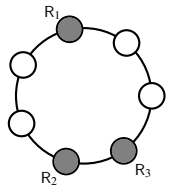
Parameterized Verification of Algorithms for Oblivious Robots on a Ring

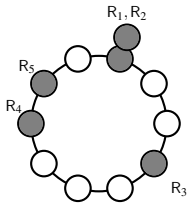
Arnaud Sangnier
IRIF, Univ Paris Diderot

Nathalie Sznajder,
Maria Potop-Butucaru
and Sébastien Tixeuil
Sorbonne Universités, UPMC Univ Paris 06, LIP6

***Abstract*—We study verification problems for autonomous swarms of mobile robots that self-organize and cooperate to solve global objectives. In particular, we focus in this paper on the model proposed by Suzuki and Yamashita of anonymous robots evolving in a discrete space with a finite number of locations (here, a ring). A large number of algorithms have been proposed working for rings whose size is not a priori fixed and can be hence considered as a parameter. Handmade correctness proofs of these algorithms have been shown to be error-prone, and recent attention had been given to the application of formal methods to automatically prove those. Our work is the first to study the verification problem of such algorithms in the parameterized case. We show that safety and reachability problems are undecidable for robots evolving asynchronously. On the positive side, we show that safety properties are decidable in the synchronous case, as well as in the asynchronous case for a particular class of algorithms. Several properties on the protocol can be decided as well. Decision procedures rely on an encoding in Presburger arithmetics formulae that can be verified by an SMT-solver. Feasibility of our approach is demonstrated by the encoding of several case studies.**



(a) A disoriented robot R_1



(b) A configuration with a tower

Fig. 1

I. Introduction

We consider sets of mobile oblivious robots evolving in a discrete space (modeled as a ring shaped graph). For our purpose, rings are seen as discrete graphs whose vertices represent the different positions available to host a robot, and edges model the possibility for a robot to move from one position to another. Robots follow the seminal model by Suzuki and Yamashita [23]: they do not remember their past actions, they cannot communicate explicitly, and are disoriented.

However, they can sense their environment and detect the positions of the other robots on the ring. If several robots share the same position on the ring (forming a *tower*, or multiplicity point), other robots may or may not detect the tower. If robots have *strong* multiplicity detection, as assumed in this paper, they are able to count the exact number of robots on a given position.

Robots are anonymous and execute the same deterministic algorithm to achieve together a given objective. Different objectives for ring shaped discrete spaces have been studied in the literature [17]: gathering – starting from any initial configuration, all the robots must gather on the same node, not known beforehand, and then stop [18], exploration with stop – starting from any initial configuration, the robots reach a configuration where they all are idle and, in the meanwhile, all the positions of the ring have been visited by a robot [16], exclusive perpetual exploration – starting from any tower-free configuration, each position of the ring is visited infinitely often and no multiplicity point ever appears [6], [11].

Each robot behaves according to the following cycle: it takes a snapshot of its environment, then it computes its next move (either stay idle or move to an adjacent node in the ring), and at the end of the cycle, it moves according to its computation. Such a cycle is called a look-compute-move cycle.

Since robots cannot rely on a common sense of direction, directions that are computed in the compute phase are only *relative* to the robot. To tell apart its two sides, a robot relies on a description of the ring in both clockwise and counter-clockwise direction, which gives it two views of the configuration. There are two consequences to this fact. First, if its two views are identical, meaning that the robot is on an axis of symmetry, it cannot distinguish the two directions and thus either decides to stay idle, or to move. In the latter case, the robot moves becomes a non-deterministic choice between the two available directions. Second, when two robots have the same two views of the ring, the protocol commands them to move in the same relative direction, but this might result in moves in actual opposite directions for the two robots. Such a symmetrical situation is pictured in Figure 1.

This work has been partly supported by the ANR research program PACS (ANR-14-CE28-0002)

Existing execution models consider different types of synchronization for the robots: in the fully synchronous model (FSYNC), all robots evolve simultaneously and complete a full look-compute-move cycle. The semi-synchronous model (SSYNC) consider runs that evolve in phases: at each phase, an arbitrary subset of the robots is scheduled for a full look-compute-move cycle, which is executed simultaneously by all robots of the subset. Finally, in the asynchronous model (ASYNC), robots evolve freely at their own pace: in particular, a robot can move according to a computation based on an obsolete observation of its environment, as others robots may have moved in between. Algorithms in the literature are typically parameterized by the number of robots and/or number of positions in the ring. In this work we focus on formally verifying algorithms parameterized by the number of ring positions only, assuming a a fixed number of robots.

A. Related work

Designing and proving mobile robot protocols is notoriously difficult. Formal methods encompass a long-lasting path of research that is meant to overcome errors of human origin. Unsurprisingly, this mechanized approach to protocol correctness was successively used in the context of mobile robots [7], [13], [5], [2], [20], [9], [4], [22], [3].

When robots are *not* constrained to evolve on a particular topology (but instead are allowed to move freely in a bidimensional Euclidian space), the Pactole (`http://pactole.lri.fr`) framework has been proven useful. Developed for the Coq proof assistant, Pactole enabled the use of high-order logic to certify impossibility results [2] for the problem of convergence: for any positive ε, robots are required to reach locations that are at most ε apart. Another classical impossibility result that was certified with Pactole is the impossibility of gathering starting from a bivalent configuration [9]. Recently, positive certified results for SSYNC gathering with multiplicity detection [10], and for FSYNC gathering without multiplicity detection [3] were provided. However, as of now, no Pactole library is dedicated to robots that evolve on discrete spaces.

In the discrete setting that we consider in this paper, model-checking proved useful to find bugs in existing literature [5], [14] and assess formally published algorithms [13], [5], [22]. Automatic program synthesis (for the problem of perpetual exclusive exploration in a ring-shaped discrete space) 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.* [20] for the problem of gathering in a discrete ring network. As all aforementioned approaches are designed for a bounded setting where both the number of locations and the number of robots are known, they cannot permit to establish results that are valid for any number of locations.

Recently, Aminof *et al.* [22] presented a general framework for verifying properties about mobile robots evolving on graphs, where the graphs are a parameter of the problem. While our model could be encoded in their framework, their undecidability proof relies on persistent memory used by the robots, hence is not applicable to the case of oblivious robots we consider here. Also, they obtain decidability in a subcase that is not relevant for robot protocols like those we consider. Moreover, their decision procedure relies on MSO satisfiability, which does not enjoy good complexity properties and cannot be implemented efficiently for the time being.

B. Contributions

In this work, we tackle the more general problem of verifying protocols for swarms of robots for any number of locations.

We provide a formal definition of the problem, where the protocol can be described as a quantifier free Presburger formula. This logic, weak enough to be decidable, is however powerful enough to express existing algorithms in the literature. Objectives of the robots are also described by Presburger formulae and we consider two problems: when the objective of the robots is a safety objective – robots have to avoid the configurations described by the formula (SAFE), and when it is a reachability objective (REACH). We show that if REACH is undecidable in any semantics, SAFE is decidable in FSYNC and SSYNC. We also show that when the protocol is *uniquely-sequentializable*, safety properties become decidable even in the asynchronous case.

Finally, we show practical applicability of this approach by using an SMT-solver to verify safety properties for some algorithms from the literature.

Hence, we advocate that our formalism should be used when establishing such protocols, as a formal and non-ambiguous description, instead of the very informal and sometimes unclear definitions found in the literature. Moreover, if totally automated verification in the parameterized setting seems unfeasible, our method could be used as a "sanity check" of the protocol, and to automatically prove intermediate lemmas, that can then be used as formally proved building blocks of a handmade correction proof.

II. Model of Robots Evolving on a Ring

A. Formal model

In this section we present the formal language to describe mobile robots protocols as well as the way it is interpreted.

1) Preliminaries: For $a,b \in \mathbb{Z}$ such that $a \leq b$, we denote by $[a,b]$ the set $\{c \in \mathbb{Z} \mid a \leq c \leq b\}$. For $a \in \mathbb{Z}$ and $b \in \mathbb{N}$, we write $a \odot b$ the natural $d \in [0,(b-1)]$ such that there exists $j \in \mathbb{Z}$ and $a = b.j + d$ (for instance $-1 \odot 3 = 2$). Note that $\odot$ corresponds to the modulo operator, but for sake of clarity we recall its definition when a is negative.

We recall the definition of Existential Presburger (EP) formulae. Let Y be a countable set of variables. First we define the grammar for terms $\mathtt{t} ::= x \mid \mathtt{t}+\mathtt{t} \mid a \cdot \mathtt{t} \mid \mathtt{t} \bmod a$, where $a \in \mathbb{N}$ and $x \in Y$ and then the grammar for formulae is given by $\phi ::= \mathtt{t} \bowtie b \mid \phi \wedge \phi \mid \phi \vee \phi \mid \exists x.\phi$ where $\bowtie \in \{=,\leq,\geq,<,>\}$, $x \in Y$ and $b \in \mathbb{N}$. We sometimes write a formula ϕ as $\phi(x_1,\ldots,x_k)$ to underline that $x_1,\ldots,x_k$ are the free variables of ϕ. The set of Quantifier Free Presburger (QFP) formulae is obtained by the same grammar deleting the

elements $\exists x.\phi$. Note that when dealing with QFP formulae, we allow as well negations of formulae.

We say that a vector $V = \langle d_1, \ldots, d_k \rangle$ satisfies an EP formula $\phi(x_1, \ldots, x_k)$, denoted by $V \models \phi$, if the formula obtained by replacing each x_i by d_i holds. Given a formula ϕ with free variables $x_1, \ldots, x_k$, we write $\phi(d_1, \ldots, d_k)$ the formula where each x_i is replaced by d_i. We let $[\![\phi(x_1, \ldots, x_k)]\!] = \{\langle d_1, \ldots, d_k \rangle \in \mathbb{N}^k \mid \phi(d_1, \ldots, d_k) \models \phi\}$ be the set of models of the formula. In the sequel, we use Presburger formulae to describe configurations of the robots, as well as protocols.

2) Configurations and robot views: In this paper, we consider a fixed number $k > 0$ of robots and, except when stated otherwise, we assume the identities of the robots are $\mathcal{R} = \{R_1, \ldots, R_k\}$. We may sometimes identify $\mathcal{R}$ with the set of indices $\{1, \ldots, k\}$. On a ring of size $n \geq k$, a *(k,n)-configuration* of the robots (or simply a configuration if n and k are clear from the context) is given by a vector $\mathbf{p} \in [0, n-1]^k$ associating to each robot R_i its position $\mathbf{p}(i)$ on the ring. We assume w.l.o.g. that positions are numbered in the clockwise direction.

A *view* of a robot on this configuration gives the distances between the robots, starting from its neighbor, i.e. the robot positioned on the next occupied node (a distance equals to 0 meaning that two robots are on the same node). A *view* $\mathbf{V} = \langle d_1, \ldots, d_k \rangle \in [0, n]^k$ is a k-tuple such that $\sum_{i=1}^{k} d_i = n$ and $d_1 \neq 0$. We let $\mathcal{V}_{n,k}$ be the set of possible views for k robots on a ring of size n. Notice that all the robots sharing the same position should have the same view. For instance, suppose that, on a ring of size 10, 2 robots R_1, and R_2 are on the same position of the ring (say position 1), R_3 is at position 4, R_4 is at position 8, and R_5 is at position 9 (see Figure 1b). Then, the view of R_1 and R_2 is $\langle 3, 4, 1, 2, 0 \rangle$. It is interpreted by the fact that there is a robot at a distance 3 (it is R_3), a robot a at distance $3+4$ (it is R_4) and so on. We point out that all the robots at the same position share the same view. We as well suppose that in a view, the first distance is not 0 (this is possible by putting 0 at the 'end' of the view instead). As a matter of fact in the example of Figure 1b, there is a robot at distance $3+4+1+2 = 10$ from R_1 (resp. R_2), which is R_2 (resp. R_1). The sum of the d_i corresponds always to the size of the ring and here the fact that in the view of R_1 we have as last element 0 signifies that there is a distance 0 between the last robot (here R_2) and R_1. When looking in the opposite direction, their view becomes: $\langle 2, 1, 4, 3, 0 \rangle$. Formally, for a view $\mathbf{V} = \langle d_1, \ldots, d_k \rangle \in [0, n]^k$, we note $\overleftarrow{\mathbf{V}} = \langle d_j, \ldots, d_1, d_k, \ldots, d_{j+1} \rangle$ the corresponding view when looking at the ring in the opposite direction, where j is the greatest index such that $d_j \neq 0$.

Given a configuration $\mathbf{p} \in [0, n-1]^k$ and a robot $R_i \in \mathcal{R}$, the view of robot R_i when looking in the clockwise direction, is given by $\mathbf{V}_{\mathbf{p}}[i \rightarrow] = \langle d_i(i_1), d_i(i_2) - d_i(i_1), \ldots, n - d_i(i_{k-1}) \rangle$, where, for all $j \neq i$, $d_i(j) \in [1, n]$ is such that $(\mathbf{p}(i) + d_i(j)) \odot n = \mathbf{p}(j)$ and $i_1, \ldots, i_k$ are indexes pairwise different such that $0 < d_i(i_1) \leq d_i(i_2) \leq \cdots \leq d_i(i_{k-1})$. When robot R_i looks in the opposite direction, its view according to the configuration $\mathbf{p}$ is $\mathbf{V}_{\mathbf{p}}[\leftarrow i] = \overleftarrow{\mathbf{V}_{\mathbf{p}}[i \rightarrow]}$.

3) Protocols: In our context, a protocol for networks of k robots is given by a QFP formula respecting some specific constraints.

Definition 1 (Protocol): A protocol is a QFP formula $\phi(x_1, \ldots, x_k)$ such that for all views $\mathbf{V}$ the following holds: if $\mathbf{V} \models \phi$ and $\mathbf{V} \neq \overleftarrow{\mathbf{V}}$ then $\overleftarrow{\mathbf{V}} \not\models \phi$

A robot uses the protocol to know in which direction it should move according to the following rules. As we have already stressed, all the robots that share the same position have the same view of the ring. Given a configuration $\mathbf{p}$ and a robot $R_i \in \mathcal{R}$, if $\mathbf{V}_{\mathbf{p}}[i \rightarrow] \models \phi$, then the robot R_i moves in the clockwise direction, if $\mathbf{V}_{\mathbf{p}}[\leftarrow i] \models \phi$ then it moves in the opposite direction, if none of $\mathbf{V}_{\mathbf{p}}[i \rightarrow]$ and $\mathbf{V}_{\mathbf{p}}[\leftarrow i]$ satisfies ϕ then the robot should not move. The conditions expressed in Definition 1 imposes hence a direction when $\mathbf{V}_{\mathbf{p}}[i \rightarrow] \neq \mathbf{V}_{\mathbf{p}}[\leftarrow i]$. In case $\mathbf{V}_{\mathbf{p}}[i \rightarrow] = \mathbf{V}_{\mathbf{p}}[\leftarrow i]$, the robot is disoriented and it can hence move in one direction or the other. For instance, consider the configuration $\mathbf{p}$ pictured on Figure 1a. Here, $\mathbf{V}_{\mathbf{p}}[1 \rightarrow] = \langle 3, 1, 3 \rangle = \mathbf{V}_{\mathbf{p}}[\leftarrow 1]$. Note that such a semantics enforces that the behavior of a robot is not influenced by its direction. In fact consider two symmetrical configurations $\mathbf{p}$ and $\mathbf{p}'$ such that $\mathbf{V}_{\mathbf{p}}[i \rightarrow] = \overleftarrow{\mathbf{V}_{\mathbf{p}'}[i \rightarrow]}$ for each robot R_i. If $\mathbf{V}_{\mathbf{p}}[i \rightarrow] \models \phi$ (resp. $\mathbf{V}_{\mathbf{p}}[\leftarrow i] \models \phi$), then necessarily $\mathbf{V}_{\mathbf{p}'}[\leftarrow i] \models \phi$ (resp. $\mathbf{V}_{\mathbf{p}'}[i \rightarrow] \models \phi$), and the robot in $\mathbf{p}'$ moves in the opposite direction than in $\mathbf{p}$ (and the symmetry of the two configurations is maintained).

We now formalize the way movement is decided. Given a protocol ϕ and a view $\mathbf{V}$, the moves of any robot whose clockwise direction view is $\mathbf{V}$ are given by:

$$move(\phi, V) = \begin{cases} \{+1\} & \text{if } \mathbf{V} \models \phi \text{ and } \mathbf{V} \neq \overleftarrow{\mathbf{V}} \\ \{-1\} & \text{if } \overleftarrow{\mathbf{V}} \models \phi \text{ and } \mathbf{V} \neq \overleftarrow{\mathbf{V}} \\ \{-1, +1\} & \text{if } \mathbf{V} \models \phi \text{ and } \mathbf{V} = \overleftarrow{\mathbf{V}} \\ \{0\} & \text{otherwise} \end{cases}$$

Here $+1$ (resp. -1) stands for a movement of the robot in the clockwise (resp. anticlockwise) direction.

B. Different possible semantics

We now describe different transition relations between configurations. Robots have a two-phase behavior : (1) look at the ring and (2) according to their view, compute and perform a movement. In this context, we consider three different modes. In the *semi-synchronous mode*, in one step, some of the robots look at the ring and move. In the *synchronous mode*, in one step, all the robots look at the ring and move. In the *asynchronous mode*, in one step a single robot can either choose to look at the ring, if the last thing it did was a movement, or to move, if the last thing it did was to look at the ring. As a consequence, its movement decision is a consequence of the view of the ring it has in its memory. In the remainder of the paper, we fix a protocol ϕ and we consider a set $\mathcal{R}$ of k robots.

1) Semi-synchronous mode: We begin by providing the semantics in the semi-synchronous case. For this matter we define the transition relation $\hookrightarrow_{\phi} \subseteq [0, n-1]^k \times [0, n-1]^k$ (simply noted $\hookrightarrow$ when ϕ is clear from the context) between configurations. We have $\mathbf{p} \hookrightarrow \mathbf{p}'$ if there exists a subset $I \subseteq \mathcal{R}$

of robots such that, for all $i \in I$, $\mathbf{p}'(i) = (\mathbf{p}(i) + m) \odot n$, where $m \in move(\phi, \mathbf{V_p}[i \rightarrow])$, and for all $i \in \mathcal{R} \setminus I$, $\mathbf{p}'(i) = \mathbf{p}(i)$.

2) Synchronous mode: The transition relation $\Rightarrow_\phi \subseteq [0, n-1]^k \times [0, n-1]^k$ (simply noted $\Rightarrow$ when ϕ is clear from the context) describing synchronous movements is very similar to the semi-synchronous case, except that all the robots have to move. Then $\mathbf{p} \Rightarrow \mathbf{p}'$ if $\mathbf{p}'(i) = (\mathbf{p}(i) + m) \odot n$ with $m \in move(\phi, \mathbf{V_p}[i \rightarrow])$ for all $i \in \mathcal{R}$.

3) Asynchronous mode: The definition of transition relation for the asynchronous mode is a bit more involved, for two reasons: first, the move of each robot does not depend on the current configuration, but on the last view of the robot. Second, in one step a robot either look or move. As a consequence, an *asynchronous configuration* is a tuple $(\mathbf{p}, \mathbf{s}, \mathbf{V})$ where $\mathbf{p} \in [0, n-1]^k$ gives the current configuration, $\mathbf{s} \in \{\mathbf{L}, \mathbf{M}\}^k$ gives, for each robot, its internal state ($\mathbf{L}$ stands for ready to look and and $\mathbf{M}$ stands for compute and move) and $\mathbf{V} \in \mathcal{V}^k_{n,k}$ stores, for each robot, the view (in the clockwise direction) it had the last time it looked at the ring.

The transition relation for asynchronous mode is hence defined by a binary relation $\rightsquigarrow_\phi$ (or simply $\rightsquigarrow$) working on $[0, n-1]^k \times \{\mathbf{L}, \mathbf{M}\}^k \times \mathcal{V}^k_{n,k}$ and defined as follows: $\langle \mathbf{p}, \mathbf{s}, \mathbf{V} \rangle \rightsquigarrow \langle \mathbf{p}', \mathbf{s}', \mathbf{V}' \rangle$ iff there exist $R_i \in \mathcal{R}$ such that the following conditions are satisfied:

- for all $R_j \in \mathcal{R}$ such that $j \neq i$, $\mathbf{p}'(j) = \mathbf{p}(j)$, $\mathbf{s}'(j) = \mathbf{s}(j)$ and $\mathbf{V}'(j) = \mathbf{V}(j)$,
- if $\mathbf{s}(i) = \mathbf{L}$ then $\mathbf{s}'(i) = \mathbf{M}$, $\mathbf{V}'(i) = \mathbf{V_p}[i \rightarrow]$ and $\mathbf{p}'(i) = \mathbf{p}(i)$, i.e. if the robot that has been scheduled was about to look, then the configuration of the robots won't change, and this robot updates its view of the ring according to the current configuration and change its internal state,
- if $\mathbf{s}(i) = \mathbf{M}$ then $\mathbf{s}'(i) = \mathbf{L}$, $\mathbf{V}'(i) = \mathbf{V}(i)$ and $\mathbf{p}'(i) = (\mathbf{p}(i) + m) \odot n$, with $m \in move(\phi, \mathbf{V}(i))$, i.e. if the robot was about to move, then it changes its internal state and moves according to the protocol, and *its last view of the ring*.

4) Runs: A semi-synchronous (resp. synchronous) ϕ-run (or a run according to a protocol ϕ) is a (finite or infinite) sequence of configurations $\rho = \mathbf{p}_0 \mathbf{p}_1 \ldots$ where, for all $0 \leq i < |\rho|$, $\mathbf{p}_i \hookrightarrow_\phi \mathbf{p}_{i+1}$ (resp. $\mathbf{p}_i \Rightarrow_\phi \mathbf{p}_{i+1}$). Moreover, if $\rho = \mathbf{p}_0 \cdots \mathbf{p}_n$ is finite, then there is no $\mathbf{p}$ such that $\mathbf{p}_n \hookrightarrow_\phi \mathbf{p}$ (respectively $\mathbf{p}_n \Rightarrow_\phi \mathbf{p}$). An asynchronous ϕ-run is a (finite or infinite) sequence of asynchronous configurations $\rho = \langle \mathbf{p}_0, \mathbf{s}_0, \mathbf{V}_0 \rangle \langle \mathbf{p}_1, \mathbf{s}_1, \mathbf{V}_1 \rangle \cdots$ where, for all $0 \leq i < |\rho|$, $\langle \mathbf{p}_i, \mathbf{s}_i, \mathbf{V}_i \rangle \rightsquigarrow_\phi \langle \mathbf{p}_{i+1}, \mathbf{s}_{i+1}, \mathbf{V}_{i+1} \rangle$ and such that $\mathbf{s}_0(i) = \mathbf{L}$ for all $i \in [1, k]$. Observe that the value of $\mathbf{V}_0$ has no influence on the actual asynchronous run, since any robot starts its computation by a look, hence changing the value of $\mathbf{V}_0$.

We let $\mathbf{Post}_{\mathrm{ss}}(\phi, \mathbf{p}) = \{\mathbf{p}' \mid \mathbf{p} \hookrightarrow_\phi \mathbf{p}'\}$, $\mathbf{Post}_{\mathrm{s}}(\phi, \mathbf{p}) = \{\mathbf{p}' \mid \mathbf{p} \Rightarrow_\phi \mathbf{p}'\}$ and $\mathbf{Post}_{\mathrm{as}}(\phi, \mathbf{p}) = \{\mathbf{p}' \mid$ there exist $\mathbf{V}, \mathbf{s}', \mathbf{V}'$ s.t. $\langle \mathbf{p}, \mathbf{s}_0, \mathbf{V} \rangle \rightsquigarrow_\phi \langle \mathbf{p}', \mathbf{s}', \mathbf{V}' \rangle\}$, with $\mathbf{s}_0(i) = \mathbf{L}$ for all $i \in [1, k]$. Note that in the asynchronous case we impose all the robots to be ready to look. We respectively write $\hookrightarrow^*_\phi$, $\Rightarrow^*_\phi$ and $\rightsquigarrow^*_\phi$ for the reflexive and transitive closure of the relations $\hookrightarrow_\phi$, $\Rightarrow_\phi$ and $\rightsquigarrow_\phi$ and we define $\mathbf{Post}^*_{\mathrm{ss}}(\phi, \mathbf{p})$, $\mathbf{Post}^*_{\mathrm{s}}(\phi, \mathbf{p})$ and $\mathbf{Post}^*_{\mathrm{as}}(\phi, \mathbf{p})$ by replacing in the definition $\mathbf{Post}_{\mathrm{ss}}(\phi, \mathbf{p})$,$\mathbf{Post}_{\mathrm{s}}(\phi, \mathbf{p})$ and $\mathbf{Post}_{\mathrm{as}}(\phi, \mathbf{p})$ the relations $\hookrightarrow_\phi$, $\Rightarrow_\phi$ and $\rightsquigarrow_\phi$ by their reflexive and transitive closure accordingly.

We now come to our first result that shows that when the protocols have a special shape, the three semantics are identical.

Definition 2: A protocol ϕ is said to be *uniquely-sequentializable* if, for all configuration $\mathbf{p}$, there is at most one robot $R_i \in \mathcal{R}$ such that $move(\phi, \mathbf{V_p}[i \rightarrow]) \neq \{0\}$.

When ϕ is uniquely-sequentializable at any moment at most one robot moves. Consequently, in that specific case, the three semantics are equivalent as stated by the following theorem.

Theorem 1: If a protocol ϕ is uniquely-sequentializable, then for all configuration $\mathbf{p}$, $\mathbf{Post}^*_{\mathrm{s}}(\phi, \mathbf{p}) = \mathbf{Post}^*_{\mathrm{ss}}(\phi, \mathbf{p}) = \mathbf{Post}^*_{\mathrm{as}}(\phi, \mathbf{p})$.

C. Problems under study

In this work, we aim at verifying properties on protocols where we assume that the number of robots is fixed (equals to $k > 0$) but the size of the rings is parameterized and satisfies a given property. Note that when executing a protocol the size of the ring never changes. For our problems, we consider a ring property that is given by a QFP formula $\texttt{Ring}(y)$, a set of bad configurations given by a QFP formula $\texttt{Bad}(x_1, \ldots, x_k)$ and a set of good configurations given by a QFP formula $\texttt{Goal}(x_1, \ldots, x_k)$. We then define two general problems to address the verification of such algorithms: the SAFE_m problem, and the REACH_m problem, with $m \in \{\mathrm{ss}, \mathrm{s}, \mathrm{as}\}$.

The SAFE_m problem is to decide, given a protocol ϕ and two formulae $\texttt{Ring}$ and $\texttt{Bad}$ whether there exists a size $n \in \mathbb{N}$ with $n \in [\![\texttt{Ring}]\!]$, and a (k, n)-configuration $\mathbf{p}$ with $\mathbf{p} \notin [\![\texttt{Bad}]\!]$, such that $\mathbf{Post}^*_m(\phi, \mathbf{p}) \cap [\![\texttt{Bad}]\!] \neq \emptyset$.

The REACH_m problem is to decide given a protocol ϕ and two formulae $\texttt{Ring}$ and $\texttt{Goal}$ whether there exists a size $n \in \mathbb{N}$ with $n \in [\![\texttt{Ring}]\!]$ and a (k, n)-configuration $\mathbf{p}$, such that $\mathbf{Post}^*_m(\phi, \mathbf{p}) \cap [\![\texttt{Goal}]\!] = \emptyset$. Note that the two problems are not dual due to the quantifiers.

As an example, we can state in our context the SAFE_m problem that consists in checking that a protocol ϕ working with three robots never leads to collision (i.e. to a configuration where two robots are on the same position on the ring) for rings of size strictly bigger than 6. In that case we have $\texttt{Ring} := y > 6$ and $\texttt{Bad} := x_1 = x_2 \ \vee \ x_2 = x_3 \ \vee \ x_1 = x_3$.

III. Undecidability results

In this section, we present undecidability results for the two aforementioned problems. The proofs rely on the encoding of a deterministic k-counter machine run. A deterministic k-counter machine consists of k integer-valued registers (or counters) called $c_1, \ldots, c_k$, and a finite list of labelled instructions L. Each instruction is either of the form $\ell : c_i = c_i + 1; \texttt{goto } \ell'$, or $\ell : \texttt{if } c_i > 0 \texttt{ then } c_i = c_i - 1; \texttt{goto } \ell'; \texttt{else goto } \ell''$, where $i \in [1, k]$. We also assume the existence of a special instruction $\ell_h : \texttt{halt}$. Configurations of a k-counter machine are elements of $L \times \mathbb{N}^k$, giving the current instruction and the current values of the registers. The initial configuration is $(\ell_0, 0, \ldots, 0)$, and the set of halting configurations is $\mathsf{HALT} = \{\ell_h\} \times \mathbb{N}^k$.

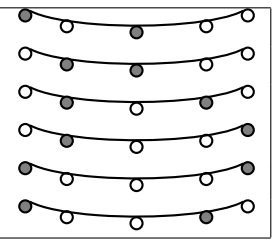
Given a configuration $(\ell, n_1, \ldots, n_k)$, the successor configuration $(\ell', n'_1, \ldots, n'_k)$ is defined in the usual way and we note $(\ell, n_1, \ldots, n_k) \vdash (\ell', n'_1, \ldots, n'_k)$. A run of a k-counter machine is a (finite or infinite) sequence of configurations $(\ell_0, n_1^0, \ldots, n_k^0), (\ell_1, n_1^1, \ldots, n_k^1) \cdots$, where $(\ell_0, n_1^0, \ldots, n_k^0)$ is the initial configuration, and, for all $i \geq 0$, $(\ell_i, n_1^i, \ldots, n_k^i) \vdash (\ell_{i+1}, n_1^{i+1}, \ldots, n_k^{i+1})$. The run is finite if and only if it ends in a halting configuration, i.e. in a configuration in HALT.

Theorem 2: $\mathsf{SAFE}_{\mathrm{as}}$ is undecidable.

Sketch of proof. The proof relies on a reduction from the halting problem of a deterministic two-counter machine M to $\mathsf{SAFE}_{\mathrm{as}}$ with $k = 42$ robots. It is likely that an encoding using less robots might be used for the proof, but for the sake of clarity, we do not seek the smallest possible amount of robots. The halting problem is to decide whether the run of a given deterministic two-counter machine is finite; this problem is undecidable [21]. The idea is to simulate the run of M in a way that ensures that a collision occurs if and only if M halts. Positions of robots on the ring are used to encode values of counters and the current instruction of the machine. The k-protocol makes sure that movements of the robots simulate correctly the run of M. Moreover, one special robot moves only when the initial configuration is encoded, and another only when the final configuration is encoded. The collision is ensured in the following sequence of actions of the robots: when the initial configuration is encoded, the first robot computes its action but does not move immediately. When the halting configuration is reached, the second robot computes its action and moves, then the first robot finally completes its move, entailing the collision. Note that if the ring is not big enough to simulate the counter values then the halting configuration is never reached and there is no collision.

Instead of describing configurations of the robots by applications giving positions of the robots on the ring, we use a sequence of letters F or R, representing respectively a free node and a node occupied by a robot. When a letter $\mathsf{A} \in \{\mathsf{F}, \mathsf{R}\}$ is repeated i times, we use the notation A^i, when it is repeated an arbitrary number or times (including 0), we use A^*. To distinguish between the two representations of the configurations, we use respectively the terms configurations or word-configurations. The correspondence between a configuration and a word-configuration is obvious. A *machine-like* (word-)configuration is a configuration of the form $\mathsf{B}_3\mathsf{F}^*\mathsf{RF}^*\mathsf{B}_4\mathsf{F}^*\mathsf{RF}^*\mathsf{B}_5\mathsf{F}^*\mathsf{RF}^*\mathsf{B}_6\mathsf{F}^*\mathsf{RF}^*\mathsf{B}_7\mathsf{F}^*\mathsf{RF}^*\mathsf{B}_8 P_1P_2P_3P_4P_5$ RFR, where B_i is a shorthand for $\mathsf{FR}^i\mathsf{F}$, and $P_1P_2 \in \{\mathsf{RF}, \mathsf{FR}\}$ and exactly one $P_i = \mathsf{R}$ for $i \in \{3,4,5\}$, $P_j = \mathsf{F}$ for $j \in \{3,4,5\} \setminus \{i\}$ (see Table I for a graphic representation of the section $P_1P_2P_3P_4P_5$ of the ring). Observe that the different blocks B_i yield for every robot in the ring a distinct view, since it allows the robots to locate their position on the ring. Hence, in the rest of the proof we abuse notations and describe the protocol using different names for the different robots, according to their position in the ring, even if they are formally anonymous. We let $\mathcal{R}$ be the set of robots involved. A machine-like (word-)configuration $\mathsf{B}_3\mathsf{F}^{n_1}\mathsf{R}_{c_1}\mathsf{F}^*\mathsf{B}_4\mathsf{F}^{n_2}\mathsf{R}_{c_2}\mathsf{F}^*\mathsf{B}_5\mathsf{F}^m\mathsf{R}_c\mathsf{F}^n\mathsf{B}_6\mathsf{F}^i\mathsf{R}_\ell\mathsf{F}^{i'}\mathsf{B}_7\mathsf{F}^p\mathsf{R}_{\ell'}\mathsf{F}^r\mathsf{B}_8$ $\mathsf{R}_{tt}\mathsf{FR}_t\mathsf{FFR}_g\mathsf{FR}_d$ is said to be *stable* because of the positions of robots R_t and R_{tt} (see Table I). Moreover, it encodes the configuration (ℓ_i, n_1, n_2) of M (due to the relative positions of robots R_{c_1}, R_{c_2} and R_ℓ respectively to B_3, B_4 and B_6). We say that a configuration $\mathbf{p}$ is machine-like, stable, etc. if its corresponding word-configuration is machine-like, stable, etc. In the following, we distinguish configurations of the 2-counter machine, and configurations of the robots, by calling them respectively M-configurations and ϕ-(word)-configurations. For a stable and machine-like ϕ-configuration $\mathbf{p}$, we let $M(\mathbf{p})$ be the M-configuration encoded by $\mathbf{p}$. We first present the part of the algorithm simulating the behavior of M. We call this algorithm ϕ'. Since the machine is deterministic, only one instruction is labelled by ℓ_i, known by every robot. The simulation follows different steps, according to the positions of the robots R_t and R_{tt}, as pictured in Table I.

TABLE I: Different types of configurations

stable configuration	$\mathsf{R}_{tt}\mathsf{FR}_t\mathsf{FF}$	
moving1 configuration	$\mathsf{FR}_{tt}\mathsf{R}_t\mathsf{FF}$	
moving2 configuration	$\mathsf{FR}_{tt}\mathsf{FR}_t\mathsf{F}$	
moving3 configuration	$\mathsf{FR}_{tt}\mathsf{FFR}_t$	
stabilizing1 configuration	$\mathsf{R}_{tt}\mathsf{FFFR}_t$	
stabilizing2 configuration	$\mathsf{R}_{tt}\mathsf{FFR}_t\mathsf{F}$	

We explain the algorithm ϕ' on the configuration (ℓ_i, n_1, n_2) with the transition ℓ_i : `if` $c_1 > 0$ `then` $c_1 = c_1 - 1$;`goto` ℓ_j;`else goto` $\ell_{j'}$.

- When in a stable configuration, robot R_{tt} first moves to obtain a moving1 configuration.
- In a moving1 configuration, robot R_c moves until it memorizes the current value of c_1. More precisely, in a moving1 configuration where $n_1 \neq m$, robot R_c moves : if $n_1 > m$, and $n \neq 0$, R_c moves towards B_6, if $n_1 < m$, it moves towards B_5, if $n_1 > m$ and $n = 0$, it does not move.
- In a moving1 configuration where $n_1 = m$, R_t moves to obtain a moving2 configuration.
- In a moving2 configuration, if $n_1 = m \neq 0$, then R_{c_1} moves towards B_3, hence encoding the decrementation of c_1.
- In a moving2 configuration, if $n_1 = m = 0$ or if $n_1 \neq m$, (then the modification of c_1 is either impossible, or already done), robot $\mathsf{R}_{\ell'}$ moves until it memorizes the position of robot R_ℓ: if $p < i$, and $r \neq 0$, $\mathsf{R}_{\ell'}$ moves towards B_8; if $p > i$, $\mathsf{R}_{\ell'}$ moves towards B_7.
- In a moving2 configuration, if $p = i$, then R_t moves to obtain a moving3 configuration.
- In a moving3 configuration, if $n_1 = m = 0$, and robot $\mathsf{R}_{\ell'}$ encodes ℓ_i (i.e. $p = i$), then $c_1 = 0$ and robot R_ℓ has to move until it encodes $\ell_{j'}$. If on the other hand $n_1 < m$, then robot R_ℓ moves until it encodes ℓ_j. More precisely, if $n_1 = m = 0$, and the position encoded by R_ℓ is smaller than j' ($i < j'$), and if $i' \neq 0$, then R_ℓ moves towards B_7. If $n_1 = m = 0$, and the position encoded by R_ℓ is greater than j', R_ℓ moves towards B_6. If $n_1 < m$, then robot R_ℓ

moves in order to reach a position where it encodes ℓ_j (towards B_6 if $i > j$, towards B_7 if $i < j$ and $i' \neq 0$).

- In a moving3 configuration, if the position encoded by $\mathsf{R}_{\ell'}$ is ℓ_i, if $n_1 = m = 0$ and the position encoded by R_ℓ is $\ell_{j'}$, or if $n_1 \neq m$, and the position encoded by R_ℓ is ℓ_j, then the transition has been completely simulated : the counters have been updated and the next transition is stored. The robots then return to a stable configuration: robot R_{tt} moves to obtain a stabilizing1 configuration.
- In a stabilizing1 configuration, robot R_t moves to obtain a stabilizing2 configuration.
- In a stabilizing2 configuration, robot R_t moves to obtain a stable configuration.

For other types of transitions, the robots move similarly. When in a stable configuration encoding a configuration in HALT, no robot moves. We describe now the algorithm ϕ that simply adds to ϕ' the two following rules. Robot R_g (respectively R_d) moves in the direction of R_d (respectively in the direction of R_g) if and only if the robots are in a stable machine-like configuration, and the encoded configuration of the machine is $(\ell_0, 0, 0)$ (respectively is in HALT), (since the configuration is machine-like, the distance between R_g and R_d is 2). Observe that if the sub-algorithm ϕ' is uniquely-sequentializable, ϕ is not.

On all configurations that are not machine-like, the algorithm makes sure that no robot move. This implies that once R_g or R_d has moved, no robot with a view up-to-date ever moves. One can easily be convinced that the algorithm can be expressed by a QFP formula ϕ.

Let the formulae $\mathtt{Bad}(\mathbf{p}_1, \ldots, \mathbf{p}_{42}) = \bigvee_{\substack{i,j \in [1,42] \\ i \neq j}} (\mathbf{p}_i = \mathbf{p}_j)$ that is satisfied by all the configurations where two robots share the same position and $\mathtt{Ring}(y) = y \geq 0$. We can show that M halts if and only if there exits a size $n \in [\![\mathtt{Ring}]\!]$, a $(42, n)$-configuration $\mathbf{p}$ with $\mathbf{p} \notin [\![\mathtt{Bad}]\!]$, such that R_g and R_d eventually collide, i.e., $\mathbf{Post}^*_{\mathrm{as}}(\phi, \mathbf{p}) \cap [\![\mathtt{Bad}]\!] \neq \emptyset$. Note that R_g and R_d can collide only in an asynchronous run.

□

Theorem 3: REACH_m is undecidable, for $m \in \{\mathrm{ss}, \mathrm{s}, \mathrm{as}\}$.

Sketch of proof. The proof relies on a reduction from the repeated reachability problem of a deterministic three-counter zero-initializing bounded-strongly-cyclic machine M, which is undecidable [19]. A counter machine is zero-initializing if from the initial instruction ℓ_0 it first sets all the counters to 0. Moreover, an infinite run is said to be *space-bounded* if there is a value $K \in \mathbb{N}$ such that all the values of all the counters stay below K during the run. A counter machine M is bounded-strongly-cyclic if every space-bounded infinite run starting from any configuration visits ℓ_0 infinitely often. The repeated reachability problem we consider is expressed as follows: given a 3-counter zero-initializing bounded-strongly-cyclic machine M, does there exist an infinite space-bounded run of M? A configuration of M is encoded in the same fashion than in the proof of Theorem 2, with 3 robots encoding the values of the counters. A transition of M is simulated by the algorithm in the same way than above *except that if a counter is to be increased, the corresponding robot moves accordingly even if there is no room to do it, yielding a collision*. Since the machine is bounded-strongly-cyclic and zero-initializing, any infinite run will eventually visit $(\ell_0, 0, 0, 0)$, so any infinite execution of the robots encode an infinite space-bounded run of M starting in $(\ell_0, 0, 0, 0)$.

□

IV. Decidability results and case study

In this section, we show that even if $\mathsf{SAFE}_{\mathrm{as}}$, $\mathsf{REACH}_{\mathrm{as}}$, $\mathsf{REACH}_{\mathrm{ss}}$ and $\mathsf{REACH}_{\mathrm{s}}$ are undecidable, the other cases $\mathsf{SAFE}_{\mathrm{s}}$ and $\mathsf{SAFE}_{\mathrm{ss}}$ can be reduced to the satisfiability problem for EP formulae, which is decidable and NP-complete [8].

A. Reducing safety to successor checking

The first step towards decidability is to remark that to solve $\mathsf{SAFE}_{\mathrm{s}}$ and $\mathsf{SAFE}_{\mathrm{ss}}$ it is enough to look at the one-step successor. Let ϕ be a protocol over k robots and $\mathtt{Ring}$ and $\mathtt{Bad}$ be respectively a ring property and a set of bad configurations. We have then the following lemma.

Lemma 1: Let $n \in \mathbb{N}$ such that $n \in [\![\mathtt{Ring}]\!]$ and $m \in \{\mathrm{s}, \mathrm{ss}\}$. There exists a (k,n)-configuration $\mathbf{p}$ with $\mathbf{p} \notin [\![\mathtt{Bad}]\!]$, such that $\mathbf{Post}^*_m(\phi, \mathbf{p}) \cap [\![\mathtt{Bad}]\!] \neq \emptyset$ iff there exists a (k,n)-configuration $\mathbf{p}'$ with $\mathbf{p}' \notin [\![\mathtt{Bad}]\!]$, such that $\mathbf{Post}_m(\phi, \mathbf{p}') \cap [\![\mathtt{Bad}]\!] \neq \emptyset$.

This last result may seems strange at a first sight but it can easily be explained by the fact that robots protocols are most of the time designed to work without any assumption on the initial configuration, except that it is not a bad configuration.

B. Encoding successor computation in Presburger

We now describe various EP formulae to be used to express the computation of the successor configuration in synchronous and semi-synchronous mode.

First we show how to express the view of some robot R_i in a configuration $\mathbf{p}$, with the following formula:

$$\begin{gathered}\mathtt{ConfigView}_i(y, p_1, \ldots, p_k, d_1, \ldots, d_k) := \\ \exists d'_1, \ldots, d'_{k-1}.i_1, \ldots, i_{k-1} \cdot \textstyle\bigwedge_{j=1}^{k-2} d'_j \leq d'_{j+1} \wedge \\ \textstyle\bigwedge_{\ell=1}^{k-1} (\bigvee_{j=1, j\neq i}^{k} p_j = (p_i + d'_\ell \mod y) \wedge i_\ell = j) \wedge \\ 0 < d'_1 \wedge \textstyle\bigwedge_{j=1}^{k-1} d'_j \leq y \wedge \bigwedge_{\ell \neq j} i_\ell \neq i_j \wedge \\ d_1 = d'_1 \wedge \textstyle\bigwedge_{j=2}^{k-1} d_j = d'_j - d'_{j-1} \wedge d_k = y - d'_{k-1},\end{gathered}$$

Note that this formula only expresses in the syntax of Presburger arithmetic the definition of $\mathbf{V}_{\mathbf{p}}[i \rightarrow]$ where the variable y is used to store the length of the ring, $p_1, \ldots, p_k$ represent $\mathbf{p}$ and the variables $d_1, \ldots, d_k$ represent the view.

We also use the formula $\mathtt{ViewSym}(d_1, \ldots, d_k, d'_1, \ldots, d'_k)$ that is useful to compute the symmetric of a view.

$$\begin{gathered}\mathtt{ViewSym}(d_1, \ldots, d_k, d'_1, \ldots, d'_k) := \\ \textstyle\bigvee_{j=1}^{k} (\bigwedge_{\ell=j+1}^{k} (d_\ell = 0 \wedge d'_\ell = 0) \wedge \bigwedge_{\ell=1}^{j} d'_\ell = d_{j-\ell+1} \wedge d_j \neq 0)\end{gathered}$$

We are now ready to introduce the formula $\mathtt{Move}_i^\phi(y, p_1, \ldots, p_k, d_1, \ldots, d_k, p')$, which is true if and only if, on a ring of size n (represented by the variable y), the move of robot R_i according to the protocol $\phi(d_1, \ldots, d_k)$ from the configuration $\mathbf{p}$ yields to the new position p'. Here the variables $p_1, \ldots, p_k$ characterizes $\mathbf{p}$ and $d_1, \ldots, d_k$ the view of

process i. Note that in the asynchronous semantics, the view of i might completely differ from the current configuration.

$$\begin{gathered}\texttt{Move}_i^{\phi}(y,p_1,\ldots,p_k,d_1,\ldots,d_k,p') := \\ \exists d'_1,\cdots,d'_k \cdot \texttt{ViewSym}(d_1,\ldots,d_k,d'_1,\ldots,d'_k)\wedge \\ \Big(\phi(d_1,\ldots,d_k)\wedge\big((p_i<y-1\wedge p'=p_i+1) \\ \vee(p_i=y-1\wedge p'=0)\big)\Big)\vee \\ \Big(\phi(d'_1,\ldots,d'_k)\wedge\big((p_i>0\wedge p'=p_i-1)\vee(p_i=0\wedge p'=y-1)\big)\Big) \\ \vee\Big(\neg\phi(d_1,\ldots,d_k)\wedge\neg\phi(d'_1,\ldots,d'_k)\wedge(p'=p_i)\Big)\end{gathered}$$

Now, given two (k,n)-configurations $\mathbf{p}$ and $\mathbf{p}'$, and a k-protocol ϕ, it is easy to express the fact that $\mathbf{p}'$ is a successor configuration of $\mathbf{p}$ according to ϕ in a semi-synchronous run (resp. synchronous run); for this we define the two formulae $\texttt{SemiSyncPost}_{\phi}(y,p_1,\ldots,p_k,p'_1,\ldots,p'_k)$ and $\texttt{SyncPost}_{\phi}(y,p_1\ldots,p_k,p'_1,\ldots,p'_k))$ as follows:

$$\begin{gathered}\texttt{SemiSyncPost}_{\phi}(y,p_1,\ldots,p_k,p'_1,\ldots,p'_k) := \exists d_1^1,\ldots,d_k^1,\ldots \\ d_1^k,\ldots,d_k^k \cdot \textstyle\bigwedge_{j=1}^k \texttt{ConfigView}_j(y,p_1,\ldots,p_k,d_1^j,\ldots,d_k^j)\wedge \\ \textstyle\bigvee_{i=1}^k\big(\texttt{Move}_i^{\phi}(y,p_1,\ldots,p_k,d_1^i,\ldots,d_k^i,p'_i)\wedge \\ \textstyle\bigwedge_{j=1,j\neq i}^k((p'_j=p_j)\vee\texttt{Move}_j^{\phi}(y,p_1,\ldots,p_k,d_1,\ldots,d_k,p'_j))\big)\end{gathered}$$

$$\begin{gathered}\texttt{SyncPost}_{\phi}(y,p_1,\ldots,p_k,p'_1,\ldots,p'_k) := \exists d_1^1,\ldots,d_k^1,\ldots, \\ d_1^k,\ldots,d_k^k \cdot \textstyle\bigwedge_{i=1}^k\big(\texttt{ConfigView}_i(y,p_1,\ldots,p_k,d_1^i,\ldots,d_k^i)\wedge \\ \texttt{Move}_i^{\phi}(y,p_1,\ldots,p_k,d_1^i,\ldots,d_k^i,p'_i)\big)\end{gathered}$$

Lemma 2: For all $n\in\mathbb{N}$ and all (k,n)-configurations $\mathbf{p}$ and $\mathbf{p}'$, we have:

1) $\mathbf{p}\hookrightarrow\mathbf{p}'$ if and only if $n,\mathbf{p},\mathbf{p}'\models\texttt{SemiSyncPost}_{\phi}$,
2) $\mathbf{p}\Rightarrow\mathbf{p}'$ if and only if $n,\mathbf{p},\mathbf{p}'\models\texttt{SyncPost}_{\phi}$.

C. Results

Now since to solve $\mathsf{SAFE}_{\mathrm{ss}}$ and $\mathsf{SAFE}_{\mathrm{s}}$, we only need to look at the successor in one step, as stated by Lemma 1, and thanks to the formulae $\texttt{SemiSyncPost}_{\phi}$ and $\texttt{SyncPost}_{\phi}$ and their properties expressed by Lemma 2, we deduce that these two problems can be expressed in Presburger arithmetic.

Theorem 4: $\mathsf{SAFE}_{\mathrm{s}}$ and $\mathsf{SAFE}_{\mathrm{ss}}$ are decidable and in NP.

Proof: We consider a ring property $\texttt{Ring}(y)$, a protocol ϕ for k robots (which is a QFP formula) and a set of bad configurations given by a QFP formula $\texttt{Bad}(x_1,\ldots,x_k)$. We know that there exists a size $n\in\mathbb{N}$ with $n\in[\![\texttt{Ring}]\!]$, and a (k,n)-configuration $\mathbf{p}$ with $\mathbf{p}\notin[\![\texttt{Bad}]\!]$, such that $\mathbf{Post}_{\mathrm{s}}^*(\phi,\mathbf{p})\cap[\![\texttt{Bad}]\!]\neq\emptyset$ if and only if there exists a (k,n)-configuration $\mathbf{p}'$ with $\mathbf{p}'\notin[\![\texttt{Bad}]\!]$, such that $\mathbf{Post}_m(\phi,\mathbf{p}')\cap[\![\texttt{Bad}]\!]\neq\emptyset$. By Lemma 2, this latter property is true if and only if the following formula is satisfiable:

$$\begin{gathered}\texttt{SyncPost}_{\phi}(y,p_1,\ldots,p_k,p'_1,\ldots,p'_k)\wedge \\ \texttt{Ring}(y)\wedge\neg\texttt{Bad}(p_1,\ldots,p_k)\wedge \\ \texttt{Bad}(p'_1,\ldots,p'_k)\end{gathered}$$

For the semi-synchronous case, we replace the formula $\texttt{SyncPost}_{\phi}$ by $\texttt{SemiSyncPost}_{\phi}$. The NP upper bound is obtained by the fact that the built formula is an EP formula. □

When the protocol ϕ is uniquely-sequentializable, i.e. when in each configuration at most one robot make the decision to move then Theorem 1 leads us to the following result.

Corollary 1: When the protocol ϕ is uniquely-sequentializable, $\mathsf{SAFE}_{\mathrm{as}}$ is decidable.

D. Expressing other interesting properties

Not only the method consisting in expressing the successor computation in Presburger arithmetic allows us to obtain the decidability for $\mathsf{SAFE}_{\mathrm{s}}$ and $\mathsf{SAFE}_{\mathrm{ss}}$, but they also allow us to express other interesting properties. For instance, we can compute the successor configuration in asynchronous mode for a protocol ϕ working over k robots thanks to the formula $\texttt{AsyncPost}_{\phi}(y,p_1,\ldots,p_k,s_1,\ldots,s_k,v_1\ldots,v_k,p'_1,\ldots,p'_k,s'_1,\ldots,s'_k,v'_1,\ldots,v'_k)$, which is given by:

$$\begin{gathered}\texttt{AsyncPost}_{\phi} := \exists d_1,\ldots,d_k\cdot \\ \textstyle\bigvee_{i=1}^k\Big(\bigwedge_{j\neq i}(p'_j=p_j\wedge s'_j=s_j\wedge v'_j=v_j)\wedge \\ s'_i=1-s_i\wedge\big((s_i=0\wedge v'_i=\langle d_1,\ldots,d_k\rangle\wedge \\ \texttt{ConfigView}_i(y,p_1,\ldots,p_k,d_1,\ldots,d_k)\wedge p'_i=p_i)\vee \\ (s_i=1\wedge v'_i=v_i\wedge\texttt{Move}_i^{\phi}(y,p_1,\ldots,p_k,d_1,\ldots,d_k,p'_i)\big)\Big)\end{gathered}$$

To prove the correctness of this formula for an asynchronous configuration $(\mathbf{p},\mathbf{s},\mathbf{V})$ with k robots we make the analogy between the flags $\mathbf{L}$ and $\mathbf{M}$ and the naturals 0 and 1, which means that in the definition of the vector $\mathbf{s}\in\{\mathbf{L},\mathbf{M}\}^k$, we encode $\mathbf{L}$ by 0 and $\mathbf{M}$ by 1 and we then apply the definition of $\rightarrow_{as}$.

One can also express the fact that one configuration is a predecessor of the other in a straightforward way.

It is as well possible to check whether a protocol ϕ over k robots fits into the hypothesis of Corollary 1, i.e. whether it is uniquely-sequentializable. We define the formula $\texttt{UniqSeq}_{\phi}$ that is satisfiable if and only if ϕ is uniquely-sequentializable.

$$\begin{gathered}\texttt{UniqSeq}_{\phi} := \neg\exists y.p_1,\ldots,p_k,p'_1,\ldots,p'_k\cdot \\ \textstyle\bigvee_{i\neq j,1\leq i,j\leq k}(\texttt{Move}_i^{\phi}(n,p_1,\ldots,p_k,p'_i)\wedge\texttt{Move}_j^{\phi}(n,p_1,\ldots,p_k,p'_j) \\ \wedge p'_i\neq p_i\wedge p'_j\neq p_j.\end{gathered}$$

Hence we deduce the following statement.

Theorem 5: Checking whether a protocol ϕ is uniquely-sequentializable is decidable.

E. Applications

We have considered the *exclusive perpetual exploration* algorithms proposed by Blin *et al.* [6], and generated the formulae to check that no collision are encountered for different cases. We have used the SMT solver Z3 [12] to verify whether the generated formulae were satisfiable or not. We have been able to prove that, in the synchronous case, the algorithm using a minimum of 3 robots was safe for any ring of size greater than 10 and changing a rule of the algorithm has allowed us to prove that we could effectively detect bugs in the Algorithm. In fact, in this buggy case, the SMT solver provides a configuration leading to a collision after one step. We have then looked for absence of collision for the algorithms using a maximum number of robots, always in the synchronous case. Here, the

verification was not parametric as the size of the ring is fixed and depends on the number of robots (it is exactly 5 plus the number of robots). The algorithm in [6] was designed to work for any number of robots k odd and co-prime with $k+5$. However, we found bugs for $k = 7, 9$. Note that for 11 robots, the SMT solver Z3 was taking more than 10 minutes and we did not let him finish its computation. We observe that when there is a bug, the SMT solver goes quite fast to generate a bad configuration but it takes much more time when the algorithm is correct. The files containing the SMT formulae are all available on the webpage [15] in the SMTLIB format.

V. Conclusion

We have addressed two main problems concerning formal verification of protocols of mobile robots, and answered the open questions regarding decidability of the verification of such protocols, when the size of the ring is given as a parameter of the problem. Note that in such algorithms, robots can start in any position on the ring. Simple modifications of the proofs in this paper allow to obtain undecidability of both the reachability and the safety problem in any semantics, when the starting configuration of the robots is given. Hence we give a precise view of what can be achieved in the automated verification of protocols for robots in the parameterized setting, and provide a means of partially verifying them. Of course, to fully demonstrate the correctness of a tentative protocol, more properties are required (like, all nodes are visited infinitely often) that are not handled with our approach. Nevertheless, as intermediate lemmas (for arbitrary n) are verified, the whole process of proof writing is both eased and strengthened.

An application of Corollary 1 and Theorem 5 deals with robot program synthesis as depicted in the approach of Bonnet *et al.* [7]. To simplify computations and save memory when synthesizing mobile robot protocols, their algorithm only generates uniquely-sequentializable protocols (for a given k and n). Now, given a protocol description for a given n, it becomes possible to check whether this protocol remains uniquely-sequentializable for any n. Afterwards, regular safety properties can be devised for this tentative protocol, for all models of computation (that is, FSYNC, SSYNC, and ASYNC). Protocol design is then driven by the availability of a uniquely-serializable solution, a serious asset for writing handwritten proofs (for the properties that cannot be automated).

Last, we would like to mention possible applications of our approach for problems whose core properties seem related to reachability only. One such problem is exploration with stop [5]: robots have to explore and visit every node in a network, then stop moving forever, assuming that all robots initial positions are distinct. All of the approaches published for this problem make use of *towers*, that is, locations that are occupied by at least two robots, in order to distinguish the various phases of the exploration process (initially, as all occupied nodes are distinct, there are no towers). Our approach still makes it possible to check if the number of created towers remains acceptable (that is below some constant, typically 2 per block of robots that are equally spaced) from any given configuration in the algorithm, for any ring size n. As before, such automatically obtained lemmas are very useful when writing the full correctness proof.

References

[1] K. R. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, 1986.

[2] C. Auger, Z. Bouzid, P. Courtieu, S. Tixeuil, and X. Urbain. Certified Impossibility Results for Byzantine-Tolerant Mobile Robots. In *Proc. of SSS'13*, volume 8255 of *LNCS*, pages 178–186. Springer, 2013.

[3] T. Balabonski, A. Delga, L. Rieg, S. Tixeuil, and X. Urbain. Synchronous gathering without multiplicity detection: A certified algorithm. In *Proc. of SSS'16*, volume 10083 of *LNCS*, pages 7–19. Springer, 2016.

[4] B. Bérard, P. Courtieu, L. Millet, M. Potop-Butucaru, L. Rieg, N. Sznajder, S. Tixeuil, and X. Urbain. Formal Methods for Mobile Robots: Current Results and Open Problems. *Int. J. Inform. Soc.*, 7(3):101–114, 2015. Invited Paper.

[5] B. Bérard, P. Lafourcade, L. Millet, M. Potop-Butucaru, Y. Thierry-Mieg, and S. Tixeuil. Formal verification of mobile robot protocols. *Distr. Comp.*, 2016.

[6] L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil. Exclusive perpetual ring exploration without chirality. In *Proc. of DISC'10*, volume 6343 of *LNCS*, pages 312–327. Springer, 2010.

[7] F. Bonnet, X. Défago, F. Petit, M. Potop-Butucaru, and S. Tixeuil. Discovering and assessing fine-grained metrics in robot networks protocols. In *Proc. of SRDS'14*, pages 50–59. IEEE Press., 2014.

[8] I. Borosh and L. Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Amer. Math. Soc.*, 55:299–304, 1976.

[9] P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain. Impossibility of Gathering, a Certification. *Inf. Process. Lett.*, 115:447–452, 2015.

[10] P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain. Certified universal gathering in \mathbb R ^2 for oblivious mobile robots. In *Proc. of DISC'16*, volume 9888 of *LNCS*, pages 187–200. Springer, 2016.

[11] G. D'Angelo, G. D. Stefano, A. Navarra, N. Nisse, and K. Suchan. A unified approach for different tasks on rings in robot-based computing systems. In *Proc. of IPDPSW'13*, pages 667–676. IEEE Press., 2013.

[12] L. M. de Moura and N. Bjørner. Z3: an efficient SMT solver. In *TACAS'08*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[13] S. Devismes, A. Lamani, F. Petit, P. Raymond, and S. Tixeuil. Optimal Grid Exploration by Asynchronous Oblivious Robots. In *Proc. of SSS'12*, volume 7596 of *LNCS*, pages 64–76. Springer, 2012.

[14] H. T. T. Doan, F. Bonnet, and K. Ogata. Model checking of a mobile robots perpetual exploration algorithm. In *Proc. of SOFL+MSVL, Revised Selected Papers*, volume 10189 of *LNCS*, pages 201–219, 2016.

[15] https://www.irif.fr/~sangnier/robots.html.

[16] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. *Algorithmica*, 65(3):562–583, 2013.

[17] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synt. Lect. Distr. Comp. Th. Morgan & Claypool Publishers, 2012.

[18] E. Kranakis, D. Krizanc, and E. Markou. *The Mobile Agent Rendezvous Problem in the Ring*. Synt. Lect. Distr. Comp. Th. Morgan & Claypool Publishers, 2010.

[19] R. Mayr. Undecidable problems in unreliable computations. *Theoret. Comput. Sci.*, 297(1-3):337–354, 2003.

[20] L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In *Proc. of SSS'14*, volume 8756 of *LNCS*, pages 237–251. Springer, 2014.

[21] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[22] S. Rubin, F. Zuleger, A. Murano, and B. Aminof. Verification of asynchronous mobile-robots in partially-known environments. In *Proc. of PRIMA'15*, volume 9387 of *LNCS*, pages 185–200. Springer, 2015.

[23] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999.