the tree for P is finite, for any y, z with $0 < x < z < y$. Then we prove by induction that it holds for any x using the first proof as base case.

Base-case: x=1

- If $z = y - 1$, the tree is:

$$
\begin{tikzcd}
 & (RLC, RLC, RLC, [C^n(1, y, y-1)]) \arrow[d, "RC3"] & \\
 & (\mathit{Front}, RLC, RLC, [C^n(1, y, y-1)]) \arrow[d, "\mathit{Move}"] & \\
 & (RLC, RLC, RLC, [B^n(0, y, y)]) \arrow[dl, "RC2"'] \arrow[dr, "RC2"] & \\
B_B \arrow[d, "\mathit{Move}"] & & B_F \arrow[d, "\mathit{Move}"] \\
AL & & AL
\end{tikzcd}
$$

where:
$B_B = (RLC, RLC, \mathit{Back}, [B^n(0, y, y)])$ and $B_F = (RLC, RLC, \mathit{Front}, [B^n(0, y, y)])$
and if $y = 4$, $AL = (RLC, RLC, RLC, [L3^n(3, 0, 5)])$
otherwise $y > 4$, $AL = (RLC, RLC, RLC, [A^n(y-1, 0, y+1)])$
Note that in both cases, the move from B_B or B_F leads to the same equivalence class of configurations: an $L3^n$ class when $y = 4$ and an A^n class otherwise. From Lemma 6, the result holds for $x = 1$ and $z = y - 1$.

- If $2 < z < y - 1$ (Recall that if $z = 2$, since $x = 1$, it is a $L2^n$ configuration), the moves are:
  $succ_{RC3}\ ((RLC, RLC, RLC, [C^n(1, y, z)])) = (\mathit{Front}, RLC, RLC, [C^n(1, y, z)])$,
  $succ_{Move}((\mathit{Front}, RLC, RLC, [C^n(1, y, z)]) = (RLC, RLC, RLC, [A^n(0, y, z+1)])$.
  The last configuration is an A^n configuration since $2 < z < y - 1$. Similarly as above, the property results from Lemma 6.

Finally, it results from the above cases that the tree $\mathcal{T}(RLC, RLC, RLC, [C^n(1, y, z)])$ is finite for any y, z.

Induction step: We now assume that the tree of root $(RLC, RLC, RLC, [C^n(x, y, z)])$ is finite for any y, z, and prove that the same is true for $\mathcal{T}(RLC, RLC, RLC, [C^n(x+1, y, z)]$.

- If $z = y - 1$, the tree is

$$\begin{tikzcd}
& (RLC, RLC, RLC, [C^n(x+1, y, y-1)]) \arrow[d, "RC3"] & \\
& (Front, RLC, RLC, [C^n(x+1, z, z-1)]) \arrow[d, "Move"] & \\
& (RLC, RLC, RLC, [B^n(x, y, y)]) \arrow[dl, "RC2"'] \arrow[dr, "RC2"] & \\
B_B \arrow[d, "Move"] & & B_F \arrow[d, "Move"] \\
C & & C
\end{tikzcd}$$

where

$$\begin{array}{lcl}
B_B & = & (RLC, RLC, Back, [B^n(x, y, y)]) \\
B_F & = & (RLC, RLC, Front, [B^n(x, y, y)]) \\
C & = & (RLC, RLC, RLC, [C^n(x, y+1, y-1)])
\end{array}$$

Thanks to the induction hypothesis, we can conclude that the property holds in this case.

- If $2 < z < y - 1$, applying the algorithm yields the movements:
  $succ_{RC3}((RLC, RLC, RLC, [C^n(x+1, y, z)])) = (Front, RLC, RLC, [C^n(x+1, y, z)])$,
  $succ_{Move}((Front, RLC, RLC, [C^n(x+1, y, z)]) = (RLC, RLC, RLC, [C^n(x, y, z+1)])$.
  The last configuration is a C^n configuration since $2 < z < y-1$, hence the property holds from the induction hypothesis.

Finally all trees $\mathcal{T}(RLC, RLC, RLC, [C^n(x, y, z)])$ with $0 < x < z < y$ and $(x, z) \neq (1, 2)$ are finite. □

Lemma 8. *The tree $\mathcal{T}(P)$ is finite for*
$P = (RLC, RLC, RLC, [B^n(x, y, x), x > 0 \wedge x \neq y])$.

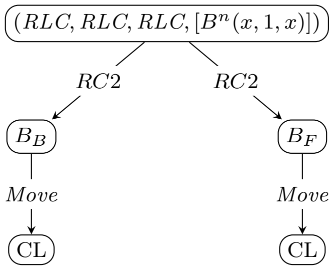
Proof. We handle two cases: $x > y$ and $x < y$.
Case $x > y$: We first handle the subcases $y = 0$ and $y = 1$.

- When $y = 0$, applying the algorithm yields the moves:

$$\begin{tikzcd}
& (RLC, RLC, RLC, [B^n(x, 0, x)]) \arrow[dl, "RC2"'] \arrow[dr, "RC2"] & \\
B_B \arrow[d, "Move"] & & B_F \arrow[d, "Move"] \\
AL & & AL
\end{tikzcd}$$

where
$B_B = (Back, RLC, RLC, [B^n(x, 0, x)])$
$B_F = (Front, RLC, RLC, [B^n(x, 0, x)])$
and if $x = 4$, $AL = (RLC, RLC, RLC, [L3^n(3, 0, 5)])$
otherwise $x > 4$, $AL = (RLC, RLC, RLC, [A^n(x - 1, 0, x + 1)])$
From B^n configurations, where $y = 0$, the moves lead to an $L3^n$ configurations when $x = 4$, and to an A^n configuration otherwise. Hence, from Lemma 6, the property holds when $y = 0$ for any x.

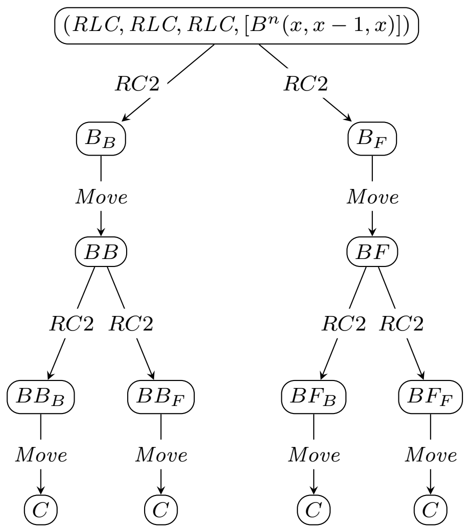
- When $y = 1$, the tree representing the algorithm is the following:



where
$B_B = (Back, RLC, RLC, [B^n(x, 1, x)])$
$B_F = (Front, RLC, RLC, [B^n(x, 1, x)])$
and if $x = 3$, $CL = (RLC, RLC, RLC, [L2^n(1, 4, 2)])$
otherwise $x > 3$, $CL = (RLC, RLC, RLC, [C^n(1, x + 1, x - 1)])$.
Similarly as above there are two cases when $x = 3$ or $x > 3$. In the first case the moves reach $L2^n$ configurations, and in the second one to C^n configurations. From Lemma 7, the property holds when $y = 1$ for any x.

We now show the moves for any x, y when $x > y > 1$. We need two subcases when $x - 1 = y$ and $x - 1 > y$.

- If $x - 1 = y$, the moves are:

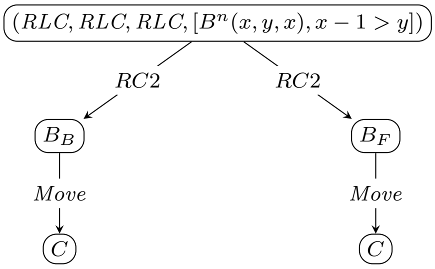


where

$$\begin{array}{lcl}
B_B & = & (Back, RLC, RLC, [B^n(x, x-1, x)]) \\
B_F & = & (Front, RLC, RLC, [B^n(x, x-1, x)]) \\
BB & = & (RLC, RLC, RLC, [B^n(x+1, x-1, x-1)]) \\
BF & = & (RLC, RLC, RLC, [B^n(x-1, x-1, x+1)]) \\
BB_B & = & (RLC, RLC, Back, [B^n(x+1, x-1, x-1)]) \\
BB_F & = & (RLC, RLC, Front, [B^n(x+1, x-1, x-1)]) \\
BF_B & = & (RLC, Back, RLC, [B^n(x-1, x-1, x+1)]) \\
BF_F & = & (RLC, Front, RLC, [B^n(x-1, x-1, x+1)]) \\
C & = & (RLC, RLC, RLC, [C^n(x-2, x+1, x)])
\end{array}$$

The property holds in this case, thanks to Lemma 7.

- If $x-1 > y$, and $y > 1$ the tree is:



where

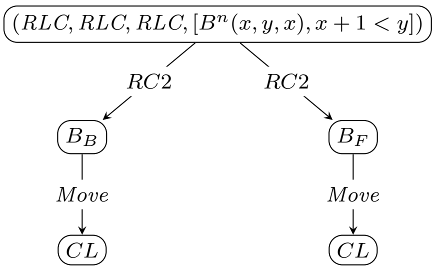
$$\begin{array}{lcl}
B_B & = & (Back, RLC, RLC, [B^n(x, y, x)]) \\
B_F & = & (Front, RLC, RLC, [B^n(x, y, x)]) \\
C & = & (RLC,\ RLC, RLC, [C^n(x+1, y, x-1)])
\end{array}$$

and Lemma 7 entails the result.

Hence, $\mathcal{T}(RLC, RLC, RLC, [B^n(x, y, x), x > y]))$ is finite.

Case $x < y$: We handle two subcases when $y > x+1$ and $y = x+1$.

- If $x+1 < y$, then the moves are:



where

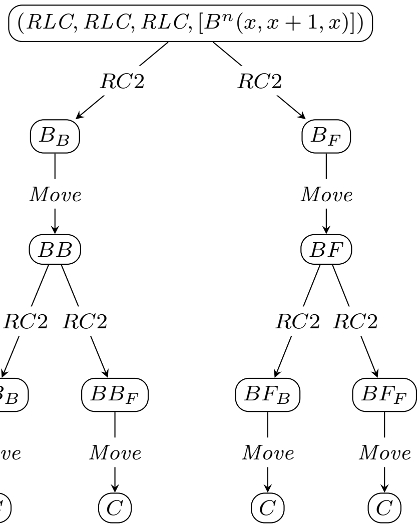
$$\begin{array}{lcl}
B_B & = & (Back, RLC, RLC, [B^n(x, y, x), x+1 < y)]) \\
B_F & = & (Front, RLC, RLC, [B^n(x, y, x), x+1 < y)
\end{array}$$

and if $x = 1$

$$\begin{array}{lcl}
CL & = & (RLC,\ RLC, RLC, [L1^n(2, y, 0)])
\end{array}$$

otherwise ($x > 1$):
$CL \;=\; (RLC, RLC, RLC, [C^n(x-1, y, x+1)])$
When $x = 1$ the moves lead to $L1^n$ configurations and to C^n configurations otherwise. Hence, again thanks to Lemma 7, the property holds for any x, y when $x + 1 < y$.

- If $x + 1 = y$, the tree is:



where

$$
\begin{array}{lcl}
B_B & = & (Back, RLC, RLC, [B^n(x, x+1, x)]) \\
B_F & = & (Front, RLC, RLC, [B^n(x, x+1, x)]) \\
BB & = & (RLC, RLC, RLC, [B^n(x+1, x+1, x-1)]) \\
BF & = & (RLC, RLC, RLC, [B^n(x-1, x+1, x+1)]) \\
BB_B & = & (RLC, Back, RLC, [B^n(x+1, x+1, x-1)]) \\
BB_F & = & (RLC, Front, RLC, [B^n(x+1, x+1, x-1)]) \\
BF_B & = & (RLC, RLC, Back, [B^n(x-1, x+1, x+1)]) \\
BF_F & = & (RLC, RLC, Front, [B^n(x-1, x+1, x+1)]) \\
C & = & (RLC, RLC, RLC, [C^n(x, x+2, x-1)])
\end{array}
$$

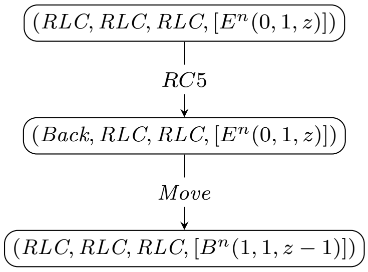
Since $\mathcal{T}(RLC, RLC, RLC, [C^n(x, y, z)])$ is finite (by Lemma 7), the property holds.

Finally the trees $\mathcal{T}(RLC, RLC, RLC[B^n(x, y, x), x < y])$ are also finite, which concludes the proof.

□

Lemma 9. *The tree $\mathcal{T}(P)$ is finite for $P = (RLC, RLC, RLC, [E^n(0, 1, z)])$.*

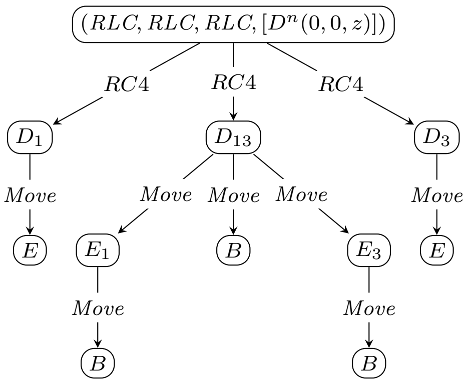
Proof. In the case of E^n configurations, we have:



and the result holds thanks to Lemma 8. □

Lemma 10. *The tree $\mathcal{T}(P)$ is finite for $P = (RLC, RLC, RLC, [D^n(0, 0, z)])$.*

Proof. From a D^n configuration, it is also possible to schedule two robots with their respective planned moves. The various cases lead to either an E^n configuration with or without a pending movement, or a B^n configuration:



where

$$
\begin{array}{lcl}
D_1 & = & (Back, RLC, RLC, [D^n(0, 0, z)]) \\
D_{13} & = & (Back, RLC, Back, [D^n(0, 0, z)]) \\
D_3 & = & (RLC, RLC, Back, [D^n(0, 0, z)]) \\
E & = & (RLC, RLC, RLC, [E^n(0, 1, z-1)]) \\
E_1 & = & (RLC, RLC, Back, [E^n(1, 0, z-1)]) \\
E_3 & = & (Back, RLC, RLC, [E^n(0, 1, z-1)]) \\
B & = & (RLC, RLC, RLC, [B^n(1, 1, z-2)])
\end{array}
$$

and the result holds from the previous lemmas 8 and 9. □

Together these lemmas imply Theorem 5. Finally, Theorems 4 and 5 give the result for perpetual exploration. Moreover, since all reachable configurations from any of the initial configurations are tower-free, the *Exclusivity* property follows (recall that the *No_ collision* property implies the *No_ switch* property in the asynchronous case as mentioned in Section 4.1). This concludes the correctness proof of the algorithm.

Part II

Synthesis of mobile robot protocols

Chapter 5

Synthesis of synchronous mobile robot protocols

In this chapter, we introduce the use of formal methods for automatic synthesis of autonomous mobile robot algorithms, in the discrete space model. As a case study, we consider the problem of gathering all robots at a particular position, not known beforehand. We propose an encoding of the gathering problem in a synchronous execution model as a reachability game, the players being the robot algorithm on one side and the scheduling adversary (that can also dynamically decide robot chirality at every activation) on the other side. Our encoding is general enough to encompass classical execution models for robots evolving on ring-shaped networks, including when several robots are located at the same node and when symmetric situations occur. Our encoding allow us to automatically generate an *optimal* distributed algorithm, in the Fsync model, for three robots evolving on a fixed size ring. The optimality criterion refers to the number of robot moves that are necessary to actually achieve gathering.

5.1 Gathering games

In this section we recall classical notions (from [Maz01]) related to two-player reachability games, simply called games in the sequel.

A game is composed of an *arena* and *winning conditions*.

Arena An arena for a two-player game is a graph $\mathcal{A} = (V, E)$ in which the set of vertices $V = V_p \uplus V_a$ is partitioned into V_p, the player locations, and V_a the adversary locations. The set of edges $E \subseteq V \times V$ allows to define the set of successors of some given vertex v, noted $vE = \{v' \in V \mid (v, v') \in E\}$. In the sequel, we only consider finite arenas.

Plays To play on an arena, a token is positioned on an initial vertex. Then the token is moved by the players from one vertex to one of its successors. Each player can move the token only if it is on one of her own vertices. Formally, a play is a path in the graph. Moreover, we only consider maximal plays: a play is maximal if it is either infinite or finite such that the last vertex of the play has no successor ($\pi = v_0 v_1 \ldots, v_n$, with $v_n E = \emptyset$).

Strategies A strategy for the player determines to which position she will bring the token whenever it is her turn to play. To do so, the player takes into account the history of the play, and the current vertex. Formally, a strategy for the player is a (partial) function $\sigma : V^* \cdot V_p \to V$ such that, for any sequence (representing the current history) $w \in V^*$, any $v \in V_p$, $\sigma(w \cdot v) \in vE$ (*i.e.*, the move is possible with respect to the arena). A strategy σ is *memoryless* if it does not depend on the history. Formally, it means that for all $w, w' \in V^*$, for all $v \in V_p$, $\sigma(w \cdot v) = \sigma(w' \cdot v)$. In that case, we may simply see the strategy as a mapping $\sigma : V_p \to V$.

Given a strategy σ for the player, a play $\pi = v_0 v_1 \cdots \in V^\infty$ is said to be σ-*consistent* if for all $0 < i < |\pi|$, if $v_{i-1} \in V_p$, then $v_i = \sigma(v_0 \cdots v_{i-1})$. Given an initial vertex v_{init}, the *outcome* of a strategy σ is the set of plays starting in v_{init} that are σ-consistent. Formally, given an arena $\mathcal{A} = (V, E)$, an intial vertex v_{init} and a strategy $\sigma : V^* V_p \to V$, we let $Outcome(\mathcal{A}, v_{init}, \sigma) = \{\pi \in V^\infty \mid v_0 = v_{init}$ and π is a maximal play and is σ-consistent$\}$.

Winning conditions, winning plays, winning strategies We define the *winning condition* for the player as a subset of the plays $Win \subseteq V^\infty$. Then, a play π is *winning* for the player if $\pi \in Win$. In this work, we focus on the simple case of reachability games: the winning condition is then expressed according to a subset of vertices $T \subseteq V$ called the *target* by $Reach(T) = \{\pi = v_0 v_1 \cdots \in V^\infty \mid \pi$ is maximal and $\exists i, 0 \leq i < |\pi| : v_i \in T\}$. This means that the player wins a play whenever the token is brought on a vertex belonging to the set T. Once it has happened, the play is winning, regardless of the following actions of the adversary.

Given an arena $\mathcal{A} = (V, E)$, an initial vertex $v_{init} \in V$ and a winning condition Win, a *winning strategy* σ for the player is a strategy such that any σ-consistent play is winning. In other words, a strategy σ is winning if $Outcome(\mathcal{A}, v_{init}, \sigma) \subseteq Win$. The player wins the game $(\mathcal{A}, v_{init}, Win)$ if she has a winning strategy for $(\mathcal{A}, v_{init}, Win)$. We say that σ is winning on a subset $U \subseteq V$ if it is winning starting from any vertex in U: $Outcome(\mathcal{A}, v, \sigma) \subseteq Win$ for all $v \in U$. A subset $U \subseteq V$ of the vertices is *winning* if there exists a strategy σ that is winning on U. The *winning region* is the maximal winning set.

Solving a reachability game Given an arena $\mathcal{A} = (V, E)$, a subset $T \subseteq V$, one wants to determine the winning region $U \subseteq V$ of the player for $Reach(T)$, and a strategy $\sigma : V^* V_p \to V$ for the player, that is winning on U.

Example 10. *Figure 5.1 represents a reachability two-player game. Player vertices are here represented by rectangles and adversary vertices by circles. The winning condition is* $Reach(\{P3\})$. *From P2 the player has no winning strategy, since from O1 the adversary can always bring the game back to P2, producing an infinite loop that never goes to the target. From P1 a (memoryless) winning strategy is to go to O2. The winning region of the player is* $\{P1, P3\}$.

We recall now a well-known result on reachability games [Mar75]:

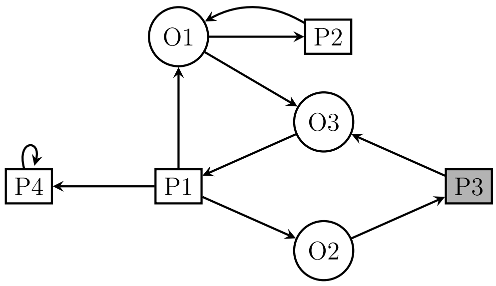


Figure 5.1: A two player game with a reachability objective.

Theorem 11. *The winning region for the player in a reachability game can be computed in linear time in the size of the arena. Moreover, from any location, the player has a winning strategy if and only if she has a memoryless winning strategy.*

5.2 Synthesis for synchronous robots

5.2.1 Arena construction

We build an arena for a reachability game, such that the player has a winning strategy if and only if one can design an algorithm for synchronous robots to gather on a single node, starting from any configuration. We consider the arena $\mathcal{A}_{\text{gather}} = (V_p \uplus V_a, E)$, where:

- the set of player locations is $V_p = (Obs/\!\equiv)$, the set of equivalence classes of observations (see Definition 8 in Chapter 2),
- the set of adversary locations is $V_a = Obs \times \Delta^k$, with $\Delta = \{Front, Back, Doubt, Idle\}$ the set of actions as defined in 2.4.4.

The size of the arena is thus linear in n and exponential in k. The edge relation E is detailed in the rest of the subsection and will ensure a strict alternation between the two players: $E \subseteq (V_p \times V_a) \cup (V_a \times V_p)$.

Edges from V_p to V_a

From a player location, representing an equivalence class of observations, the play continues on an adversary location memorizing the different movements decided by each robot. Such a move is possible if, in a given equivalence class of observations, the robots with the same view take the same decision. Recall that an algorithm $\mathcal{S}$ is given by a decision function (see Definition 12 in Chapter 2).

We now define the edge relation from a player location to an adversary location. For $v \in V_p$, recall that we can consider the canonical configuration c_o for $o = rep(v)$. Then

$(v, v') \in E$ with $v' = \big(o, (a_1, \ldots, a_k)\big)$ if and only if there exists a decision function ∂ such that $a_i = \partial(view(r_i, c_o))$ for all i in $[1..k]$.

Edges from V_a to V_p

The moves of the adversary lead the game into the configuration of the system resulting from the application of the decisions of all robots. If a robot decides to move, but is disoriented, then the adversary chooses the actual move (*Front* or *Back*). The next configuration is then determined by the actions chosen and the decisions taken by the adversary, defined as a k-tuple of movements $m = (m_i)_{1 \le i \le k} \in (\Delta \setminus \{Doubt\})^k$.

Let r be a robot and let c' be the configuration resulting from the move m applied to configuration c. We write $obs(r, c') = obs(r, c) \oplus m$ the corresponding observation. The next result states that the equivalence classes are consistent with equivalent movements of the robots.

Proposition 12. *Let $c \approx c'$ be two equivalent configurations, and let o and o' be observations of c and c' respectively. Then, for any move $m = (m_1, \ldots, m_k) \in (\Delta \setminus \{Doubt\})^k$, there exists $m' = (m'_1, \ldots, m'_k) \in (\Delta \setminus \{Doubt\})^k$ such that $o \oplus m \equiv o' \oplus m'$.*

Proof. Let $c, c' \in \mathcal{C}$ with $c' \approx c$ and let $o, o' \in Obs$ be observations of c and c' respectively. For a move $m = (m_i)_{1 \le i \le k} \in (\Delta \setminus \{Doubt\})^k$ from c, we define the move m' that will represent the same decisions, on configuration c'. Thanks to Proposition 2, we know that o and o' are in the same equivalence class for $\equiv$, hence if $o = \{(f_1, \cdots, f_k), (f_k, \cdots, f_1)\}$ then $o' = \{(f_i, \ldots, f_k, f_1, \ldots, f_{i-1}), (f_{i-1}, \ldots, f_1, f_k, \ldots, f_i)\}$ for some i, $1 \le i \le k$. Now, ordering the two tuples of o' in lexicographical order, we consider two cases. If $(f_i, \ldots, f_k, f_1, \ldots, f_{i-1})$ is the smallest observation, then $m'_j = m_{(i+j-1)}$ for all $1 \le j \le k$. Otherwise, for all $1 \le j \le k$:

$$m'_j = \begin{cases} \textit{Front} & \text{if } m_{i-j} = \textit{Back} \\ \textit{Back} & \text{if } m_{i-j} = \textit{Front} \\ \textit{Idle} & \text{if } m_{i-j} = \textit{Idle} \end{cases}$$

We have $o \oplus m \equiv o' \oplus m'$. □

We now define v-consistent moves where the adversary in some vertex v resolves all the *Doubt* actions by choosing in which directions disoriented robots will move.

Definition 19. *For a state $v = (o, (a_1, \ldots, a_k)) \in V_a$, a move $m = (m_1, \ldots, m_k) \in (\Delta \setminus \{Doubt\})^k$ is* v-consistent *if:*

- *for all $1 \le i \le k$ such that $a_i \neq Doubt$, $m_i = a_i$,*
- *for all $1 \le i \le k$ such that $a_i = Doubt$, $m_i \neq Idle$.*

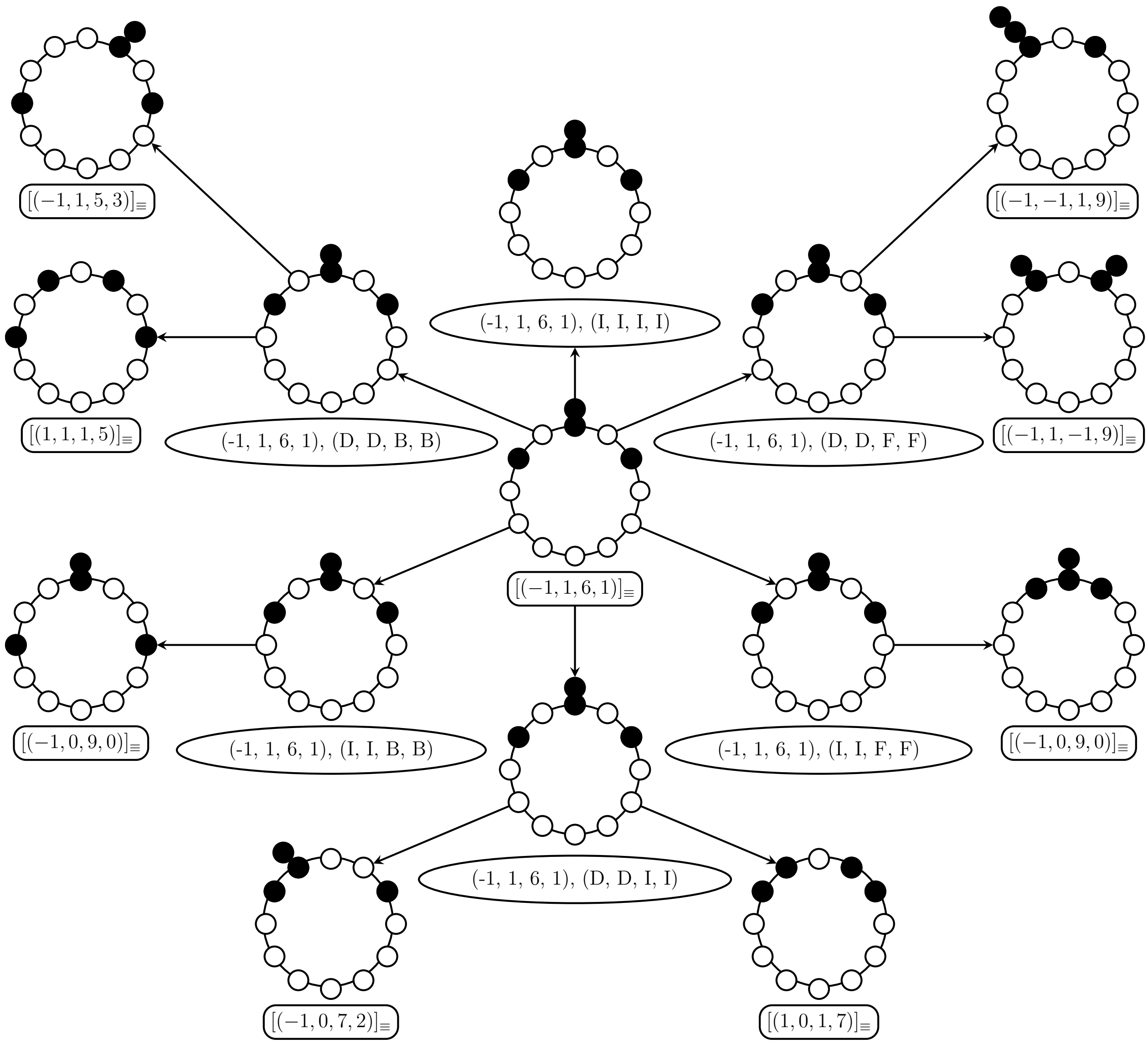


Figure 5.2: A part of the gathering arena for 4 robots in a 12 nodes ring.

The edge relation from an adversary location to a player location is then defined by: for $v = (o, (a_1, \dots, a_k)) \in V_a$, and $v' \in V_p$, the edge (v, v') belongs to E if and only if there exists a v-consistent move m such that $v' = [o \oplus m]_\equiv$.

The Figure 5.2 represents a part of the gathering arena for 4 robots on a ring of size 12.

Gathering is defined as follows:
Let $v_T = [(-1, \cdots, -1, n-1), (n-1, -1, \cdots, -1)]_\equiv \in V_p$ be the equivalence class of all configurations corresponding to all robots positioned on a single node. For the game on $\mathcal{A}_{\text{gather}}$ with winning condition $\mathit{Reach}(\{v_T\})$, the winning region corresponds exactly to the set of configurations from which robots can achieve the gathering.

We must show now that solving this reachability game amounts to automatically synthesizing a deterministic algorithm achieving the gathering for this system. Let $\mathcal{S}$ be an algorithm given by its decision function $\partial_\mathcal{S}$. The notion of consistency is adapted to $\mathcal{S}$ as follows: for a configuration $c \in \mathcal{C}$ and a corresponding observation $o \in \mathit{Obs}$, $m \in (\Delta \setminus \{\mathit{Doubt}\})^k$ is a $(\mathcal{S}, c)$-consistent move if it is a v-consistent move for $v = (o, (a_1, \dots, a_k))$ with $a_i = \partial_\mathcal{S}(\mathit{view}(r_i, c))$, $1 \leq i \leq k$. We denote by $M(\mathcal{S}, c)$ the set of all $(\mathcal{S}, c)$-consistent moves.
Let c, c' be two configurations, and let o and o' be observations such that $o = \mathit{obs}(r, c)$ and $o' = \mathit{obs}(r, c')$ for some robot r. We denote by $c \xrightarrow{m} c'$ the application to configuration c of the move m, where m is $(\mathcal{S}, c)$-consistent, leading to configuration c', defined by: $o' = o \oplus m$.

For a configuration c, we note $\mathit{succ}(\mathcal{S}, c)$ the set of configurations produced by applying $M(\mathcal{S}, c)$ on c, more formally:

$$\mathit{succ}(\mathcal{S}, c) = \{c' \mid \exists m \in M(\mathcal{S}, c) \textit{ such that } c \xrightarrow{m} c'\}.$$

Proposition 12 implies that, for two equivalent configurations $c \approx c'$, $\mathit{succ}(\mathcal{S}, c) = \mathit{succ}(\mathcal{S}, c')$.

The next result gives the correspondence between algorithms and winning strategies in the reachability game.

Theorem 13. *There exists an algorithm which achieves gathering if and only if there exists a memoryless winning strategy for the reachability game $\mathcal{A}_{gather}$ with winning condition $\mathit{Reach}(\{v_T\})$.*

Proof. We first show that if some algorithm $\mathcal{S}$ achieves gathering then we can build a winning strategy $\sigma : V_p \to V_a$. Conversely we show that given a winning region and a memoryless strategy, one can define a unique algorithm for the robots.

From $\mathcal{S}$ we construct the memoryless strategy σ defined for any $v \in V_p$ by:

$$\sigma(v) = (\mathit{rep}(v), (a_1, \dots, a_k)) \text{ with } a_i = \partial_\mathcal{S}(\mathit{view}(r_i, c_{\mathit{rep}(v)})), 1 \leq i \leq k.$$

We now have to prove that if $\mathcal{S}$ achieves the gathering then σ is winning. Our interest is on the plays in the *Outcome* of σ: these plays are σ-consistent and any edge $(v, v') \in V_a \times V_p$

in these plays results from a v-consistent move: For such an edge (v, v'), let c be the configuration in v, with observation o, there exists a v-consistent move m such that $v' = [o']_\equiv$, with $o' = o \oplus m$. Since m is also $(\mathcal{S}, c)$-consistent, then $c' \in \mathit{succ}(\mathcal{S}, c)$ and thus, any configuration from which the algorithm ensures gathering is winning for the game.

Conversely, let W be the winning region, and let σ be a (memoryless, from Theorem 11) winning strategy. For a configuration $c \in \mathcal{C}$, o a corresponding observation, assume that the strategy is defined by: $\sigma([o]_\equiv) = (\mathit{rep}([o]_\equiv), (a_1, \ldots, a_k))$. Then, we define a unique algorithm $\mathcal{S}$ by extracting the decision function underneath the strategy as follows: $\partial_{\mathcal{S}}(\mathit{view}(r_i, c)) = a_i$, for all robots $r_i \in \mathit{Rob}$. We now show that this algorithm achieves gathering from any configuration c such that its observation class $[o]_\equiv$ belongs to W. Let c' be a configuration in $\mathit{succ}(\mathcal{S}, c)$, then there exists a $(\mathcal{S}, c)$-consistent move $m \in M(\mathcal{S}, c)$ such that $c \xrightarrow{m} c'$. For all succession of configurations $c \xrightarrow{m} c' \rightarrow \ldots$ obtained by successive applications of the algorithm, we have a corresponding play that is σ-consistent, in the game. Since $[o]_\equiv$ is a winning position, the play is winning and $\mathcal{S}$ achieves gathering. $\square$

In the following subsection we explain in details how the robot moves have been implemented. In particular the $o \oplus m$ notation is defined explicitely here.

5.2.2 Implementation details

For efficiency reasons, our implementation does not handle configurations but only observations. More precisely we only work on the smallest k-tuple in $\mathcal{F}$ of an observation, from which we must recover the relevant information to perform robot moves. Recall (from Subsection 2.4.2) that from an observation class $\mathit{obs} \in \mathit{Obs}/\equiv$, we extract an observation $\mathit{rep}(\mathit{obs}) = o = \{F, \tilde{F}\}$ with $F = (f_1, \ldots, f_k)$ in $\mathcal{F}$ minimal among all tuples in obs. We associate with o a canonical configuration c_o defined by $c_o(r_1) = 0$, $c_o(r_2) = f_1 + 1, \ldots, c_o(r_k) = \Sigma_{i=1}^{k} f_i +_n k$ where the robot r_i is at distance f_i of robot $r_{i+_k 1}$.

Recall that a robot movement in $\mathit{Front}, \mathit{Back}, \mathit{Idle}$ is given according to the robot minimal view (see the paragraph 2.4.4). When a robot r_i moves, it modifies the distances f_i and f_{i-1} (increasing one of these two distances by one, and decreasing by one the other). Formally, the effect on an observation of a configuration, of any movement m_i of robot r_i can be described by a mapping $\varepsilon : \{1, \ldots, k\} \times \{\mathit{Front}, \mathit{Back}, \mathit{Idle}\} \times F \rightarrow \{-1, 0, 1\}^k$. This mapping denotes the translated moves that permit to apply real movements on the observations class and is defined by:

- If $\mathit{view}^{\min}{}_{-}{}^{r} \circlearrowleft^m F$ for some m then $\varepsilon(i, \mathit{Back}, F) = \varepsilon_{i,\mathit{Back}}$, and $\varepsilon(i, \mathit{Front}, F) = \varepsilon_{i,\mathit{Front}}$.
- If there exists F' in $\mathcal{F}$ such that $\mathit{view}^{\min}{}_{-}{}^{r} \sim F'$ and $F' \circlearrowleft^m F$ for some m then $\varepsilon(i, \mathit{Back}, F) = \varepsilon_{i,\mathit{Front}}$, and $\varepsilon(i, \mathit{Front}, F) = \varepsilon_{i,\mathit{Back}}$,
- and $\varepsilon(i, \mathit{Idle}, F) = 0^k$.

where $\varepsilon_{i,Back}$ and $\varepsilon_{i,Front}$ are defined by:

$\varepsilon_{i,Back} = (\varepsilon_{i,h})_{1\leq h\leq k}$ with:

- $\varepsilon_{i,i-1} = -1$,
- $\varepsilon_{i,i} = 1$,
- $\varepsilon_{i,h} = 0$ for $h \notin \{i-1, i\}$.

$\varepsilon_{i,Front} = (\varepsilon_{i,h})_{1\leq h\leq k}$ with:

- $\varepsilon_{i,i} = -1$,
- $\varepsilon_{i,i-1} = 1$,
- $\varepsilon_{i,h} = 0$ for $h \notin \{i-1, i\}$.

The idea is to add (in an element-by-element fashion) the current observation to all the vectors representing the movements of the robots to obtain the next configuration. However, when the movements of two adjacent robots imply that they switch their positions in the ring, some absurd values (-2 or -3) may appear in the obtained configuration, if the sum is naively performed, so a careful treatment of these particular cases must be done. To obtain the correct configuration, one should recall that robots are anonymous, hence if two robots switch their positions, it has the same effect as if none of them has moved. Also, if in a tower, some robots want to move *Front*, and the others want to move *Back*, the exact robots that will move are of no importance: only the number of robots that move in each direction is important. We will then reorganize the movements between the robots, in order to keep correct values in our configurations:

- in a tower, we assume that the robots that move *Back* always are the bottom ones, and robots that move *Front* are the top ones,
- when a robot moves *Front* and joins a tower, we assume that it is placed at the bottom of the tower,
- and when it moves *Back* and joins a tower, it is placed at the top of the tower.

These conventions ensure that when adding the configuration and the different movements, we obtain correct values.

We define $PosTower(F)$, a set that contains all towers, encoded by the identities of the first and the last robot in it:

$$PosTower(F) = \{(i,j) \mid f_{i-1} \neq -1, f_j \neq -1 \text{ and } \forall h, i \leq h < j, f_h = -1\}$$

We then define

$$Pos(F) = PosTower(F) \cup \{(i,i) \mid 1 \leq i \leq k, f_i \neq -1 \text{ and } \ f_{i-1} \neq -1\}$$

that contains all the towers, and the identities of all the other robots (not part of a tower).

We first reorganize the movements of the robots in the towers such that robot moving to the *Front* are the top ones and robots moving to the *Back* are the bottom ones. Given a move $(m_i)_{1\leq i\leq k}$ in $(\Delta \setminus \{Doubt\})^k$, and a tower $(i,j) \in Pos(F)$, let $N^{Front}_{(i,j)}$ be the number of robots with *Front* movement in this tower, defined by

$$N^{Front}_{(i,j)} = |\{\varepsilon_{\ell,Front} \mid i \leq \ell \leq j\}|$$

and let $N^{Back}_{(i,j)}$ be the number of robots that go *Back* in this tower, defined by:

$$N^{Back}_{(i,j)} = |\{\varepsilon_{\ell,Front} \mid i \leq \ell \leq j\}|.$$

The modified movement of robot ℓ denoted by ε'_ℓ is then defined by:

- if ℓ is part of tower $(i,j) \in \mathit{PosTower}(F)$, then $\varepsilon'_\ell = \varepsilon_{\ell,\mathit{Back}}$, if $i \leq \ell \leq \left(N^{\mathit{Back}}_{(i,j)}+i-1\right)$
- if ℓ is part of tower $(i,j) \in \mathit{PosTower}(F)$, then $\varepsilon'_\ell = \varepsilon_{\ell,\mathit{Front}}$ if $\left(N^{\mathit{Back}}_{(i,j)}+i\right) \leq \ell \leq j$.
- For all other robots $\varepsilon'_\ell = \varepsilon_\ell$.

Now, we modify the ε vectors in order to delete pointless moves, corresponding to robots switching positions. Let $(i,j) \in \mathit{Pos}(F)$ be the element of $\mathit{Pos}(F)$ considered and let ε' be the current k-tuple of k-vectors encoding the moves.

- If $f_j \neq 0$, $\varepsilon''_\ell = \varepsilon'_\ell$ (there is no robot in the neighboring node in the clockwise direction).
- Otherwise, let $s \in [1..k]$ such that $(j+1,s) \in \mathit{Pos}(F)$ (note that if $s = j+1$, there is only one single robot on the neighboring node).
  - If $N^{\mathit{Front}}_{(i,j)} \geq N^{\mathit{Back}}_{(j+1,s)}$, then
    * $\varepsilon''_\ell = \varepsilon_{\ell,\mathit{Front}}$ for all $j - N^{\mathit{Front}}_{(i,j)} + N^{\mathit{Back}}_{(j+1,s)} + 1 \leq \ell \leq j$,
    * $\varepsilon''_\ell = \varepsilon_{\ell,\mathit{Idle}}$ for all $\begin{array}{l} j - N^{\mathit{Front}}_{(i,j)} + 1 \leq \ell \leq j - N^{\mathit{Front}}_{(i,j)} + N^{\mathit{Back}}_{(j+1,s)} \\ j+1 \leq \ell \leq j + N^{\mathit{Back}}_{(j+1,s)} \end{array}$
    * and $\varepsilon''_\ell = \varepsilon'_\ell$ for all other ℓ.
  - If $N^{\mathit{Front}}_{(i,j)} < N^{\mathit{Back}}_{(j+1,s)}$, then the modification is symmetrical.

When all the elements of $\mathit{Pos}(F)$ have been visited, we obtain a tuple $(\varepsilon''_\ell)_{1\leq\ell\leq k}$.

Proposition 14. *Given an observation class $[o]_\equiv$ and a tuple $m = (m_i)_{1\leq i\leq k}$ of movements in $(\Delta \setminus \{\mathit{Doubt}\})^k$, the successor $[o \oplus m]_\equiv$ is obtained from $\mathit{rep}([o]_\equiv) = \{F, \tilde{F}\}$ as the class of $o' = \{F', \tilde{F}'\}$ where $F' = F + \sum_{i=1}^k \varepsilon''_i$, and $(\varepsilon''_i)_{1\leq i\leq k}$ has been obtained as described above.*

Proof. Let $o \in \mathit{Obs}$ with $\mathit{rep}([o]_\equiv) = \{F, \tilde{F}\}$, $F = (f_1, \ldots, f_k)$, and let $m = (m_i)_{1\leq i\leq k}$ be a move. For all $1 \leq i \leq k$, if $f_i = 0$, then if the robot i wants to move in the direction of $i+1$, and the robot $i+1$ wants to move in the direction of i, then by our construction, $\varepsilon''_i = \varepsilon_{i,\mathit{Idle}}$ and $\varepsilon''_{i+1} = \varepsilon_{i+1,\mathit{Idle}}$, and the resulting distance will stay 0. For all other decisions of the robots, the distance obtained will be positive. If $f_i = -1$, by the reorganization of the robots on a tower, it is impossible that robot i wants to move in the clockwise direction and that the robot $i+1$ wants to move in the counterclockwise direction. Hence, the distance obtained is never less than -1. In all other cases, the obtained distance is necessarily positive. $\square$

5.3 Results of synthesis

In the case of a system with three robots, there are 6 distinct types of configuration classes:

- The 3-robots tower configuration, which is the configuration to reach, where observations are in the observations class $[\{(-1,-1,n-1),(n-1,-1,-1)\}]_{\equiv}$. From this class of configuration the arena leads to $(\{(-1,-1,n-1),(n-1,-1,-1)\},(a_1,a_1,a_1))$ with $a_1 \in \{\mathit{Idle}, \mathit{Doubt}\}$. However, these edges are of no interest for us since the gathering property is verified.

- The disoriented tower is a configuration where there is an axis of symmetry passing through a two robots tower and the third robot (it occurs only when n is odd). Observation belongs to the classes: $[\{(-1,\frac{n}{2}-1,\frac{n}{2}-1),(\frac{n}{2}-1,\frac{n}{2}-1,-1)\}]_{\equiv}$. In this case, all robots are disoriented and thus the possible actions are: $(a_1,\overline{a_1},a_2)$ with $a_1,a_2 \in \{\mathit{Idle}, \mathit{Doubt}\}$.

- The remaining tower configurations are the ones with observations in the classes $[\{(-1,f_2,f_3),(f_3,f_2,-1)\}]_{\equiv}$, with as representative $\{(-1,f_2,f_3),(f_3,f_2,-1)\}$ with $-1 < f_2 < f_3 \in \mathbb{N}$. The actions are of the form $(a_1,\overline{a_1},a_2\}$ with $a_1,a_2 \in \{\mathit{Back}, \mathit{Front}, \mathit{Idle}\}$.

- The symmetrical configuration is a configuration where its observations are in $[\{(f_1,f_1,f_2),(f_2,f_1,f_1)\}]_{\equiv}$ with $-1 \neq f_1 \neq f_2$ and $-1 \neq f_2$. Note that with 3 robots there is an axis of symmetry that goes through an occupied node. If $f_1 < f_2$, the edges lead to $(\{(f_1,f_1,f_2),(f_2,f_1,f_1)\},(a_1,a_2,\overline{a_1})$ with $a_1 \in \{\mathit{Front}, \mathit{Back}, \mathit{Idle}\}$ and $a_2 \in \{\mathit{Idle}, \mathit{Doubt}\}$, otherwise (when $f_1 > f_2$) edges lead to $(\{(f_2,f_1,f_1),(f_1,f_1,f_2)\},(a_1,\overline{a_1},a_2))$.

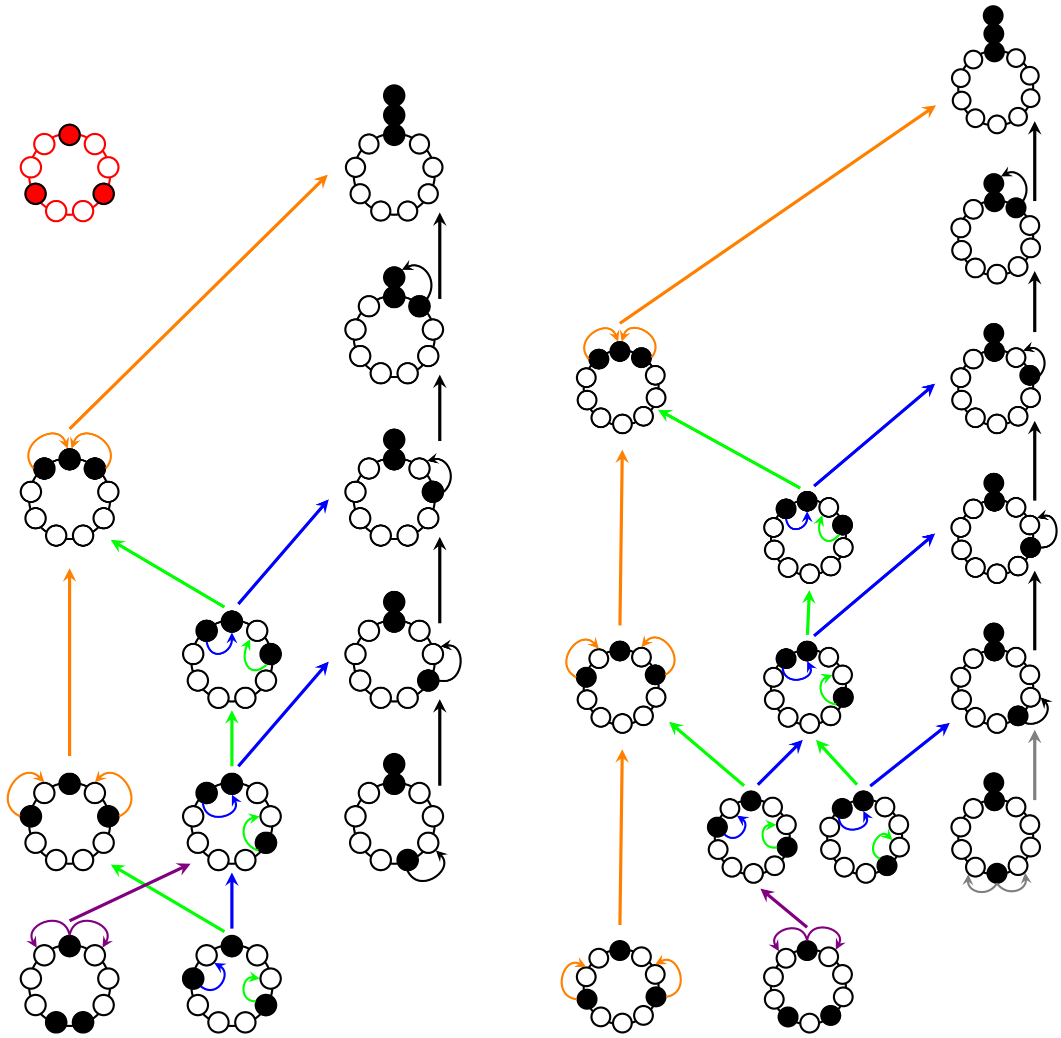
- The rigid configurations are all other configurations which do not fall into any of the above categories, the outgoing edges go to $(\{(f_1,f_2,f_3),(f_3,f_2,f_1)\},(a_1,a_2,a_3))$ with $-1 < f_1 < f_2 < f_3$ and $a_1,a_2,a_3 \in \{\mathit{Front}, \mathit{Back}, \mathit{Idle}\}$.

We implemented the arena for three robots and different ring sizes, in the game-solver tool Uppaal Tiga [Beh+07], and we verified the impossibility of the gathering from periodic configurations. On the other hand, the winning positions for the player are all vertices in V_p corresponding to non periodic configurations.

Moreover, recall that we look for optimal strategies, minimizing the number of robot moves achieving gathering. For this, we enrich the model by adding weights on edges: each edge is weighted by the number of robots that move. We are looking for a strategy on this graph that minimizes the sum of the weight, whatever the adversary does.

5.3.1 A synthesized algorithm

The strategies obtained for 3 robots on rings of size 9 and 10 are depicted in the graphs of Figure 5.3. The red configuration is the configuration from which there is no strategy



a) Strategies for 3 robots on an 9-nodes ring

b) Strategies for 3 robots on a ring of 10 nodes

Figure 5.3: Strategies from Uppaal Tiga

since it is periodical. For each other configuration, the actions chosen by the strategy are depicted by small arrows on the ring, path on the graph show the result of these actions in a synchronous execution model (it represents the adversary actions). Paths and actions are colored in order to distinguish the different types of configurations: The black ones are for the tower configurations without symmetries, grey ones for tower configurations with an axe of symmetry. The orange ones depict the symmetrical configurations: $[\{(f_1, f_1, f_2), (f_2, f_1, f_1)\}]_{\equiv}$ with $-1 < f_1 < f_2$, the symmetrical configurations: $[\{(f_1, f_1, f_2), (f_2, f_1, f_1)\}]_{\equiv}$ with $-1 < f_2 < f_1$ are the violet ones. The blue and green ones depict the two different strategies from rigid configurations.

From the given strategies we extract a pattern and thus a parameterized strategy:

- If 2 robots form a tower the last robot takes the shortest path to the tower:
  - From $[\{(-1, \frac{n}{2} - 1, \frac{n}{2} - 1), (\frac{n}{2} - 1, \frac{n}{2} - 1, -1)\}]_{\equiv}$ the edge relation leads to $(\{(-1, \frac{n}{2} - 1, \frac{n}{2} - 1), (\frac{n}{2} - 1, \frac{n}{2} - 1, -1)\}, (\mathit{Idle}, \mathit{Idle}, \mathit{Doubt}))$.
  - From $[\{(-1, f_1, f_2), (f_2, f_1, -1)\}]_{\equiv}$ when $f_1 \neq \frac{n}{2} - 1$ the edge relation leads to $(\{(-1, f_1, f_2), (f_2, f_1, -1)\}$with $-1 < f_1 < f_2, (\mathit{Idle}, \mathit{Idle}, \mathit{Back}))$.
- If the configuration is symmetrical, in$[\{(f_1, f_1, f_2), (f_2, f_1, f_1)\}]_{\equiv}$ with $-1 < f_1$, $-1 < f_2$, and $f_1 \neq f_2$, the proposed strategy depends on whether $f_1 < f_2$ or $f_2 < f_1$:
  - If $f_1 < f_2$ then the two symmetrical robots get closer to the last robot. The edge relation leads to $((f_1, f_1, f_2), (\mathit{Front}, \mathit{Idle}, \mathit{Back}))$.
  - If $f_1 > f_2$ then the disoriented robot moves. The edge relation leads to $((f_2, f_1, f_1), (\mathit{Idle}, \mathit{Idle}, \mathit{Doubt}))$.
- If the configuration is rigid $([\{(f_1, f_2, f_3), (f_3, f_2, f_1)\}]_{\equiv}$ with $-1 < f_1 < f_2 < f_3)$ there is three possible algorithms:
  - The robot with the minimum view gets closer to its nearest neighbor. In this case the edge relation leads to $(\{(f_1, f_2, f_3), (f_3, f_2, f_1)\}, (\mathit{Front}, \mathit{Idle}, \mathit{Idle}))$.
  - The robot with the maximum view gets closer to its nearest neighbor. In this case the edge relation leads to $(\{(f_1, f_2, f_3), (f_3, f_2, f_1)\}, (\mathit{Idle}, \mathit{Idle}, \mathit{Back}))$.
  - The robot with the minimum view and the robot with the maximum view get closer to their nearest neighbor. In this case the edge relation leads to $(\{(f_1, f_2, f_3), (f_3, f_2, f_1)\}$, $(\mathit{Front}, \mathit{Idle}, \mathit{Back}))$. This strategy is the two above strategies made simultaneously.

From Theorem 13, one can translate the winning strategies for each configuration into a distributed algorithm. We present the possible strategies in Table 5.1. For robot views not present in the Table, the robot movement is *Idle*. This algorithm is correct by construction for various values of n ($3 \leq n \leq 15$, $n = 100$).

3-gathering algorithm:					
Rule: :	Condition	$\wedge$	$view(r,c)$	$\rightarrow$	Move
$RS1$: :			$(R_1, F_{\frac{n}{2}-1}, T_2, F_{\frac{n}{2}-1})$	$\rightarrow$	$r.Doubt$
$RS2$: :	$f_1 \neq f_2$	$\wedge$	$(R_1, F_{f_1}, T_2, F_{f_2})$	$\rightarrow$	$r.Doubt$
$RS3$: :	$-1 < f_1 < f_2$	$\wedge$	$(R_1, F_{f_1}, R_1, F_{f_1}, R_1, F_{f_2})$	$\rightarrow$	$r.Front$
$RS4$: :	$-1 < f_2 < f_1$	$\wedge$	$(R_1, F_{f_1}, R_1, F_{f_2}, R_1, F_{f_1})$	$\rightarrow$	$r.Doubt$
$RS5_1$: :	$-1 < f_1 < f_2 < f_3$	$\wedge$	$(R_1, F_{f_1}, R_1, F_{f_2}, R_1, F_{f_3})$	$\rightarrow$	$r.Front$
$RS5_2$: :	$-1 < f_1 < f_2 < f_3$	$\wedge$	$(R_1, F_{f_2}, R_1, F_{f_1}, R_1, F_{f_3})$	$\rightarrow$	$r.Front$

Table 5.1: Rules of the synthesized algorithm for a robot r

5.3.2 Proof of the 3-gathering algorithm

Let $C = [\{(x,y,z),(z,y,x)\}]_\equiv$ a class of configurations such that $rep(C) = (\{x,y,z),(z,y,x)\})$ when $x \leq y \leq z$, and $d = x + y$ be the *separating-distance* of C.

Theorem 15. *Starting from any configuration (except periodic ones) the 3-gathering algorithm eventually reaches a gathering configuration.*

Proof. The theorem is correct if any of the movements produced by the above strategy is decreasing the separating-distance. The proof directly follows from the Lemmas 16, 17, 18, 19 below. Each one of these lemmas addresses a possible initial class of configurations. □

Lemma 16. *Starting from a class of configuration $[(-1, d_2, d_3)]_\equiv$, which is the class of configuration where there is one tower of size 2, the system executing the 3-gathering algorithm eventually reaches a gathering configuration.*

Proof.

Base-case: Consider the minimal separating distance. For any ring size $n \in \mathbb{N}$, the execution starting from $[\{(-1,0,n-2),(n-2,0,-1)\}]_\equiv$ leads to $[\{(-1,-1,n-1),(n-1,-1,-1)\}]_\equiv$ which is the gathering class of configuration.

Induction: Assume that starting in a configuration where the separating-distance is $i \in \mathbb{N}$, the system executing the strategy eventually reaches a gathering class of configuration. We prove in the following that the above also holds when the *separating-distance* is $i+1$:
From $[\{(-1,i+2,d_3),(d_3,i+2,-1)\}]_\equiv$ the execution of the strategy leads to $[\{(-1,i+1,d_3+1),(d_3+1,i+1,-1)\}]_\equiv$. The proof follows from the induction hypothesis.

□

Lemma 17. *Starting from a rigid configuration $[\{(d_1,d_2,d_3),(d_3,d_2,d_1)\}]_\equiv$, the system executing the* 3-gathering *algorithm eventually reaches a gathering configuration.*

From a rigid configuration $[\{(d_1, d_2, d_3), (d_3, d_2, d_1)\}]_\equiv$ where the representative is $\{(d_1, d_2, d_3), (d_3, d_2, d_1)\}$, with $-1 < d_1 < d_2 < d_3$, the algorithm reaches a gathering configuration. In order to prove this lemma for the last edge we fix $d_1 = 0$ and prove that for any d_2 and any d_3 the strategy is correct, and then we use this proof as a base case to prove the lemma for any d_1, d_2 and d_3.

Proof.

Base-case2: When $d_1 = 0$, for any d_2, d_3, from $[\{(0, d_2, d_3), (d_3, d_2, 0)\}]_\equiv$ the strategy leads to $[\{(-1, d_2 - 1, d_3 + 2), (d_3 + 2, d_2 - 1, -1)\}]_\equiv$ where the *separating-distance* is decreased by two. Hence, the lemma is true, thanks to lemma 16.

Induction2: We assume that the gathering is made for any d_2, d_3 for an $d_1 = i$, and we show that it is also made when $d_1 = i + 1$. From $[\{(i + 1, d_2, d_3), (d_3, d_2, i + 1)\}]_\equiv$ the algorithm leads to $[\{(i - 1, d_2 - 1, d_3 + 2), (d_3 + 2, d_2 - 1, i - 1)\}]_\equiv$. Thanks to our induction hypothesis the lemma is true.

□

Lemma 18. *Starting from a symmetrical class of configurations* ($[\{(d_1, d_1, d_2), (d_2, d_1, d_1)\}]_\equiv$) *without tower where* $rep([\{(d_1, d_1, d_2), (d_2, d_1, d_1)\}]_\equiv) = \{(d_1, d_1, d_2), (d_2, d_1, d_1)\}$, *with* $-1 < d_1$, $-1 < d_2$, *and* $d_1 \neq d_2$. *The 3-gathering algorithm eventually reaches a gathering configuration.*

Proof.

Base-case: Consider the class of configurations where the *separating-distance* is minimal: $[\{(0, 0, n - 3), (n - 3, 0, 0)\}]_\equiv$. For any ring size $n \in \mathbb{N}$, the system executing the 3-gathering algorithm starting in $[\{(0, 0, n - 3), (n - 3, 0, 0)\}]_\equiv$ reaches $[\{(-1, -1, n - 1), (n - 1, -1, -1)\}]_\equiv$ (the gathering class of configurations).

Induction: Assume that when the separating-distance equals $2i, i \in \mathbb{N}$ a gathering configuration is eventually reached, and prove that it is also true when the separating distance is $2i + 2$. From $[\{(i + 1, i + 1, d_3), (d_3, i + 1, i + 1)\}]_\equiv$ the system executing the strategy eventually reaches $[\{(i, i, d_3+2), (d_3+2, i, i)\}]_\equiv$. The lemma follows from the induction hypothesis.

□

Lemma 19. *Starting from a symmetrical class of configurations* $[\{(d_1, d_1, d_2), (d_2, d_1, d_1)\}]_\equiv$ *without tower where* $rep([\{(d_1, d_1, d_2), (d_2, d_1, d_1)\}]_\equiv) = \{(d_2, d_1, d_1), (d_2, d_1, d_1)\}$, *with* $-1 < d_1$, $-1 < d_2$, *and* $d_1 \neq d_2$. *The 3-gathering algorithm eventually reaches a gathering configuration.*

Proof. First observe that $[\{(d_1, d_1, d_2), (d_2, d_1, d_1)\}]_\equiv$ leads to $[\{(d_1-1, d_1+1, d_2), (d_2, d_1+1, d_1 - 1)\}]_\equiv$. The obtained configuration is a symmetric configuration if $d_2 = d_1 - 1$ or a rigid configuration otherwise. Thanks to Lemma 18 and Lemma 17, we know that from these configurations, a gathering configuration is eventually reached. □

We proposed a formal method based on reachability games that permits to automatically generate distributed algorithms for mobile autonomous robots solving a global task. The task of gathering on a ring-shaped network was used as a case study. We hereby discuss current limitations and future works.

While our construction generates algorithms for a particular number of robots k and ring size n, the game encoding we propose enables to easily tackle the gathering problem for any given k and n, provided as inputs, since k and n are parameters of the arena. Also, we focused on the atomic Fsync and Ssync models. Breaking the atomicity of Look-Compute-Move cycles (that is, considering automatic algorithm production for the Async model) implies that robots cannot maintain a current global view of the system (their own view may be outdated), nor be aware of the view of other robots (that may be outdated as well). Then, our two-players game encoding is not feasible anymore. A natural approach would be to use distributed games, but they are generally undecidable as previously stated. So, a completely new approach is required for the automatic generation of non-atomic mobile robot algorithms.

Chapter 6

Synthesis of asynchronous mobile robot protocols

In this chapter we consider the asynchronous model. We first show how finding an algorithm for gathering asynchronous robots can be seen as a two players game with partial information. In order to fight the combinatorial explosion due to the asynchronous model we propose a recursive algorithm that permits to obtain a gathering protocol in the asynchronous model thanks to synchronous synthesis and model-checking.

Recall that in the asynchronous model, some robots can be in a computation state while others are in a moving state, leading to the possibility for a robot to move according to an obsolete observation. Therefore, the game must be modified from the synchronous case: We show in the sequel how this new setting corresponds to a *partial observation game*.

6.1 Partial observation games

Games can be classified according to the knowledge of the players about the game: In a partial observation game, the set of locations is partitioned into sets called observations. A player cannot see the current location of the game, but only its observation. For example, in the asynchronous model, a robot cannot see the states of other robots. There are three types of games according to observations: complete-observation games, where both players have complete knowledge of the game (as was the case of the game in the previous section), partial-observation games, where each player only has a partial view about the state and the moves of the other player, and one-sided partial-observation games, where one player has partial knowledge and the other one has complete knowledge of the game.

Two-player games with partial observations are considerably more complicated than games with complete observations. Decision problems for partial observation games usually lie in higher complexity classes than complete-observation games [Rei84].

In this section we recall classical notions (from [DR]). A partial observation game is composed of an *arena* and *winning conditions*.

An arena for a game with partial observation is a graph $\mathcal{A} = (V, E, O)$ in which the set of vertices $V = V_p \uplus V_a$ is partitioned into V_p, the player locations, and V_a the adversary locations. The set of edges is $E \subseteq V \times V$, and $O \subseteq 2^V$ is a finite set of observations that partitions the set V of vertices. It induces a mapping $\mathcal{O} : V \to O$ that associates with each vertex its unique observation. For each play $\pi = v_0 v_1 \ldots$ in V^∞, we denote by $\mathcal{O}(\pi)$ the sequence $\mathcal{O}(v_0)\mathcal{O}(v_1)\ldots$, we analogously extend observations to histories, sets of plays, etc.

A game is played in the same way as in the perfect information case, but now only the observation of the current location is revealed to the player who owns the location. The effect of uncertainty about the history of the play is formally captured by the notion of observation-based strategy. An observation-based strategy for the player is a function $\sigma^{\mathcal{O}} : V^* \cdot V_p \to V$ such that $\sigma^{\mathcal{O}}(\pi) = \sigma^{\mathcal{O}}(\pi')$ for all histories π, π' in $V^* \cdot V_p$ with $\mathcal{O}(\pi) = \mathcal{O}(\pi')$.

Example 11. *In the game with partial observation depicted in Figure 6.1 player vertices are represented by rectangles and adversary positions by circles. The observations of the player are $O1 = \{P1, P2\}$, $O2 = \{P3, P4\}$. The transitions are shown as labeled edges, and the initial state is $A1$. The objective of the player is Reach(T). This game is not winning for the player. If the strategy is to take the action a in $O1$ it is only winning in $P2$, and if the strategy is to take the action b in $O1$, it is only winning in $P1$.*

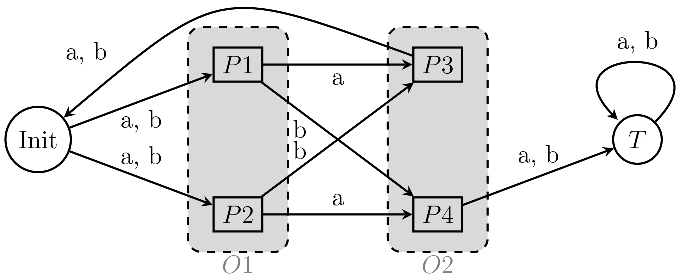


Figure 6.1: A partial observation game with a reachability objective

6.2 The arena for asynchronous robots

We now explain how a gathering algorithm for the robots in the asynchronous model can be obtained as a winning strategy in a one-sided partial observation game. Like before, the adversary is the scheduler that can dynamically decide robot chirality at every activation, with complete observation. In order to take into account asynchronous moves of robots in the scheduling, the vertices of the associated arena must now contain robot states and planned move of robots that are not yet scheduled (and can thus become obsolete). On the other hand, the player is the algorithm deciding robot moves in every state. Since robot execution of a rule is only based on its view, the player has a partial observation of the location.

Robot states are described by tuples in $S = \{\mathit{Front}, \mathit{Back}, \mathit{Idle}, \mathit{RLC}\}^k$, where RLC denotes the "Ready to Look-Compute" state (see Figure 2.7). For the implementation, all robot movements must be given according to a common sense of direction. We consider the arena $\mathcal{A} = (V_p \uplus V_a, E, O)$, where:

- The set of player vertices is $V_p = (Obs/\!\!\equiv \times S)$;
- The set of adversary vertices is $V_a = Obs \times S \times \Delta^k$;
- As in the synchronous arena the edge relation ensures a strict alternation between the two players: $E \subseteq (V_p \times V_a) \cup (V_a \times V_p)$;
- The observation simply abstracts away the robot states, hence $O = Obs/\!\!\equiv$ and the mapping $\mathcal{O}$ is only defined for player vertices by $\mathcal{O}(v) = obs$ for $v = (obs, s) \in V_p$.

These elements are detailed in the sequel.

6.2.1 Edges from V_p to V_a

A player vertex $v = (obs, s) \in V_p$ is built as follows. We consider the canonical configuration c_o associated with $o = rep(obs)$, defined by $c_o(r_1) = 0$, $c_o(r_2) = f_1 + 1, \ldots, c_o(r_k) = \Sigma_{i=1}^{k} f_i +_n k$, where robot r_i is at distance f_i of robot $r_{i+_k 1}$, and $s = (s_1, \ldots, s_k)$ where s_i is the state of r_i.

The player chooses robot movements by applying some decision function related to the views of the equivalence class $\mathcal{O}(v) = obs$, similarly to the synchronous case: There is an edge (v, v') in E with $v' = \big(o, s, (a_1, \ldots, a_k)\big)$ if and only if there exists a decision function ∂ such that $a_i = \partial(view(r_i, c_o))$ for all i in $[1..k]$.

The play continues on an adversary position memorizing the different movements decided for each robot.

6.2.2 Edges from V_a to V_p

From a position $v = \big(o, s, (a_1, \ldots, a_k)\big) \in V_a$, the adversary chooses a set $Sched \subseteq Rob$ of robots, and schedules them to act according to their current state s. When disoriented robots are scheduled for their LC actions the adversary also chooses the direction in which robots will move, producing the next player vertex of the form $(obs', s') \in V_p$.

To describe the edge more precisely, given a set $Sched$ and $v \in V_a$, we define the corresponding $(v, Sched)$-move and the $(v, Sched)$-consistent states, in the vein of v-consistent movements of Definition 19 in Chapter 5.

The $(v, Sched)$-move m is the unique movement associated with v when subset $Sched$ of robots is scheduled.

Definition 20. *For a state $v = \big(o, s, (a_1, \ldots, a_k)\big) \in V_a$ and a set of robots $Sched \subseteq Rob$, the $(v, Sched)$-move $m = (m_1, \ldots, m_k) \in (\Delta \cup \{-\} \setminus \{Doubt\})^k$ is defined for all $1 \leq i \leq k$ by:*

$$m_i = \begin{cases} s_i & \text{if } r_i \in Sched \text{ and } s_i \neq RLC \\ - & \text{otherwise} \end{cases}$$

The $-$ represents an absence of movement for a robot, corresponding to the robot being either not scheduled or in its *RLC* state (see Definition 3). Note that movements in m have been determined on some previous adversary position.

A (v, Sched)-consistent state s' represents the scheduler choices in term of direction for disoriented robots, resolving *Doubt* moves. When a robot is not scheduled its state stays the same. If the robot is scheduled, its new state depends on its current state: if the robot state is in either *Front* or *Back* then it actually performs the move (we have $m_i = s_i$) and its state becomes *RLC*. If the robot state is *RLC* then its new state is a_i if $a_i \neq \mathit{Doubt}$, and otherwise the scheduler chooses among *Front* or *Back*.

Definition 21. *For a vertex $v = \big(o, s, (a_1, \ldots, a_k)\big) \in V_a$ and a set $\mathit{Sched} \subseteq \mathit{Rob}$ of robots, a (v, Sched)-consistent state $s' = (s'_1, \ldots, s'_k) \in S$ is defined for all $1 \leq i \leq k$ by:*

- *if $r_i \notin \mathit{Sched}$, then $s'_i = s_i$,*
- *if $r_i \in \mathit{Sched}$ and $s_i \neq \mathit{RLC}$, then $s'_i = \mathit{RLC}$,*
- *if $r_i \in \mathit{Sched}$, $s_i = \mathit{RLC}$ and $a_i \neq \mathit{Doubt}$, then there are two cases,*
  - *If $\mathit{view}^{\min\_r_i} \circlearrowright^m \mathit{view}^{\min\_r_0}$ for some m then $s'_i = a_i$.*
  - *Otherwise $s'_i = \overline{a_i}$.*

  (We choose the direction sense of $\mathit{view}^{\min\_r_0}$ as the common sense of direction.)
- *otherwise $s'_i \in \{\mathit{Front}, \mathit{Back}\}$.*

Note that the last case corresponds to $r_i \in \mathit{Sched}$, $s_i = \mathit{RLC}$ and $a_i = \mathit{Doubt}$.

The effect on an observation, of any movement m_i of robot r_i can be described by a mapping $\varepsilon : \{1, \ldots, k\} \times \{\mathit{Front}, \mathit{Back}, \mathit{Idle}\} \to \{-1, 0, 1\}^k$. This mapping denotes the translated moves that permit to apply real movements on the observation and is defined by:

- $\varepsilon(i, \mathit{Back}) = (\varepsilon_{i,h})_{1 \leq h \leq k}$ with:
  $\varepsilon_{i,i-1} = -1$, $\varepsilon_{i,i} = 1$, and $\varepsilon_{i,h} = 0$ for $h \notin \{i-1, i\}$,
- $\varepsilon(i, \mathit{Front}) = (\varepsilon_{i,h})_{1 \leq h \leq k}$ with:
  $\varepsilon_{i,i} = -1$, $\varepsilon_{i,i-1} = 1$, and $\varepsilon_{i,h} = 0$ for $h \notin \{i-1, i\}$,
- $\varepsilon(i, \mathit{Idle}) = 0^k$.

Similarly as in the synchronous case, the idea is to add (in an element-by-element fashion) the current observation to all the vectors representing the movements of the robots to obtain the next configuration. However, when the movements of two adjacent robots imply that they switch their positions in the ring, some absurd values (-2 or -3) may appear in the obtained configuration, if the sum is naively performed, so a careful treatment of these particular cases must be done. To obtain the correct configuration, one should recall that robots are anonymous, hence if two robots switch their positions, it has the same effect as if none of them has moved. Also, if in a tower, some robots want to move *Front*, and the others want to move *Back*, the exact robots that will move are of no importance: only the number of robots that move in each direction is important. We will then reorganize the movements between the robots, in order to keep correct values in our configurations:

- in a tower, we assume that the robots that move *Back* always are the bottom ones, and robots that move *Front* are the top ones,
- when a robot moves *Front* and joins a tower, we assume that it is placed at the bottom of the tower,
- and when it moves *Back* and joins a tower, it is placed at the top of the tower.

These conventions ensure that when adding the configuration and the different movements, we obtain correct values.

We define $PosTower(F)$, a set that contains all towers, encoded by the identities of the first and the last robot in it:

$$PosTower(F) = \{(i,j) \mid f_{i-1} \neq -1, f_j \neq -1 \text{ and } \forall h, i \leq h < j, f_h = -1\}$$

We then define

$$Pos(F) = PosTower(F) \cup \{(i,i) \mid 1 \leq i \leq k, f_i \neq -1 \text{ and } \ f_{i-1} \neq -1\}$$

that contains all the towers, and the identities of all the other robots (not part of a tower).

We first reorganize the movements of the robots in the towers such that robot moving to the *Front* are the top ones and robots moving to the *Back* are the bottom ones. Given a move $(m_i)_{1\leq i\leq k}$ in $(\Delta \setminus \{Doubt\})^k$, and a tower $(i,j) \in Pos(F)$, let $N^{Front}_{(i,j)}$ be the number of robots with *Front* movement in this tower, defined by

$$N^{Front}_{(i,j)} = |\{\varepsilon_{\ell,Front} \mid i \leq \ell \leq j\}|$$

and let $N^{Back}_{(i,j)}$ be the number of robots that go *Back* in this tower, defined by:

$$N^{Back}_{(i,j)} = |\{\varepsilon_{\ell,Front} \mid i \leq \ell \leq j\}|.$$

The modified movement of robot ℓ denoted by ε'_ℓ is then defined by:

- if ℓ is part of tower $(i,j) \in PosTower(F)$, then $\varepsilon'_\ell = \varepsilon_{\ell,Back}$, if $i \leq \ell \leq \left(N^{Back}_{(i,j)}+i-1\right)$
- if ℓ is part of tower $(i,j) \in PosTower(F)$, then $\varepsilon'_\ell = \varepsilon_{\ell,Front}$ if $\left(N^{Back}_{(i,j)}+i\right) \leq \ell \leq j$.
- For all other robots $\varepsilon'_\ell = \varepsilon_\ell$.

Now, we modify the ε vectors in order to delete pointless moves, corresponding to robots switching positions. Let $(i,j) \in Pos(F)$ be the element of $Pos(F)$ considered and let ε' be the current k-tuple of k-vectors encoding the moves.

- If $f_j \neq 0$, $\varepsilon''_\ell = \varepsilon'_\ell$ (there is no robot in the neighboring node in the clockwise direction).
- Otherwise, let $s \in [1..k]$ such that $(j+1, s) \in Pos(F)$ (note that if $s = j+1$, there is only one single robot on the neighboring node).
  - If $N^{Front}_{(i,j)} \geq N^{Back}_{(j+1,s)}$, then
    - $\varepsilon''_\ell = \varepsilon_{\ell,Front}$ for all $j - N^{Front}_{(i,j)} + N^{Back}_{(j+1,s)} + 1 \leq \ell \leq j$,
    - $\varepsilon''_\ell = \varepsilon_{\ell,Idle}$ for all $\begin{array}{l} j - N^{Front}_{(i,j)} + 1 \leq \ell \leq j - N^{Front}_{(i,j)} + N^{Back}_{(j+1,s)} \\ j+1 \leq \ell \leq j + N^{Back}_{(j+1,s)} \end{array}$
    - and $\varepsilon''_\ell = \varepsilon'_\ell$ for all other ℓ.
  - If $N^{Front}_{(i,j)} < N^{Back}_{(j+1,s)}$, then the modification is symmetrical.

When all the elements of $Pos(F)$ have been visited, we obtain a tuple $(\varepsilon''_\ell)_{1\leq \ell \leq k}$.

From an adversary position $v = \big(o, s, (a_1, \ldots, a_k)\big) \in V_a$, once a subset *Sched* of robots is chosen and disoriented robots are given an orientation, a new observation o' is obtained by applying m on $o = \{F, \tilde{F}\}$, such that $o' = \{F', \tilde{F}'\}$ where $F' = F + \sum_{i=1}^{k} \varepsilon''_i$, and $(\varepsilon''_i)_{1\leq i \leq k}$ has been obtained as described above. We note $o' = o \oplus m$ the effect of m on o.

To formally define the edge relation from an adversary position to a player position, we also need to introduce a normalizing operation. This normalization maintains a standard form in player locations of the form $v = (obs, s) \in V_p$, such that the the robot states m are coherent with obs *i.e.*, if c_o is the canonical configuration associated with $o = rep(obs)$, defined by $c_o(r_1) = 0$, $c_o(r_2) = f_1 + 1, \ldots, c_o(r_k) = \Sigma_{i=1}^{k} f_i +_n k$, where the robot r_i is at distance f_i of robot $r_{i+_k 1}$, then $s = (s_1, \ldots, s_k)$ where s_i is the state of r_i.

To normalize s' according to the observation o', let c' be the canonical configuration associated with o', and let $\hat{c}$ be the the canonical configuration associated with $rep([o']_\equiv)$. From Proposition 2, we have $\hat{c} \approx c'$. Then, according to definition 11, we can define the tuple $\hat{s}'$ associated with $\hat{c}$ by:

- If $\hat{c} = \pi^h \circ \overline{\pi} \circ c' \circ \beta$ for some h and some robot permutation β, then for all $1 \leq i \leq k$, $\hat{s}'_i = \overline{s_j}$ where $r_j = \beta(r_i)$,
- If $\hat{c} = \pi^h \circ c' \circ \beta$ or $\hat{c} = \pi^h \circ \overline{\pi} \circ c' \circ \beta$ for some h and some robot permutation β then for all $1 \leq i \leq k$, $\hat{s}'_i = s_j$ where $r_j = \beta(r_i)$.

Example 12. *This operation is illustrated in Figure 6.2. Let a be the canonical configuration associated with observation*

$$o = \{(-1,-1,0,1,-1,0,-1,1,2,1,0),(0,1,2,1,-1,0,-1,1,0,-1,-1)\}.$$

Let $v_a = (o, s, \{Back\}^k) \in V_a$ *be the adversary location where all robots want to move Back and* $s_1 = Back$*,* $s_2 = s_3 = s_4 = Front$*,* $s_5 = s_6 = Front$*,* $s_7 = Back$*,* $s_8 = Idle$*,* $s_9 = Back$*,* $s_{10} = Idle$*, and* $s_{11} = RLC$*.*

If all robots except r_8 are scheduled, $Sched = Rob \setminus \{r_8\}$, and if m is the $(v_a, Sched)$-move, the resulting configuration is b with robot states in s', where all robots are in RLC except for r_8 and r_{11} that are in states Idle and Back respectively. The canonical configuration of $rep([o \oplus m]_{\equiv})$ is $c = \pi^{-4} \circ \overline{\pi} \circ b \circ \beta$, where β is the robot permutation defined by $\beta(r_1) = r_6$, $\beta(r_2) = r_7$, $\beta(r_3) = r_8$, $\beta(r_4) = r_5$, $\beta(r_5) = r_4$, $\beta(r_6) = r_2$, $\beta(r_7) = r_3$, $\beta(r_8) = r_1$, $\beta(r_9) = r_{11}$, $\beta(r_{10}) = r_{10}$, $\beta(r_{11}) = r_9$. The tuple of robot states $\hat{s}'$ associated with c is such that all robot states are in RLC except for $\hat{s}'_3 =$ Idle and $\hat{s}'_9 =$ Front, since $\beta(r_3) = r_8$ and $\beta(r_9) = r_{11}$.

Then the player location resulting from move m is $([o \oplus m]_{\equiv}, \hat{s}')$.

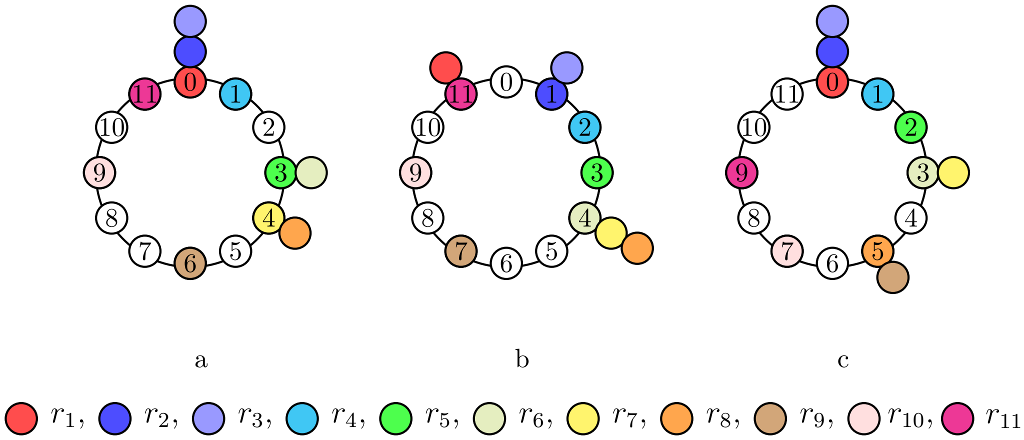


Figure 6.2: Example of a V_a to V_p transition from a to b, normalized in c.

Given a adversary vertex $v = \big(o, s, (a_1, \ldots, a_k)\big) \in V_a$ and a player vertex $v' \in V_p$. The edge (v, v') belongs to E if and only if there exists a non empty subset *Sched* of *Rob*, a $(v, Sched)$-consistent state s' and a $(v, Sched)$-move m such that $v' = \big([o \oplus m]_{\equiv}, \hat{s}'\big)$ where $\hat{s}'$ has been obtained by normalizing s'.

6.3 An algorithm for asynchronous synthesis

In order to fight the combinatorial explosion due to the asynchronous model we propose a method to obtain a gathering protocol in the asynchronous model combining synchronous synthesis and model-checking.

We know that all executions in the synchronous model are also executions of the asynchronous models, then if a protocol is correct under the asynchronous execution model it is also correct under the synchronous model. Conversely, if a protocol is not correct under the synchronous model it cannot be correct under the asynchronous one. Thus, we use synchronous algorithm synthesis coupled with model-checking on the resulting strategy of the synthesis in an asynchronous execution model. If the strategy is correct then we have the desired protocol otherwise we search for a distinct synchronous strategy.

The Algorithm AsyncSynth takes as input the arena for ring size n and k robots. It constructs a tree of all synchronous strategies for the gathering and tests each one by model-checking in the asynchronous setting. If an asynchronous strategy achieves the gathering then the tree construction is stopped.

Each node of the tree is labeled by a strategy $\partial = (\partial_1, \partial_2, \ldots, \partial_{|O|})$ where for each i, ∂_i is the set of actions associated with the i^{th} observation class, one action for each robot view. The root of the tree is the strategy ∂_{init} resulting from the call $SS(\emptyset)$, where no rule is forbidden. A node of the tree is labeled by a strategy $\partial = (\partial_1, \partial_2, \ldots, \partial_{|O|})$ resulting from some call $SS(L)$, where L is the list of rules forbidden in ∂. This node has as many children as the number of observation classes, where the label of the i^{th} child is the strategy denoted by ∂^i, resulting from the call $SS(L \cup \{\partial_i\})$.

Algorithm AsyncSynth:

```
Function SS(CrList): /* call to synchronous synthesis:  asking
for a strategy that does not contain any rule of the CrList
list                                                          */
    Result: A strategy as a rule list, if there is none it returns the empty
            List ∅
Function MC(CrList): /* call to model checking on the algorithm
composed of the rule list CrList in asynchronous model        */
    Result: true or false
Function AsyncSynth(CrList L): /* The recursive function that
calls synchronous synthesis and asynchronous model-checking   */
    Data: CrList CurrentProc
    Result: An algorithm as a List of Rules or ∅
    CurrentProc = SS(L)
    if isEmpty(CurrentProc) then
        return ∅
    else
        if MC(CurrentProc) then
            return CurrentProc
        else
            for i = 0 to CurrentProc.size()-1 do
                CrList L' = L + CurrentProc[i]
                CrList Proc = AsyncSynth(L')
                if not isEmpty(Proc) then
                    return Proc
        return ∅

Algorithm AsyncSynth()
    Data: Int n, k: the size of the ring and the number of robots
    Result: A correct by construction algorithm as a List of Rules or ∅
    return AsyncSynth(∅)
```

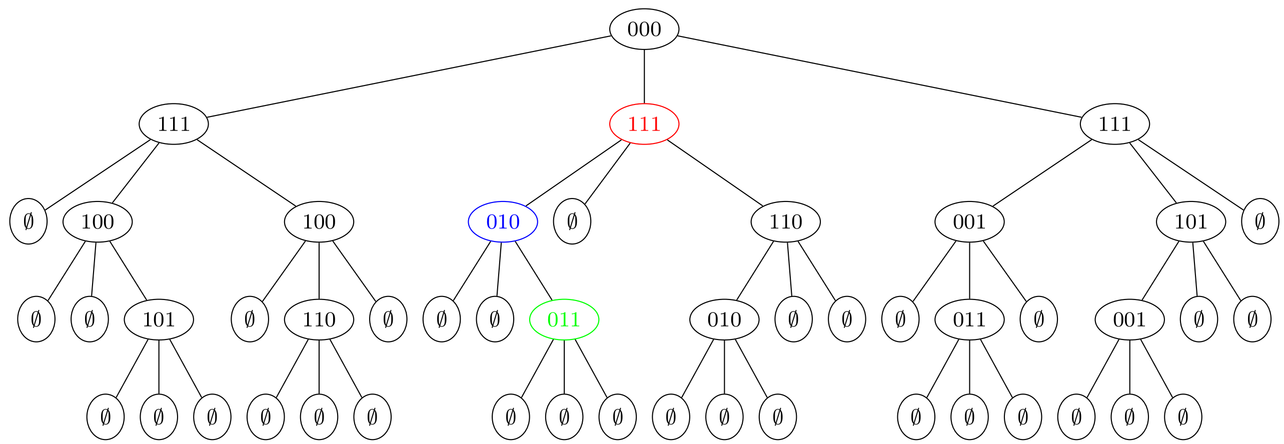


Figure 6.3: Example of a tree constructed by AsyncSynth

Example 13. *The Algorithm AsyncSynth builds a tree of all possible synchronous strategies until it finds one that also achieve the gathering in the asynchronous execution model. Figure 6.3 represents one of the possible trees constructed by the algorithm (depending on the synthesis function) for 2 robots in a ring where there are only three observation classes and two possible strategies (in* $\{0,1\}$*) in each configuration class. In this example, we assume that every strategy achieves the gathering in a synchronous setting but not in an asynchronous one, which permits to build the complete tree of synchronous strategies. A strategy is written as a word* $\partial_1\partial_2\partial_3$*, where* ∂_i *is a binary digit representing the strategy in the* i^{th} *class of observations. The root of the tree corresponds to the strategy returned by the function* $SS(\emptyset)$*. The red, blue and green nodes correspond respectively to the strategies given by* $SS(\sigma_2 = 1)$, $SS(\sigma_2 = 1, \sigma_1 = 0)$ *and* $SS(\sigma_2 = 1, \sigma_1 = 0, \sigma_3 = 0)$.

If there is no synchronous strategy that performs the gathering in the asynchronous model, then it means that no protocol exists for this model. Moreover, if there is a gathering protocol our algorithm will find it.

Lemma 20. *The Algorithm AsyncSynth terminates.*

Proof. Since the size of the ring and the number of robots are known and finite, the number of configuration classes is known and finite (note that this would also be the case for any other finite graph). The number of strategies is then bounded by $|\Delta|^{k|O|}$. At each step the algorithm increments the number of different forbidden rules, and thus decrement the number of strategies. If no asynchronous protocol is found the number of forbidden rules for an observation class will be equal to the number of possible strategies for this class, hence there is no more synchronous strategy and thus the algorithm terminates. □

Lemma 21. *The Algorithm AsyncSynth is complete.*

Proof. To show that the algorithm is complete, we proceed by contradiction. Assume that the algorithm cannot find an existing asynchronous gathering protocol. This means that every leaf of the tree is $\emptyset$, hence at least one rule of the protocol is forbidden in each leaf, thus these rules were present in the leaf's ancestors. Assume that the algorithm is currently building node x and that no rule of the protocol is forbidden in this step, but none of its subtrees contains the protocol. If none of its subtrees contains the protocol, it means that at least one rule of the protocol is forbidden in every child. If a rule is forbidden in a child and not in its parent it means that the parent strategy contains this rule. Then the node x must contain the protocol, hence we have a contradiction. □

Theorem 22. *There is a gathering protocol if and only if Algorithm AsyncSynth returns a non empty list.*

Proof. The algorithm is correct since it terminates (Lemma 20), it is correct because any protocol produced by the synthesis (Theorem 13) is tested by model checking and hence is a solution to asynchronous gathering, and it is complete (Lemma 21). □

CHAPTER 7

Conclusion and perspectives

In this thesis, we demonstrate the feasibility of applying formal methods to mobile robot protocols in a discrete space. We propose a formal model which represents the robots as automata communicating via shared variables. This model is general enough to handle the various settings, from synchronous to asynchronous execution models, and to express all the particularities of mobile robot algorithms. We then provide some results for the distributed algorithms community, applying both model checking and synthesis techniques on this model.

Summary

The techniques of model checking and synthesis are known to suffer from combinatorial explosion, due to the large size of the models of the system and the properties to be verified. Previous formal works on robot protocols were only able to handle synchronous robots, in the Fsync model, where the set of executions is smaller than in the asynchronous one (Async). Moreover, even in this synchronous model, they only solve very small instances, for instance 3 robots exploring 3×3 grids for the model checking, or 5 robots perpetually exploring a 10 nodes ring for the synthesis.

Using equivalence classes with respect to symmetries, we show that it is possible to reduce the size of the model, allowing us to deal with larger instances in the asynchronous execution model.

Our results demonstrate the interest of model-checking tools, that use concise data and parallel algorithms, permitting to verify protocols for which hand made proofs were painful and sometimes erroneous. Indeed, we exhibited a counter example for an exploration protocol, leading to a better understanding of the system and a corrected version of the algorithm. Our work also differs from previous approaches, since we verify some complex properties, assuming fairness, and not only invariant properties. We validated our approach by two case studies in the asynchronous model:

- The verification of Flocchini algorithm for exploration with stop. We show that for many instances of k and n not covered in the original paper, the algorithm is still correct.
- The verification of the Min algorithm for exclusive perpetual exploration. The DiVinE tool produces a counter-example in the asynchronous setting, where two robots collide. We correct the original protocol and verify the new one via model checking for several instances of the ring size.

For this last problem, we also provide a correctness proof for any ring size with an inductive approach.

Going one step further, we investigate the automatic synthesis of robot protocols. Concerning synthesis in the Fsync model, an encoding of the gathering problem as a reachability game allows us to automatically generate an optimal distributed algorithm for three robots evolving on a fixed size ring. Our optimality criterion refers to the number of robot moves necessary to actually achieve gathering. The previous attempt to synthesize robot algorithms was not fully automatic: The method was a construction of all possible algorithms, followed by manual verification performed on each of these algorithms. Moreover, this solution only handles rigid configurations and does not take into account symmetric configurations or those containing a tower. In contrast, our synthesis technique handles all configurations, and permits to highlight initial configurations from which the gathering is not solvable. Once these configurations are extracted from the set of initial configurations, our technique automatically generates an algorithm. Thus, this approach also permits to obtain impossibility results as side results. Indeed when there exists some periodic initial configurations, the tool exhibits these configurations as not being part of the winning set.

In the Async model, we show that synthesis can be seen as a two players game with partial information. In order to fight the combinatorial explosion due to the asynchronous model we propose a recursive algorithm producing a gathering protocol in the asynchronous model thanks to synchronous synthesis and model checking. This approach is the first one that deals with synthesis of an algorithm for mobile robots in the asynchronous model.

These contributions lead to the following set of reviewed publications:

International journals

[IJIS] B. Bérard, P. Courtieu, L. Millet, M. Potop-Butucaru, L. Rieg, N. Sznajder, S. Tixeuil, and X. Urbain. “Formal Methods for Mobile Robots: Current Results and Open Problems”. In: *International Journal of Informatics Society* 7 (accepted).

International conferences

[SSS’14] L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. “On the Synthesis of Mobile Robots Algorithms: The Case of Ring Gathering.” In: *Proc. of 16th Int. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS’14)*. Vol. 8756. Lecture Notes in Computer Science. Springer, 2014, pp. 237–251.

National conferences

[AlgoTel'13] B. Bérard, L. Millet, M. Potop-Butucaru, S. Tixeuil, and Y. Thierry-Mieg. "Verification formelle et robots mobiles". In: *15èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel'13)*. 2013.

[AlgoTel'15] L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. "Synthèse d'algorithmes pour robots mobiles : le cas du regroupement sur un anneau". In: *17èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel'15)*. 2015.

Submitted

[DISC] B. Bérard, P. Lafourcade, L. Millet, M. Potop-Butucaru, S. Tixeuil, and Y. Thierry-Mieg. "Formal Verification of Mobile Robot Protocols". In: *Distributed Computing* (under revision).

Ongoing works and perspectives

In the short term, we would like to apply synthesis to the gathering of 4 robots in the Async model, to obtain impossibility results and/or protocols. While algorithms already exist for restricted sets of initial configurations, we are looking for a protocol that could handle the maximal set of initial configurations. In this setting (still with n and 4 coprime), a particular class of initial configurations for which no result exists should be investigated, namely the SP4 configurations: defining an interval as a maximal sequence of free nodes, such configurations are *symmetric ones of type node-edge, such that the odd interval cut by the axis is bigger than the even one* (an SP4 configuration is depicted in Figure 1.2).

We also would like to extend the Pactole framework [Cou+15a] to certify the proofs given in Section 4.4 and 5.3.2, for the Min algorithm and the gathering solution respectively. This framework is a part of Coq devoted to robot algorithms and already permitted to certify a gathering algorithm in the plane for the Ssync execution model. According to the authors, adapting the tool to the discrete environment would not require too much modifications. On the other hand, extension to the Async model would be more difficult. For this point, we think that schemes like those depicted in Figure 4.2 and 4.3, describing the Min algorithm as a parametrized graph, could be useful to translate algorithms in this tool.

Moreover, several decidability questions remain open for the robot model. The first one is the decidability of parameterized verification for this model, with the ring size (and possibly the number of robots) as parameter. While there is a general undecidability result [AK86], several positive answers have been obtained in restricted frameworks, from [EFM99] for some broadcast protocols, to more recent work on byzantine consensus [Ami+14; KVW15]. Hence, the question remains of interest. When there is only one robot and the parameter of the system is the size of the graph, Rubin [Rub15] proves the decidability of the parameterized verification problem. This result is obtained by a reduction from classic questions in automata theory and monadic second order logic. Unfortunately, in this restricted setting, only a few problems are interesting and all answers are negative, which conforms to the intuition. We would like to see how his work could be extended to multiple robots. The second one is the decidability of synthesis of parameterized algorithm for this model, in view of the undecidability result for the general case [PR90].

In the longer term, we plan to extend this work and look for semi-automatic verification of parametrized algorithms, for the cases where no decidability results can be obtained. Since the first proposals in [MP94] or [CGJ95], a classical line of work combined model-checking with other techniques like abstraction, induction, etc. These methods are usually sound but incomplete and were largely used since, for instance in [Bjø+96; Alf+97; CMM01; Aro+01]. A preliminary step not described in this manuscript consists in the development of a prototype, where model-checking and inductive proofs are combined.

Bibliography

[AK86] K. R. Apt and D. Kozen. "Limits for Automatic Verification of Finite-State Concurrent Systems". In: *Information Processing Letters* 22.6 (1986), pp. 307–309.

[Alf+97] L. de Alfaro, Z. Manna, H. B. Sipma, and T. E. Uribe. "Visual Verification of Reactive Systems". In: *Proc. of 3d Int. Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS'97)*. Vol. 1217. Lecture Notes in Computer Science. Springer, 1997, pp. 334–350.

[Alm+12] J. B. Almeida, M. Barbosa, E. Bangerter, G. Barthe, S. Krenn, and S. Z. Béguelin. "Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols". In: *ACM Conference on Computer and Communications Security, CCS'12*. 2012, pp. 488–500.

[Ami+14] B. Aminof, T. Kotek, S. Rubin, F. Spegni, and H. Veith. "CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings". In: *25th International Conference on Concurrency Theory,CONCUR'14*. Ed. by P. Baldan and D. Gorla. Vol. 8704. Lecture Notes in Computer Science. Springer, 2014, pp. 109–124.

[And+99] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. "Distributed memoryless point convergence algorithm for mobile robots with limited visibility". In: *T. Robotics and Automation* 15.5 (1999), pp. 818–828.

[AP06] N. Agmon and D. Peleg. "Fault-Tolerant Gathering Algorithms for Autonomous Mobile Robots". In: *SIAM* 36.1 (2006), pp. 56–82.

[Aro+01] T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. D. Zuck. "Parameterized Verification with Automatically Computed Inductive Assertions". In: *Proc. of 13th Int. Conf. on Computer Aided Verification (CAV'01)*. Vol. 2102. Lecture Notes in Computer Science. Springer, 2001, pp. 221–234.

[AS85] B. Alpern and F. B. Schneider. "Defining Liveness". In: *Information Processing Letters* 21.4 (1985), pp. 181–185.

[Aug+13] C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain. "Certified Impossibility Results for Byzantine-Tolerant Mobile Robots". In: *15th International Symposium on Stabilization, Safety, and Security of Distributed Systems - 15th International Symposium, SSS'13*. 2013, pp. 178–190.

[Bal+08a] R. Baldoni, F. Bonnet, A. Milani, and M. Raynal. "Anonymous graph exploration without collision by mobile robots". In: *Information Processing Letters* 109.2 (2008), pp. 98–103.

[Bal+08b] R. Baldoni, F. Bonnet, A. Milani, and M. Raynal. "On the Solvability of Anonymous Partial Grids Exploration by Mobile Robots". In: *12th International Conference Principles of Distributed Systems, OPODIS'08*. Vol. 5401. Lecture Notes in Computer Science. Springer, 2008, pp. 428–445.

[Bar+13] J. Barnat, L. Brim, V. Havel, J. Havlícek, J. Kriho, M. Lenco, P. Rockai, V. Still, and J. Weiser. "DiVinE 3.0 - An Explicit-State Model Checker for Multithreaded C & C++ Programs". In: *25th International Conference on Computer Aided Verification (CAV'13)*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 863–868.

[BBN12] L. Blin, J. Burman, and N. Nisse. "Brief Announcement: Distributed Exclusive and Perpetual Tree Searching". In: *26th International Symposium on Distributed Computing, DISC'12*. Vol. 7611. Lecture Notes in Computer Science. Springer, 2012, pp. 403–404.

[BBN13] L. Blin, J. Burman, and N. Nisse. "Exclusive Graph Searching". In: *21st Annual European Symposium onAlgorithms, ESA'13*. Vol. 8125. Lecture Notes in Computer Science. Springer, 2013, pp. 181–192.

[Beh+07] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. "UPPAAL-Tiga: Time for Playing Games!" In: *19th International Conference on Computer Aided Verification, CAV'07*. Vol. 4590. Lecture Notes in Computer Science. Springer, 2007, pp. 121–125.

[Bjø+96] N. Bjørner, A. Browne, E. Y. Chang, M. Colón, A. Kapur, Z. Manna, H. Sipma, and T. E. Uribe. "STeP: Deductive-Algorithmic Verification of Reactive and Real-Time Systems". In: *8th International Conference on Computer Aided Verification (CAV'96)*. Vol. 1102. Lecture Notes in Computer Science. Springer, 1996, pp. 415–418.

[BK08] C. Baier and J. P. Katoen. *Principles of model checking*. MIT press, 2008.

[BL69] J. R. Büchi and L. H. Landweber. "Solving sequential conditions by finite-state strategies". In: *Trans. Amer. Math. Soc.* 138 (1969), pp. 295–311.

[Bli+10] L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil. "Exclusive Perpetual Ring Exploration without Chirality". In: *24th International Symposium in Distributed Computing (DISC'10)*. Vol. 6343. Lecture Notes in Computer Science. Springer, 2010, pp. 312–327.

[Bon+11] F. Bonnet, A. Milani, M. Potop-Butucaru, and S. Tixeuil. "Asynchronous exclusive perpetual grid exploration without sense of direction". In: *5th International Conference on Principles of Distributed Systems (OPODIS'11)*. Vol. 7109. Lecture Notes in Computer Science. 2011, pp. 251–265.

[Bon+12] F. Bonnet, X. Défago, F. Petit, M. Potop-Butucaru, and S. Tixeuil. "Brief Announcement: Discovering and Assessing Fine-grained Metrics in Robot Networks Protocols". In: *14th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'12)*. Vol. 7596. Lecture Notes in Computer Science. Springer, 2012, pp. 282–284.

[CE81] E. M. Clarke and E. A. Emerson. "Design and synthesis of synchronization skeletons using branching time temporal logic". In: *IBM Workshop on Logics of Programs*. 1981.

[CGJ95] E. M. Clarke, O. Grumberg, and S. Jha. "Veryfying Parameterized Networks using Abstraction and Regular Languages". In: *Proc. of 6th Int. Conf. on Concurrency Theory (CONCUR'95)*. Vol. 962. Lecture Notes in Computer Science. Springer, 1995, pp. 395–407.

[CGP09] J. Czyzowicz, L. Gasieniec, and A. Pelc. "Gathering few fat mobile robots in the plane". In: *Theoretical Computer Science* 410.6-7 (2009), pp. 481–499.

[Cha+10] J. Chalopin, P. Flocchini, B. Mans, and N. Santoro. "Network Exploration by Silent and Oblivious Robots". In: *36th International Workshop on Graph Theoretic Concepts in Computer Science, WG'10*. Vol. 6410. Lecture Notes in Computer Science. 2010, pp. 208–219.

[Chu63] A. Church. "Logic, arithmetic, and automata". In: *International Proceedings Congr. Mathematicians (Stockholm, 1962)*. Djursholm: Inst. Mittag-Leffler, 1963, pp. 23–35.

[Cie+03] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. "Solving the Robots Gathering Problem". English. In: *30th International Colloquium on Automata, Languages and Programming (ICALP'03)*. Ed. by J. Baeten, J. Lenstra, J. Parrow, and G. Woeginger. Vol. 2719. Lecture Notes in Computer Science. Springer, 2003, pp. 1181–1196.

[Cie04] M. Cieliebak. “Gathering Non-oblivious Mobile Robots”. In: *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings*. Vol. 2976. Lecture Notes in Computer Science. Springer, 2004, pp. 577–588.

[Cla+96] E. M. Clarke, S. Jha, R. Enders, and T. Filkorn. “Exploiting Symmetry in Temporal Logic Model Checking”. In: *Formal Methods in System Design* 9.1 (1996), pp. 77–104.

[Cle+08] A. Clerentin, M. Delafosse, L. Delahoche, B. Marhic, and A. Jolly-Desodt. “Uncertainty and imprecision modeling for the mobile robot localization problem”. In: *Autonomous Robots* 24.3 (2008), pp. 267–283.

[CM15] S. G. Chaudhuri and K. Mukhopadhyaya. “Leader election and gathering for asynchronous fat robots without common chirality”. In: *J. Discrete Algorithms* 33 (2015), pp. 171–192.

[CMM01] D. Cansell, D. Méry, and S. Merz. “Diagram Refinements for the Design of Reactive Systems”. In: *J. Univ. Comp. Sci.* 7.2 (2001), pp. 159–174.

[Col+13] M. Colange, S. Baarir, F. Kordon, and Y. Thierry-Mieg. “Towards Distributed Software Model-Checking using Decision Diagrams”. In: *25th International Conference on Computer Aided Verification (CAV’13)*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 830–845.

[Cou+15a] P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain. “A Certified Universal Gathering Algorithm for Oblivious Mobile Robots”. In: *CoRR* abs/1506.01603 (2015).

[Cou+15b] P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain. “Impossibility of gathering, a certification”. In: *Information Processing Letters* 115.3 (2015), pp. 447–452.

[CP02] M. Cieliebak and G. Prencipe. “Gathering Autonomous Mobile Robots”. In: *9th International Colloquium on Structural Information and Communication Complexity, SIROCCO’02*. Vol. 13. Proceedings in Informatics. Carleton Scientific, 2002, pp. 57–72.

[DAn+12] G. D’Angelo, G. D. Stefano, R. Klasing, and A. Navarra. “Gathering of Robots on Anonymous Grids without Multiplicity Detection”. In: *Structural Information and Communication Complexity - 19th International Colloquium, SIROCCO 2012, Reykjavik, Iceland, June 30-July 2, 2012, Revised Selected Papers*. Vol. 7355. Lecture Notes in Computer Science. Springer, 2012, pp. 327–338.

[DAn+13] G. D'Angelo, G. D. Stefano, A. Navarra, N. Nisse, and K. Suchan. "A Unified Approach for Different Tasks on Rings in Robot-Based Computing Systems". In: *27th International Symposium on Parallel & Distributed Processing Workshops, IPDPSW'13*. IEEE, 2013, pp. 667–676.

[Dat+13] A. K. Datta, A. Lamani, L. L. Larmore, and F. Petit. "Ring Exploration by Oblivious Robots with Vision Limited to 2 or 3". In: *15th International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS'13*. Vol. 8255. Lecture Notes in Computer Science. Springer, 2013, pp. 363–366.

[Déf+06] X. Défago, M. Gradinariu, S. Messika, and P. R. Parvédy. "Fault-Tolerant and Self-stabilizing Mobile Robots Gathering". In: *20th International Symposium on Distributed Computing, DISC'06*. Vol. 4167. Lecture Notes in Computer Science. Springer, 2006, pp. 46–60.

[Des+06] A. Dessmark, P. Fraigniaud, D. R. Kowalski, and A. Pelc. "Deterministic Rendezvous in Graphs". In: *Algorithmica* 46.1 (2006), pp. 69–96.

[Dev+12] S. Devismes, A. Lamani, F. Petit, P. Raymond, and S. Tixeuil. "Optimal Grid Exploration by Asynchronous Oblivious Robots". In: Lecture Notes in Computer Science 7596 (2012), pp. 64–76.

[Dev+14] S. Devismes, A. Lamani, F. Petit, and S. Tixeuil. "Optimal Torus Exploration by Oblivious Mobile Robots". unpublished. Jan. 2014.

[DPT13] S. Devismes, F. Petit, and S. Tixeuil. "Optimal probabilistic ring exploration by semi-synchronous oblivious robots". In: *Theoretical Computer Science* 498 (2013), pp. 10–27.

[DR] L. Doyen and J.-F. Raskin. "Games with Imperfect Information: Theory and Algorithms". unpublished.

[DSN11a] G. D'Angelo, G. D. Stefano, and A. Navarra. "Gathering of Six Robots on Anonymous Symmetric Rings". In: *Proc. of 18th Int. Coll. on Structural Information and Communication Complexity (SIROCCO'11)*. Vol. 6796. Lecture Notes in Computer Science. Springer, 2011, pp. 174–185.

[DSN11b] G. D'Angelo, G. D. Stefano, and A. Navarra. In: *18th International Colloquium on Structural Information and Communication Complexity, SIROCCO'11*. Vol. 6796. Lecture Notes in Computer Science. Springer, 2011, pp. 174–185.

[DSN12] G. D'Angelo, G. D. Stefano, and A. Navarra. "How to Gather Asynchronous Oblivious Robots on Anonymous Rings". In: *26th International Symposium on Distributed Computing, DISC'12*. Vol. 7611. Lecture Notes in Computer Science. Springer, 2012, pp. 326–340.

[DSN14] G. D'Angelo, G. D. Stefano, and A. Navarra. "Gathering on rings under the Look-Compute-Move model". In: *Distributed Computing* 27.4 (2014), pp. 255–285.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. "On the Verification of Broadcast Protocols". In: *14th Annual Symposium on Logic in Computer Science*. IEEE, 1999, pp. 352–359.

[ES96] E. A. Emerson and A. P. Sistla. "Symmetry and Model Checking". In: *Formal Methods in System Design* 9.1 (1996), pp. 105–131.

[Flo+05] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. "Gathering of asynchronous robots with limited visibility". In: *Theoretical Computer Science* 337.1-3 (2005), pp. 147–168.

[Flo+10] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. "Remembering without memory: Tree exploration by asynchronous oblivious robots". In: *Theoretical Computer Science* 411.14-15 (2010), pp. 1583–1598.

[Flo+11] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. "How many oblivious robots can explore a line". In: *Information Processing Letters* 111.20 (2011), pp. 1027–1031.

[Flo+13] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. "Computing Without Communicating: Ring Exploration by Asynchronous Oblivious Robots". In: *Algorithmica* 65.3 (2013), pp. 562–583.

[FP08] P. Fraigniaud and A. Pelc. "Deterministic Rendezvous in Trees with Little Memory". In: *22nd International Symposium on Distributed Computing, DISC'08*. Vol. 5218. Lecture Notes in Computer Science. Springer, 2008, pp. 242–256.

[FPS12] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2012.

[GP11] S. Guilbault and A. Pelc. "Gathering Asynchronous Oblivious Agents with Local Vision in Regular Bipartite Graphs". In: *18th International Colloquium on Structural Information and Communication Complexity, SIROCCO'11*. Vol. 6796. Lecture Notes in Computer Science. Springer, 2011, pp. 162–173.

[Hal+91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. "The synchronous dataflow programming language Lustre". In: *Proceedings of the IEEE* 79.9 (1991), pp. 1305–1320.

[HPT14] A. Honorat, M. Potop-Butucaru, and S. Tixeuil. "Gathering fat mobile robots with slim omnidirectional cameras". In: *Theor. Comput. Sci.* 557 (2014), pp. 1–27.

[Izu+10] T. Izumi, T. Izumi, S. Kamei, and F. Ooshita. "Mobile Robots Gathering Algorithm with Local Weak Multiplicity in Rings". In: *17th International Colloquium on Structural Information and Communication Complexity, SIROCCO'10*. Vol. 6058. Lecture Notes in Computer Science. Springer, 2010, pp. 101–113.

[Kam+11] S. Kamei, A. Lamani, F. Ooshita, and S. Tixeuil. "Asynchronous Mobile Robot Gathering from Symmetric Configurations without Global Multiplicity Detection". In: *18th International Colloquium on Structural Information and Communication Complexity, SIROCCO'11*. Vol. 6796. Lecture Notes in Computer Science. Springer, 2011, pp. 150–161.

[Kam+12] S. Kamei, A. Lamani, S. Kamei, F. Ooshita, and S. Tixeuil. "Gathering an Even Number of Robots in an Odd Ring without Global Multiplicity Detection". In: *37th International Symposiumon Mathematical Foundations of Computer Science, MFCS'12*. Vol. 7464. Lecture Notes in Computer Science. Springer, 2012, pp. 542–553.

[KKM10] E. Kranakis, D. Krizanc, and E. Markou. *The Mobile Agent Rendezvous Problem in the Ring*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2010.

[KKN08] R. Klasing, A. Kosowski, and A. Navarra. "Taking Advantage of Symmetries: Gathering of Asynchronous Oblivious Robots on a Ring". In: *Theoretical Computer Science*. Lecture Notes in Computer Science 5401 (2008). Ed. by T. Baker, A. Bui, and S. Tixeuil, pp. 446–462.

[Kle+10] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. "seL4: formal verification of an operating-system kernel". In: *Commun. ACM* 53.6 (2010), pp. 107–115.

[KMP06] R. Klasing, E. Markou, and A. Pelc. "Gathering Asynchronous Oblivious Mobile Robots in a Ring". In: *Algorithms and Computation, 17th International Symposium, ISAAC 2006, Kolkata, India, De-*

cember 18-20, 2006, Proceedings. Vol. 4288. Lecture Notes in Computer Science. Springer, 2006, pp. 744–753.

[Kor10] M. Koren. *Gathering small number of mobile asynchronous robots on ring*. Tech. rep. 18:325–331. Zeszyty Naukowe Wydzialu ETI Politechniki Gdanskiej. Technologie Informacyjne, 2010.

[KVW15] I. Konnov, H. Veith, and J. Widder. "SMT and POR Beat Counter Abstraction: Parameterized Model Checking of Threshold-Based Distributed Algorithms". In: *27th International Conference on Computer Aided Verification, CAV'15*. Ed. by D. Kroening and C. S. Pasareanu. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 85–102.

[Ler09] X. Leroy. "A Formally Verified Compiler Back-end". In: *J. Autom. Reasoning* 43.4 (2009), pp. 363–446.

[LMA07] J. Lin, A. S. Morse, and B. D. O. Anderson. "The Multi-Agent Rendezvous Problem. Part 2: The Asynchronous Case". In: *SIAM J. Control and Optimization* 46.6 (2007), pp. 2120–2147.

[LPT10] A. Lamani, M. G. Potop-Butucaru, and S. Tixeuil. "Optimal deterministic ring exploration with oblivious asynchronous robots". In: *Proc. of 17th Int. Coll. in Structural Information and Communication Complexity (SIROCCO'10)*. Vol. 6058. Lecture Notes in Computer Science. Springer, 2010, pp. 183–196.

[Lyn96a] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[Lyn96b] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[Mar75] D. A. Martin. "Borel Determinacy". English. In: *Annals of Mathematics*. Second Series 102.2 (1975), pp. 363–371.

[Maz01] R. Mazala. "Infinite Games". In: *Automata, Logics, and Infinite Games*. Ed. by E. Grädel, W. Thomas, and T. Wilke. Vol. 2500. Lecture Notes in Computer Science. Springer, 2001, pp. 23–42.

[MP94] Z. Manna and A. Pnueli. "Temporal Verification Diagrams". In: *Proc.of Int. Conf. on Theoretical Aspects of Computer Software (TACS'94)*. Vol. 789. Lecture Notes in Computer Science. Springer, 1994, pp. 726–765.

[MW84] Z. Manna and P. Wolper. "Synthesis of Communicating Processes from Temporal Logic Specifications". In: *ACM Trans. Program. Lang. Syst.* 6.1 (1984), pp. 68–93.

[Pel11] A. Pelc. “DISC 2011 Invited Lecture: Deterministic Rendezvous in Networks: Survey of Models and Results”. In: *Distributed Computing - 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings*. Vol. 6950. Lecture Notes in Computer Science. Springer, 2011, pp. 1–15.

[PR79] G. L. Peterson and J. H. Reif. “Multiple-Person Alternation”. In: *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*. 1979, pp. 348–363.

[PR90] A. Pnueli and R. Rosner. “Distributed Reactive Systems Are Hard to Synthesize”. In: *31st Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1990, pp. 746–757.

[Pre00] G. Prencipe. *A New Distributed Model to Control and Coordinate a Set of Autonomous Mobile Robots: The Corda Model*. 2000.

[Pre05] G. Prencipe. “On the Feasibility of Gathering by Autonomous Mobile Robots”. In: *Structural Information and Communication Complexity, 12th International Colloquium, SIROCCO 2005, Mont Saint-Michel, France, May 24-26, 2005, Proceedings*. Ed. by A. Pelc and M. Raynal. Vol. 3499. Lecture Notes in Computer Science. Springer, 2005, pp. 246–261.

[Pre13] G. Prencipe. “Autonomous Mobile Robots: A Distributed Computing Perspective”. In: *Algorithms for Sensor Systems - 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS 2013, Sophia Antipolis, France, September 5-6, 2013, Revised Selected Papers*. Ed. by P. Flocchini, J. Gao, E. Kranakis, and F. M. auf der Heide. Vol. 8243. Lecture Notes in Computer Science. Springer, 2013, pp. 6–21.

[PRT11] M. Potop-Butucaru, M. Raynal, and S. Tixeuil. “Distributed Computing with Mobile Robots: An Introductory Survey”. In: *The 14th International Conference on Network-Based Information Systems, NBiS 2011, Tirana, Albania, September 7-9, 2011*. Ed. by L. Barolli, F. Xhafa, and M. Takizawa. IEEE Computer Society, 2011, pp. 318–324.

[Rei84] J. H. Reif. “The complexity of two-player games of incomplete information”. In: *Journal of Computer and System Sciences* 29.2 (1984), pp. 274–301.

[Rub15] S. Rubin. "Parameterised Verification of Autonomous Mobile-Agents in Static but Unknown Environments". In: *International Conference on Autonomous Agents and Multiagent Systems, AAMAS'15*. Ed. by G. Weiss, P. Yolum, R. H. Bordini, and E. Elkind. ACM, 2015, pp. 199–208.

[SDY09] S. Souissi, X. Défago, and M. Yamashita. "Using eventually consistent compasses to gather memory-less mobile robots with limited visibility". In: *TAAS* 4.1 (2009).

[SN13] G. D. Stefano and A. Navarra. "Optimal Gathering of Oblivious Robots in Anonymous Graphs". In: *Proc. of SIROCCO*. Vol. 8179. Lecture Notes in Computer Science. Springer, 2013, pp. 213–224.

[SY99] I. Suzuki and M. Yamashita. "Distributed Anonymous Mobile Robots: Formation of Geometric Patterns". In: *SIAM Journal on Computing* 28.4 (1999), pp. 1347–1363.

[Tel01] G. Tel. *Introduction to Distributed Algorithms*. 2nd. New York, NY, USA: Cambridge University Press, 2001.

[Val96] A. Valmari. "The State Explosion Problem". In: *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*. Ed. by W. Reisig and G. Rozenberg. Vol. 1491. Lecture Notes in Computer Science. Springer, 1996, pp. 429–528.