A Foundational Framework for Certified Impossibility Results with Mobile Robots on Graphs

Thibaut Balabonski
LRI, Université Paris-Sud, France

Robin Pelle
LRI, Université Paris-Sud, France

Lionel Rieg
Yale University, USA

Sébastien Tixeuil
UPMC Sorbonne Universtités, France

ABSTRACT

Swarms of mobile robots recently attracted the focus of the Distributed Computing community. One of the fundamental problems in this context is that of exploration: the robots must coordinate to visit all locations that are reachable from their initial positions. Despite its apparent simplicity, this problem proved quite hard to characterise fully, due to many model variants, leading to informal error-prone reasoning.

Over the past few years, a significant effort permitted to set up a formal framework, relying on the Coq proof assistant, which was used to provide certified results when robots evolve in a continuous bi-dimensional Euclidean space. However, the most challenging issues with exploration arise in the discrete setting (a.k.a. graph), where locations are modeled as vertices and where edges between vertices denote the ability for a robot to move from one location to the next.

We present a formal model to tackle problems and reason about robot algorithms arising in the discrete setting. Our approach extends and generalises previous research efforts focusing on the continuous model. As case studies, we consider fundamental impossibility results for exploration with stop in the discrete model. To our knowledge, those are the first certified results in this context. This framework paves the way for a general certification workflow dedicated to mobile robots on graphs.

CCS CONCEPTS

• **Computer systems organization** **Embedded systems**; *Redundancy*; Robotics; • **Networks** Network reliability;

KEYWORDS

Mobile Robots, Proof Assistant, Exploration, Impossibility results

ACM Reference format:
Thibaut Balabonski, Robin Pelle, Lionel Rieg, and Sébastien Tixeuil. 2018. A Foundational Framework for Certified Impossibility Results with Mobile Robots on Graphs. In *Proceedings of 19th International Conference on Distributed Computing and Networking, Varanasi, India, January 4–7, 2018 (ICDCN '18),* 10 pages.
https://doi.org/10.1145/3154273.3154321

ACM ISBN 978-1-4503-6372-3/18/01…$15.00
https://doi.org/10.1145/3154273.3154321

1 INTRODUCTION

Networks of mobile robots captured the attention of the distributed computing community, as they promise new application (rescue, exploration, surveillance) in potentially dangerous environments. Originally introduced in 1999 by Suzuki and Yamashita [32], the model has been refined since by many authors while growing in popularity (see [24] for a comprehensive textbook). From a theoretical point of view, the interest lies in characterising, for each of these various refinements, the exact conditions under which a particular task can be solved or not.

1.1 The Look-Compute-Move model

In the model we consider, all robots operate using the same embedded program through repeated Look-Compute-Move cycles. In each cycle a robot first "looks" at its environment and obtains a snapshot containing some information about the locations of all robots, expressed in the robot's own self-centered coordinate system, whose scale and orientation might not be consistent with the other robot's coordinate systems (or even with the same robot's coordinate system from a previous cycle). Then the robot "computes" a destination, still in its own coordinate system, based only on the snapshot it just obtained (which means the robot is oblivious, in the sense that its behaviour is independent of the past history of execution). Finally the robot "moves" towards the computed destination.

Different levels of synchronisation have been considered. The weakest [24] is the asynchronous model (ASYNC), where each robot performs its own Look, Compute and Move actions at its own pace, which needs not be consistent with that of other robots. The strongest [32] is the fully synchronous model (FSYNC), where all robots perform simultaneously and atomically all of these three steps. In this paper, we consider both the FSYNC model and a intermediate level [32] called semi-synchronous (SSYNC), where the computation is organised in rounds and only a subset of the robots are active at any given round. The active robots in a round perform exactly one atomic Look-Compute-Move cycle in that round. The subset of active robots is chosen arbitrarily by a scheduler (seen as an adversary and called demon). We have absolutely no control over the choices made by the scheduler and only assume that the scheduler is fair, in the sense that at any round, each robot is guaranteed to be activated within a finite number of rounds. The FSYNC model can be seen as a particular case of SSYNC, which is realised when an SSYNC scheduler chooses to activate all the robots at each round.

A task is considered to be solved if one provides an algorithm guaranteeing that *for any admissible initial configuration of the robots and for any admissible schedule* the task will eventually be accomplished (which means that a task solved in the SSYNC model is also solved in the FSYNC model, and that an algorithm that does not solve a task in the FSYNC model does not solve it in the SSYNC model either). On the contrary, showing that some conditions are necessary for a task to be solvable requires reasoning on all the possible algorithms and proving that no algorithm can solve the task when said conditions are not met.

The general model is agnostic to the shape of the space where the robots evolve, which can be the real line, a two dimensional euclidean space, a discrete space, or even another space with a more intricate topology.

In this paper we use a general methodology that is adapted to the general case, and study its instantiation on discrete graphs.

1.2 Robots on graphs

We consider a finite set of k robots evolving in a discrete space made of a finite set of n positions, where both k and n are unspecified, arbitrarily large numbers. Our results are valid for all values of k and n, and will only depend on some basic relations between them, such as $k < n$ or k divides n. A given position may harbour several robots, in which case it is said that a *tower* of robots is located there. The k robots are all identical and anonymous (a robot cannot distinguish between two fellow robots), and they do not share any reference point or orientation.

A configuration is a map from robot identifiers to positions. All robots run the same *robogram*: an algorithm that designates a new position for a robot when it is activated. The destination position can only be chosen within a neighbourhood of the current position, which is defined by modelling the space as an undirected, simple graph whose vertices are the possible positions of the robots. The edges of this graph describe the neighbourhood of a position and the possible moves of the robots.

During the Look phase, a robot obtains an inaccurate snapshot of the whole configuration: since it is possibly disoriented, the robot sees an arbitrary graph isomorphic to the underlying graph, instead of the actual underlying graph. Moreover, since all the other robots are anonymous, the robot only perceives its own position and the number of robots inhabiting each vertex.

In this context, typical problems are *terminating exploration* [11, 18–20, 22, 23, 28], *exclusive perpetual exploration* [3, 6–8, 17], *exclusive searching* [5, 16, 17], and *gathering* [12, 17, 25–27]. While our formalisation approach is universal and can be used to tackle any of those problems, the case studies we consider in this paper are related to the *terminating exploration* (or simply *exploration*) problem, which requires that robots collectively explore the whole graph and stop upon completion. For this problem to be interesting, we will assume throughout the paper that the number k of robots is strictly less than the numbers n of positions to be explored.

With respect to the (terminating) exploration problem, the main metric considered in existing literature is the necessary and sufficient number of robots for exploring particular classes of graphs. The only result available for exploration in general graphs [11] considers that edges are labeled in such a way that the network configuration is asymmetric. In this extended model, three robots are not sufficient to explore all asymmetric configurations, and four robots are sufficient to explore all asymmetric configurations. Note that exploring the set of asymmetric configurations is strictly weaker than exploring the complete underlying graph, especially when the graph is highly symmetric. The rest of the literature is thus dedicated to a weaker model, where edges are not labeled. One extreme case in this weak model is the set of tree-shaped networks, as in general, $\Omega(n)$ robots are necessary and sufficient to explore a tree network of n nodes deterministically [22]. The other extreme case is the set of grid-shaped networks [18], where three robots are necessary and sufficient to explore deterministically any grid of at least three nodes (except for the grids of size 2×2 and 3×3, where four – respectively five – robots are necessary and sufficient). However, this result is mainly due to the fact that grids are not regular graphs: they contain nodes of degrees 2, 3, and 4. So, this topological property implies less symmetries.

By contrast, rings and tori are regular graphs, and consequently more intricate. In ring-shaped networks [23], the fact that the number k of robots and the ring size n must be coprime yields to the lower bound $\Omega(\log n)$ on the number of robots required to explore a n-size ring. Indeed, the smallest non-divisor of n evolves as $\log n$ in the worst case. However, notice that Lamani *et al.* also provide [28] an algorithm that allows 5 robots to explore deterministically any ring whose size is coprime with 5. The large number of robots and the constraint on the ratio between the number of robots and the ring size induced by the deterministic setting in ring-shaped networks hinted at a possible more efficient solution when robots can make use of probabilities [20]. As a matter of fact, four robots are necessary and sufficient to explore probabilistically any ring of size at least four. While the gain in going probabilistic is only one robot when n is not divisible by 5, a logarithmic factor is obtained in the general case. A probabilistic extension to the case of torus-shaped networks was presented by Devismes *et al.* [19], and four robots are also necessary and sufficient in this case.

1.3 The Certification Path

The design and the verification of protocols for swarms of robots is notoriously difficult, and Formal Methods have been recently put into use to get rid of errors introduced by humans in that context [1, 2, 4, 7, 9, 14, 18, 29–31].

Model-Checking proved useful to find bugs in existing literature [4, 21] and check formally the correctness of published algorithms [4, 18, 30]. However, the current models do not permit to establish impossibility results, only to assess the correctness of candidate solutions. *Automatic program synthesis* for the problem of perpetual exclusive exploration in a discrete ring is due to Bonnet *et al.* [7], and can be used to obtain automatically algorithms that are "correct-by-design". The approach was refined by Millet *et al.* [29] for the problem of gathering in a discrete ring network. In principle, program synthesis satisfying completeness permits to establish impossibility results (no algorithm satisfying the specification can be found). However, to date only very small problem instances (small rings shaped networks, and a fixed - 3 - number of robots) were considered. A recent result by Sangnier *et al.* [31] gives little hope about the scalability of model-checking/program synthesis approaches beyond a few nodes, even for a fixed number of robots: even in the most simple case FSYNC, parameterized verification of reachability properties (that are required for verifying exploration with stop) is undecidable. Recently, Aminof *et al.* [30] presented a general framework for verifying properties about mobile robots evolving on graphs, where the graphs are a parameter of the problem. As they consider non-oblivious robots, most interesting properties are also undecidable. Overall, despite its appealing simplicity, no existing model can provide either scalability or generality for the problem we consider.

An approach based on *Formal Proof* has been introduced with the framework Pactole.[1] On the contrary to model-checking, formal proofs do not suffer from a scalability issue: anything that can be proven on paper can be made formal (provided the proof is correct of course!). The downside is that this technique is much less automated and requires more work from the user: the main job of the proof assistant is to check the correctness of the proof the user builds. To ease the creation of these formal proofs, interactive proof assistants also provide tools and guidance during the construction of the proof itself.

The Pactole formal model is developed in the formal language of the Coq interactive proof assistant,[2] a very expressive λ-calculus: the *Calculus of Inductive Constructions* (CIC) [13]. In this (functional) language, datatypes, objects, algorithms, theorems and proofs can be expressed in a unified way, as terms. λ-abstraction is denoted `fun x:T ⇒ t`, and application is denoted `t u`. The Curry-Howard isomorphism associates proofs and programs, types and logical propositions. Writing a proof of a theorem in this setting amounts to building (interactively in most cases but with the help of tactics) a term the type of which corresponds to the theorem statement. As a term is indeed a *proof* of its type, ensuring the soundness of a proof thus simply consists in type-checking a λ-term.

Unlike previous approaches which are devised for a discrete space where robots move according to a pre-existing graph, Pactole has been applied to the case where robots move freely in a bidimensional Euclidian space. Pactole is to our knowledge the only formal framework for robots swarms on continuous spaces. It provides positive certified results for SSYNC gathering with multiplicity detection [15], and for FSYNC gathering without multiplicity detection [2]. Using higher-order logic, Pactole was also successfully used to certify *impossibility results*, notably for the problem of (Byzantine) Convergence [1] where robots are required to reach positions that are arbitrarily close to each other, or for the problem of Gathering starting from a *bivalent* configuration [14], that is, a configuration with exactly two distinct towers, each consisting of half the robots. With respect to scalability and generality of the expressed properties, Pactole is an ideal candidate for our purpose, but its initial design toward continuous spaces makes it difficult to assess its relevance for discrete spaces.

[1]http://pactole.lri.fr
[2]https://coq.inria.fr

1.4 Our Contribution

We present a formal model to tackle problems, and reason about robot algorithms arising in the discrete setting, and its development in the Coq proof assistant. Our approach extends and generalises previous research efforts focusing on the continuous model. In addition to the core model extension, we provide convenient interfaces to ease the specification of graphs and their properties.

We illustrate the adequacy of our framework by considering fundamental impossibility results for exploration with stop in the discrete model, where we obtain, to our knowledge, the first certified results in this context. The Coq langage allowing the use of quantifiers, the results we obtain apply for any number of robots and any size of graph, thus putting aside the issue of scalability. Our approach paves the way for a general certification workflow dedicated to mobile robots on graphs.

The formal development related to this paper is available online at http://pactole.lri.fr/pub/cier/html/toc, and the whole Pactole project at http://pactole.lri.fr/. A symbol ⚭ in the margin is a link to a specific file of the development. ⚭

2 IMPOSSIBILITY RESULTS

Two kinds of results help us in characterising the conditions under which a task is feasible: positive results identify *sufficient* conditions for the task to be feasible by providing algorithms that provably solve the task when some conditions are met. On the contrary, negative results identify *necessary* conditions, by proving that no algorithm can solve the task when some conditions are not met. We get a good understanding of the considered task when we can provide both positive and negative results, under conditions that are as close as possible to each other.

We shall consider in this paper a specific shape of graphs: a *ring* of size n, and the task of *Exploration with Stop* of this ring with k robots.

We provide hereafter a formalisation of robots and graphs that can express both the fully synchronous model FSYNC and the semi-synchronous model SSYNC, and prove that some conditions on k and n make the task impossible in both models (since FSYNC can be seen as a particular case of an SSYNC scheduler, it suffices to prove the impossibility of the task for this particular scheduler). In particular, we provide a formalisation of the proof that exploration with stop requires n to be not divisible by k.

2.1 Main arguments for impossibility in graphs

An algorithm solves the exploration with stop of a ring if, for every admissible initial configuration and every demon, the two following properties hold:

(1) each node of the graph is eventually visited by at least one robot, and
(2) all robots eventually stop forever.

Thus, proving that a given algorithm does not solve the exploration with stop amounts to providing an admissible initial configuration and a demon falsifying at least one of our two properties. Since we cannot enumerate all algorithms for an arbitrary number k of robots on a graph of arbitrary size n, proving that no algorithm can solve exploration requires classifying all the algorithms into a finite set of classes, and providing generic counter-examples that apply to whole classes.

In other words, we have to exhibit admissible initial configurations and demons such that, for any class of algorithms, we can prove that:

- either the algorithm stops, but we can prove that not all nodes have been visited,
- or some robots move, but we can prove that the resulting configuration is similar enough to the initial configuration to ensure that robots will never stop moving.

We accept as initial configurations any configuration where no two robots are on the same node of the graph[3]. For our counter-examples to apply to both the FSYNC and SSYNC cases, we will also preferentially chose demons that fulfil the conditions of fully synchronous executions. Since such a demon can also be seen as a semi-synchronous demon (which happens to activate always the full subset of the robots), the same proof will cover both execution models.

2.2 Examples

We develop our approach to certify the impossibility of exploration with stop in two cases: less than two robots, and at least two robots when their number divides the size of the ring.

2.2.1 Too few robots. We start with the following very simple but essential remark: due to the restriction that the number of robots k is strictly less than the size of the graph n, no initial configuration explores all nodes in the graph.

Thus, an algorithm that stops at an initial configuration cannot solve the exploration. This implies that, on any configuration that would be admissible as an initial configuration, an algorithm solving the exploration has to make at least one move, which in our case applies to any configuration where no two robots are on the same node. Hence an algorithm solving the exploration *with stop* must reach a configuration that is not admissible as an initial configuration, i.e., a configuration where at least two robots are on the same node.

A simple consequence of this remark is that there is no algorithm solving the exploration with stop with only one robot.

2.2.2 Number of robots dividing the size of the ring. If we have at least two robots, and if the number of robots divides the size of the ring, then we can build an initial configuration that is symmetric enough to ensure that all robots have exactly the same behaviour, and that the global configuration remains in a state similar to the initial state: we choose the initial configuration where all robots are regularly spaced on the ring. In this configuration, all robots have the exact same view of their environment and choose the same move if they are activated. Since exploration with stop is solved only by a protocol that succeeds for all possible schedulers and disorientations of the robots, it suffices to exhibit one demon that makes all protocols fail with this initial configuration. We choose the demon that activates all robots at each round and gives them all the same orientation.

[3]This is not the most general notion of admissible initial configuration that could be used here, but it allows a clean presentation of the formalisation. In particular, it gives a simple way to characterize a family of configurations that cannot be final.

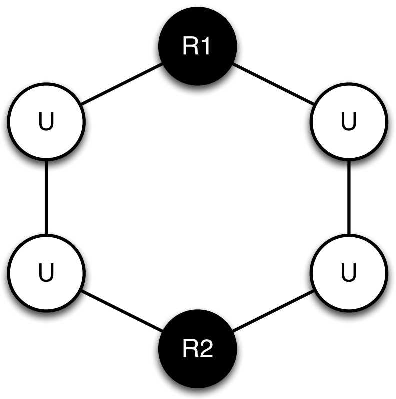

Figure 1: Two robots and four unvisited nodes

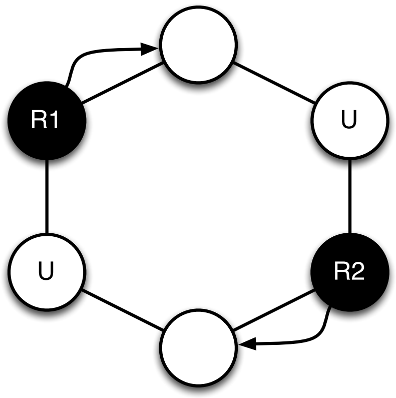

Figure 2: Two robots going to their left

Then any algorithm will have one of the following behaviours:

- either it orders to each robot not to move, which implies that the algorithm stops (but not all nodes have been visited, see Figure 1 for an example with two robots in a six-nodes ring),
- or it orders to each robot to move in the same direction, which implies that the next configuration will be undistinguishable from the initial one, and that all robots will go on moving forever, see Figure 2 for an example with two robots moving in a six-nodes ring (the robots will move forever since they are not able to distinguish visited from unvisited nodes).

From this we deduce that no algorithm can solve the exploration with stop of a ring when the number of robots divides the size of the ring.

3 CERTIFYING IMPOSSIBILITY RESULTS

3.1 A general pattern for graphs in Pactole

An important axis in the development of Pactole is genericity. To be as expressive as possible with reference to the many variations in the model of Suzuki & Yamashita [32], an important part of the framework is kept abstract. In particular, the space where robots move is encapsulated into a module `Location` that provides a core type `Location.t` denoting the positions in the actual space, together with useful functions (a decidable equality, an origin, a distance, etc.). This module may be instantiated with $\mathbb{R}$ if one considers the real line, or $\mathbb{R} \times \mathbb{R}$ with the relevant arithmetics for the real plane, etc.

So as to allow for a comfortable use of graphs in Pactole, we provide a general template for graphs, that the user can instantiate with the kind of graphs needed. The graph theory we provide is rather lightweight and restricted compared to, for instance, the library Loco for local computation on graphs [10]. It is however fully integrated in Pactole and connects naturally with the main signature for the spaces where robots evolve, which provides simple means of specifying discrete spaces and reasoning about them in Pactole. In Section 3.2, we will provide in particular an instance for rings of a certain size.

Technically, Coq provides a module system with signatures (called **`Module Type`**) that can be implemented by modules. A signature Σ may declare some objects that are intended to be defined in modules implementing Σ. For instance, a parameter `x` of type `A` can be introduced with the declaration **`Parameter`** `x : A`. A signature or module can also define an object `x` to have the value `a` with **`Definition`** `x := a`. A module `M` implementing a signature Σ can define any parameter `x : A` of Σ by providing a value of the corresponding type `A`. In the same fashion, a signature can "declare" a property that has to be satisfied by its parameters (that is, an **`Axiom`**), and one can "define" such a property in a module by providing a proof for it (that is, by turning the axiom into a **`Lemma`**).

In Pactole, we provide a general signature for graphs (`Graph`) as well as a more precise signature for finite graphs (`FiniteGraph`). Signature `Graph` is thus a **`Module Type`** whose parameters are two data types `V` and `E` for vertices and edges, together with some basic definitions and properties. To ease the implementation, each edge is directed and connects a source vertex to a target vertex, given by respective functions `src` and `tgt`. Since the graphs we are interested in Pactole are undirected, we will systematically model the fact that two vertices are neighbouring locations by connecting them with two directed opposite edges. The main parameters of a graph are then:

```
Parameter V : Set.
Parameter E : Set.
Parameter src tgt : E → V.
```

Remark that `Set` here denotes the data types of Coq, which do not coincide with the mathematical sets.

To check if two vertices are neighbours, we finally need a function that, given two vertices, returns an *option* over the potential edge, that is either `Some` e if there is such an edge e, or `None` if there is not.

```
Parameter find_edge : V → V → option E.
```

As all the other parameters the function are only declared at this point and not defined, we add some axioms to ensure that this function enjoys all the needed properties. In particular: if two vertices are respectively the source and the target of an edge, then the function will return the option type `Some` e, with e being the edge And if, given two vertices, the function returns the option type `None`, then no edge has (both) the first vertex as source, and the second vertex as target.

As the demon may change the frame of reference of a robot, what a robot perceives is a graph that is isomorphic to, but not necessary equal to, the actual underlying graph. In other words, the robot perceives the shape of the graph, but cannot distinguish between two similar vertices. We thus include in Pactole a notion of graph isomorphism , defined by a pair consisting of a bijection `sim_V` on the vertices, and a bijection `sim_E` on the edges. Both functions are compatible: if e is an edge between two vertices v_1 and v_2, then its image `sim_E`(e) is an edge between the two images `sim_V`(v_1) and `sim_V`(v_2).

```
Record Isomorphism :=
{
  sim_V : Bijection V;
  sim_E : Bijection E;
  sim_morphism : ∀ e,
     Veq (sim_V (src e)) (src (sim_E e))
   ∧ Veq (sim_V (tgt e)) (tgt (sim_E e))
}.
```

where `Bijection T` is the type of bijective functions on `T`.

The set of parameters and axioms we just described defines the most general theory of graphs used in Pactole. This theory can now be refined to more specific cases of graphs, by adding new parameters or axioms, or by providing concrete definitions for some of the parameters, and proofs for the corresponding axioms.

We may for instance obtain a signature `FiniteGraph` for finite graphs with `n` vertices by adding a new parameter `n` and by defining the type `V` of vertices to represent a finite set with `n` elements. Technically we use the type `Fin.t n` from the Coq library, which denotes the elements of a finite set with `n` elements. We leave the parameter `E` abstract, and obtain the following declarations:

```
Module Type FiniteGraph <: Graph :=
  Parameter  n : nat.
  Definition V := Fin.t n.
  Parameter  E : Set.
...
```

where the subtyping notation `<:` declares that `FiniteGraph` is a refinement of the signature `Graph`.

3.2 A particular graph: the ring of size n

We instantiate the signature `FiniteGraph` described above on a specific shape of finite graphs: rings . To avoid some trivial case distinction in our proofs, we assume $n > 1$; the exploration problem is indeed not very interesting when there is zero or one position.

A ring contains, for each vertex, a directed forward edge towards its successor and a directed backward edge towards its predecessor. Hence we will represent a directed edge by a source vertex and a direction. Since the directed edges represent the possible actions of the robots and a robot may choose to remain at its current position, for the sake of simplicity we also add a direction `AutoLoop` denoting an empty move.

```
Inductive direction :=
  | Forward | Backward | AutoLoop.
```

We thus characterise our graph with

```
Module Ring <: FiniteGraph :=
  Parameter n  : nat.
  Axiom n_sup1 : 1 < n.
  Definition V := Fin.t n.
  Definition E := (Fin.t n * direction).
  ...
```

The implementation of the relevant functions is based on $\mathbb{Z}/n\mathbb{Z}$ as a model for the set of vertices. A position on the ring is seen as an integer from $\mathbb{Z}$ taken modulo *n*. Predecessor, Successor, and remaining functions are obtained by straightforward arithmetics. Throughout this paper, we use the notations $\mathtt{i}_V$ for the vertex corresponding to the integer i, and $\mathtt{v}_{\mathbb{Z}}$ for the integer encoding the vertex v.

With these definitions we obtain a graph that is a slight variation tuned for Pactole of the usual notion of ring, on which we will be able to conduct Coq proofs.

3.3 Look-Compute-Move Model

The formalisation of the Look-Compute-Move model in Pactole has been described in [1, 2, 15]. We briefly recall what we need here.

Robots are just encoded as identifiers.

The embedded program the robots use to define their moves consists of a function pgm that simply returns a destination when given a perception of the environment (the type of a perception, named Spect.t the code and *spectrum* in the body of the paper, will be made explicit in the next section). Said function is required to return a destination that is reachable through an edge from the current vertex of the robot, which is expressed by the property pgm_range, packed with the declaration of the function pgm to form what we call a *robogram*.

```
Record robogram :=
{
  pgm : Spect.t → Location.t;
  pgm_range : ∀ (spect: Spect.t),
    ∃ e, find_edge (get_current spect) (pgm spect)
         = Some e
}.
```

Remark that, since any vertex contains an AutoLoop edge to itself, the constraint pgm_range does not prevent a robot from staying in place.

Depending on the robots' capabilities, the perception may not be as accurate as the complete configuration: anonymous robots cannot see names, they may lack detection of multiplicity, frames of reference may not be shared, vision can be limited, etc. The forbidden information is pruned from the configuration, using the function from_config which returns a *spectrum*, which is the only input of the robogram's pgm. Spectra form an arbitrary type that is part of the description of the model and contributes to its genericity.

In a SSYNC model, subsets of robots are activated to perform (synchronously) their atomic Look-Compute-Move cycle. A demon is thus an infinite sequence (stream) of *demonic actions*. A demonic action is defined by two functions. The one in which we are most interested here is the function step, which both selects the subset of robots to be activated at this round (that is, all robots if in a FSYNC setting) and provides the activated robots with a graph isomorphism defining their new frame of reference. The selection of activated robots is encoded in an option type (whose values are either None of Some v where v is the actual content we are interested in). To ease the definition of interesting demonic actions, the graph isomorphism provided to a robot depends on the *status* of this robot, hence the function type robot_status → Iso.t in the following code fragment. The notion of status of a robot (robot_status) will be discussed later, when its definition needs to be unfolded.

```
Record demonic_action :=
{
  relocate_byz : robot → robot_status;
  step : robot → option (robot_status → Iso.t);
}.
```

Finally, *executions* are obtained from a robogram and a demon by executing successively the robogram against the demonic action described by the demon for each round. To this end, a round function computes the configuration obtained after one round of executing a robogram against a demonic action da starting from a configuration. This is done in the following consecutive steps for each robot identifier r:

(1) If the robot r is not activated, return the same position.
(2) If r is a Byzantine robot, it is relocated by the demonic action da.
(3) Use the local frame of reference provided by da to compute the local configuration.
(4) Transform this local configuration into a spectrum using from_config.
(5) Apply the robogram on this spectrum.
(6) Convert the new position from the local frame to the global one.

To define a full execution, the function execute rbg d config iterates round starting from the configuration config, using the robogram rbg and the demon d.

In order to easily express properties about executions and schedules (demons), both seen as streams, we define the usual temporal operators $\Diamond$ (the property will become valid within a finite number of rounds), $\circ$ (the property will be valid in the next round), and $\square$ (the property is valid forever, starting from this round) to help expressing temporal properties about executions. In our formalisation, these three operators are written respectively Stream.eventually, Stream.next, and Stream.forever Yet, the logic of Coq is much more expressive, and one can define new temporal operators or new properties directly on an execution. Such a formalisation allows for a convenient handling of demons, including their theoretical study. The framework provides in particular definitions for different flavours of *fairness*, with relevant theorems.

4 A FORMALISATION OF THE IMPOSSIBILITY OF EXPLORATION WITH STOP

4.1 Context

On a finite discrete ring, we consider oblivious and anonymous robots, endorsed with multiplicity detection, and moving in an SSYNC fashion. We assume there are no byzantine robots: since exploration with stop is impossible even without byzantine robots, it surely is with them too. The ring's vertices have no name, and there is no special vertex known to robots as a reference point.

With anonymous robots, the spectrum must be clear of all information about names. With robots equipped with detection of multiplicity, a pointed multiset of inhabited positions is a suitable spectrum (that is, a multiset with a distinguished element denoting the position of the considered robot).

Robots are oblivious, hence at each activation they must be associated to a new frame of reference. In the case of a graph-based space, an (graph-)isomorphism is applied to the graph before extracting a spectrum so as to get rid of any information relying on vertices identifiers. Any SSYNC demonic action in this context will simply return for each robot:

- either `None` if it is not activated, or
- `Some` f if the robot is activated, where f is a function that, depending on the position of the robot, provides the isomorphism to apply before computing a spectrum for it.

4.2 Exploration with Stop

The problem of terminating exploration requires the robots to visit collectively all the vertices in the graph, and to stop eventually once the exploration is complete. To formalise this specification, we provide a predicate `Will_be_visited` such that, if `v` is a vertex and `exc` is an execution, `Will_be_visited v e` is true if and only if at least one robot visits the vertex `v` in at least one round of `exc`:

```
Definition Will_be_visited v exc :=
  Stream.eventually (Visited_now v) exc.
```

where the predicate `Visited_now v` is true of any execution `exc` whose head configuration has a robot on the vertex `v`.

We characterise a terminating execution by the existence of a round from which the execution is *stopped*, that is from which it *stalls* forever. This is easily achieved by combining the usual temporal operators:

```
Definition Stall (exc : execution) :=
  Config.eq (hd exc) (hd (tl exc)).

Definition Stopped (exc : execution) :=
  Stream.forever Stall exc.

Definition Will_stop (exc : execution) :=
  Stream.eventually Stopped exc.
```

where `hd` and `tl` denote as usual the first element and the remaining elements of a stream.

Then we can characterise a solution to the exploration with stop to be a robogram such that, for any admissible initial configuration and for any admissible demon, the execution of the robogram satisfies the previous properties. We consider an initial configuration `c` to be admissible, or *valid*, whenever it has no tower, that is whenever no two robots are on the same vertex. The corresponding predicate is named `Valid_starting_conf` in the Coq development. We consider a demon to be admissible when it is fair, which we define in two steps as follows: the proposition `LocallyFairForOne r d` specifies that the demon `d` will activate the robot `r` within a finite number of rounds (be it *now* or *later*), and the proposition `Fair d` requires `LocallyFairForOne` to hold forever for any robot. In the following code, the function `step` applies to a demonic action `da` and a robot `r` and returns the activation information for `r`.

```
Inductive LocallyFairForOne r d :=
  | Now   : step (hd d) r ≠ None →
            LocallyFairForOne r d
  | Later : step (hd d) r = None →
            LocallyFairForOne r (tl d) →
            LocallyFairForOne r d.

Definition Fair d :=
  Stream.forever
    (fun d ⇒ ∀ r, LocallyFairForOne r d) d.
```

Finally, we can write the predicate that characterises the solutions to the exploration with stop:

```
Definition Explores_and_stops (rbg : robogram) :=
  ∀ (c : configuration) (d : demon),
    Valid_starting_conf c →
    Fair d →
    (∀ l, Will_be_visited l (execute rbg d c))
    ∧ Will_stop (execute rbg d c).
```

4.3 Use Case 1: k Divides n

The first case study consists in establishing that the aforementioned exploration is impossible to realise when the number k of robots divides the size n of the ring. The main theorem to be proved can be directly stated:

```
Theorem no_exploration_k_divides_n :
  (n mod k) = 0 → ∀ rbg, ¬ (Explores_and_stops rbg).
```

4.3.1 *Expressing the counter-example.*

We assume that k divides n, and we define an initial configuration where robots are equidistant from each other, that is here each robot is at n/k from the previous one, as well as a demon that always activates all the robots.

For this we rely again on basic arithmetic on $\mathbb{Z}/n\mathbb{Z}$ to assign a vertex to each robot: the function `loc_equi` takes as parameter a robot `r`, that is an element of type `Fin.t k`, to which an integer $\texttt{r}_{\mathbb{Z}}$ between 0 and $k-1$can be associated. Then the integer $\texttt{r}_{\mathbb{Z}}$ characterising the robot can be multiplied by n/k to produce the integer characterising the vertex on which the robot is. Finally, the function `loc_equi` returns the obtained vertex.

The initial configuration `conf_equi` is then obtained by assigning to each robot the vertex given by the previous function:

```
Definition conf_equi := fun r ⇒
  match r with
  | Good r ⇒ {| loc   := loc_equi r;
                state := tt        |}
  | Byz r  ⇒ {| loc   := origin;
                state := tt    |}
  end.
```

This definition reveals two generic aspects of Pactole. Firstly, a robot r can be either a *good*, properly functioning robot, or a *byzantine*, potentially adversary robot. However, the impossibility result we formalise in this paper holds even without any byzantine robot, and so we may assume in the development that all robots are good. The locations assigned to hypothetical byzantine robots (here origin, the vertex corresponding to the integer 0) are thus irrelevant, and the case of a byzantine robot will always be discarded straight away without being a burden for the proof.

Secondly, the configuration might assign not only a location loc, but also a *state* to each robot, the pair of these two elements being called the *status* of a robot. Here again, we do not use this possibility, since we only deal with oblivious robots. Any robot can be given a neutral state tt, and this parameter of the model becomes essentially transparent in the proof.

Some basic properties can be derived on the configuration conf_equi, such as the fact that a vertex is inhabited if and only if the corresponding integer is divisible by n/k, and that an inhabited vertex contains only one robot. This implies in particular that conf_equi is a valid initial configuration:

```
Lemma equi_valid : Valid_starting_conf conf_equi.
```

We now define a demon which, taken together with the configuration conf_equi, will define a counter-exemple to exploration with stop for any algorithm. Our goal is to ensure that, if a robot moves, then all of them move the same way, leading to a new configuration that is indistinguishable from the initial configuration conf_equi. Hence the demon should ensure that all the robots: (1) are activated at each round, (2) perceive the same spectrum, and (3) move in the same direction. To give the same perception to all robots, we define for every vertex v an isomorphism trans v that rotates the whole graph in such a way that the vertex v is mapped to origin. The name trans comes from the fact that, on the space $\mathbb{Z}/n\mathbb{Z}$ underlying the graph, this "rotation" is implemented as a translation: the same value $v_\mathbb{Z}$ is subtracted from every vertex to compute its image.

Finally, we define as follows a unique demonic action da_equi that is intended to be repeated at each round of an execution:

```
Definition da_equi : demonic_action :=
{|
  relocate_byz := fun r ⇒ origin;
  step := fun r ⇒
          Some (fun r_status ⇒ trans (loc r_status))
|}.

Definition demon_equi : demon :=
  Stream.constant da_equi.
```

The demonic action da_equi gives a default destination to byzantine robots (again this is irrelevant in our case), and the interesting part is the function step, that decides whether each robot is activated or not and defines the isomorphism applied to the perception of this robot: applied to any robot r, the function activates r by returning a value Some f that is independent of r. The chosen f is a function that gives to any robot the new frame of reference obtained with trans and placing r at the origin[4].

Since the demon demon_equi activates all robots at each round, it is straightforwardly proved to be fair.

```
Lemma equi_fair : Fair demon_equi.
```

Actually, demon_equi fits into the FSYNC model, which is even stronger and will allow us to conclude that the exploration with stop is impossible when k divides n, even assuming full synchronisation.

4.3.2 The proof.

An interesting property of our initial configuration conf_equi and of the frame of reference provided by the demonic action da_equi to the robots is that every robot in conf_equi will have the same perception of its environment (that is, it will compute its destination based on the same spectrum).

```
Lemma same_Spectrum : ∀ r₁ r₂,
  spect_equi conf_equi r₁ = spect_equi conf_equi r₂.
```

where spect_equi c r is the spectrum computed for the robot r in the configuration c based on the frame of reference given to r by the demonic action da_equi: it is the multiset of the inhabited positions translated to place r on the vertex 0_V.

From this preliminary property we deduce that, in the configuration conf_equi and under the demonic action da_equi, any robogram rbg decide the same move for all the robots. Moreover, in the ring the possible moves are restricted to: staying still, going forward, or going backward.

```
Lemma ring_range : ∀ (rbg : robogram) (r : robot),
  let m := rbg (spect_equi config_equi r) in
  m = 1_V ∨ m = 0_V ∨ m = (-1)_V.
```

These three possibilities define a classification of the algorithms, and we check in each of these cases that rbg does not solve the exploration with stop.

Let rbg be a robogram, and m the common move computed by rbg for every robot.

Case $m = 0$. In that case we first check that the execution stalls forever on the configuration conf_equi.

```
Lemma conf_equi_stalls :
  round rbg da_equi conf_equi = conf_equi.
```

Then we deduce that there is at least one vertex that will never be visited:

```
Lemma conf_equi_no_expl : ∃ v,
  ¬ Will_be_visited v (execute rbg demon_equi conf_equi).
```

The vertices that will never be visited are exactly those vertices that are not inhabited in the initial configuration conf_equi. This is for instance the case of the vertex 1_V.

[4]the actual step function is wrapped in a lift function that extends the function presented here to deal with byzantine robots (that is, to ignore them).

Case $m = 1$. In this case each of the robots will move to the neighbour vertex on its right. As a result, the new configuration after one round will be isomorphic to the initial configuration `conf_equi`, through the translation function `trans (-1)`$_V$ we already presented. To conclude the proof of this case, we prove that the successive configurations of the execution will be similar enough to each other for all the robots to keep on moving the same way forever. Hence the execution will never stop in this case.

Central to this proof is a notion of equivalent configurations that contains all the configurations that will be visited in the execution. We define `equiv_conf c₁ c₂` to hold when the configuration `c₂` can be obtained by applying, for some `v`, the isomorphism `trans v` to `c₁`.

The main property of this equivalence relation is that, under the demonic action `da_equi`, a robot perceives the same spectrum in any two equivalent configurations. In particular, this applies to `conf_equi` and any configuration `c` equivalent to `conf_equi`:

```
Lemma equiv_spectrum : ∀ c,
  equiv_conf c conf_equi →
  ∀ r, spect_equi c r = spect_equi conf_equi r.
```

Then we have to state and prove the fact that an execution starting from `conf_equi` and scheduled by the demon `demon_equi` will traverse only configurations that are equivalent to `conf_equi`:

```
Definition Always_equiv c e :=
  Stream.forever (fun e1 ⇒ equiv_conf c (hd e1)) e.

Lemma Always_equiv_equi :
  Always_equiv conf_equi
               (execute rbg demon_equi conf_equi).
```

From this we deduce that all the robots keep moving, and that the execution is never stopped. This proves that `rbg` does not solve exploration with stop, and applies to the whole class of robograms defined by $m = 1$.

The case $m = -1$ is similar. This concludes the proof of theorem `no_exploration_k_divides_n`: no algorithm can solve the exploration with stop of ring of size n by k robots when k divides n.

4.4 Use Case 2: Towers Are Required

The second case study consists in establishing that the aforementioned exploration is impossible to realise when there are less that two robots. The theorem to be proved is:

```
Theorem no_exploration_k_inf_2 :
  ∀ rbg, Explores_and_stops rbg → k > 1.
```

The most important lemma in this proof expresses that a robogram solving exploration with stop cannot stop at a configuration that would be admissible as initial configuration:

```
Lemma no_stop_on_starting_conf : ∀ rbg c d,
  Explores_and_stops rbg →
  Valid_starting_conf c →
  Fair d →
  ¬ Stopped (execute rbg c d).
```

Indeed, if we assume that the robogram `rbg` stops on some valid starting configuration `c` when scheduled by a fair demon `d`, then we can take the configuration `c` and the demon `d` as a counter-example to the fact that `rbg` solves exploration with stop.

5 CONCLUSION

We presented a dedicated formal framework for certifying results about mobile oblivious robots evolving in discrete spaces (*a.k.a.* graphs). Emphasis was put on generality and modularity, in order to enable expressing various other problems that are relevant in this context (*e.g.* exclusive perpetual exploration, gathering, graph searching, etc.). As case studies for our framework, we considered foundational impossibility results for the exploration with stop problem, and certified those results correct with the Coq proof assistant. We would like to mention two intriguing open problems:

1. *Certifying positive results*, that is, certifying the correctness of algorithms for robots operating in discrete space. This problem was previously investigated using model checking [4] and program synthesis [29], but only for fixed size instances. Now, verifying arbitrary sized instances through parameterised model-checking is undecidable for non-trivial specifications [31]. By contrast, our approach seems capable to tackle the case of arbitrary sized instances.
2. *Characterising continuous vs. discrete space.* The modularity that is offered by our framework permits to abstract many notions in a uniform manner, and hint at the possibility to establish bridges between the two domains. Actual computers (and hence, robots) only operate over a discrete variable space, while the environment robots evolve in can be seen as continuous. Bridging the reality of the environment and the vision robots have about it in a meaningful certified manner would permit the development of realistic reliable software for those robots.

ACKNOWLEDGMENTS

This work was partially funded by the CNRS PEPS OCAAA 2017 project *CYBORG* and the Université Claude Bernard Lyon 1 BQR 2017 project *PREFER*.

The authors would like to thank Xavier Urbain for his decisive role in this work, and the other members of the Pactole project.

REFERENCES

[1] Cédric Auger, Zohir Bouzid, Pierre Courtieu, Sébastien Tixeuil, and Xavier Urbain. 2013. Certified Impossibility Results for Byzantine-Tolerant Mobile Robots. In *Stabilization, Safety, and Security of Distributed Systems - 15th International Symposium (SSS 2013) (Lecture Notes in Computer Science)*, Teruo Higashino, Yoshiaki Katayama, Toshimitsu Masuzawa, Maria Potop-Butucaru, and Masafumi Yamashita (Eds.), Vol. 8255. Springer-Verlag, Osaka, Japan, 178–186. https://doi.org/10.1007/978-3-319-03089-0_13

[2] Thibaut Balabonski, Amélie Delga, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. 2016. Synchronous Gathering Without Multiplicity Detection: A Certified Algorithm. In *Stabilization, Safety, and Security of Distributed Systems - 18th International Symposium, (SSS 2016) (Lecture Notes in Computer Science)*, Borzoo Bonakdarpour and Frank Petit (Eds.), Vol. 10083. Springer-Verlag, Lyon, France. https://doi.org/10.1007/978-3-319-49259-9

[3] Roberto Baldoni, François Bonnet, Alessia Milani, and Michel Raynal. 2008. Anonymous graph exploration without collision by mobile robots. *Inf. Process. Lett.* 109, 2 (2008), 98–103. https://doi.org/10.1016/j.ipl.2008.08.011

[4] Béatrice Bérard, Pascal Lafourcade, Laure Millet, Maria Potop-Butucaru, Yann Thierry-Mieg, and Sébastien Tixeuil. 2016. Formal Verification of Mobile Robot Protocols. *Distributed Computing* 29, 6 (2016), 459–487. https://doi.org/10.1007/s00446-016-0271-1

[5] Lélia Blin, Janna Burman, and Nicolas Nisse. 2017. Exclusive Graph Searching. *Algorithmica* 77, 3 (2017), 942–969. https://doi.org/10.1007/s00453-016-0124-0

[6] Lélia Blin, Alessia Milani, Maria Potop-Butucaru, and Sébastien Tixeuil. 2010. Exclusive Perpetual Ring Exploration without Chirality. In *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings (Lecture Notes in Computer Science)*, Nancy A. Lynch and Alexander A. Shvartsman (Eds.), Vol. 6343. Springer, 312–327. https://doi.org/10.1007/978-3-642-15763-9_29

[7] François Bonnet, Xavier Défago, Franck Petit, Maria Potop-Butucaru, and Sébastien Tixeuil. 2014. Discovering and Assessing Fine-Grained Metrics in Robot Networks Protocols. In *33rd IEEE International Symposium on Reliable Distributed Systems Workshops, SRDS Workshops 2014, Nara, Japan, October 6-9, 2014*. IEEE, 50–59. https://doi.org/10.1109/SRDSW.2014.34

[8] François Bonnet, Alessia Milani, Maria Potop-Butucaru, and Sébastien Tixeuil. 2011. Asynchronous Exclusive Perpetual Grid Exploration without Sense of Direction. In *Principles of Distributed Systems - 15th International Conference, OPODIS 2011, Toulouse, France, December 13-16, 2011. Proceedings (Lecture Notes in Computer Science)*, Antonio Fernández Anta, Giuseppe Lipari, and Matthieu Roy (Eds.), Vol. 7109. Springer, 251–265. https://doi.org/10.1007/978-3-642-25873-2_18

[9] Béatrice Bérard, Pierre Courtieu, Laure Millet, Maria Potop-Butucaru, Lionel Rieg, Nathalie Sznajder, Sébastien Tixeuil, and Xavier Urbain. 2015. Formal Methods for Mobile Robots: Current Results and Open Problems. *International Journal of Informatics Society* 7, 3 (2015), 101–114. http://www.infsoc.org/journal/vol07/IJIS_07_3_101-114.pdf Invited Paper.

[10] Pierre Castéran, Vincent Filou, and Mohamed Mosbah. 2009. Certifying Distributed Algorithms by Embedding Local Computation Systems in the Coq Proof Assistant. In *Symbolic Computation in Software Science (SCSS'09)*, Adel Bouhoula and Tetsuo Ida (Eds.).

[11] Jérémie Chalopin, Paola Flocchini, Bernard Mans, and Nicola Santoro. 2010. Network Exploration by Silent and Oblivious Robots. In *Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, Zarós, Crete, Greece, June 28-30, 2010 Revised Papers (Lecture Notes in Computer Science)*, Dimitrios M. Thilikos (Ed.), Vol. 6410. 208–219. https://doi.org/10.1007/978-3-642-16926-7_20

[12] François Bonnet, Maria Potop-Butucaru, and Sébastien Tixeuil. 2016. Asynchrnous gathering in rings with four robots. In *Ad-hoc, Mobile, and Wireless Networks - 15th International Conference, ADHOC-NOW 2015, Lille, France, 2016, Proceedings (Lecture Notes in Computer Science)*. Springer.

[13] Thierry Coquand and Christine Paulin-Mohring. 1990. Inductively Defined Types. In *International Conference on Computer Logic (Colog'88) (Lecture Notes in Computer Science)*, Per Martin-Löf and Grigori Mints (Eds.), Vol. 417. Springer-Verlag, 50–66.

[14] Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. 2015. Impossibility of Gathering, a Certification. *Inform. Process. Lett.* 115 (2015), 447–452. https://doi.org/10.1016/j.ipl.2014.11.001

[15] Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil, and Xavier Urbain. 2016. Certified Universal Gathering Algorithm in $\mathbb{R}^2$ for Oblivious Mobile Robots. In *Distributed Computing - 30th International Symposium, (DISC 2016) (Lecture Notes in Computer Science)*, Cyril Gavoille and David Ilcinkas (Eds.), Vol. 9888. Springer-Verlag, Paris, France.

[16] Gianlorenzo D'Angelo, Alfredo Navarra, and Nicolas Nisse. 2017. A unified approach for gathering and exclusive searching on rings under weak assumptions. *Distributed Computing* 30, 1 (2017), 17–48. https://doi.org/10.1007/s00446-016-0274-y

[17] Gianlorenzo D'Angelo, Gabriele Di Stefano, Alfredo Navarra, Nicolas Nisse, and Karol Suchan. 2015. Computing on Rings by Oblivious Robots: A Unified Approach for Different Tasks. *Algorithmica* 72, 4 (2015), 1055–1096. https://doi.org/10.1007/s00453-014-9892-6

[18] Stéphane Devismes, Anissa Lamani, Franck Petit, Pascal Raymond, and Sébastien Tixeuil. 2012. Optimal Grid Exploration by Asynchronous Oblivious Robots. In *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium (SSS 2012) (Lecture Notes in Computer Science)*, Andréa W. Richa and Christian Scheideler (Eds.), Vol. 7596. Springer-Verlag, Toronto, Canada, 64–76.

[19] Stéphane Devismes, Anissa Lamani, Franck Petit, and Sébastien Tixeuil. 2015. Optimal Torus Exploration by Oblivious Robots. In *Networked Systems - Third International Conference, NETYS 2015, Agadir, Morocco, May 13-15, 2015, Revised Selected Papers (Lecture Notes in Computer Science)*, Ahmed Bouajjani and Hugues Fauconnier (Eds.), Vol. 9466. Springer, 183–199. https://doi.org/10.1007/978-3-319-26850-7_13

[20] Stéphane Devismes, Franck Petit, and Sébastien Tixeuil. 2013. Optimal probabilistic ring exploration by semi-synchronous oblivious robots. *Theoretical Computer Science* 498 (2013), 10–27. https://doi.org/10.1016/j.tcs.2013.05.031

[21] Ha Thi Thu Doan, François Bonnet, and Kazuhiro Ogata. 2016. Model Checking of a Mobile Robots Perpetual Exploration Algorithm. In *Structured Object-Oriented Formal Language and Method - 6th International Workshop, SOFL+MSVL 2016, Tokyo, Japan, November 15, 2016, Revised Selected Papers (Lecture Notes in Computer Science)*, Shaoying Liu, Zhenhua Duan, Cong Tian, and Fumiko Nagoya (Eds.), Vol. 10189. 201–219. https://doi.org/10.1007/978-3-319-57708-1_12

[22] Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. 2010. Remembering without memory: Tree exploration by asynchronous oblivious robots. *Theoretical Computer Science* 411, 14-15 (2010), 1583–1598. https://doi.org/10.1016/j.tcs.2010.01.007

[23] Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. 2013. Computing Without Communicating: Ring Exploration by Asynchronous Oblivious Robots. *Algorithmica* 65, 3 (2013), 562–583. https://doi.org/10.1007/s00453-011-9611-5

[24] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. 2012. *Distributed Computing by Oblivious Mobile Robots*. Morgan & Claypool Publishers.

[25] Tomoko Izumi, Taisuke Izumi, Sayaka Kamei, and Fukuhito Ooshita. 2010. Mobile Robots Gathering Algorithm with Local Weak Multiplicity in Rings. In *Structural Information and Communication Complexity, 17th International Colloquium, SIROCCO 2010, Sirince, Turkey, June 7-11, 2010. Proceedings (Lecture Notes in Computer Science)*, Boaz Patt-Shamir and Tinaz Ekim (Eds.), Vol. 6058. Springer, 101–113. https://doi.org/10.1007/978-3-642-13284-1_9

[26] Sayaka Kamei, Anissa Lamani, Fukuhito Ooshita, and Sébastien Tixeuil. 2011. Asynchronous Mobile Robot Gathering from Symmetric Configurations without Global Multiplicity Detection. In *Structural Information and Communication Complexity - 18th International Colloquium, SIROCCO 2011, Gdansk, Poland, June 26-29, 2011. Proceedings (Lecture Notes in Computer Science)*, Adrian Kosowski and Masafumi Yamashita (Eds.), Vol. 6796. Springer, 150–161. https://doi.org/10.1007/978-3-642-22212-2_14

[27] Sayaka Kamei, Anissa Lamani, Fukuhito Ooshita, and Sébastien Tixeuil. 2012. Gathering an Even Number of Robots in an Odd Ring without Global Multiplicity Detection. In *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings (Lecture Notes in Computer Science)*, Branislav Rovan, Vladimiro Sassone, and Peter Widmayer (Eds.), Vol. 7464. Springer, 542–553. https://doi.org/10.1007/978-3-642-32589-2_48

[28] Anissa Lamani, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. 2010. Optimal Deterministic Ring Exploration with Oblivious Asynchronous Robots. In *Structural Information and Communication Complexity, 17th International Colloquium, SIROCCO 2010, Sirince, Turkey, June 7-11, 2010. Proceedings (Lecture Notes in Computer Science)*, Boaz Patt-Shamir and Tinaz Ekim (Eds.), Vol. 6058. Springer, 183–196. https://doi.org/10.1007/978-3-642-13284-1_15

[29] Laure Millet, Maria Potop-Butucaru, Nathalie Sznajder, and Sébastien Tixeuil. 2014. On the Synthesis of Mobile Robots Algorithms: The Case of Ring Gathering. In *Stabilization, Safety, and Security of Distributed Systems - 16th International Symposium, (SSS 2014) (Lecture Notes in Computer Science)*, Pascal Felber and Vijay K. Garg (Eds.), Vol. 8756. Springer-Verlag, Paderborn, Germany, 237–251. https://doi.org/10.1007/978-3-319-11764-5_17

[30] Sasha Rubin, Florian Zuleger, Aniello Murano, and Benjamin Aminof. 2015. Verification of Asynchronous Mobile-Robots in Partially-Known Environments. In *PRIMA 2015: Principles and Practice of Multi-Agent Systems - 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings (Lecture Notes in Computer Science)*, Qingliang Chen, Paolo Torroni, Serena Villata, Jane Yung-jen Hsu, and Andrea Omicini (Eds.), Vol. 9387. Springer-Verlag, 185–200. https://doi.org/10.1007/978-3-319-25524-8_12

[31] Arnaud Sangnier, Nathalie Sznajder, Maria Potop-Butucaru, and Sébastien Tixeuil. 2017. Parameterized Verification of Algorithms for Oblivious Robots on a Ring. In *Formal Methods in Computer Aided Design*. Vienna, Austria. http://arxiv.org/abs/1706.05193

[32] Ichiro Suzuki and Masafumi Yamashita. 1999. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *SIAM Journal of Computing* 28, 4 (1999), 1347–1363.