[Invited Paper] Formal Methods for Mobile Robots: Current Results and Open Problems

Béatrice Bérard[†#], Pierre Courtieu[‡], Laure Millet[†#], Maria Potop-Butucaru[†#], Lionel Rieg[*], Nathalie Sznajder[†#], Sébastien Tixeuil[†#‖], and Xavier Urbain[⋆]

[†]UPMC Sorbonne Universités, LIP6-CNRS UMR 7606, France
[#] CNRS, LIP6-CNRS UMR 7606, France
[‡]CNAM, Cédric EA 4629, France
[*] Collège de France, France
[‖] Institut Universitaire de France, France
[⋆] ENSIIE, Cédric EA4629 and LRI CNRS UMR 8623, France

Abstract - Mobile robot networks emerged in the past few years as a promising distributed computing model. Existing work in the literature typically ensures the correctness of mobile robot protocols via *ad hoc* handwritten proofs, which are both cumbersome and error-prone.

This paper surveys state-of-the-art results about applying formal methods approaches (namely, model-checking, program synthesis, and proof assistants) to the context of mobile robot networks. Those methods already proved useful for bug-hunting in published literature, designing correct-by-design optimal protocols, and certifying impossibility results. We also present related open questions to further develop this path of research.

Keywords: Formal Methods, Mobile Robots, Distributed Algorithms, Model Checking, Program Synthesis, Proof Certification, Proof Assistant.

1 INTRODUCTION

The variety of tasks that can be performed by autonomous robots and their complexity are both increasing [1], [2]. Many applications envision groups of mobile robots that are self-organising and cooperating toward the resolution of common objectives, in the absence of any central coordinating authority.

The seminal model introduced by Suzuki and Yamashita [3] features a distributed system of k mobile robots that have limited capabilities: they are identical and *anonymous* (they execute the same algorithm and they cannot be distinguished using their appearance), they are *oblivious* (they have no memory of their past actions) and they have neither a common sense of direction, nor a common *handedness* (chirality). Furthermore these robots do not communicate by sending or receiving messages. However they have the ability to sense the environment and see all positions of the other robots.

Robots operate in cycles of three phases: *Look*, *Compute* and *Move*. During the *Look* phase robots take a snapshot of the positions of the other robots (in their own coordinate system). The collected information is used in the *Compute* phase where robots decide to move or to stay idle. In the *Move* phase, robots may move according to the computation of the previous phase.

In the original model [3], some non-empty subset of robots execute the three phases synchronously and atomically, giving rise to two variants: FSYNC, for the fully-synchronous model where all robots are scheduled at each step to execute a full cycle, and SSYNC, for the semi-synchronous model, where a strict subset of robots can be scheduled. This model had a huge impact on the community and was instrumental in deriving many new core problems in the area of distributed mobile entities. It was later generalised by Flocchini *et al.* [4] to handle full asynchrony and remove atomicity constraints (this model is called ASYNC [1], for asynchronous, in the sequel). One of the key differences between the fully- or semi-synchronous models, and the asynchronous model in the discrete setting is that in the ASYNC model, a robot can compute its next move based on an *outdated* view of the system. It is notorious that handwritten proofs for protocols operating in the ASYNC model are hard to write and read, due to many instances of case-based reasoning that is both cumbersome and error-prone.

Outline. The goal of the survey is to present recent advances in using formal methods for mobile robots following the model of Suzuki and Yamashita and derivatives. Formal methods are needed to certify that obtained results are correct, as previously published solutions were in fact incorrect.

We consider in this paper three main proposals in the domain of formal methods: model-checking, algorithm synthesis, and proof assistants.

The model itself may seem limited (robots have extremely few capabilities, compared to real life robots), but it permits to establish fundamental results (what are the tasks that are feasible, and what are those which are not). That is, it is a computability-centric model (as opposed to a efficiency-centric model).

In Section 2, we recall basic notions on model-checking, synthesis and games and proof assistants. We also briefly describe previous work on formal methods applied to robot algorithms. We present in Section 3.1 an overview of the various settings, as well as the formal models used in the sequel. In Section 4, we survey results in the three directions of model-checking, synthesis and proof-assistants. We conclude in Section 5 with several problems open for future research.

2 PRELIMINARIES

2.1 Model Cheking

Model-checking [5], [6] is an appealing technique that was developed for the verification of various models: finite ones but also in some cases infinite, parameterised, or even timed and probabilistic models. It has been successfully used for the verification of distributed systems from classical shared memory (consensus, transactional memory) to population protocols [7]–[12]. Unfortunately, it was proved in [13] that parameterised model checking is undecidable, and this general result was followed by several stronger ones for specific models, for instance in [14]. In such cases, a classical line of work consisted in combining model-checking with other techniques like abstraction, induction, etc., as first proposed in [15] or [16]. These techniques were largely used since, for instance in [17]–[20]. Although the problem is still open, we conjecture that parameterised model checking is undecidable for the robot model which leads to follow combined approaches.

2.2 Games and Protocols Synthesis

In the formal methods community, automatically synthesising programs that would be correct by design is a problem that raised interest early [21]–[24]. Actually, this problem goes back to Church [25], [26]. When the program to generate is intended to work in an open system, maintaining an on-going interaction with a (partially) unknown environment, it is known since [26] that seeing the problem as a *game* between the system and the environment is a successful approach. The system and its environment are considered as opposite players that play a game on some graph, the winning condition being the specification the system should fulfill whatever the environment behavior. 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. The case of mobile autonomous robots that we focus on in this paper falls in this category of problems: the robots may evolve (possibly indefinitely) on a ring, making decisions based on some global state of the system at each time instant. The vertices of graph on which the players will play would then be some representation of the different global positions of the robots on the ring. The presence of an opposite player (or environment) is motivated by the absence of chirality of the robots: when a robot is on an axis of symmetry, it is unable to distinguish its two sides one from another, hence to choose exactly *where* it moves ; this decision is supposed to be taken by the opposite player.

2.3 Certification and Proof Assistants

Mechanical proof assistants are proof management systems in which a user can express data, programs, theorems and proofs. In sharp contrast with automated provers, they are mostly interactive, and thus require some kind of expertise from their users. Skeptical proof assistants provide an additional guarantee by checking mechanically the soundness of proofs *after* it has been interactively developped.

Various proof assistants emerged since the 60's, to name a few: Agda [27], NqThm [28] and its relative ACL2 [29], PVS [30], Mizar [31], COQ [32], Isabelle/HOL [33], etc.

In the context of program verification, Isabelle/HOL and COQ are amongst the most widely used; both are based on type theory. They have been successfully employed for various tasks such as the formalisation of programming language semantics [34], certification of an OS kernel [35], verification of cryptographic protocols [36], certification of RSA keys [37], mathematical developments as involved as the 4-colours threorem [38], the Feit-Thompson theorem [39], or the Kepler Conjecture [40].

During the last twenty years, the use of tool-assisted verification has extended to the validation of distributed processes.

In the context of process algebras, which can be used to describe and verify algorithms built from merge, sequential composition and encapsulation, Fokkink [41] and Bezem *et al.* [42] use a proof assistant to prove the equality between two processes, one of them being a specification.

TLA/TLAPS [43], [44] can enjoy an Isabelle back-end for its provers [45]. Gascard and Pierre [46] focus on interconnection networks that are symmetric: rings, tori, hypercubes. Based on a compositional approach of certified components, their work makes use of Nqthm.

Cansell and Méry's contribution to the RIMEL project [47] addresses the class of local computation (LC) algorithms. A catalogue of case studies like election algorithms, spanning tree construction and even Mazurkiewicz's enumeration algorithm have been developed in Event-B. The code of these algorithms is obtained by successive refinements starting from an abstract machine that translates directly to a specification. This code is *annotated* with logical formulas — mainly invariants on the state of the system — the proofs of which generate verification conditions through a calculus of weakest preconditions.

Küfner *et al.* [48] propose a methodology to develop (using Isabelle) proofs of properties of fault-tolerant distributed algorithms in an asynchronous message passing style setting. They focus on correctness proofs only.

Chou's methodology [49] is based on the HOL proof assistant. It aims at proving properties of concrete distributed algorithms through simulation with abstract ones. The methodology does not allow to prove impossibility results.

Castéran *et al.* [50] use COQ to state and prove invariants but also generic results about subclasses of LC systems, thanks to Castéran and Filou's library Loco [51]. Genericity is worth emphasising here as the approach is not limited to *particular instances* of algorithms. Castéran *et al.* actually propose proofs of negative results in COQ for some kinds of distributed algorithms in this graph relabelling setting.

Deng and Monin [52] use COQ to prove the correctness of distributed self-stabilising protocols in the population protocol model. This model permits to describe interactions of an arbitrary large size of mobile entities, however the considered entities lack movement control and geometric awareness that are characteristic of robot networks.

As a matter of fact, surprisingly few works consider using mechanised assistance for networks of *mobile entities*.

2.4 Previous Attempts for Mobile Robots

To our knowledge, in the context of mobile robots operating in discrete space, only two previous attempts, by Devismes *et al.* [53] and by Bonnet *et al.* [54], [55], investigate the possibility of automated verification of mobile robots protocols. The first paper uses LUSTRE [56] to describe and verify the problem of exploration with stop of a 3×3 grid by 3 robots in the SSYNC model, and to show by exhaustive searching that no such protocol can exist. The second paper considers the perpetual exclusive exploration by k robots of n-sized rings, and generates mechanically all *unambiguous* protocols for k and n in the SSYNC model (that is, all protocols that do *not* have symmetrical configurations). Those two works are restricted to the simpler SSYNC model rather than the more general and more complex ASYNC model. Second, they are either specific to a hard-coded topology (*e.g.*, a 3×3 grid [53]) that prevents easy reuse in more generic situations, or make additional assumptions about configurations and protocols to be verified (*e.g.* unambiguous protocols [54], [55]) that prevent combinatorial explosion but forbid reuse for proof-challenging protocols, which would most benefit from automatic verification.

3 FORMAL MODELLING

This section reviews the classical model for mobile robots that is due to Suzuki and Yamashita [3] (Section 3.1), then surveys formal modelling schemes that are tailored for model-cheking (Section 2.1), protocol synthesis (Section 3.3), and proof assistants (Section 3.4).

3.1 A Model for Mobile Robot Networks

Robots We consider a set of k mobile entities called robots, that are endowed with sensing, computing, and moving capabilities. They can observe (sense) the positions of other robots in the space they evolve in and based on these observations, they perform some local computations that can drive them to move to other locations.

Sensing Robots are usually endowed with visibility sensors that permit them to obtain the location of other robots. The obtained location is either *fine grained* (which usually denotes an arbitrary degree of precision in the such obtained location) or *coarse grained* (robots can only be observed at some specific discrete locations, each location being adjacent to at least one another). In the first case, the literature mostly refers to the *continuous space* model, while in the latter case, it is the *discrete space* model.

In some problem instances, robots may share the same position, which is called a *multiplicity point* or a *tower*. The ability for a robot to detect multiplicity is crucial to solve 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. The multiplicity detector may be local or global. The *local* detector returns information only at the current position of the robot, while the *global* multiplicity detector return information about all observed positions.

A third characteristic of robot sensing capabilities is their visibility radius. It can be *infinite* (that is a robot is able to sense the position of all other robots) or finite. In the latter case, there exists a bound (that can be expressed either by a distance – in case of continuous space, or by a number of hops – in case of discrete space) beyond which a robot cannot sense anything.

It should be understood that all the sensing performed by robots are presented in the robot's own ego-centric coordinate system (that is called the local coordinate system in the sequel). The local coordinate systems of robots is not necessarily the same for all robots with respect to *origin* (the local coordinate systems are self-centered), *direction* (all robots need not agree on a common vertical north direction), and *chirality* (robots may have different sensing of left and right).

Computing As in classical distributed systems, robots are assumed to be able to perform the computing steps in negligible time.

Robots may be *oblivious* in the sense that they do not remember previously executed steps. Hence, volatile memory can be used to perform computing tasks in a single Look-Compute-Move loop, but the contents of the memory used in the computation are erased before the next loop occurs. By contrast, robots may have non-volatile memory: in this case they are *non-oblivious*.

Moving Robots may move only to the location computed in the computing phase of the current loop. In some instances, due to symmetry, the computed location may be ambiguous. To model this case, it is assumed that the actual move is decided by an adversary (also called demon, or scheduler). The demon can be viewed as an opponent in the game context. 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. With the *rigid* assumption, a move always perfoms to completion (that is, the robot is never interrupted). In the original model, a robot may be interrupted by an adversary before it finishes its move, but not before it has moved at least a minimum distance $\delta > 0$, where δ is a parameter of the model (unknown to the robots).

Atomicity There are two main models for atomicity. The historical model is the atomic model, where look-compute-move loops are executed in a lock-step fashion. In particular, in the atomic model, the robots that are selected for execution all sense at the same time, all compute at the same time, all move at the same time. In the current terminology [1], the atomic model is either referred to as the FSYNC (in the case where *all* robots execute at the same time) or as the SSYNC (in the case where a non-empty subset of the robots execute at the same time) model.

A less constrained model is the asynchronous and non-atomic model (or ASYNC in the current terminology [1]),

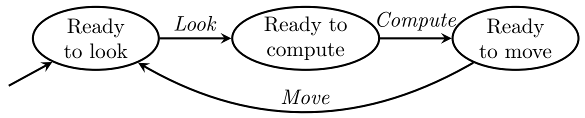


Figure 1: A generic automaton for the robot behaviour.

where robots look-compute-move loops are completely non-atomic and can each last an arbitrary period of time. In particular, in the ASYNC model, it is possible for a robot to observe another robot while it moves, or to perform the computing (and moving) phase with an observation that is long outdated. Of course, all executions in the atomic model are also valid in the ASYNC model. Thus, impossibility results for the atomic model extend in the ASYNC model, and protocols for the ASYNC model are also valid for the atomic model, but the converse is not true.

Demons Demons are an abstraction to characterise the degree of asynchrony in the robot network [57]. Demons can be seen as a predicate on system executions, that is, only executions matching the demon predicate can appear in a given context. The larger the set of executions in the predicate is, the more powerfull (and more general) the demon is. The most general demon in the context of mobile robots is the *fair* demon, which guarantees that in any configuration, any robot is activated within a finite number of steps. If the demon is k-*fair*, then between any two actions of a particular robot, any other robot is activated at most k times. Finally, the *synchronous* demon activates all robots all the time, always.

Faults Robots usually operate without failures (in which case they are said to be *correct*). Yet, some unexpected behaviours may occur. In the worst case, robots are *Byzantine*, meaning that they can behave arbitrarily. Note that to have an impact on the others, the only part of the misbehaviour to take into account is the move part. A less serious fault is the *crash* fault, where a robot unexpectedly stops moving forever.

3.2 A Formal Model for Robots on Graphs

In this section we describe the model proposed by Bérard *et al.* for the robots (in Section 3.2.1), the demons (in Section 3.2.2), and the system resulting from their interactions (in Section 3.2.3). This model encompasses all three FSYNC, SSYNC, and ASYNC operating modes, but assumes that individual robots can only operate in a discrete setting (that is, a graph).

3.2.1 Robot Modelling

All robots execute the same algorithm [1], hence the behaviour of each of them can be described by the finite automaton of Fig. 1. They operate in *Look*, *Compute*, and *Move cycles*.

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 along an edge of the graph according to its previous computation, this effective movement is represented by the *Move* transition.

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.

Note that the original model [3] 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. This 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".

3.2.2 Demon Modelling

Unlike robots that have the same behaviour regardless of the model, the demon is parameterised by the execution model and by the number of robots. It is also modelled by a finite automaton, one for each variant of the execution model. By synchronising one of these demons with robot automata, we obtain an automaton that represents the global behaviour of robots in the chosen model.

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

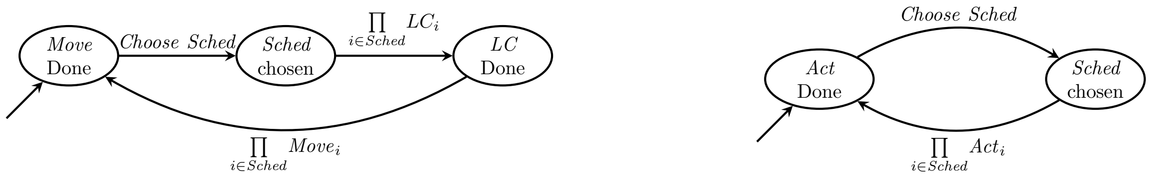
In the SSYNC model, a non-empty subset of robots is scheduled for execution at every phase, and operations are executed synchronously. In this case, the automaton is a cycle, where a set $\mathit{Sched} \subseteq \mathit{Rob}$ is first chosen. In this cycle the *LC* and *Move* phases are synchronised for this set of robots. A generic automaton for SSYNC is described in Fig. 2(a). 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 instance of the SSYNC model, where all robots are scheduled for execution at every phase, and operate synchronously thereafter: In each global cycle, $\mathit{Sched} = \mathit{Rob}$, hence all global cycles are identical.

The ASYNC model is totally asynchronous: 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 demon is depicted in Fig. 2(b).

3.2.3 System Modelling

To describe the global model, we denote by $\mathit{Pos} = \{0, \ldots, n-1\} \subseteq \mathbb{N}$ the set of positions on the graph. A configuration of the system is a mapping $c : \mathit{Rob} \to \mathit{Pos}$ associating with each



(a) SSYNC model.

(b) ASYNC model.

Figure 2: The Demons automata.

robot r its position $c(r) \in \mathit{Pos}$. Hence, in a graph of n nodes with k robots, there are n^k possible configurations.

The model of the system is an automaton

$$M = (S, s_0, A, T)$$

obtained by the synchronised product of k robot automata and all the possible configurations, as defined above, the demon is used to define the synchronisation function. The alphabet of actions is $A = \prod_{r_i \in \mathit{Rob}} A_i$, with $A_i = \mathit{LC}_i \cup \mathit{Move}_i$ for each robot r_i. In this product, states are of the form $s = (s_1, \ldots, s_k, c)$ where s_i is the local state of robot r_i, and c the configuration. An initial state is of the form $s_0 = (s_{1,0}, \ldots, s_{k,0}, c)$ where $s_{i,0}$ is the initial local state of robot r_i and c is an arbitrary configuration.

A transition of the system is labelled by a tuple

$$a = (a_1, \ldots, a_k)$$

where $a_i \in A_i \cup \{\varepsilon, -\}$ for all $1 \leq i \leq k$ and

$$(s_1, \ldots, s_k, c) \xrightarrow{a} (s'_1, \ldots, s'_k, c')$$

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 $a_i \in \mathit{Move}_i$. To represent the scheduling, we denote by $\prod_{r_i \in \mathit{Sched}} \mathit{Act}_i$ the action $(a_1, \ldots, a_k)$ such that $a_i = -$ if $r_i \notin Sched$ and $a_i \in \mathit{LC}_i \cup \mathit{Move}_i \cup \{\varepsilon\}$ otherwise.

3.3 Protocol Synthesis and Reachability Games

To enable robot protocol synthesis (that is, the automatic generation of robot protocols for a given problem in a given setting), the approach of Millet *et al.* [58] is to reuse the modelling presented in Section 3.1 for robots, schedulers, and their interactions, and to revisit reachability games in this context.

We now present classical notions on this subject. If A is a set of symbols, A^* is the set of finite sequences of elements of A (also called *words*), and A^ω the set of infinite such sequences, with ε the empty sequence. We note $A^+ = A^* \setminus \{\varepsilon\}$, and $A^\infty = A^* \cup A^\omega$. For a sequence $w \in A^\infty$, we denote its *length* by $|w|$. If $w \in A^*$, $|w|$ is equal to its number of elements. If $w \in A^\omega$, $|w| = \infty$. For all 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 noted $w \cdot w' = a_1 \cdots a_k a'_1 \cdots$. We sometimes omit the symbol and simply write ww'. If $L \subseteq A^*$ and $L' \subseteq A^\infty$, we define $L \cdot L' = \{w \cdot w' \mid w \in L, w' \in L'\}$.

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

Arena An arena is a graph $\mathcal{A} = (V, E)$ in which the set of vertices $V = V_p \uplus V_o$ is partitioned into V_p, the vertices of the protagonist, and V_o the vertices of the opponent. 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 following, 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, i.e., a finite or infinite sequence of vertices $\pi = v_0 v_1 \cdots \in V^\infty$, where for all $0 < i < |\pi|$, $v_i \in v_{i-1}E$. Moreover, a play is finite only if the token has been taken to a position without any successor (where it is impossible to continue the game): if π is finite with $|\pi| = n$, then $v_{n-1}E = \emptyset$.

Strategies A strategy for the protagonist determines where she brings 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 protagonist is a (partial) function $\sigma : V^* \cdot V_p \to V$ such that, for all sequence (representing the current history) $w \in V^*$, all $v \in V_p$, $\sigma(w \cdot v) \in vE$ (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 function $\sigma : V_p \to V$.

Given a strategy σ for the protagonist, 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_0, the *outcome* of a strategy σ is the set of plays starting in v_0 that are σ-consistent. Formally, given an arena $A = (V, E)$, an intial vertex v_0 and a strategy $\sigma : V^* V_p \to V$, we let

$$\mathit{Outcome}(A, v_0, \sigma) = \left\{ v_0\pi \in V^\infty \mid v_0\pi \begin{array}{c}\text{is a play and}\\ \text{is } \sigma\text{-consistent}\end{array} \right\}$$

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

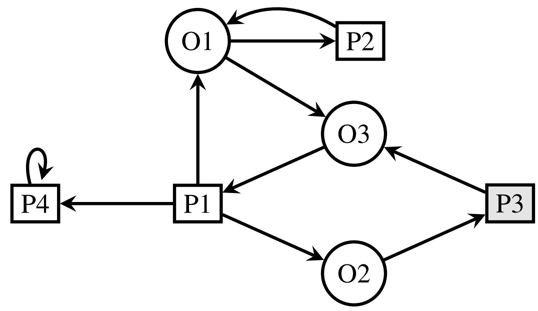


Figure 3: A two-player game. In this figure protagonist vertices are represented by rectangles and antagonist vertices by circles. The winning condition is $Reach(\{P3\})$. Any path in the graph is a play. From P2 the protagonist has no winning strategy. From P1 a (memoryless) winning strategy is to go to O2. Winning positions are $\{P1, P3\}$.

happened, the play is winning, regardless of the following actions of the players.

Given an arena $\mathcal{A} = (V, E)$, an initial vertex $v_0 \in V$ and a winning condition Win, a *winning strategy* σ for the protagonist is a strategy such that any σ-consistent play is winning. In other words, a strategy σ is winning if $Outcome(\mathcal{A}, v_0, \sigma) \subseteq Win$. The protagonist wins the game $(\mathcal{A}, v_0, Win)$ if she has a winning strategy for $(\mathcal{A}, v_0, Win)$. We say that σ is winning on a subset $U \subseteq V$ if it is winning starting from any vertex in U: if $Outcome(\mathcal{A}, v_0, \sigma) \subseteq Win$ for all $v_0 \in U$. A subset $U \subseteq V$ of the vertices is *winning* if there exists a strategy σ that is winning on U.

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

Figure 3 represents a reachability 2-player game. We recall now a well-known result [59] on reachability games:

Theorem 1 *The set of winning positions for the protagonist in a reachability game can be computed in linear time in the size of the arena. Moreover, from any position, the protagonist has a winning strategy if and only if she has a memoryless winning strategy.*

3.4 A Formal Model with Coq for Robots in Continuous Spaces

In this section, we survey the modelling in COQ that was introduced by Auger *et al.* [60], [61] and by Courtieu *et al.* [62]. This model enables to deal with FSYNC and SSYNC execution models in a two-dimensional Euclidian space setting (where coordinates are modeled by real numbers), but assumes the *rigid* model of movement, where move phases always complete.

Since there is a wide variety of different assumptions, the model must be highly flexible. The higher-order expressiveness of proof assistants allows many aspect of the model to remain abstract. In a particular setting, one may instantiate carefully the abstract parts with concrete definitions corresponding to the assumptions under consideration. We provide such examples of particular instances in the following.

The formal framework is parameterised by the following: (1) The number of correct and Byzantine robots. (2) The topological space in which robots move, i.e. the type of *locations* (infinite line, discrete grid, discrete ring network, etc). (3) The observing capabilities of robots, i.e. what kind of *spectrum* do they receive from their sensors. This is where anonymity and multiplicity assumptions are specified for example. (4) The distributed protocol running on each non-Byzantine robot, which we call the *robogram*. (5) The execution model (FSYNC, etc.) and the degree of fairness under consideration. Proof of distributed systems are supposed to state properties for any execution, i.e. for any infinite sequence of successive activations of robots that obeys the assumptions under consideration (fairness, etc.). Traditionally, such an infinite sequence is called a *demon*. Characterising the authorised executions through the definition of a given demon is one of the crucial step of instantiating our framework on a particular setting.

3.4.1 Robots

We consider the union of two given disjoint finite sets of (robot) *identifiers*: G referring to robots that behave correctly, and B referring to the set of Byzantine ones. Note that at this level, in order to express any kind of properties about programs, all robots can be identified. The behaviour of correct and byzantine robots is defined later.

```
Variable nG nB: nat.
(* Number of good and byz. robots.
Left abstract *)
Definition G := Finite nG.
(* Type of good robots *)
Definition B := Finite nB.
(* Type of byzantine robots *)
Inductive ident := Good: G → ident
| Byz: B → ident.
(* Disjoint union *)
```

In some cases the assumptions require that local robograms cannot tell robots apart (anonymity), or detect whether they are correct or Byzantine. This restriction of the model can be ensured by the notion of *spectrum*, described below, which characterises what a robot can see of the global position.

Locations, Positions, Similarities Robots are distributed in space, at places called *locations*. *Positions* are functions from the set of identifiers to the space of locations. The space of location is left abstract in the model, it can be instantiated by any type: the infinite line $\mathbb{Q}$ [60], [61] or $\mathbb{R}$ [62], the plan $\mathbb{R} \times \mathbb{R}$, a ring network $\mathbb{Z}/n\mathbb{Z}$, a line network $[i, j]$, etc.

`gp` denotes a position for correct robots, and `bp` a position for Byzantine ones. The position of all robots is then given by the combination `gp` ⊎ `bp` defined by a record in COQ.

```
Variable location : Type.
(* Space occupied by robots. Left abstract. *)
```

```
Record position := {
gp: G → location ;
bp: B → location
}.
```

Spectrum Generally speaking, robots compute their target position from the configuration they perceive of their siblings in the considered space. Depending on assumptions (*e.g.* anonymity, multiplicity detection, etc.) the observation may be more or less accurate. To allow for different assumptions to be studied, we leave the type *spectrum*, together with the notion of spectrum of a position, abstract.

```
Variable spectrum : Type.
Variable spectrum_of : position → spectrum.
```

In the following we distinguish a *demon* position (resp. spectrum), that is expressed in the global frame of reference (viewed from nominal position, orientation and zoom), from a *robot* position (resp. spectrum), that is expressed in the robot's frame of reference. At each step of the distributed protocol (see definition of `round` below) the demon position and spectrum are transformed (i.e., recentered, rotated and scaled) into the considered robots ones before being given as parameters to robograms. Depending on the assumptions under consideration, the zoom and rotation factors may be fixed for each robot or chosen by the demon at each step. They may also be shared by all robots or not, etc.

Example 1 *In a framework where anonymity holds and where robots do not enjoy multiplicity detection, one can define a spectrum as a set of robot locations (each element of the set is a location occupied by at least one robot), and* `spectrum_of` *as a function returning the set of locations occurring in its parameter* `p`.

```
Definition location := R.
Definition spectrum := set location.
Definition spectrum_of p : spectrum
    := collect_locations p.
```

Notice that a spectrum being a set *in this example, it masks the number of robots occupying the same location, thus ensuring that multiplicity is undetected. To account for multiplicity, one may define another instance where spectra are* multisets, *and* `collect_locations` *keeps record of redundant locations.*

Robogram The robogram is a function computing a target location from a spectrum.

```
Definition robogram := spectrum → location.
```

More precisely it computes the target location from the robot spectrum, that is: expressed in the robot's own frame of reference.

3.4.2 Demonic action and round

Assuming the SSYNC model, at each round the demon selects the new location of byzantine robots, the set of correct robots to be activated, and a frame for each of them. More precisely the frame is a way to change the frame of reference. Depending on the space the robots move in, it can be for instance rotation and scale factors. The type of *demonic action* is left abstract in the model but it should provide all these operations.

Example 1 (continued) *We continue on the previous example where we suppose the set of locations to be the infinite real line. The frame can be expressed by a real number as follows: the absolute value denotes the scaling with reference to the demon's point of view, a negative number means that the position is rotated (in this case: swapped), and the special* 0 *value means that the robot is actually not activated at this round.*

```
Record demonic_action := {
  locate_byz  : B → location ;
  frame       : G → R }
```

From these definitions we can formalise what it is for the distributed algorithm to perform a *round*. In an SSYNC context, a round consists in the computation of the new position of correct robots (i.e. a function of type `G →location`) from a robogram, a demonic action and the previous position. The function `round` defined below is thus a function returning a function. For each robot `g` it computes its new location by feeding the robogram with the spectrum recentered and distorted by the demon.

```
Definition round (r : robogram)
(da : demonic_action) (gp : G → location) :
G → location :=
fun g:G ⇒
  let l := gp g in
  (* current location of g *)
  let k := da.(frame) g in
  (* scale and rotation factor for g *)
  if k = 0 then l
  (* g not activated, g stays at l *)
  else
    let pos := repos gp da.(locate_byz) k l in
    (* position viewed from g *)
    let newloc := r (spectrum_of pos) in
    (* apply r on g's spectrum *)
    l + /k * newloc.
    (* Uncenter, unscale, unrotate *)
```

Where `repos gp bp k l` returns the `l`-centered, `k`-zoomed and rotated version of position `{|gp;bp|}`.

Demon, Fairness An actual *demon* is simply an infinite sequence (stream) of demonic actions, that is a coinductive object [63]. Coinductive types are of invaluable help to express in a direct way infinite behaviours, infinite datatypes and properties on them. The COQ proof assistant provides means for the developer to define *and to quantify* over both inductive and coinductive types, so as to express inductive and coinductive properties. Roughly, coinduction is used for properties that hold forever, and induction for properties that hold eventually.

```
CoInductive demon :=
  NextDemon : demonic_action → demon → demon.
```

The set of authorised demons also depends on the assumptions under consideration. For example, we define below the well-known notion of being a *fair* demon by a coinductive property over demons, which state that at each step of the demon any robot is activated after a finite number of steps.

```
Inductive LocallyFairForOne g (d : demon) :
  Prop :=
  | ImmediatelyFair :
      frame (demon_head d) g ≠ 0
            → LocallyFairForOne g d
  | LaterFair :
      frame (demon_head d) g = 0
            → LocallyFairForOne g
                    (demon_tail d)
            → LocallyFairForOne g d.

CoInductive Fair (d : demon) :
  Prop :=
  AlwaysFair :
    (∀ g, LocallyFairForOne g d)
               → Fair (demon_tail d)
               → Fair d.
```

Some of those definitions may be shortened, but this is a rather direct and generic way to express that, at each point of an *infinite* execution, a property holds *eventually*.

4 SURVEY OF RESULTS

Making use of the formal modelling presented in the previous section, recent papers were able to use formal methods to verify existing algorihtms (Section 4.1), synthesise new algorithms that are correct by design (Section 4.2), and provide certified impossibility results (Section 4.3). In this section, we review the main contributions published so far.

4.1 Model Checking

The model checking approach of Bérard *et al.* was used for studying the *Min-Algorithm* presented by Blin *et al.* [64]. The followed approach was to outline the properties that need to be satisfied for the particular problem of perpetual exclusive exploration, using LTL logic.

Problem specification The *Exclusive Perpetual Exploration* problem in [64] is defined in the general asynchronous model as follows.

For any graph G of size n and any initial configuration where robots are located on different vertices, an algorithm solves the perpetual exclusive exploration problem if it guarantees two properties: the *exclusivity* property and the *liveness* property. The first one requires that no two robots visit the same vertex or traverse the same edge at the same time, whereas the *liveness* property requires that each robot visits each vertex infinitely often.

In the considered models an execution where no robot is ever scheduled can happen, as well as an execution where a particular robot is never scheduled. To prevent such executions a fairness assumption has to be added: All robots have to be scheduled infinitely often. Thus the *liveness* property is satisfied only on executions where the *fairness* assumption holds.

Min-Algorithm In [64] the authors proposed an algorithm called *Min-Algorithm*, for $k = 3$ robots in a ring of size $n \geq 10$, such that n is not a multiple of 3. Starting from tower-free configurations (where no two robots occupy the same position), this algorithm ensures exclusive and perpetual exploration. It is based on a classification of tower-free configurations and a specific action to be taken by the robot in any recognized configuration. An equivalence class of tower-free configurations on the ring is described by a sequence of symbols R and F, indexed by integers: R_i stands for i consecutive nodes occupied by a robot, and F_j stands for j consecutive nodes free of robots. The algorithm is presented in Tables 1 and 2.

Verification The previous algorithm was modeled then implemented into the DiVinE [65] model-checker, using a ring of size 10, the smallest advertized size for the algorithm to work. The algorithm was verified to work properly in the FSYNC and SSYNC model, but a counter-example was found when run using the ASYNC model, among the 13.10^6 possible movements. This counter-example ends up in two robots colliding (and thus breaking the exclusion property), as explicited in Fig. 4.

In this counter example every ring represents a configuration, a configuration change occurs when a robot moves, in each configuration a computation is represented by a full arrow, and outdated computation by a dotted arrow.

Following the verification, a simple fix on the rule

$$RC5 \;::\; (R_2, F_1, R_1, F_z) \;\rightarrow\; (R_1, F_1, R_1, F_1, R_1, F_{z-1})$$

allowed to correct the algorithm.

4.2 Algorithm Synthesis

The algorithm synthesis approach of Millet *et al.* [58] was used to produce a mobile robot protocol for the gathering problem in a ring shaped network. The followed approach was to encode a particular arena for the gathering task, and later use the UPPAAL TIGA tool to generate a winning strategy that can be developed into an algorithm.

Problem specification The *Gathering* problem is defined in the general asynchronous model as follows.

For any graph G of size n and any initial configuration, an algorithm solves the gathering problem if it guarantees that in any execution, all robots meet at the same vertex (not known beforehand) and remain there infinitely thereafter. Similarly as in the previous section, all robots have to be scheduled infinitely often.

4.2.1 Arena Encoding for Gathering

The authors construct an arena so that the player has a winning strategy if and only if a mobile robot algorithm permits robots to gather at a particular node independentely of the

Table 1: Rules for the legitimate phase of *Min-Algorithm*.

Legitimate Phase: $z \neq \{0,1,2,3,4\}$			
$RL1$::	(R_2, F_2, R_1, F_z)	$\rightarrow$	$(R_1, F_1, R_1, F_2, R_1, F_{z-1})$
$RL2$::	$(R_1, F_1, R_1, F_2, R_1, F_z)$	$\rightarrow$	(R_2, F_3, R_1, F_z)
$RL3$::	(R_2, F_3, R_1, F_z)	$\rightarrow$	(R_2, F_2, R_1, F_{z+1})

Table 2: Rules for the convergence phase of *Min-Algorithm*.

Convergence Phase:				
$RC1$::	(R_2, F_y, R_1, F_z)	$\rightarrow$	$(R_2, F_{min(y,z)}, R_1, F_{max(y,z)+1})$	avec $y \neq z \neq \{1,2,3\}$
$RC2$::	$(R_1, F_x, R_1, F_y, R_1, F_y)$	$\rightarrow$	$(R_1, F_x, R_1, F_{y-1}, R_1, F_{y+1})$	avec $x \neq y \neq 0$
$RC3$::	$(R_1, F_x, R_1, F_y, R_1, F_z)$	$\rightarrow$	$(R_1, F_{x-1}, R_1, F_{y+1}, R_1, F_z)$	avec $x < y < z$
$RC4$::	(R_3, F_z)	$\rightarrow$	(R_2, F_1, R_1, F_{z-1})	if a single robot moves
		$\rightarrow$	$(R_1, F_1, R_1, F_1, R_1, F_{z-2})$	if two robots move
$RC5$::	(R_2, F_1, R_1, F_z)	$\rightarrow$	(R_2, F_2, R_1, F_{z-1})	

initial configuration. In each configuration, the robots can choose among the following actions: $\Delta = \{\curvearrowright, \curvearrowleft, \uparrow, ?\}$, which contains $\mathbb{M} = \{\curvearrowright, \curvearrowleft, \uparrow\}$, the set of possible movements, and "?", used by disoriented robots indicating their will to move, yet unability to decide the exact direction of movement (*e.g.* due to symmetry). We note $\overline{\curvearrowright} = \curvearrowleft$, $\overline{\curvearrowleft} = \curvearrowright$, $\overline{\uparrow} = \uparrow$ and $\overline{?} = ?$.

The arena is $\mathcal{A}_{\text{gather}} = (V_p \uplus V_o, E)$, with $V_p = (\mathcal{C}/\equiv)$ denoting the player states, and $V_o = \mathcal{C} \times (\Delta^k)$ denoting the environment states. The size of the arena is then linear in n and exponential in k. The arcs in the arena are defined by relation E as a strict alternating sequence of states between the two players: $E \subseteq (V_p \times V_o) \cup (V_o \times V_p)$.

From a player state, the player chooses for each robot a move. There is the additional constraint that in any equivalence class, two robots with the same view take the same decision (the robot algorithm is deterministic).

A *decision function* f is a function that proposes a move based on a robot view. It is defined by $f : \mathcal{V} \rightarrow \Delta$ such that, for any view $V \in \mathcal{V}$, if $|V| = 1$ then $f(V) \in \{\uparrow, ?\}$, and if $f(V) = ?$ then $|V| = 1$ (a disoriented robot can only choose to move or not to move). When a decision function is run, the robots moves must be coherent with a global sense of orientation. Since $C = (d_1, \ldots, d_k) \in \mathcal{C}$, and $f : \mathcal{V} \rightarrow \Delta$, for any $1 \leq i \leq k$, we define $f(C, i) = f(view_i(C))$ if $(d_i, \cdots, d_k, d_1, \cdots d_{i-1})$ is the smallest element of $view_i(C)$ (in lexicographic order), and $f(C, i) = \overline{f(view_i(C))}$ otherwise.

Then, for every $v \in V_p, v' \in V_o$, $(v, v') \in E$ *iff* there exists a decision function f such that $v' = \big(C, (a_1, \ldots, a_k)\big)$ with $C = rep(v) = (d_1, \ldots, d_k)$, and for every $1 \leq i \leq k$, $a_i = f(C, i)$.

The game then continues from an enviroment position where the previous choices of the player are remembered. If a disoriented robot has deciced to move, the environment chooses the move to be performed by the robot among $\{\curvearrowright, \curvearrowleft\}$.

In $v' = (C, (a_1, \ldots, a_k)) \in V_o$, a set of movements

$$(m_i)_{i \in \{1, \ldots, k\}} \in \mathbb{M}^k$$

is *v'-compatible* if: for every $1 \leq i \leq k$ such that $a_i \neq ?$, $m_i = a_i$, and for every $1 \leq i \leq k$ such that $a_i = ?$, $m_i \neq \uparrow$.

Getting from an environment state to a player state is then expressed as: for every $v \in V_p$, $v' = (C, (a_1, \ldots, a_k)) \in V_o$, $(v', v) \in E$ *iff* there exists a tuple that is v'-compatible $(m_i)_{i \in \{1, \ldots, k\}}$ and such that $v = [C \oplus (m_i)_{i \in \{1, \ldots, k\}}]_\equiv$.

Theorem 2 *The winning position for the player in the*

$$(\mathcal{A}_{gather}, W)$$

game corresponds exactly to the gatherable configurations.

4.2.2 Synthesis of a Gathering Algorithm for Three Robots

The aforementioned arena permits to synthesise a deterministic protocol for the gathering problem of k robots in a n-sized ring. Let $T = [(-1, \cdots, -1, n-1)]_\equiv \in V_p$ be the equivalence class of all configurations where all robots are gahtered at a single node. Millet *et al.* [58] implemented the arena for three robots and various ring sizes ($n \in [3, 15]$ et $n = 100$) using the game resolution tool UPPAAL TIGA [66]. It was possible to confirm the impossibility of gathering from a starting configuration that is periodic, and possibility of gathering otherwise (that is, there exists a winning strategy in those remaining cases).

To obtain optimal strategies (with respect to the overall number of movements), one can use weighted arcs in the arena depending on the number of moving robots on that arc.

Figure 5 presents classes of configurations (satisfying some constraint φ), and the strategy found in this class (in the "Strategy" column). The "Robot Algorithm" column presents the corresponding robot algorithm executed by Robot r when its view $view_r$ satisfies φ. For all other views, the robot algorithm is $\uparrow$. This algorithm is correct by construction for $n \in [3, 15]$ and $n = 100$. An induction proof is given in [58], extending the results to any ring size n.

4.3 Certification of Impossibility Results

So far the aforementioned formalism proved to be useful and with a (relative) ease of use to certify impossibility results regarding oblivious and anonymous mobile robots [62], even when one allows for byzantine behaviours [60], [61].

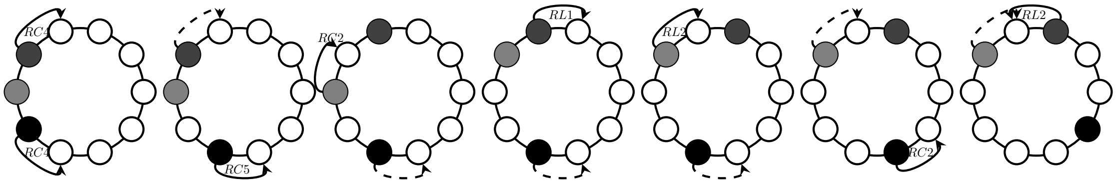


Figure 4: Counter example.

Strategy		
V_p	φ	V_o
$[(-1,-1,n-1)]_\equiv$		$((-1,-1,n-1),(\uparrow,\uparrow,\uparrow))$
$[(-1,d_1,d_2)]_\equiv$	$d_1 < d_2$	$((-1,d_1,d_2),(\uparrow,\uparrow,\curvearrowleft))$
$[(-1,\frac{n-1}{2},\frac{n-1}{2})]_\equiv$		$((-1,\frac{n-1}{2},\frac{n-1}{2}),(\uparrow,\uparrow,?))$
$[(d_1,d_1,d_2)]_\equiv$	$rep = (d_1,d_1,d_2)$	$((d_1,d_1,d_2),(\curvearrowright,\uparrow,\curvearrowleft))$
$[(d_1,d_1,d_2)]_\equiv$	$rep = (d_2,d_1,d_1)$	$((d_2,d_1,d_1),(\uparrow,\uparrow,?))$
$[(d_1,d_2,d_3)]_\equiv$	$d_1 < d_2 < d_3$	$(\curvearrowright,\uparrow,\uparrow))$

Robot Algorithm		
$view_r$	φ	$f : view_r$
$\{(d_1,-1,d_2),(d_2,-1,d_1)\}$	$d_1 < d_2$	$\curvearrowright$
$\{(\frac{n-1}{2},-1,\frac{n-1}{2})\}$		?
$\{(d_1,d_1,d_2),(d_2,d_1,d_1)\}$	$d_1 < d_2$	$\curvearrowright$
$\{(d_1,d_2,d_1)\}$	$d_2 < d_1$	?
$\{(d_2,d_1,d_3),(d_3,d_1,d_2)\}$	$d_1 < d_2 < d_3$	$\curvearrowright$

Figure 5: A Strategy and its corresponding algorithm.

From the point of view of the person who specifies the model and the properties, the theorems are stated in a natural way: mostly by quantifying over relevant demons, protocols (robograms), and concluding with a negation of the solution characterisation.

For instance, the impossibility of gathering for an even number of oblivious and anonymous mobile entities moving along $\mathbb{R}$ in [62] is simply expressed as follows:

```
Theorem noGathering :
  ∀ (G : finite) (r : robogram (G ⊎ G)),
  inhabited G
  → ∀ k : nat, (1 <= k)
    → ¬ (∀ d, kFair k d → solGathering r d)
```

It can be read as "for every finite set G that is non-empty, for every robogram r distributed over twice the cardinal of G robots (thus an even number), for every fairness constraint k, there is a k-fair demon d for which r fails to gather all robots".

Its proof amounts to showing that for a non-null even number of robots, any `k` and any robogram `r` there exists a k-fair demon that prevents r to gather all robots.

From the point view of the person with proof-assistant expertise who develops the actual (interactive) proof, the size of the development is reasonably short as it makes a fair use of the provided libraries. The size of the specialised development for the relevant notions and the aforementioned theorems (thus excluding for example the complete library for reals) is approximately 480 lines of specifications and 430 lines only of proofs. The file dedicated to the theorem itself is about 200 lines of specifications for 250 lines of proof scripts. This is a good indication on how adequate the framework is, as proofs are not too intricate and remain human readable.

Proving negative results has been emphasised here, yet it is worth noticing that this approach is not limited to impossibility results. Indeed, protocols can also be proved correct using this formal development, as it is easy to write an actual program within the language of Coq, a functional language. The statements are then of the form: for all demons, for any number of robots and initial positions that fulfill some constraint, the given robogram is a solution to some problem.

5 OPEN PROBLEMS

We surveyed recent results that make use of formal methods in the context of mobile robot networks. Model checking and algorithm synthesis were used in the discrete space model to find errors in existing literature (and possibly relieve protocol designers from the burden of manually checking small instances of the problem, thus permitting them to concentrate on abstract configurations where some global invariants hold) and general protocols that are correct by design in this context, while proof assistant was used to devise general impossibility results in the continuous space model. Many open challenges remain, we list a few of them in the sequel, hoping to pave the way for future research.

Arbitrary Sized Networks The main limitations implied by the model-checking and algorihtm synthesis approaches is that the space where robots evolve is *bounded*. That is, the number of robots k and the size of the ring n are given as parameter to generate the possible configuration. This permits to keep the modelling of the system simple, and to enumerate all possible situations. Getting generic results for any size n still requires a handmade approach, taking the mechanically verified instances as a base case for human generated induction. Mechanising the second part (*e.g.* with Coq or another similar tool) is a promising path.

Discrete *vs.* Continuous Space Going from the discrete space to the continuous space is another challenge (in the case of model-cheking and algorithm synthesis). Then, it becomes impossible to enumerate all possible configurations of robots, yet a completely different modelling of the configurations (*e.g.* based on some geometric invariants observed by the robots) could lead to limiting their classes to a tractable number. However, in this case, none of the presented approaches so far can be reused.

On the positive side, thanks to the abstraction level of the Pactole framework [67], setting the space to be $\mathbb{R}$, thus both unbounded and continuous, is not as complicated as one could imagine; it emphasises the relevance of a formal proof approach and how it is complementary to other formal verification techniques.

Atomic *vs.* Non-atomic Executions For the algorithm synthesis and proof assistant, we focused on the atomic FSYNC and SSYNC models. Breaking the atomicity of the individual Look-Compute-Move cycles (that is, considering automatic algorithm production for the ASYNC model [1], or writing impossibility results that are specific to that 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, the two-players game encoding of Millet *et al.* [58] 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.

The modelling of ASYNC is feasible in a proof assistant, and should not bring any additional difficulties in the specification of properties in that context. However, it would currently have a significant cost in terms of intricacy of the associated proofs. A really manageable formal development in an ASYNC model requires more automation at the proof level.

Toward Weaker Requirements A noteworthy added benefit of the COQ abstract framework is that keeping the abstractions as general as possible may lead to relaxing premises of theorems, thus potentially discovering new results (*e.g.* formalising weaker demons [57] and weaker forms of Byzantine behaviours could lead to stronger impossibility results).

Toward New Robotic Problems Solved While the modelling in the discrete space approaches is generic, the encoding of the problem has to be specific (LTL logic for model checking, identifying the winning configurations in the algorithm synthesis approach). The COQ approach remains generic with respect to the algorithm thank to its higher-order logic capabilities, however the suitability of the approach to obtain positive results (that is, certified algorihtms solving a praticular problem) has not been demonstrated yet on practical examples. This issue remains challenging as expertise is required to design the proper encoding in each formal model. Facilitating this step for algorithm designer is a long term research goal.

REFERENCES

[1] P. Flocchini, G. Prencipe, and N. Santoro, Distributed Computing by Oblivious Mobile Robots, Morgan & Claypool Publishers (2012).

[2] M. Potop-Butucaru, M. Raynal, and S. Tixeuil, "Distributed computing with mobile robots: An introductory survey," Network-Based Information Systems (NBiS), 2011 14th International Conference on, pp.318–324 (2011).

[3] I. Suzuki and M. Yamashita, "Distributed anonymous mobile robots: Formation of geometric patterns," SIAM Journal on Computing, pp.1347–1363 (1999).

[4] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, "Gathering of asynchronous robots with limited visibility," Theoretical Computer Science, pp.147–168 (2005).

[5] E. Clarke, O. Grumberg, and D. Peled, Model Checking, MIT Press (1999).

[6] C. Baier and J.P. Katoen, Principles of model checking, MIT press (2008).

[7] R. Guerraoui, T.A. Henzinger, and V. Singh, "Model checking transactional memories," Distributed Computing, pp.129–145 (2010).

[8] I. Chatzigiannakis, O. Michail, and P.G. Spirakis, "Algorithmic verification of population protocols," Stabilization, Safety, and Security of Distributed Systems, pp.221–235, Springer Berlin Heidelberg (2010).

[9] J. Clément, C. Delporte-Gallet, H. Fauconnier, and M. Sighireanu, "Guidelines for the verification of population protocols," Distributed Computing Systems, pp.215–224, IEEE (2011).

[10] T. Lu, S. Merz, and C. Weidenbach, "Towards verification of the pastry protocol using tla^{+}," Formal Techniques for Distributed Systems, pp.244–258, Springer Berlin Heidelberg (2011).

[11] T. Tsuchiya and A. Schiper, "Verification of consensus algorithms using satisfiability solving," Distributed Computing, pp.341–358 (2011).

[12] M.Z. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic verification of herman's self-stabilisation algorithm," Formal Asp. Comput., Vol.24, No.4-6, pp.661–670 (2012).

[13] K.R. Apt and D. Kozen, "Limits for automatic verification of finite-state concurrent systems," Inf. Process. Lett., Vol.22, No.6, pp.307–309 (1986).

[14] J. Esparza, A. Finkel, and R. Mayr, "On the verification of broadcast protocols," 14th Annual Symp. on Logic in Computer Science, pp.352–359, IEEE (1999).

[15] Z. Manna and A. Pnueli, "Temporal verification diagrams," Proc.of Int. Conf. on Theoretical Aspects of Computer Software (TACS'94), LNCS, Vol.789, pp.726–765, Springer (1994).

[16] E.M. Clarke, O. Grumberg, and S. Jha, "Veryfying parameterized networks using abstraction and regular languages," Proc. of 6th Int. Conf. on Concurrency Theory (CONCUR'95), LNCS, Vol.962, pp.395–407, Springer (1995).

[17] 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," Proc. of 8th Int. Conf. on Computer Aided Verification (CAV'96), LNCS, Vol.1102, pp.415–418, Springer (1996).

[18] L. de Alfaro, Z. Manna, H.B. Sipma, and T.E. Uribe, "Visual verification of reactive systems," Proc. of 3d Int. Workshop on Tools and Algorithms for Construc-

tion and Analysis of Systems (TACAS'97), LNCS, Vol.1217, pp.334–350, Springer (1997).

[19] D. Cansell, D. Méry, and S. Merz, "Diagram refinements for the design of reactive systems," J. Univ. Comp. Sci., Vol.7, No.2, pp.159–174 (2001).

[20] T. Arons, A. Pnueli, S. Ruah, J. Xu, and L.D. Zuck, "Parameterized verification with automatically computed inductive assertions," Proc. of 13th Int. Conf. on Computer Aided Verification (CAV'01), LNCS, Vol.2102, pp.221–234, Springer (2001).

[21] E.M. Clarke and E.A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," Proc. ofIBM Workshop on Logics of Programs (1981).

[22] Z. Manna and P. Wolper, "Synthesis of communicating processes from temporal logic specifications," ACM Trans. Program. Lang. Syst., Vol.6, No.1, pp.68–93 (1984).

[23] M. Abadi, L. Lamport, and P. Wolper, "Realizable and unrealizable specifications of reactive systems," Proc. ofICALP'89, LNCS, Vol.372, pp.1–17, Springer (1989).

[24] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," Proc. ofPOPL'89, pp.179–190, ACM (1989).

[25] A. Church, "Logic, arithmetics, and automata," Proc. ofInt. Congr. of Mathematicians, pp.23–35 (1963).

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

[27] Agda, `http://wiki.portal.chalmers.se/agda/pmwiki.php`.

[28] R.S. Boyer and J.S. Moore, A Computational Logic Handbook, Academic Press (1988).

[29] ACL2, `http://www.cs.utexas.edu/users/moore/acl2/`.

[30] PVS, `http://pvs.csl.sri.com/`.

[31] Mizar, `http://mizar.uwb.edu.pl/`.

[32] Coq, `https://coq.inria.fr/`.

[33] T. Nipkow, L.C. Paulson, and M. Wenzel, Isabelle/HOL — A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, Vol.2283, Springer-Verlag (2002).

[34] X. Leroy, "A Formally Verified Compiler Back-End," Journal of Automated Reasoning, Vol.43, No.4, pp.363–446 (2009).

[35] 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," Communications of the ACM, Vol.53, No.6, pp.107–115 (2010).

[36] 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," ACM Conference on Computer and Communications Security, ed. T. Yu, G. Danezis, and V.D. Gligor, pp.488–500, ACM (2012).

[37] L. Théry and G. Hanrot, "Primality Proving with Elliptic Curves," 20th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2007), ed. K. Schneider and J. Brandt, Lecture Notes in Computer Science, Vol.4732, Kaiserslautern, Germany, pp.319–333, Springer-Verlag, Sept. (2007).

[38] G. Gonthier, "Formal Proof—The Four-Color Theorem," in Notices of the AMS, Vol.55, p.1370, december (2008).

[39] G. Gonthier, "Engineering Mathematics: the Odd Order Theorem Proof," POPL, ed. R. Giacobazzi and R. Cousot, pp.1–2, ACM (2013).

[40] F.D. Team, "The Flyspeck Project." `https://code.google.com/p/flyspeck/`.

[41] W. Fokkink, Modelling Distributed Systems, EATCS Texts in Theoretical Computer Science, Springer-Verlag (2007).

[42] M. Bezem, R. Bol, and J.F. Groote, "Formalizing Process Algebraic Verifications in the Calculus of Constructions," Formal Aspects of Computing, Vol.9, pp.1–48 (1997).

[43] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," ACM Transactions on Programming Languages and Systems, Vol.4, No.3, pp.382–401 (1982).

[44] L. Lamport, "Byzantizing Paxos by Refinement," DISC, ed. D. Peleg, Lecture Notes in Computer Science, Vol.6950, pp.211–224, Springer (2011).

[45] D. Cousineau, D. Doligez, L. Lamport, S. Merz, D. Ricketts, and H. Vanzetto, "TLA + Proofs," FM, ed. D. Giannakopoulou and D. Méry, Lecture Notes in Computer Science, Vol.7436, Paris, France, pp.147–154, Springer-Verlag, Aug. (2012).

[46] E. Gascard and L. Pierre, "Formal Proof of Applications Distributed in Symmetric Interconnexion Networks," Parallel Processing Letters, Vol.13, No.1, pp.3–18 (2003).

[47] D. Cansell and D. Méry, Logics of Specification Languages, ch. The Event-B Modelling Method: Concepts and Case Studies, pp.47–152, Springer-Verlag (2007).

[48] P. Küfner, U. Nestmann, and C. Rickmann, "Formal Verification of Distributed Algorithms - From Pseudo Code to Checked Proofs," IFIP TCS, ed. J.C.M. Baeten, T. Ball, and F.S. de Boer, Lecture Notes in Computer Science, Vol.7604, Amsterdam, The Netherlands, pp.209–224, Springer-Verlag, Sept. (2012).

[49] C.T. Chou, "Mechanical Verification of Distributed Algorithms in Higher-Order Logic," The Computer Journal, Vol.38, pp.158–176 (1995).

[50] P. Castéran, V. Filou, and M. Mosbah, "Certifying Distributed Algorithms by Embedding Local Computation Systems in the Coq Proof Assistant," Symbolic Computation in Software Science (SCSS'09), ed. A. Bouhoula and T. Ida (2009).

[51] Loco, `http://www.labri.fr/~casteran/Loco`.

[52] Y. Deng and J.F. Monin, "Verifying Self-stabilizing Population Protocols with Coq," Third IEEE International Symposium on Theoretical Aspects of Software Engi-

neering (TASE 2009), ed. W.N. Chin and S. Qin, Tianjin, China, pp.201–208, IEEE Computer Society, July (2009).

[53] S. Devismes, A. Lamani, F. Petit, P. Raymond, and S. Tixeuil, “Optimal grid exploration by asynchronous oblivious robots,” Proc. of SSS, pp.64–76, Springer (2012).

[54] F. Bonnet, X. Défago, F. Petit, M. Potop-Butucaru, and S. Tixeuil, “Brief announcement: Discovering and assessing fine-grained metrics in robot networks protocols,” Proc. of SSS 2012, LNCS, Vol.7596, pp.282–284, Springer (2012).

[55] F. Bonnet, X. Défago, F. Petit, M. Potop-Butucaru, and S. Tixeuil, “Discovering and assessing fine-grained metrics in robot networks protocols,” 33rd IEEE International Symposium on Reliable Distributed Systems Workshops, SRDS Workshops 2014, Nara, Japan, October 6-9, 2014, pp.50–59, IEEE (2014).

[56] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, “The synchronous dataflow programming language lustre,” Proceedings of the IEEE, Vol.79, No.9, pp.1305–1320, September (1991).

[57] S. Dubois and S. Tixeuil, “A Taxonomy of Daemons in Self-stabilization,” Tech. Rep. 1110.0334, ArXiv eprint, October (2011).

[58] L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil, “On the synthesis of mobile robots algorithms: The case of ring gathering,” Stabilization, Safety, and Security of Distributed Systems - 16th International Symposium, SSS 2014, Paderborn, Germany, September 28 - October 1, 2014. Proceedings, ed. P. Felber and V.K. Garg, Lecture Notes in Computer Science, Vol.8756, pp.237–251, Springer (2014).

[59] E. Grädel, W. Thomas, and T. Wilke, eds., Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001], Lecture Notes in Computer Science, Vol.2500, Springer (2002).

[60] C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain, “Brief announcement: Certified impossibility results for byzantine-tolerant mobile robots,” International Symposium on Distributed Computing (DISC), ed. Y. Afek, Lecture Notes in Computer Science, Vol.8205, Jerusalem, Israel, pp.577–578, Springer, October (2013).

[61] C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain, “Certified impossibility results for byzantine-tolerant mobile robots,” SSS, ed. T. Higashino, Y. Katayama, T. Masuzawa, M. Potop-Butucaru, and M. Yamashita, Lecture Notes in Computer Science, Vol.8255, Osaka, Japan, pp.178–190, Springer, November (2013).

[62] P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain, “Impossibility of gathering, a certification,” Information Processing Letters, Vol.115, pp.447–452 (2015).

[63] D. Sangiorgi, Introduction to Bisimulation and Coinduction, Cambridge University Press (2012).

[64] L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil, “Exclusive perpetual ring exploration without chirality,” Distributed Computing, pp.312–327 (2010).

[65] J. Barnat, L. Brim, M. Češka, and P. Ročkai, “DiVinE: Parallel Distributed Model Checker (Tool paper),” Parallel and Distributed Methods in Verification and High Performance Computational Systems Biology, pp.4–7, IEEE (2010).

[66] Uppaal Tiga, `http://people.cs.aau.dk/~adavid/tiga/`.

[67] Pactole, `http://pactole.lri.fr`.

(Received May 11, 2015)
(Revised August 22, 2015)

Béatrice Bérard is full professor at University Pierre & Marie Curie (Paris 6, France) in the Modelling and Verification group. Her research interests cover formal verification and synthesis of reactive systems, possibly integrating quantitative features like time or probabilities.

Pierre Courtieu is associate professor at CNAM, Paris (France). He received his PhD from University Paris-Sud XI in 2001, and is a specialist of theorem proving and proof of programs. He is a member of the steering committee of the COQ proof assistant.

Laure Millet is currently a third year PHD student at University Pierre & Marie Curie - Paris 6 (France). She is working on formal methods and verification of distributed algorithms for autonomous and mobile robots.

Maria Potop-Butucaru is full professor at University Pierre & Marie Curie (Paris 6, France) in the Network and Performance Analysis group. She was previously, assistant professor in University Paris 11 and associate professor in University Rennes 1. Her research area includes distributed and fault tolerant algorithms and formal methods for static and dynamic systems with special focus on sensor and robot networks.

Lionel Rieg is currently *Attaché Temporaire d'Enseignement et de Recherche* (a temporary research assistant position) at Collège de France, Paris (France). Holder of the high academic competitive examination *Agrégation* in mathematics, he defended his PhD at École Normale Supérieure de Lyon in 2014. He is a specialist in logics, in particular regarding the Curry-Howard correspondance.

Nathalie Sznajder is assistant professor at University Pierre & Marie Curie - Paris 6 (France), LIP6. She received her PhD from ENS Cachan in 2009 and spent a year as a post-doctoral researcher at Université Libre in Bruxelles. Her research interests cover formal verification and control of (distributed, timed) systems.

Sébastien Tixeuil is full professor at University Pierre & Marie Curie - Paris 6 (France) and Institut Universitaire de France, where he leads the Networks and Systems department at LIP6. He received his Ph.D. from University of Paris Sud-XI in 2000. His research interests include fault and attack tolerance in dynamic networks and systems. He has co-authored more than 150 research papers in international journal and conferences.

Xavier Urbain is associate professor at ENSIIE, Évry (France). He received his PhD from University Paris-Sud XI in 2001, as well as his Habilitation in 2010. He is a specialist of automated deduction and certified proof, in particular aimed at exhibiting properties of programs.