Formal verification of mobile robot protocols

Beatrice Bérard, Pascal Lafourcade, Laure Millet, Maria Potop-Butucaru, Yann Thierry-Mieg, Sébastien Tixeuil

Noname manuscript No.
(will be inserted by the editor)

Formal Verification of Mobile Robot Protocols

Béatrice Bérard · Pascal Lafourcade · Laure Millet · Maria Potop-Butucaru · Yann Thierry-Mieg · Sébastien Tixeuil

Received: date / Accepted: date

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, in the case of asynchronous execution models, are both cumbersome and error-prone.

Our contribution is twofold. We first propose a formal model to describe mobile robot protocols operating in a discrete space *i.e.*, with a finite set of possible robot positions, under synchrony and asynchrony assumptions. We translate this formal model into the DVE language, which is the input format of the model-checkers DiVinE and ITS tools, and formally prove the equivalence of the two models. We then verify several instances of two existing protocols for variants of the ring exploration in an asynchronous setting: exploration with stop and perpetual exclusive exploration. For the first protocol we refine the correctness bounds and for the second one, we exhibit a counter-example. This protocol is then modified and we establish the correctness of the new version with an inductive proof.

B. Bérard
Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu 75005 Paris
E-mail: beatrice.berard@lip6.fr

P. Lafourcade
University Clermont Auvergne, CNRS, LIMOS UMR 6158, Clermont, France
E-mail: pascal.lafourcade@imag.fr

L. Millet
Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu 75005 Paris
E-mail: laure.millet@lip6.fr

M. Potop-Butucaru
Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu 75005 Paris
E-mail: Maria.Potop-Butucaru@lip6.fr

Y. Thierry-Mieg
Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu 75005 Paris
E-mail: Yann.Thierry-Mieg@lip6.fr

S. Tixeuil
Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu 75005 Paris
Institut Universitaire de France
E-mail: Sebastien.Tixeuil@lip6.fr

1 Introduction

The variety of tasks that can be performed by autonomous robots and their complexity are both increasing [1]. Many applications envision groups of mobile robots self-organizing and cooperating toward the resolution of common objectives, in the absence of any central coordinating authority. Possible applications for such multi-robot systems include environmental monitoring, map construction, urban search and rescue, surface cleaning, surround-

ing or surveillance of risky areas, exploration of unknown environments, etc. Some of these applications can be critical, implying the need for formal and exhaustive verification.

In this paper we focus in particular on patrolling problems [1,2]. Patrolling consists in moving around an area in order to inspect or to protect it, which can be useful for domains where distributed surveillance is required. These problems have thus a broad area of applications ranging from military to civil [2] and are typically critical systems. They can be solved by so-called exploration protocols, for which two variants are studied in this work: exploration with stop and perpetual exclusive exploration, respectively proposed in [3] and [4]. The first one corresponds to a finite exploration phase followed by a termination phase. Exploration with exclusivity corresponds to a particular form of patrolling where the locations to be visited are too small to be occupied by more than one robot at a time.

Robot protocols. The model introduced by Suzuki & Yamashita [5] features a distributed system of k mobile robots with 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 other robots. They are also able to evaluate any predicate on the set of positions.

A recent trend was to shift from the original *continuous* setting where robots evolve in a continuous two-dimensional Euclidian space, to a discrete one, partitioned into a *finite* number of locations. The discretization process is motivated by practical aspects with respect to the unreliability of sensing devices used by the robots as well as inaccuracy of their motorization [6]. This discrete space is described by a graph, where nodes represent locations, and edges represent paths for robots from one location to the other. While the discrete setting permits to simplify the design of robot models by reasoning on finite structures, it is more sensitive to the size of constants. This can lead to a significant increase of the number of symmetric configurations when the underlying graph is also symmetric (*e.g.* a ring) and thus of the size of correctness proofs [7,8,9].

Robots operate in cycles of three phases: *Look*, *Compute* and *Move*. During the *Look* phase robots take a snapshot of the graph together with other robot positions. The collected information is used in the *Compute* phase where robots decide to move or to stay idle. In the *Move* phase, a robot may move to one of its adjacent nodes according to the computation of the previous phase, if it is scheduled.

In the original model of [5], 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, and SSYNC, for the semi-synchronous model, where a strict subset of robots can be scheduled. This model was later generalized by Flocchini *et al.* [10] 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-synchonous 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.

Comparison with classical distributed algorithms. Several particularities of robot networks, as defined by [5], make robot protocols different from classical distributed systems. First, while concurrent snapshots return an identical output in the usual setting, the lack of a common coordinate system makes robots interpret differently the snapshot of the system. Another important point is the fact that outputs of the same function computed at different nodes may be different. For example, consider the minimum function on a set of robot positions. Since these positions can be interpreted differently

by each robot, the minimum function will return different values. Therefore, just executing a classical distributed algorithm for agreement on top of these networks does not necessarily solve a robot agreement task. New algorithms should be designed and hence verified.

Verification. Model checking [11,12] is an appealing technique that was developed for the verification of various models: finite ones but also in some cases infinite, parameterized, or even timed models. It has been successfully used for the verification of distributed systems from classical shared memory (consensus, transactional memory) to population protocols [13,14,15,16,17,18,19,20]. Unfortunately, it was proved in [21] that parameterized model checking is undecidable, and this general result was followed by several stronger ones for specific models, for instance in [22,23,24].

In such cases, there are mainly two approaches. The first one consists in finding decidable properties for subclasses. This was done in [22] for safety properties in Broadcast protocols. Other decidability results were obtained by reducing model checking of a parameterized system to model checking of a finite model called a *cutoff*, for ring protocols [25] and Rendezvous systems [24]. A second line of work investigates the combination of model checking with other techniques like abstraction, induction, etc., as first proposed in [26] or [27]. These ideas were largely used since, for instance in [28,29,30,31,32]. Decidability can also be obtained when the abstractions considered are complete, as done in [32] for a class of threshold-based fault tolerant algorithms. Discrete robot protocols have two parameters: the number of robots and the graph size. Although the problem is still open, we conjecture that parameterized model checking is undecidable in this case, which leads to follow combined approaches.

To our knowledge, in the context of mobile robots operating in discrete space, only two previous attempts, by Devismes *et al.* [33] and by Bonnet *et al.* [34], investigate the possibility of automated verification of mobile robots protocols. The first paper uses LUSTRE [35] 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 mechanically generates all *unambiguous* protocols for k and n in the SSYNC model (that is, all protocols that do *not* have symmetrical configurations). Those two works differ from our proposal in several aspects. First, they are restricted to the simpler SSYNC model rather than the more general and more complex ASYNC model. Second, they are either specific to a hardcoded topology (*e.g.*, a 3×3 grid [33]) that prevents easy reuse in more generic situations, or make additional assumptions about configurations and protocols to be verified (*e.g.* unambiguous protocols [34]) that prevent combinatorial explosion but forbid reuse for proof-challenging protocols, which would most benefit from automatic verification.

Contribution. In this paper, our contribution is twofold: First, in Section 3 and 4, we provide a formal automata-based model for a network of mobile robots operating under the three execution models described above, namely FSYNC, SSYNC and ASYNC. We use the logic LTL (Linear Temporal Logic) to specify the requirements corresponding to robot tasks. Then we transform this model into DVE, the input model of two model checking tools DiVinE [36] and ITS-tools [37], and we formally prove the equivalence of the two models (in terms of possible executions). These implementation issues are discussed in Section 5.

Second, we formally verify several instances of two known protocols for variants of the ring exploration in an asynchronous setting: exploration with stop from [3] and perpetual exclusive exploration from [4]. Modeling these protocols starting from informal descriptions given in the original paper is an arduous task. Nevertheless this leads us either to a formal proof of correctness of the analyzed algorithms for particular instances or to a counter-example that shows a subtle flaw in the algorithm.

– We study the case of exploration with stop, and more particularly the protocol from [3] in Section 6. This protocol was manually proved cor-

rect when the number of robots is $k > 17$, and n (the ring size) and k are co-prime. As the necessity of this bound was not proved in the original paper, our methodology demonstrates that for many instances of k and n not covered in the original paper, the protocol is still correct. More precisely, we propose the following conjecture for all cases with $n < 18$ and $k < 17$: When k is even the algorithm is correct as long as $n < k + \lceil k/2 \rceil$ and $k \geq 10$. When k is odd the algorithm is correct for any $k \geq 5$.

– In Section 7 we study the case of the perpetual exclusive exploration protocol [4]. In this case, model checking with DiVinE produces a counter-example in the completely asynchronous setting, where safety is violated. We correct the original protocol and verify the new one via model checking for several instances of n. Additionally, we prove the correctness of the protocol for any ring size with an inductive approach.

2 Finite automata and LTL logic

We denote by $\mathbb{N}$ the set of natural numbers and we first recall the definitions of finite automata, synchronized products and LTL specifications.

Definition 1 (automaton) A finite automaton is a tuple $M = (S, s_0, A, T)$ where S is a finite set of states, $s_0 \in S$ is the initial state, A is a finite alphabet of actions and $T \subseteq S \times (A \cup \{\varepsilon\}) \times S$ is a finite set of transitions.

A transition (s, a, s'), written $s \xrightarrow{a} s'$, represents a transition of the automaton from state s to state s' by executing the action a. The empty word ε (in the set A^* of words over alphabet A) is used as a label to represent an unobservable (or internal) action.

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

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

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

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

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

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

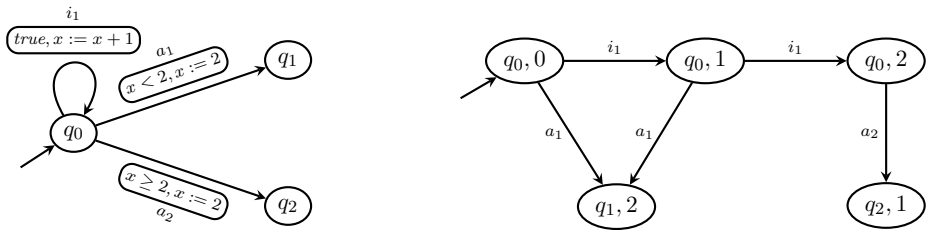
Given a set $\mathcal{P}$ of atomic propositions, the temporal logic LTL (for Linear Temporal Logic) is a specification language interpreted on infinite sequences over $2^{\mathcal{P}}$. The LTL formulae are defined by the following grammar:

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

where $p \in \mathcal{P}$, $\vee$ is the boolean disjunction, $\neg$ is the negation and X (next) and U (until) are temporal operators described below.

To interpret LTL formulae on executions of an automaton $M = (S, s_0, A, T)$, the transition relation is assumed to be *non blocking*: for each $s \in S$, there is at least one transition starting from s. Moreover, a labeling function $\mathcal{L} : S \to 2^{\mathcal{P}}$ is added to M. This function maps each state of M to a set of atomic propositions that hold in this state.

For execution $e : s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots$ of M and formula φ, we note $e, i \models \varphi$ when φ is satisfied at position i of e *i.e.*, on the execution starting in



(a) An automaton with a variable x initially 0 (b) Unfolding into a finite automaton

Fig. 1: Unfolding an automaton with a variable

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

Table 1: LTL satisfaction relation.

s_i. The satisfaction relation is defined inductively by the rules given in Table 1.

Moreover two temporal operators $\Diamond$ and $\Box$ are usually defined from U by: $\Diamond\varphi = true\mathsf{U}\varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$. The formula $\Diamond\varphi$ states that φ holds *eventually*, and $\Box\varphi$ is satisfied *iff* φ holds *forever* from now on.

Temporal and boolean operators can be nested. For instance $\Diamond\Box\varphi$ expresses that from some position in the future φ always holds, and $\Box\Diamond\varphi$ states that φ is satisfied infinitely often.

Definition 3 An automaton M, with labeling $\mathcal{L}$, satisfies φ if for each execution e of M, $e,0 \vDash \varphi$.

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

3 Formal Model for Mobile Robot Protocols

In this section we propose a model for the robots (in Section 3.1), the schedulers (in Section 3.2), and the system resulting from their interactions (in Section 3.3).

To describe the model, we use a classical extension of finite automata by adding variables with bounded range and decorating the actions labeling transitions by predicates on these variables, called *guards*, and assignment to these variables, called *updates*. Note that this extension does not add any expressive power to finite automata and is largely used as input in most verification tools [38,39,40, 41], to permit more compact and readable models. A translation into standard finite automata can be obtained by unfolding the model according to variable values [42], we give an example below.

Example 1 The automaton $\mathcal{A}$ in Figure 1(a) handles a variable $x \in \{0,1,2\}$ initialized to 0. Unfolding yields the standard finite automaton of figure 1(b). Note that this result can also be obtained by a synchronized product between $\mathcal{A}$ and the finite automaton $\mathcal{X}$ depicted in Figure 2, representing the bounded variable x. The synchronization is performed via the updates on x.

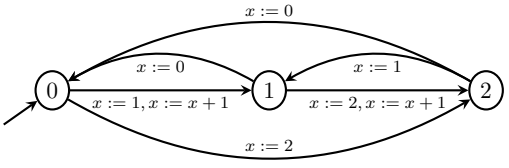


Fig. 2: Automaton $\mathcal{X}$ for variable x

We consider a set $Rob = \{r_1, \ldots, r_k\}$ of robots, evolving on a graph. A node of this graph is called

a *position*, and the set of positions is denoted by $Pos = \{0, \ldots, n-1\} \subseteq \mathbb{N}$. A configuration of the system is a mapping $c : Rob \to Pos$ associating with each robot r its position $c(r) \in Pos$. We denote by $\mathcal{C} = Pos^{Rob}$ the set of all configurations, which is then of cardinality n^k in a graph of n nodes with k robots.

3.1 Robot Modeling

All robots execute the same algorithm [1], operating in *Look*, *Compute*, and *Move cycles*, without any memory of previous actions. Hence the behaviour of each of them can be described by the generic finite automaton of Figure 3.

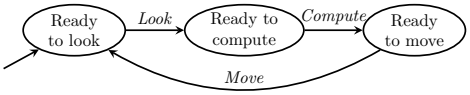


Fig. 3: A generic automaton for the robot behavior.

In the *Look* transition, the robot takes a snapshot of its environment, observing the current configuration c. Then, it computes its future position from this observation in the *Compute* transition. Finally the robot moves along an edge of the graph according to its previous computation: this effective movement corresponds to the *Move* transition.

Note that the original model of [5] abstracts the precise time constraints (like the computational power or the locomotion speed of robots) and keeps only sequences of instantaneous actions, assuming that each robot completes each cycle in finite time. The model can be reduced by combining the *Look* and *Compute* phases to obtain the *LC* phase. This is simply done by merging the two states "Ready to look" and "Ready to compute" into a single state "Ready to Look-Compute".

A robot algorithm is implemented in the *LC* phase, as a set of guarded actions. A guarded action is written as $[guard] \to action$, where the guard is a predicate on the current configuration and the action is the movement assigned to the robot by the protocol under study. In the robot automaton, the "Ready to move" state must then be divided into as many parts as possible movements. The *Move* transition actually performs the computed move by updating the configuration.

3.2 Scheduler Modeling

The scheduler organizes robot movements to obtain all possible behaviors with respect to FSYNC (Fully Synchronous), SSYNC (Semi Synchronous) or ASYNC (Asynchronous) models. Hence, unlike robots that have the same behavior regardless of the model, the scheduler is parameterized by the execution model and the number of robots. Given a fixed number of robots and a variant of the execution model, the associated scheduler can be described by a finite automaton.

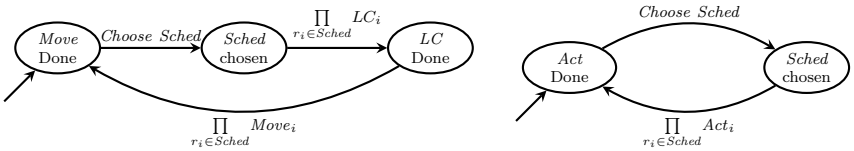
We denote by LC_i (respectively $Move_i$), the *LC* (resp. *Move*) phase of robot r_i. For a subset $Sched \subseteq Rob$, we denote the synchronization of all LC_i (respectively $Move_i$) actions of all robots in *Sched* by $\prod_{r_i \in Sched} LC_i$ (respectively $\prod_{r_i \in Sched} Move_i$).

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 model is also a cycle, where a set $Sched \subseteq Rob$ is first chosen. Then the *LC* and *Move* phases are synchronized for this set of robots. This generic automaton for SSYNC is described in Figure 4(a).

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

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 generic automaton for this scheduler is depicted in Figure 4(b).

To obtain more concrete versions of the scheduler models, we actually use a classical extension of automata consisting in labeling a transition by a sequence of actions instead of a single one [43].



(a) SSYNC model. (b) ASYNC model.

Fig. 4: The generic scheduler automata.

The automata represented in Figure 4 are compacted into a one state automaton with $2^k - 1$ loops, each one containing the choice of a non empty subset *Sched* of *Rob*, followed by the associated sequence of operations. The corresponding SSYNC scheduler automaton is depicted in Figure 5.

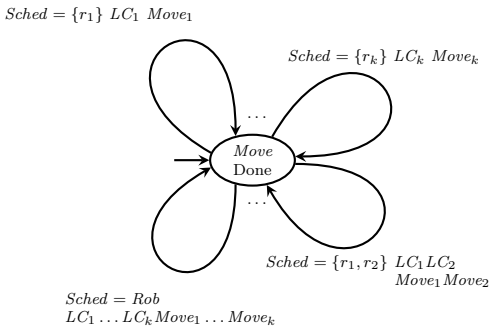


Fig. 5: The concrete version of the SSYNC scheduler automaton for k robots.

3.3 System Modeling

A global state is a tuple $s = (s_1, \dots, s_k, c)$ where s_i is the local state of robot r_i, and c the configuration. We denote by S the set of global states. The alphabet of actions is $A = \prod_{r_i \in Rob} A_i$, with $A_i = LC_i \cup Move_i$ for each robot r_i.

A transition of the system is labeled by a tuple $a = (a_1, \dots, a_k)$, where $a_i \in A_i \cup \{\varepsilon, -\}$ for all $1 \leq i \leq k$ and $(s_1, \dots, s_k, c) \xrightarrow{a} (s'_1, \dots, s'_k, c')$ if and only if for all i, $s_i \xrightarrow{a_i} s'_i$ and c' is obtained from c by updating the positions of all robots for which $a_i \in Move_i$. To represent the scheduling, we denote by $\prod_{r_i \in Sched} Act_i$ the scheduler action which will be synchronized with $(a_1, \dots, a_k)$ such that $a_i = -$ if $r_i \notin Sched$ and $a_i \in LC_i \cup Move_i \cup \{\varepsilon\}$ otherwise. We denote by T the set of transitions.

The model of the system is a family of automata $(M_{c_0})_{c_0 \in \mathcal{C}}$, where $M_{c_0} = (S, s_{c_0}, A, T)$ has initial state $s_{c_0} = (s_{1,0} \dots, s_{k,0}, c_0)$ where $s_{i,0}$ is the initial local state of robot r_i, and $c_0 \in \mathcal{C}$ is a fixed configuration.

Note that M_{c_0} is obtained by a synchronized product according to the definition of Section 2 as follows. We consider a configuration automaton $M_{conf} = (\mathcal{C}, c_0, A, T_{conf})$ with transitions $c \xrightarrow{a} c'$ labeled by $a \in A$ describing configuration updates. Then M_{c_0} is the synchronized product of the k robot automata, the scheduler automaton and M_{conf}. Since there is a single state for the scheduler component, it can be omitted in the product.

Among the protocols designed for discrete settings we choose as case studies two protocols where robots are positioned on a ring. Our model needs to be refined to take into account this particular shape.

4 Methodology for the ring

In this section, we give more details for the specific case of ring algorithms. In particular, the nodes in the set $Pos = \{0, \dots, n-1\}$ are numbered in the clockwise direction and we implicitly use arithmetic modulo n on this set.

4.1 Robot views

The robots take a snapshot of their environment to compute their future movement. Since this snapshot represents the graph, it depends on the ring

topology. This is formally described by the notion of view:

Definition 4 (View) In a given configuration of the system, we denote by d_j the number of robots at node j, called the *multiplicity* of node j. A *tower* at node j thus corresponds to $d_j > 1$: (strictly) more than one robot on this node. We note:
$\delta^{+j} = \langle d_j d_{j+1} \dots d_{j+n-1} \rangle$ and
$\delta^{-j} = \langle d_j d_{j-1} \dots d_{j-(n-1)} \rangle$.
The *view* at node j is the set $\delta^j = \{\delta^{+j}, \delta^{-j}\}$ describing the configuration of the system viewed from this node.

In a configuration $c : Rob \to Pos$, the view of robot r is thus $\delta^{c(r)}$. Ordering the tuples by lexicographical order, we define the maximal and minimal observation of robot r respectively by:
$\delta^{\max\_r} = \max\{\delta^{+c(r)}, \delta^{-c(r)}\}$,
$\delta^{\min\_r} = \min\{\delta^{+c(r)}, \delta^{-c(r)}\}$.

This definition of view as a set (with no order between the elements) takes into account the absence of chirality of robots. In particular, when $\delta^{\max\_r} = \delta^{\min\_r}$, the view contains a single element: in this case, robot r cannot distinguish between the two directions on the ring: this corresponds to a *disoriented* robot. Also note that two robots on the same node (*i.e.*, on a tower) have the same view.

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

Example 2 Consider for instance the ring configuration c depicted in Figure 6 where white nodes are free nodes and black nodes are the occupied ones. Writing $\delta^{c(r)} = (R_1, F_1, T_2, R_2, F_3, R_1, F_1)$, indicates that this set contains:
$\delta^{\max\_r} = \langle 1,0,2,1,1,0,0,0,1,0 \rangle$
and $\delta^{\min\_r} = \langle 1,0,1,0,0,0,1,1,2,0 \rangle$.

Since the configurations give absolute positions to the robots and robot protocols only depend on the robot views, we want to gather all configurations for which the robot views are the same. For this, we consider the permutations of the set *Pos* in a ring of size n. We denote by $\circ$ the composition of applications, with *id* the identity. Recall that for any permutation π of *Pos*, $\pi^0 = id$, and $\pi^{m+1} = \pi \circ \pi^m$, for any $m \in \mathbb{N}$. Only specific permutations will produce the same views:

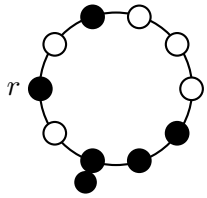


Fig. 6: Robot r of Example 1.

- If a configuration c' is obtained from c by a rotation of the absolute positions on the ring, the views will be the same. These operations are obtained by iterating the permutation σ defined by $\sigma(i) = i+1$ for all $i \in Pos$, which corresponds to a one step shift in the clockwise direction. Then $\sigma^n = id$, and we denote by σ^{-m} the inverse σ^{n-m} of σ^m.
- Two configurations that are symmetric with respect to the diameter of the ring containing node m also produce the same views. For node 0, this corresponds to the permutation $\overline{\sigma}$ defined by $\overline{\sigma}(i) = n - i$ for $i \in Pos$, with $\overline{\sigma}^2 = id$, and $\overline{\sigma}^{-1} = \overline{\sigma}$. For node m, the associated permutation is $\sigma^m \circ \overline{\sigma} \circ \sigma^{-m}$.
- Finally, when n and k are even, two configurations that are symmetric with respect to the diameter of the ring containing edge $m-(m+1)$ also produce the same views. This corresponds to the permutation $\sigma^{m+1} \circ \overline{\sigma} \circ \sigma^{-m}$.

Form these observations and the fact that $\overline{\sigma} \circ \sigma^m = \sigma^{-m} \circ \overline{\sigma}$ for any m, the set of permutations on positions producing the same views is in fact the group generated by σ and $\overline{\sigma}$.

In particular, all permutations containing $\overline{\sigma}$ correspond to symmetries and reflect the absence of chirality of robots. Moreover, to take into account robot anonymity, we introduce permutations of the set *Rob*. These considerations lead to the following definition:

Definition 5 (Equivalence and symmetry)

– The binary relation $\sim$ on the set Pos^{Rob} of configurations is defined by:
  $c \sim c'$ if there exist an integer m and some permutation β of *Rob* such that $c' = \sigma^m \circ c \circ \beta$ or $c' = \overline{\sigma} \circ \sigma^m \circ c \circ \beta$.
– Configurations c and c' are *symmetric*, written c *sym* c', if there are some m and β such that $c' = \sigma^m \circ \overline{\sigma} \circ \sigma^{-m} \circ c \circ \beta$ or $c' = \sigma^{m+1} \circ \overline{\sigma} \circ \sigma^{-m} \circ c \circ \beta$ (when n and k are even).
– Configuration c is *symmetric* if c *sym* c.

The relation $\sim$ is an equivalence relation and an *F-R-T* sequence is a representent of an equivalence class. A class is symmetrical if it contains a symmetric configuration. In the sequel, when describing a configuration (class), we simply give an *F-R-T* sequence.

4.2 Robot movements

The possible movements along edges also depend on the graph shape: on a ring there are only three possibilities, to stay idle, to move in the clockwise direction or in the anti-clockwise direction. The state "Ready to Move" (from Figure 3) is then divided into three states *r.Front*, *r.Back* and *r.Idle*. When a robot r is in state *r.Front*, it means that it will shift to its neighboring node in the direction given by $\delta^{\max\_r}$. Symmetrically, the robot in state *r.Back* will go in the opposite direction.

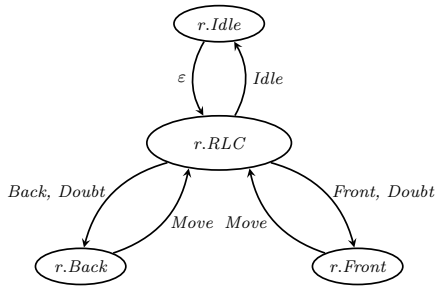


Fig. 7: Generic automaton of robot r on a ring.

These moves, determined by the algorithm, are described by the *LC* actions, which can be *Front*, *Back*, *Doubt* or *Idle*. Hence the alphabet of each robot r_i is $A_i = \{Front, Back, Doubt, Idle, Move\}$.

For a given robot r, the choice of an action depends on its own view of the configuration. If the robot chooses not to move, its action is *Idle*. If $\delta^{\max\_r} \neq \delta^{\min\_r}$, the robot can choose between the two directions, producing actions *Front* or *Back*. Otherwise the action is *Doubt*, corresponding to a non-deterministic choice between *Front* and *Back*. As explained above, the *Move* transitions actually update the configuration. A generic robot automaton for the case of the ring is depicted in Figure 7 where the state "Ready to Look Compute" is written as *r.RLC*.

4.3 Verification methodology

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

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

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

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

5 Implementation

5.1 Guarded actions formalism

Robot protocols (in FSYNC, SSYNC, or ASYNC models) are usually described by a set of rules that give a move to a robot, according to its view. We illustrate the successive translations with a toy protocol for 3 robots on a 10-nodes ring, described by the following rules for any robot r:

– if the robot view $\delta^{c(r)}$ is a singleton, *i.e.*, $\delta^{\max\_r} = \delta^{\min\_r}$, then the robot r moves in any direction,
– otherwise, if there are at least 4 free nodes adjacent to r, then it moves toward these free nodes,

– otherwise r stays idle.

Since most verification tools (like SPIN [40] or UPPAAL [44] used in [45]) use guarded actions of the form *[predicate]* $\rightarrow$ *action*, we first apply a pre-processing phase to express the rules of the algorithm in this formalism. The predicate is evaluated on the robot view and associated with an *LC* action. In our example we have:

1. $[G_{Sym}] \rightarrow$ *r.Doubt*,
   where $G_{Sym} := \delta^{\max\_r} = \delta^{\min\_r}$.
2. $[G_{Back}] \rightarrow$ *r.Back*,
   where $G_{Back} := \delta^{c(r)} \in \{(R_1,F_x,R_1,F_y,R_1,F_z) \mid x \geq 4, z < 4\}$.
3. $[G_{Idle}] \rightarrow$ *r.Idle*,
   where $G_{Idle} := \left(\delta^{\max\_r} \neq \delta^{\min\_r}\right) \wedge$
   $\delta^{c(r)} \notin \{(R_1,F_x,R_1,F_y,R_1,F_z) \mid x < 4, z < 4\}$.

In the implementation, these rules are then translated into clockwise or anti-clockwise moves, according to the current configuration considered as a global variable. For each robot r, the associated automaton has as local variables the robot identity and its position $c(r)$. The expressions for guards become:

1a $[\bigwedge_{i\in\{1,\ldots 9\}} d_{c(r)+i} = d_{c(r)-i}] \rightarrow$ *CounterClockwise*

1b $[\bigwedge_{i\in\{1,\ldots 9\}} d_{c(r)+i} = d_{c(r)-i}] \rightarrow$ *Clockwise*

2a $[\bigwedge_{i\in\{1,\ldots 4\}} d_{c(r)-i} = 0 \wedge d_{c(r)-5} \neq 2] \rightarrow$ *CounterClockwise*

2b $[\bigwedge_{i\in\{1,\ldots 4\}} d_{c(r)+i} = 0 \wedge d_{c(r)+5} \neq 2] \rightarrow$ *Clockwise*

3 $\left[\begin{array}{l} \neg\left(\bigwedge_{i\in\{1,\ldots 9\}} d_{c(r)+i} = d_{c(r)-i}\right) \\ \wedge\neg\left(\bigwedge_{i\in\{1,\ldots 4\}} d_{c(r)-i} = 0 \wedge d_{c(r)-5} \neq 2\right) \\ \wedge\neg\left(\bigwedge_{i\in\{1,\ldots 4\}} d_{c(r)+i} = 0 \wedge d_{c(r)+5} \neq 2\right) \end{array}\right] \rightarrow$ *Idle*

We thus obtain the automaton of Figure 8 that represents a robot implementing the above protocol.

For our modeling purpose, we opt for the language DVE, supported by several verification tools: DiVinE [36], for which DVE is the original modeling language, but also ITS-tools [37] and LTSmin [41]. The first one is an explicit model checker (operating on the actual synchronized product), integrating partial order reduction techniques. The two others are symbolic model checkers, operating on reduced state spaces based on decision diagram representations. For our experimentations, we use both types: the explicit one DiVinE and ITS-tools developed in our LIP6 team.

A DVE system is composed of processes, that are automata equipped with guarded actions on transitions. When two transitions can be fired, one of them is chosen nondeterministically. Transitions have so-called *effects* that actually are assignments to local or global variables. In our case, the *LC* transitions update the robot states, while the *Move* transitions update the configuration.

The DVE file containing the system is composed of a table *conf* that represents the configuration initialized at c_0, a process for each robot and a process for the scheduler.

5.2 Implementation issues

To obtain a first basic reduction of the state space in the case of the ASYNC model, we implement synchronized actions using a reordering of the Act_i actions, where look/compute actions (LC_i) are executed first, and the move actions ($Move_i$) afterward. All intermediate states are considered transient.

More precisely, we define the following system: $M' = (S', s_0, A, T')$ where S' is defined similarly to S, with the addition of a labeling of states (explained below), to indicate if the state is a transient or a steady state. The transition relation is defined as follows: Any transition $s \xrightarrow{a} s'$ in M is replaced in M' by a sequence of transitions, where all intermediate states are labeled as transient, while s and s' are steady states. More precisely, we note $\hat{a}_i = (-,\ldots,-,a_i,-,\ldots,-)$ the tuple of actions where only the robot r_i executes $a_i \in A_i$. An action $a = \prod_{r_i \in Sched} Act_i$ is executed as the sequence of actions $\hat{\ell}_1,\ldots,\hat{\ell}_k,\hat{m}_1,\ldots,\hat{m}_k$ where $\ell_i \in LC_i$ if $Act_i \in LC_i$ and $-$ otherwise, and similarly, $m_i \in Move_i$ if $Act_i \in Move_i$ and $-$ otherwise. Note that for each i, $\hat{\ell}_i$ and $\hat{m}_i$ are either $(-,\ldots,-)$ (containing only $-$, which corresponds to no action from any robot), or belong to $\{\hat{Act}_i \mid r_i \in Sched\}$.

Let $Exec(M)$ and $Exec(M')$ be respectively the set of executions of M and M'. We denote by $cf(e)$ the sequence of configurations in $e \in Exec(M)$ and by $cfs(e')$ the sequence of configurations of the

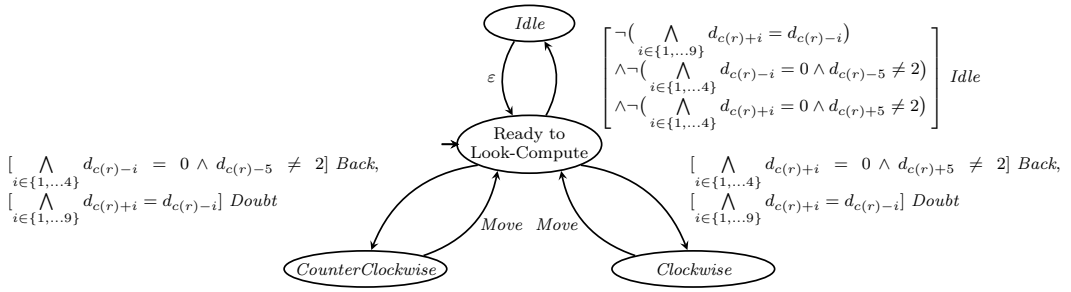


Fig. 8: Automaton of robot r implementing the toy protocol

steady states in $e' \in Exec(M')$. This notation is extended to the set of executions of M and M' by:

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

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

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

Definition 6 The models M and M' are equivalent if $cf(Exec(M)) = cfs(Exec(M'))$.

The following theorem states that our DiVinE implementation is equivalent to the original ASYNC model.

Theorem 1 *The models M and M′ are equivalent.*

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

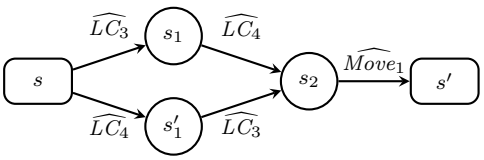
Let $e \in Exec(M)$. With any transition $t : s \xrightarrow{a} s'$ in e, with $a = (a_1, \ldots, a_k)$, we associate the execution e_t in M' defined above by:

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

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

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

Example 3 Consider for instance a system composed of 4 robots: $Rob = \{r_1, r_2, r_3, r_4\}$, and the transition between global states s and s' labeled by the tuple $a = (Move_1, -, LC_3, LC_4)$, with $Sched = \{r_1, r_3, r_4\}$. The schema below depicts all the executions equivalent to this transition,



where the actions are: $\widehat{Move_1} = (Move_1, -, -, -)$, $\widehat{LC_3} = (-, -, LC_3, -)$, and $\widehat{LC_4} = (-, -, -, LC_4)$. As depicted in this figure, the LC actions commute. Hence any execution can represent the above transition, including $s \xrightarrow{\widehat{LC_3}} s_1 \xrightarrow{\widehat{LC_4}} s_2 \xrightarrow{\widehat{Move_1}} s'$.

Among the protocols designed for discrete settings we choose as case studies the ring exploration with stop and perpetual exclusive ring exploration, with [3] and [4] respectively, as representative protocols of these two classes. The next sections are devoted to these case studies. In each case, starting from the informal description of the algorithm, we give the associated formal model and LTL formulas for the properties that these algorithms have to satisfy. We end by the verification results.

6 Ring Exploration with Stop

Flocchini *et al.* first defined in [3] the problem of exploration with stop on a ring of size n and proved that this problem cannot be solved by a deterministic algorithm when the number k of robots divides n. The authors also proposed a deterministic protocol to solve exploration with stop when $k \geq 17$, provided that n and k are co-prime.

The original paper only contains an informal description of the algorithm, thus our first contribution is to formally express the algorithm in order to remove ambiguities. Then we translate this protocol in the DiVinE language and automatically verify it on several instances.

6.1 Specification

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

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

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

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

These two properties can be expressed in LTL as follows:

- *Exploration*: $\bigwedge_{j=0}^{n-1} \Diamond(d_j > 0)$.
- *Termination*: $\bigwedge_{i=1}^{k} \Diamond\Box(\neg r_i.Front \wedge \neg r_i.Back)$.

Definition 7 A protocol solves the problem of ring exploration with stop if from any initial configuration, the following formula holds:

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

6.2 Algorithm Description

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

In order to express the algorithm in a guarded action language, some definitions and notations are introduced, related to a given configuration, and examples from the rings of Figure 9 are given along these definitions.

- The *interdistance* d is the minimum distance between all pairs of distinct robots in the configuration, where distance is counted in number of edges. Hence, an interdistance $d = 0$

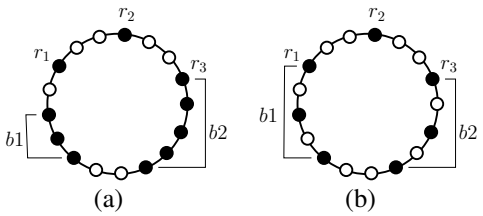


Fig. 9: Illustration of the definitions.

corresponds to the presence of (at least) two robots on a same node.

For the configuration (a) of Figure 9, the interdistance is 1, while it is 2 for (b).

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

  In configuration (a), there are two blocks of size 3 and 5, while there are two blocks of size 3 in (b).
- *Between*(b_1,b_2) is a pair of natural numbers counting the number of blocks between two blocks b_1 and b_2 (one integer for each direction). In both (a) and (b), *Between*$(b1,b2)$ is equal to $(0,0)$.

We also define the following predicates:

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

In (a), Isolated$(r_1) = true$, Isolated$(r_2) = true$ and in (b), Isolated$(r_2) = true$

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

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

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

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

Leading(x): this predicate is true if x is a leading block or a leading robot. A robot is leading if its view is maximal (among the different $\delta^{\max\_r}$). Such a robot is called a *leader*. A block b is *leading* if it has a border robot which is a leader.

In Figure 9 (a), only Leading(r_3) is true, hence also Leading$(b2)$. In Figure 9 (b), Leading(r_1) and Leading(r_3) are both true.

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

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

6.2.1 The Set-Up Phase

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

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

There are four types of configurations, namely A, B, C, D, that form a partition of all possible tower-free configurations. Configurations of type A contain isolated robots and configurations of type B, C or D contain only blocks of robots. Configurations of type D are the Set-Up final configurations, configurations of type C are similar (there are no isolated robots, and each block is a leading block) to these configurations but with an interdis-

tance $d \geq 2$. Configurations of type B are all the remaining configurations without isolated robot.

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

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

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

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

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

We also need the following predicates and sets:

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

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

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

The guarded action in type A for a robot r is thus:
$[\text{Type}A \wedge (r,b) \in \text{Closest}] \rightarrow \text{Move}(r,b)$.

Example 4 This action is illustrated in Figure 10, where the isolated robot r is at distance 9 of b_1 (which is of maximal size) and 3 of b_2. Since the two views of r are different and

$$\mathit{dist}(r,b_1) > n - \mathit{dist}(r,b_1) - (b_1.\mathit{size}-1) \times d$$

holds with $n = 21$ and $d = 2$, its move must be *r.Back*, in the direction opposite to its maximal view, as indicated by the arrow on Figure 10.

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

$$\text{Type}CD := \text{Set-Up} \wedge \neg\text{Type}A \wedge \forall b \in \mathit{Blocks} : \text{Leading}(b)$$

$$\text{Type}C := d \geq 2 \wedge \text{Type}CD$$

$$\text{Type}D := d = 1 \wedge \text{Type}CD$$

It can be seen that all C and D configurations are symmetrical. In a configuration of type C, all blocks are leading, with a maximal view for all leader robots, and no isolated robot. Hence, there are two leaders in each block. The aim of the protocol here is to reduce the interdistance. Hence the two leaders of a block will move inside their block, so that from a C configuration with interdistance d, a configuration of type A with interdistance $d-1$ will be reached. The protocol executed by robot r in this case is:
$[\text{Type}C \wedge \text{Leading}(r)] \rightarrow r.\mathit{Front}$,
meaning that the robot moves in the direction of its maximal view.

From a configuration of type D (Set-Up final) the *Tower-Creation* phase begins. Then, denoting by Tower-Creation the predicate satisfied in the *Tower-Creation* phase, we set:
$[\text{Type}D] \rightarrow$ Tower-Creation.

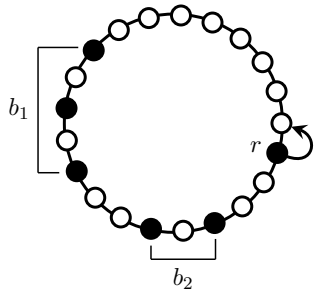


Fig. 10: Movement in a type A configuration.

Configurations of type B. When the current configuration is neither a type A configuration nor a type C or D configuration, then it is a type B configuration, which is by far the most complicated part of the algorithm:
$\text{Type}B := \mathit{Set\text{-}Up} \wedge \neg(\text{Type}A \vee \text{Type}CD)$.
Configurations of type B are divided into the two types $B1$ and $B2$: if all blocks have the same size

then the configuration is of type $B1$, otherwise it is of type $B2$.

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

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

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

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

$\text{symRobots}(r, r') := \delta^{\max\_r} = \delta^{\max\_r'} \wedge r \neq r'$

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

and the sets:

$\text{L} = \{r \in \mathit{Rob} \mid \text{Leading}(r)\}$

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

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

For subsets *prob* of Rob^2, and *Bs* of *Blocks*:

$\text{NearPair}(\mathit{prob}) = \{(r_1, r_2) \in \mathit{prob} \mid \mathit{dist}(r_1, r_2)$
$= \min\{\mathit{dist}(r, r'), (r, r') \in \mathit{prob}\}$
$\wedge \neg \text{Neighbor}(r_1, r_2)\}$

$\text{minView}(\mathit{Bs}) = \{r \in \mathit{Rob} \mid \exists b \in \mathit{Bs}, \exists r' \in \mathit{Rob}$
$\text{Border}(r, b) \wedge \text{Border}(r', b) \wedge$
$\delta^{\min\_r} < \delta^{\min\_r'}\}$

The guarded actions in type $B1$ for a robot r are:

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

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

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

$\text{m} = \min\{b.\mathit{size} \mid b \in \mathit{Blocks}\}$

$\text{M} = \max\{b_1.\mathit{size} \mid b_1 \in \mathit{Blocks}$ s.t. $\exists b_2 \in \mathit{Blocks}$:
$\text{Neighbor}(b_1, b_2) \wedge b_2.\mathit{size} = \text{m}\}$

$\text{dmin} = \min\{\mathit{dist}(b_1, b_2) \mid (b_1, b_2) \in \mathit{Blocks}^2$ s.t.
$\text{Neighbor}(b_1, b_2) \wedge b_2.\mathit{size} = \text{m} \wedge$
$b_1.\mathit{size} = \text{M}\}$

For a subset *rob* of *Rob*:

$\text{MaxV}(\mathit{rob}) = \max\{\delta^{\max\_r} \mid r \in \mathit{rob}\}$

$T = \{r \in \mathit{Rob} \mid \exists (b_1, b_2) \in \mathit{Blocks}^2 : \text{Border}(r, b_1)$
$\wedge\ b_1.\mathit{size} = \text{m} \wedge \text{Neighbor}(r, b_2) \wedge b_2.\mathit{size} = \text{M}$
$\wedge\ \mathit{dist}(b_1, b_2) = \text{dmin}\}$

The guarded action for a robot r is:

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

The Set-Up phase movements are summed up in Table 2, where scheduled robots stay idle in all cases not covered in this table. The predicates defined above to describe robots views would give rise to numerous F-R-T sequences, according to the number of blocks, between 1 and $\lfloor n/2 \rfloor$.

In the next two phases, this number of blocks is in $\{1, 2, 3\}$, and it will be possible to give the rule presentation with F-R-T sequences.

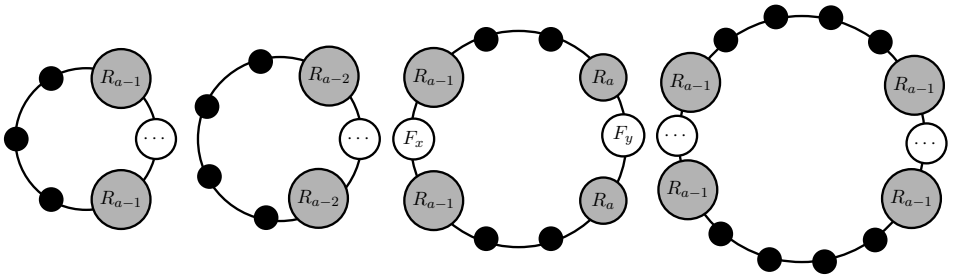
6.2.2 The Tower-Creation Phase

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

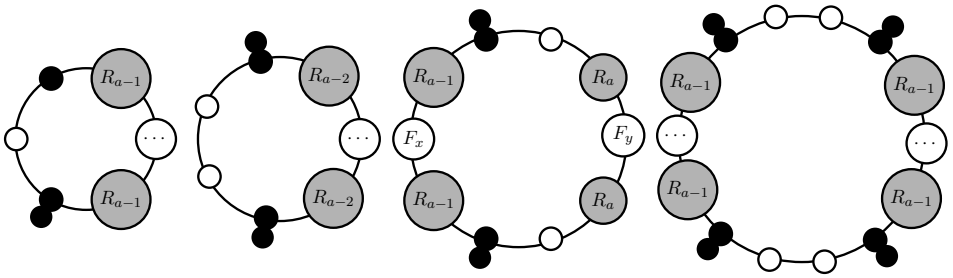
Informally, for each odd block one tower is formed by the central robot moving to its neighboring node containing the robot with the larger view. For each even block two towers are formed

Tower-Creation Phase:			
TSA_1::	Type$A \wedge (r,b) \in$ Closest $\wedge\ \delta^{\max\_r} = \delta^{\min\_r}$	$\rightarrow$	*r.Doubt*
TSA_2::	Type$A \wedge (r,b) \in$ Closest $\wedge\ \delta^{\max\_r} \neq \delta^{\min\_r} \wedge dist(r,b) > n - dist(r,b) - (b.size-1) \times d$	$\rightarrow$	*r.Back*
TSA_3::	Type$A \wedge (r,b) \in$ Closest $\wedge\ \delta^{\max\_r} \neq \delta^{\min\_r} \wedge dist(r,b) \leq n - dist(r,b) - (b.size-1) \times d$	$\rightarrow$	*r.Front*
$TSB1_1$::	Type$B1 \wedge L = \{r\}$	$\rightarrow$	*r.Back*
$TSB1_2$::	Type$B1 \wedge \lvert L\rvert = 2 \wedge \exists b \in Blocks : b.size = 2 \wedge$ Border$(r,b) \wedge r \in$ minView(SymB)	$\rightarrow$	*r.Back*
$TSB1_3$::	Type$B1 \wedge \lvert L\rvert = 2 \wedge \exists b \in Blocks : b.size \neq 2 \wedge$ Border$(r,b) \wedge r \in$ NearPair(SymR)	$\rightarrow$	*r.Back*
$TSB2$::	Type$B2 \wedge r \in T \wedge \delta^{\max_r} =$ MaxV$(T) \wedge \exists b \in Blocks :$ (Neighbor$(r,b) \wedge b.size =$ M$\wedge dist(r,b) =$ dmin)	$\rightarrow$	*r.Back*
TSC::	Type$C \wedge$ Leading(r)	$\rightarrow$	*r.Front*

Table 2: Rules of the Tower-Creation Phase for a robot r.



(a) Set-Up final configurations.



(b) Tower-Completed configurations.

Fig. 11: Tower-Creation phase from Set-Up final configurations.

by the two central robots moving to their other neighbors.

These moves are described in Table 3. In this table, as well as in Table 4, the view is given by an *F-R-T* sequence as described in Definition 4 and brackets for guards are omitted. Figure 11 illustrates the process of tower creation from the possible Set-Up final configurations. Each configuration in Figure 11(a) will produce the one just below in Figure 11(b). Big circles contain a set of adjacent nodes which are all free or all occupied. A big gray node R_x represents x adjacent occupied nodes, a big white node F_x represents x adjacent free nodes, and a white node containing dots represents a positive number of free nodes.

There are four cases:

- If there is only one block of odd size then the Set-Up final configuration looks like the first one of Figure 11(a). A unique robot can move according to rule $TC1_0$, which produces the configuration just below in Figure 11(b) (or the symmetrical one with the tower in the upper half instead of the lower half).
- If there is only one block of even size (second column), then two robots will move according to rule $TC2_0$. If one has moved before the other

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

Table 3: Rules of the Tower-Creation Phase for a robot r.

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

Table 4: Rules of the Exploration Phase for a robot r.

one could take a snapshot of the configuration then its movement is given by rule $TC2_1$.

- If there are two symmetrical blocks of odd size (third case), then two robots will move according to rule $TC3_0$. If one has moved before the other one could take a snapshot of the configuration then its movement is obtained by rule $TC3_1$. In this case, the two free segments F_x and F_y have different sizes, and the robots move in the direction opposite to the shortest one.
- If there are two blocks of even size (rightmost column) then robots move according to rule $TC4_0$. In this case also, the two free segments have different sizes. If one tower is formed, the other robot movement is given by rules $TC4_{11}$, $TC4_{12}$ and $TC4_{13}$ according to the robot view. If two of them have moved, and two towers are formed, then the moving robots compute their movements according to one of the rules in $\{TC4_{21}, TC4_{22}, TC4_{23}\}$, depending of which robots have moved before. And when all robots but one have moved, the last one moves according to rule $TC4_3$.

6.2.3 The Exploration Phase

The exploration phase is the last phase of the algorithm. It starts from tower-completed configurations and is described in Table 4 (where again robots stay idle for non covered cases). No new towers are created during this phase.

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

When k is odd, the configuration starting the exploration phase is made of two blocks, one of them containing a tower (leftmost configuration of Figure 11(b)). The explorer is the robot at the border of the block with the tower, the tower being the other border of the block. The destination of this

robot is the neighbor free node toward the block that does not contain the tower. The algorithm for the moving robot is given by rule E_1.

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

6.3 Verification Results

Experiments on this algorithm were performed with the model-checker ITS-tools, hence using a symbolic state space. As usual with this type of tools, heuristics with various variables orders have been tested, in order to optimize the exploration.

The results show that the algorithm satisfies the exploration and termination properties under fairness hypothesis (see Section 6.1). Hence the algorithm is correct for all tested instances of k and n that satisfy the constraints given in the original paper: n, k are co-prime and $n, k \geq 17$.

k	n	Time	Mem (MB)
17	18	00:00:04	61
18	19	00:00:04	66
17	19	00:25:29	1 622
19	20	00:00:08	88
18	20	00:12:10	2 131
17	20	08:08:00	22 045
20	21	00:00:08	100
19	21	01:08:12	3 632
18	21	03:00:52	9 428
17	21	18:40:07	55 287
21	22	00:00:12	124
20	22	01:58:27	5 914
19	22	08:25:22	30 243
18	22	20:32:45	100 328

Table 5: Set-Up phase model-checking.

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

Table 6: Model-checking small instances.

Since the most complex phase of the algorithm is the Set-Up phase, we present in Table 5 the verification results (time and memory) for the restriction to this particular phase, model-checking the property: every run reaches a configuration satisfying the Type$D$ predicate (corresponding to a Set-Up final configuration). The state space explosion occurring during the model-checking can be seen on these results.

We also tested the entire algorithm for some small instances not covered by the original setting, for initial configurations that are not periodic (where not periodic means that there is at most one symmetry axis in the ring). Hence, our methodology permits to refine the correctness bounds for these cases. The performances can be seen in Table 6, where the algorithm satisfies the correctness property for all values of n and k appearing in the table. In particular, we can notice that correctness holds for values ok k and n that are not co-prime ($k = 5$ and $n = 10, 15$), again for non periodic initial configurations.

From these experiments, we conjecture that the algorithm is correct for $n < 18$ in the following cases even when n and k are not co-prime, as long as the initial configuration is not periodic:

- When k is even the algorithm is correct as long as $n < k + \lceil k/2 \rceil$ and $10 \leq k < 17$.
- When k is odd the algorithm is correct if $5 \leq k < 17$.

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

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

7 Perpetual Ring Exploration

We now recall the problem of perpetual exclusive ring exploration, and present the verification results for the *Min-Algorithm* [4]. Note that the same arguments as in [3] apply to obtain impossibility when the number k of robots divides the size n of the ring. For this algorithm, model checking tools are used to exhibit a counter-example. After identifying the rule producing this counter-example, we correct the algorithm and establish the correctness of the new version by model checking small instances and providing an inductive proof.

7.1 Specification

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

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

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

These properties can be expressed in LTL as follows: the *Exclusivity* property is the conjunction of the *No_collision* and the *No_switch* properties below :

- *No_collision*: $\bigwedge_{j=0}^{n-1} \Box\Big(d_j < 2\Big)$
- *No_switch*: $\bigwedge_{j=0}^{n-1} \bigwedge_{i=1}^{k} \bigwedge_{h=1}^{k} \neg \Diamond\Big(c(r_i) = j \wedge c(r_h) = j+1 \wedge r_i.Front \wedge r_h.Back\Big)$

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

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

$$Live : \bigwedge_{j=0}^{n-1} \bigwedge_{i=1}^{k} \Box\Diamond\big(c(r_i) = j\big).$$

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

$$Liveness : Fairness \Rightarrow Live.$$

7.2 Algorithm Description and Definitions

The *Min-Algorithm* from [4] is designed to ensure that 3 robots always exclusively and perpetually explore any ring of size $n \geq 10$ where n is not a multiple of 3. It is based on a classification of the set of tower-free configurations. The *Min-Algorithm* operates in two phases: the *Convergence* phase and the *Legitimate* phase. In the *Convergence* phase system states converge towards so-called *Legitimate states*. In the *Legitimate* phase the system cycles between its legitimate states, performing the exploration.

In Definitions 8 and 9 below, each set of configurations is an equivalence class of configura-

tions, given by an *F-R-T* sequence, according to Definition 4.

Definition 8 *Legitimate configurations* are defined for $n \geq 10$ by $\mathbb{L}^n = L1^n \cup L2^n \cup L3^n$ where:
- $L1^n = (R_2, F_2, R_1, F_{n-5})$
- $L2^n = (R_1, F_1, R_1, F_{n-6}, R_1, F_2)$
- $L3^n = (R_1, F_3, R_2, F_{n-6})$

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

Definition 9 *Non legitimate configurations* are defined for $n \geq 10$, when n and 3 are co-prime, by $\mathbb{NL}^n = A^n \cup B^n \cup C^n \cup D^n \cup E^n$ as follows.

When one robot is at the same positive distance of the two others, then it is a B^n configuration:

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

Otherwise, we distinguish three sub cases depending on the number of adjacent robots.

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

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

  Note that the case $(x,z) = (1,2)$ corresponds to a $L2^n$ configuration. Hence C^n configurations only appear when $n \geq 11$.
- If only two robots are adjacent, if the minimal distance between these two robots and the third one is equal to 2 or 3, then it is a $L1^n$ or a $L3^n$ configuration. Otherwise:
  - If the minimal distance is equal to 1, then it is an E^n configuration:

$$E^n = (R_1, F_1, R_2, F_{n-4}).$$

  - Otherwise the distance is larger than 3 and then it is an A^n configuration:

$$A^n = \{(R_1, F_x, R_2, F_z) \mid 4 \leq x < z \wedge n = x + z + 3\}.$$

  Note that this type of configuration only exists when $n > 12$, since n and $k = 3$ must be co-prime.
- If the three robots are adjacent then the configuration is a D^n configuration:

$$D^n = (R_3, F_{n-3}).$$

From the disjunction of cases in Definitions 8 and 9 above, and observing that no two classes of configurations overlap, we have:

Proposition 1 *The sets of configurations: A^n, B^n, C^n, D^n, E^n, $L1^n$, $L2^n$, $L3^n$, form a partition of the set of all tower-free configurations of a ring of size $n \geq 10$, when n and $k = 3$ are co-prime.*

We now detail the two phases, described in Table 7. In the *Legitimate* phase the idea is to authorize, by exploiting the asymmetry of the network, a single robot to move at each step. Rule *RL*1 authorizes only the robot which is the farthest from the isolated robot to move. This robot goes to the only free neighboring node. Rule *RL*2 authorizes the robot which is the nearest to the other robots to move in order to minimize the distance between itself and its nearest neighbor. Rule *RL*3 authorizes the isolated robot to come closer to the other robots. After the execution of n rounds (each one of the three robots has moved n times), all robots have explored the ring once.

The *Convergence* phase brings non-legitimate configurations into legitimate ones. The main point of this algorithm is to break possible symmetries and to converge to a pattern that allows the execution of one of the *RL* rules. Rule *RC*1 (resp. *RC*2, *RC*3, *RC*4 and *RC*5) is only applied for configurations in A^n (resp. B^n, C^n, D^n, E^n). Rule *RC*1 is applied in order to reduce the distance between the isolated robot and the two other robots. Rule *RC*2 is applied when a robot is at equal distance from the two other robots. This robot will break the symmetry by a shift of one position in any direction. Rule *RC*3 is applied when robots are scattered on the ring at distances $x < z < y$. The robot authorized to move is the one that is adjacent to the free spaces of size x and z. This robot will move such that the free space x is reduced by 1. The

Legitimate Phase:					
*RL*1::			$\delta^{c(r)} = (R_2, F_2, R_1, F_{n-5})$	$\rightarrow$	*r.Back*
*RL*2::			$\delta^{c(r)} = (R_1, F_1, R_1, F_{n-6}, R_1, F_2)$	$\rightarrow$	*r.Front*
*RL*3::			$\delta^{c(r)} = (R_1, F_3, R_2, F_{n-6})$	$\rightarrow$	*r.Front*
Convergence Phase:					
*RC*1::	$4 \leq x < z$	$\wedge$	$\delta^{c(r)} = (R_1, F_x, R_2, F_z)$	$\rightarrow$	*r.Front*
*RC*2::	$x \neq y, x > 0$	$\wedge$	$\delta^{c(r)} = (R_1, F_x, R_1, F_y, R_1, F_x)$	$\rightarrow$	*r.Doubt*
*RC*3::	$0 < x < z < y \wedge (x,z) \neq (1,2)$	$\wedge$	$\delta^{c(r)} = (R_1, F_x, R_1, F_y, R_1, F_z)$	$\rightarrow$	*r.Front*
*RC*4::			$\delta^{c(r)} = (R_3, F_{n-3})$	$\rightarrow$	*r.Back*
*RC*5::			$\delta^{c(r)} = (R_1, F_1, R_2, F_{n-4})$	$\rightarrow$	*r.Back*

Table 7: Rules of *Min-Algorithm* [4] for a robot r with $n \geq 10$.

idea behind this movement is to create a block of robots. Rule *RC*4 captures the situation when the three robots are adjacent. In this case, due to the symmetry the two robots on the border can move. Rule *RC*5 is applied when the isolated robot is too close (at distance 1) to the block of robots. In this case it will move away from the block.

The specification for this algorithm is refined as follows:

(a) The *No_collision* and *No_switch* properties are satisfied.

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

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

7.3 Verification Results

Recall that the setting of the *Min-Algorithm* features 3 robots in a ring of size $n \geq 10$ where n is not a multiple of 3. Thus we first construct a model for this protocol and its properties in the model checker DiVinE [36]. Then we verify this algorithm for the smallest possible ring of size 10, for all models (FSYNC, SSYNC and ASYNC). These results are presented in Table 8, with number of states, transitions, memory used, and time spent.

More importantly, our results show that the algorithm does not satisfy the *Exclusivity* property in the ASYNC model. A counter-example is automatically generated, exhibiting a sequence of transitions leading to a collision (a tower), hence a violation of the *Exclusivity* property. It is presented

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

Table 8: Model checking of *Min-Algorithm* in the three models for the smallest ring

in details in Figure 12, with a sequence of configurations obtained by successive robot moves. In each configuration a computation is represented by an arrow, which is dotted when the computation is made from an outdated snapshot.

In the starting configuration after the *LC* phase of all robots, the gray one and the black one have decided to move according to the *RC*4 rule, and the light gray one to stay idle. The black robot moves, which produces the second configuration. Before the gray robot could move, the black one performs its *LC* phase and according to the *RC*5 rules, it chooses to roll away from the two other robots. The gray robot moves from the decision taken previously and the third configuration is reached. From this configuration the light gray robot performs its *LC* phase and chooses to move in any direction (as the configuration has a symmetry axe passing through this robot). The scheduler makes it move toward the gray one. From the fourth configuration thus obtained, the gray robot had to move according to rule *RL*1 after its *LC* phase. This movement permits to obtain the fifth configuration, from where the light gray robot chooses to move according to rule *RL*2. We obtain the sixth configuration thanks to the movement of the black robot (movement that it had chosen in the second config-

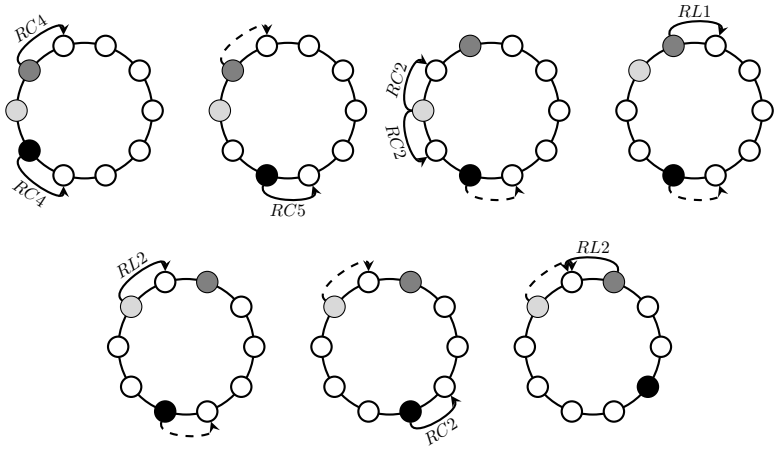


Fig. 12: Counter-example.

uration). From this configuration the black robot chooses to move according to rule *RC*4, and the move is performed. In the last configuration the gray robot performs its *LC* phase and according to the *RL*2 rule, it chooses to move toward the light gray one, on the same node where the light gray one had chosen to go in the fifth configuration. From there if these two robots move, they collide.

In this counter-example, we can see that the collision is due to the fact that there is always one movement that can be made from an outdated snapshot, hence we need to stop these movements. We now present a correction of the algorithm referred to as *Min-Algorithm-Corrected*. The modification concerns the convergence phase, leaving the legitimate phase unchanged. More precisely, only rule *RC*5 is modified to avoid collisions induced by the previous rules, when movements computed on obsolete observations are taken into account. The new *RC*5 rule is:

$$RC5 \;::\; \delta^{c(r)} = (R_2, F_1, R_1, F_{n-4}) \;\rightarrow\; r.Back$$

Note that the moving robot has changed with respect to the old rule. If this new rule is applied in the counter-example, then from the second configuration, no movements from outdated snapshots can be made any more since the *RC*5 rule requires a configuration where the light gray and the black robots have stayed idle.

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

Table 9: Model checking of the *Min-Algorithm-Corrected* for the ASYNC model.

Verification results given in Table 9 show that correctness is achieved in the ASYNC model for several instances of n. We can observe a limited blow up, due to the fact that when the number $k = 3$ of robots is fixed, the total number of configurations (and of states) is of order n^3.

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

7.4 Inductive approach

With respect to the refined specification in Section 7.2, we first prove point *(c)*: the exploration is performed by cycling within the legitimate config-

urations and point *(b)*: from all non-legitimate configurations a legitimate configuration is reached.

Given a configuration type TC (like A, B or $L1$), we express how it is parametrized by the distances (given as the numbers of free nodes) between consecutive robots. For this, we use a triple of natural numbers (x,y,z) such that the F-R-T sequence $(R_1,F_x,R_1,F_y,R_1,F_z)$ belongs to the type TC.

Notation 7.1 *The notation* $[TC^n(x,y,z),\ \varphi]$ *represents the set of configurations of type TC, with distances* x,y,z *between consecutive robots, restricted by the additional constraint* φ *on* x,y,z.

Recall that $n = x+y+z+3$ remains constant, with $n \geq 10$. Constraints defining the type itself are omitted. For instance,

$$[B^n(x,y,z) \mid x = z \wedge x \neq y \wedge x > 0 \wedge\ n = y+2x+3]$$

is simply denoted by $[B^n(x,y,x)]$.

Notation 7.2 *The tuple*

$$(s_x, s_y, s_z, [TC^n(x,y,z), \varphi])$$

denotes the set

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

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

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

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

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

An abstracted view of the algorithm is shown in Figures 13 and 14 using these notations. Gray states are initial states, more particularly the light grey ones are legitimate states. Each *Move* transition is guarded by a condition between brackets and corresponds to the choice of the scheduler to let all robots move. In the $Move_{any}$ transition, the scheduler lets only a single robot move. While this graph was merely intended to illustrate the proof, it was pointed out by an anonymous referee that it can also be viewed as a kind of visual abstraction, as proposed by Z. Manna and A. Pnueli in [26] and further studied in [28,29,30]. More precisely this graph, with three robot processes and variables x, y, z and n, is similar to a generalized verification diagram from [29], used by the STeP group, or to a predicate diagrams from [30].

7.4.1 Exploration from legitimate configurations

We prove the following theorem:

Theorem 2 *From any legitimate configuration the ring (of size* $n \geq 10$*, co-prime with* 3*) is perpetually explored.*

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

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

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

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

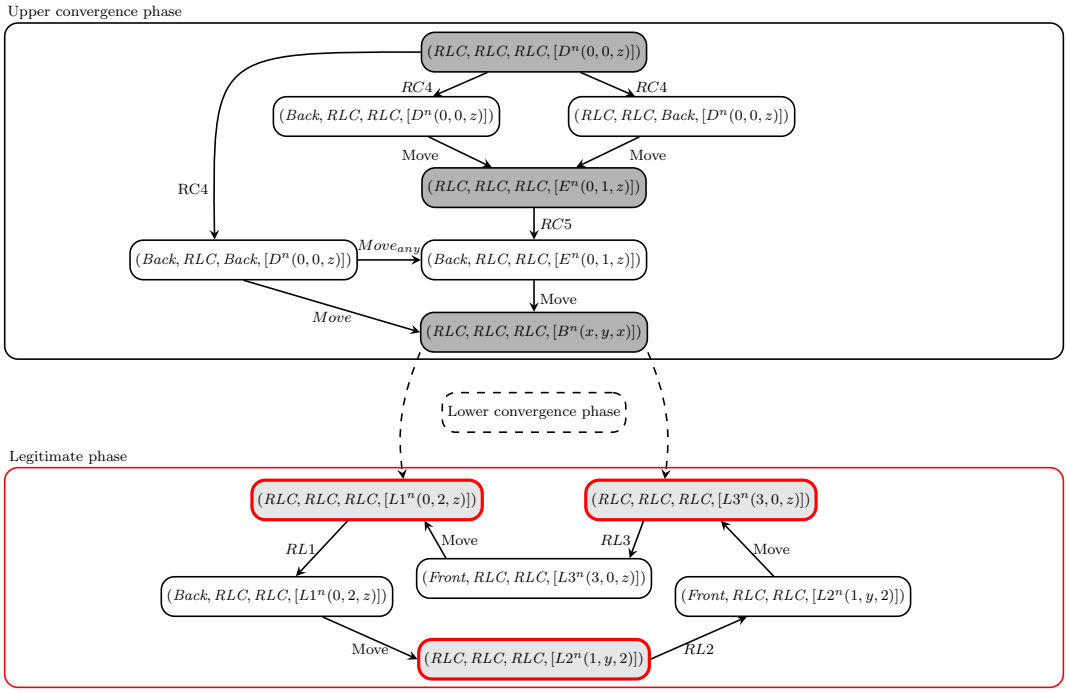


Fig. 13: Graph of Min-algorithm.

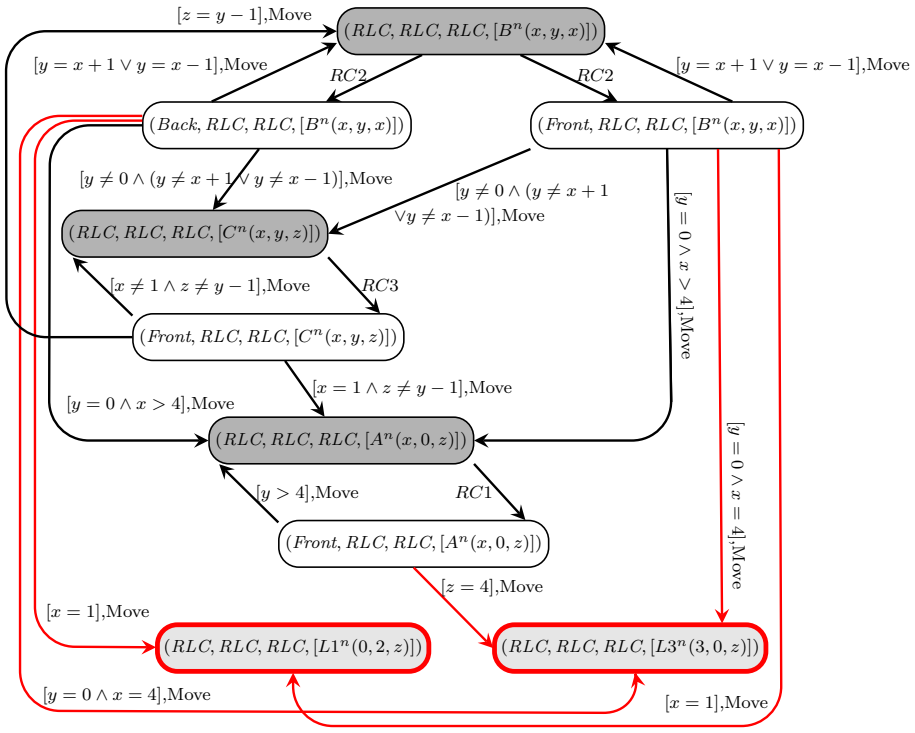


Fig. 14: Lower convergence phase.

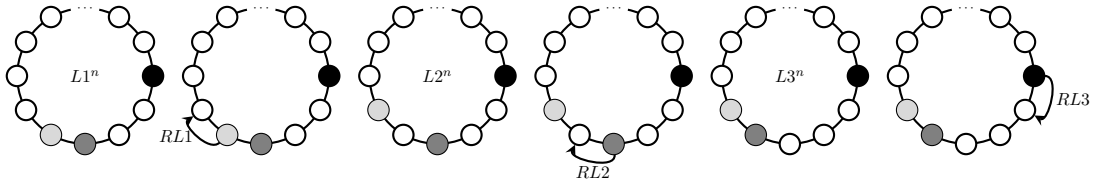


Fig. 15: A step for the exploration.

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

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

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

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

The base case for $p=0$ is trivial. For the induction step, assume that the property holds for p, hence: $(Lk \wedge r[j]=r_i) \implies \Diamond(Lk \wedge r[j+p]=r_i)$. Setting $j'=j+p$ and chaining the three moves described above, as illustrated in Figure 15, we obtain : $(Lk \wedge r[j']=r_i) \Rightarrow \Diamond(Lk \wedge r[j'+1]=r_i)$. Hence $(Lk \wedge r[j]=r_i) \Rightarrow \Diamond(Lk \wedge r[j+p+1]=r_i)$ and the property holds for $p+1$.

7.4.2 Convergence from illegitimate configurations

We prove:

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

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

Definition 10 Given the set $\mathcal{R}$ of rules of the *Min-Algorithm*, let P_0 be a subset of system states. The tree $\mathcal{T}(P_0)$ has P_0 as root and for each node P:

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

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

To prove this result, we exhaustively verify the property for all types A^n, B^n, C^n, D^n or E^n, by inductive proofs, in Lemmas 1 to 5 (where we assume $n \geq 10$ and n co-prime with 3). Note that these lemmas must be proved in the order A, C, B, E and D. Since $\mathbb{NL}^n = A^n \cup B^n \cup C^n \cup D^n \cup E^n$ the result follows.

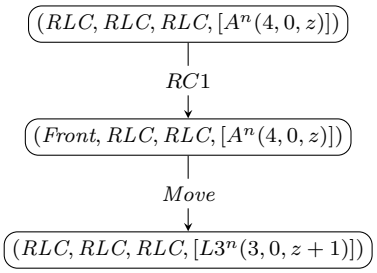
Lemma 1 *The tree $\mathcal{T}(P)$ is finite for* $P=(RLC,RLC,RLC,[A^n(x,0,z),4 \leq x < z])$.

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

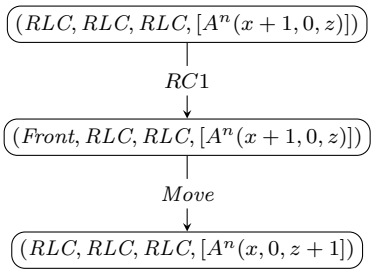
configuration otherwise. Hence the number of free nodes in the x part decreases until an $L3$ configuration is reached.

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

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



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



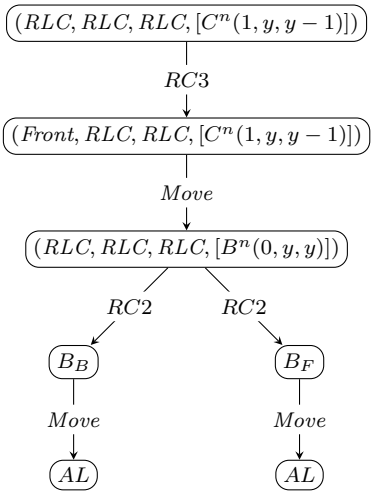
they lead to $(RLC, RLC, RLC, [A^n(x,0,z+1])$ for which the tree is finite from the induction hypothesis. This ensures the desired result.

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

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

Base-case: x=1

– If $z = y-1$, the tree is:



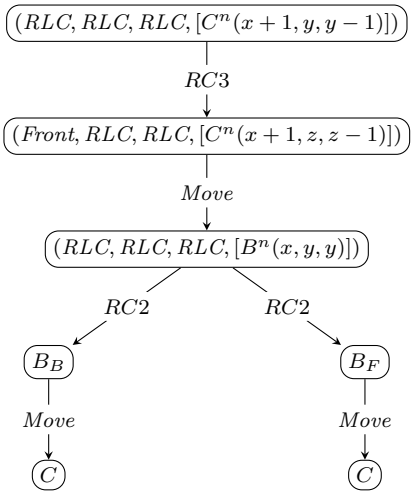
where:
$B_B = (RLC, RLC, Back, [B^n(0,y,y)])$
$B_F = (RLC, RLC, Front, [B^n(0,y,y)])$
and if $y = 4$,
$AL = (RLC, RLC, RLC, [L3^n(3,0,5)])$
otherwise ($y > 4$),
$AL = (RLC, RLC, RLC, [A^n(y-1,0,y+1)])$
Note that in both cases, the move from B_B or B_F leads to the same equivalence class of configurations: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

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

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

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

– If $z = y-1$, the tree is



where
$B_B = (RLC, RLC, Back, [B^n(x,y,y)])$
$B_F = (RLC, RLC, Front, [B^n(x,y,y)])$
$C = (RLC, RLC, RLC, [C^n(x,y+1,y-1)])$
Thanks to the induction hypothesis, we can conclude that the property holds in this case.

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

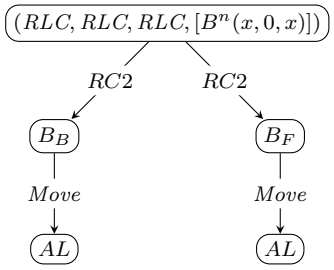
Finally all trees $\mathcal{T}(RLC, RLC, RLC, [C^n(x,y,z)])$ with $0 < x < z < y$ and $(x,z) \neq (1,2)$ are finite.

Lemma 3 *The tree $\mathcal{T}(P)$ is finite for* $P = (RLC, RLC, RLC, [B^n(x,y,x), x > 0 \wedge x \neq y])$.

Proof We handle two cases: $x > y$ and $x < y$.

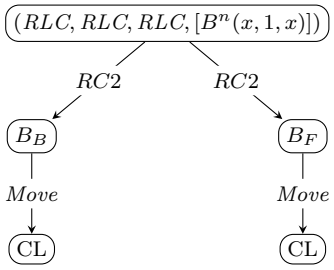
Case $x > y$: We first handle the subcases $y = 0$ and $y = 1$.

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



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

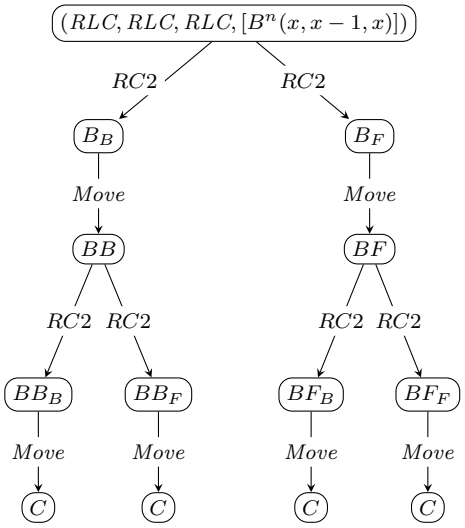
– When $y = 1$, the tree representing the algorithm is the following:



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

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

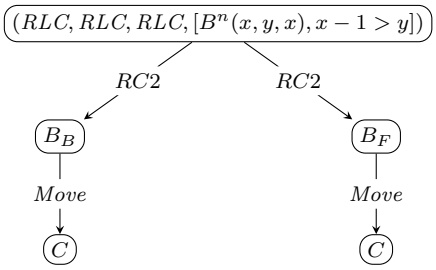
– If $x - 1 = y$, the moves are:



where
$B_B = (Back, RLC, RLC, [B^n(x, x-1, x)])$
$B_F = (Front, RLC, RLC, [B^n(x, x-1, x)])$
$BB = (RLC, RLC, RLC, [B^n(x+1, x-1, x-1)])$
$BF = (RLC, RLC, RLC, [B^n(x-1, x-1, x+1)])$
$BB_B = (RLC, RLC, Back, [B^n(x+1, x-1, x-1)])$
$BB_F = (RLC, RLC, Front, [B^n(x+1, x-1, x-1)])$
$BF_B = (RLC, Back, RLC, [B^n(x-1, x-1, x+1)])$
$BF_F = (RLC, Front, RLC, [B^n(x-1, x-1, x+1)])$
$C = (RLC, RLC, RLC, [C^n(x-2, x+1, x)])$

The property holds in this case, thanks to Lemma 2.

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

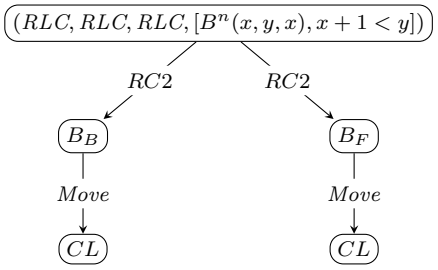


where
$B_B = (Back, RLC, RLC, [B^n(x, y, x)])$
$B_F = (Front, RLC, RLC, [B^n(x, y, x)])$
$C = (RLC, RLC, RLC, [C^n(x+1, y, x-1)])$
and Lemma 2 entails the result.

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

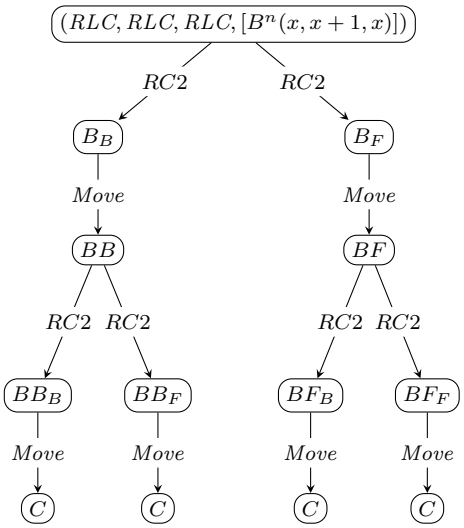
Case $x < y$: We handle two subcases when $y > x+1$ and $y = x+1$.

– If $x+1 < y$, then the moves are:



where
$B_B = (Back, RLC, RLC, [B^n(x, y, x), x+1 < y)])$
$B_F = (Front, RLC, RLC, [B^n(x, y, x), x+1 < y)$
and if $x = 1$
$CL = (RLC, RLC, RLC, [L1^n(2, y, 0)])$
otherwise ($x > 1$):
$CL = (RLC, RLC, RLC, [C^n(x-1, y, x+1)])$
When $x = 1$ the moves lead to $L1^n$ configurations and to C^n configurations otherwise. Hence, again thanks to Lemma 2, the property holds for any x, y when $x+1 < y$.

– If $x+1 = y$, the tree is:



where
$B_B = (Back, RLC, RLC, [B^n(x, x+1, x)])$
$B_F = (Front, RLC, RLC, [B^n(x, x+1, x)])$
$BB = (RLC, RLC, RLC, [B^n(x+1, x+1, x-1)])$
$BF = (RLC, RLC, RLC, [B^n(x-1, x+1, x+1)])$
$BB_B = (RLC, Back, RLC, [B^n(x+1, x+1, x-1)])$
$BB_F = (RLC, Front, RLC, [B^n(x+1, x+1, x-1)])$
$BF_B = (RLC, RLC, Back, [B^n(x-1, x+1, x+1)])$
$BF_F = (RLC, RLC, Front, [B^n(x-1, x+1, x+1)])$
$C = (RLC, RLC, RLC, [C^n(x, x+2, x-1)])$

Since $\mathcal{T}(RLC, RLC, RLC, [C^n(x, y, z)])$ is finite (by Lemma 2), the property holds.

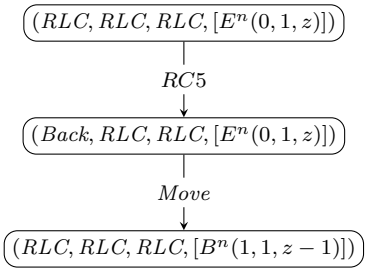
Finally the trees

$$\mathcal{T}(RLC, RLC, RLC[B^n(x, y, x), x < y])$$

are also finite, which concludes the proof.

Lemma 4 *The tree $\mathcal{T}(P)$ is finite for* $P = (RLC, RLC, RLC, [E^n(0, 1, z)])$.

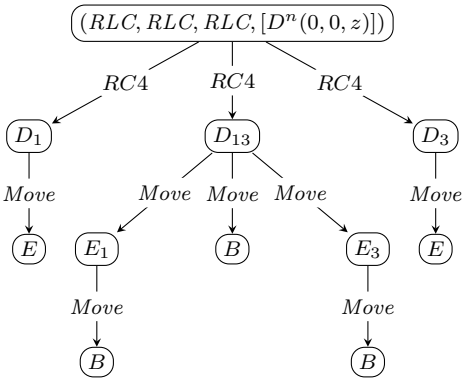
Proof In the case of E^n configurations, we have:



and the result holds thanks to Lemma 3.

Lemma 5 *The tree $\mathcal{T}(P)$ is finite for* $P = (RLC, RLC, RLC, [D^n(0, 0, z)])$.

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



where
$D_1 = (Back, RLC, RLC, [D^n(0, 0, z)])$
$D_{13} = (Back, RLC, Back, [D^n(0, 0, z)])$
$D_3 = (RLC, RLC, Back, [D^n(0, 0, z)])$
$E = (RLC, RLC, RLC, [E^n(0, 1, z-1)])$
$E_1 = (RLC, RLC, Back, [E^n(1, 0, z-1)])$
$E_3 = (Back, RLC, RLC, [E^n(0, 1, z-1)])$
$B = (RLC, RLC, RLC, [B^n(1, 1, z-2)])$
and the result holds from the previous lemmas 3 and 4.

Together these lemmas imply Theorem 3. Finally, Theorems 2 and 3 give the result for perpetual exploration. Moreover, since all reachable configurations from any of the initial configurations are tower-free, the *Exclusivity* property follows (recall that the *No_collision* property implies the *No_switch* property in the asynchronous case, as mentioned in Section 7.1). This concludes the correctness proof of the algorithm.

8 Conclusion

We demonstrated the feasibility of formal verification through model checking for mobile robot protocols in a discrete space. We verified several instances of two protocols for ring exploration. In the first case model checking helped in refining the correctness bounds of the protocol and in the second case it produced a counter-example, which permitted to correct the protocol. We then proved the correctness of this protocol by induction.

Our approach leads not only to find and correct bugs in the protocols (which is especially useful in the more challenging execution model ASYNC),

but also relieves 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.

Going one step further, we would like to provide an automated version for the proof of Theorem 3. The graph of Figures 13 and 14 could be used as a hand made start for a theorem prover, in order to exhibit all cases. Moreover, since these cases are dependent, the order in which the proof has to be carried over should be done manually. Such a proof could also be based on visual abstractions as proposed in [29] or [30], by translating the graph into a suitable diagram. A first challenge would be to define ranking functions for generalized verification diagrams or well-founded orderings for predicate diagrams.

A further step is to generate the protocol automatically from the problem. This would provide solutions that are correct by design. We believe that controller synthesis [46] and more generally, algorithm synthesis for synchronous robots in the spirit of [47], can be extended to obtain such guarantees in the asynchronous case.

Finally, it would be worthwhile to investigate the decidability issues that remain open for the robot model.

Acknowledgement. We are very grateful to anonymous referees for their careful reading and suggestions that were a great help for improving the presentation of this work.

References

1. P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.
2. A. Almeida, G. Ramalho, H. Santana, P. Azevedo Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre. Recent advances on multi-agent patrolling. In *Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence, São Luis, Maranhão, Brazil, September 29 - October 1, 2004, Proceedings*, pages 474–483, 2004.
3. P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. *Algorithmica*, 65(3):562–583, 2013.
4. L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil. Exclusive perpetual ring exploration without chirality. In *Proc. of 24th Int. Symp. in Distributed Computing (DISC'10)*, volume 6343 of *LNCS*, pages 312–327. Springer, 2010.
5. I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.
6. A. Clerentin, M. Delafosse, L. Delahoche, B. Marhic, and A. Jolly-Desodt. Uncertainty and imprecision modeling for the mobile robot localization problem. *Autonomous Robots*, 24(3):267–283, 2008.
7. G. D'Angelo, G. Di Stefano, and A. Navarra. Gathering of six robots on anonymous symmetric rings. In *Proc. of 18th Int. Coll. on Structural Information and Communication Complexity (SIROCCO'11)*, volume 6796 of *LNCS*, pages 174–185. Springer, 2011.
8. S. Kamei, A. Lamani, F. Ooshita, and S. Tixeuil. Asynchronous mobile robot gathering from symmetric configurations without global multiplicity detection. In *Proc. of 18th Int. Coll. on Structural Information and Communication Complexity (SIROCCO'11)*, volume 6796 of *LNCS*, pages 150–161. Springer, 2011.
9. A. Lamani, S. Kamei, F. Ooshita, and S. Tixeuil. Gathering an even number of robots in an odd ring without global multiplicity detection. In *Proc. of Int. Conf. on Mathematical Foundations of Computer Science (MFCS'12)*, volume 7464 of *LNCS*, pages 542–553. Springer, 2012.
10. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1-3):147–168, 2005.
11. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2001.
12. C. Baier and J. P. Katoen. *Principles of model checking*. MIT press, 2008.
13. L. Lamport and S. Merz. Specifying and verifying fault-tolerant systems. In *Proc. of Third Int. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94), organized jointly with the Working Group Provably Correct Systems - ProCoS*, volume 863 of *LNCS*, pages 41–76. Springer, 1994.
14. S. S. Kulkarni, B. Bonakdarpour, and A. Ebnenasir. Mechanical verification of automatic synthesis of fault-tolerant programs. In *Proc. of 14th Int. Symp. on Logic Based Program Synthesis and Transformation (LOPSTR' 04)*, volume 3573 of *LNCS*, pages 36–52. Springer, 2004.
15. R. Guerraoui, T. A. Henzinger, and V. Singh. Model checking transactional memories. *Distributed Computing*, 22(3):129–145, 2010.
16. I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Algorithmic verification of population protocols. In *Proc. of 12th Int. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS'10)*, volume 6366 of *LNCS*, pages 221–235. Springer, 2010.

17. J. Clément, C. Delporte-Gallet, H. Fauconnier, and M. Sighireanu. Guidelines for the verification of population protocols. In *Proc of 31st Int. Conf. on Distributed Computing Systems (ICDCS'11)*, pages 215–224. IEEE, 2011.
18. B. Charron-Bost, H. Debrat, and S. Merz. Formal verification of consensus algorithms tolerating malicious faults. In *Proc. of 13th Int. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS'11)*, volume 6976 of *LNCS*, pages 120–134. Springer, 2011.
19. T. Lu, S. Merz, and C. Weidenbach. Towards verification of the pastry protocol using TLA$^+$. In *Proc of Joint 13t Int. Conf. (FMOODS'11) 2011, and 31st Int. Conf. (FORTE'11) on Formal Techniques for Distributed Systems*, volume 6722 of *LNCS*, pages 244–258. Springer, 2011.
20. T. Tsuchiya and A. Schiper. Verification of consensus algorithms using satisfiability solving. *Distributed Computing*, 23:341–358, 2011.
21. K. R. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, 1986.
22. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *14th Annual Symp. on Logic in Computer Science*, pages 352–359. IEEE, 1999.
23. Ichiro Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28(4):213–214, July 1988.
24. B. Aminof, T. Kotek, S. Rubin, F. Spegni, and H. Veith. Parameterized model checking of rendezvous systems. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 Concurrency Theory*, volume 8704 of *Lecture Notes in Computer Science*, pages 109–124. Springer Berlin Heidelberg, 2014.
25. E. Allen Emerson and Kedar S. Namjoshi. Reasoning about rings. In *Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 85–94, New York, NY, USA, 1995. ACM.
26. Z. Manna and A. Pnueli. Temporal verification diagrams. In *Proc.of Int. Conf. on Theoretical Aspects of Computer Software (TACS'94)*, volume 789 of *LNCS*, pages 726–765. Springer, 1994.
27. E. M. Clarke, O. Grumberg, and S. Jha. Veryfying parameterized networks using abstraction and regular languages. In *Proc. of 6th Int. Conf. on Concurrency Theory (CONCUR'95)*, volume 962 of *LNCS*, pages 395–407. Springer, 1995.
28. N. Bjørner, A. Browne, E. Y. Chang, M. Colón, A. Kapur, Z. Manna, H. Sipma, and T. E. Uribe. Step: Deductive-algorithmic verification of reactive and real-time systems. In *Proc. of 8th Int. Conf. on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*, pages 415–418. Springer, 1996.
29. L. de Alfaro, Z. Manna, H. B. Sipma, and T. E. Uribe. Visual verification of reactive systems. In *Proc. of 3d Int. Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS'97)*, volume 1217 of *LNCS*, pages 334–350. Springer, 1997.
30. D. Cansell, D. Méry, and S. Merz. Diagram refinements for the design of reactive systems. *J. Univ. Comp. Sci.*, 7(2):159–174, 2001.
31. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. D. Zuck. Parameterized verification with automatically computed inductive assertions. In *Proc. of 13th Int. Conf. on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 221–234. Springer, 2001.
32. A Gmeiner, I Konnov, U Schmid, H Veith, and J Widder. Tutorial on parameterized model checking of fault-tolerant distributed algorithms. In Marco Bernardo, Ferruccio Damiani, Reiner Hhnle, Einar-Broch Johnsen, and Ina Schaefer, editors, *Formal Methods for Executable Software Models*, volume 8483 of *Lecture Notes in Computer Science*, pages 122–171. Springer International Publishing, 2014.
33. S. Devismes, A. Lamani, F. Petit, P. Raymond, and S. Tixeuil. Optimal grid exploration by asynchronous oblivious robots. In *Proc of 14th Int. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS'12)*, volume 7596 of *LNCS*, pages 64–76. Springer, 2012.
34. F. Bonnet, X. Défago, F. Petit, M. Potop-Butucaru, and S. Tixeuil. Brief announcement: Discovering and assessing fine-grained metrics in robot networks protocols. In *Proc of 14th Int. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS'12)*, volume 7596 of *LNCS*, pages 282–284. Springer, 2012.
35. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language Lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.
36. J. Barnat, L. Brim, V. Havel, J. Havlícek, J. Kriho, M. Lenco, P. Rockai, V. Still, and J. Weiser. Divine 3.0 - an explicit-state model checker for multithreaded C & C++ programs. In *Proc. of 25th Int. Conf. on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 863–868. Springer, 2013.
37. M. Colange, S. Baarir, F. Kordon, and Y. Thierry-Mieg. Towards Distributed Software Model-Checking using Decision Diagrams. In *Proc. of 25th Int. Conf. on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 830–845. Springer, 2013.
38. G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Guldstrand Larsen, and D. Lime. Uppaal-tiga: Time for playing games! In *19th International Conference on Computer Aided Verification, CAV'07*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer, 2007.
39. J. Barnat, L. Brim, M. Češka, and P. Ročkai. DiVinE: Parallel Distributed Model Checker (Tool paper). In *Parallel and Distributed Methods in Verification and High Performance Computational Systems Biology*, pages 4–7. IEEE, 2010.
40. G. Holzmann. *Spin Model Checker, the: Primer and Reference Manual*. Addison-Wesley Professional, first edition, 2003.
41. S. Blom, J. van de Pol, and M. Weber. Ltsmin: Distributed and symbolic reachability. In T. Touili,

B. Cook, and P. Jackson, editors, *Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 354–359. Springer Berlin Heidelberg, 2010.
42. B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer Publishing Company, Incorporated, 1st edition, 2010.
43. M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978.
44. J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, October 1995.
45. L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In *Proc. of 16th Int. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS'14)*, volume 8756 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2014.
46. W. M. Ramadge, P. J. Wonham. Supervisory control of a class of discrete event processes. In *Proc. of 6th Int. Conf. on Analysis and Optimization of Systems*, volume 63 of *LNCS*. Springer, 1984.
47. L. Millet, M. Potop-Butucaru, N. Sznajder, and S. Tixeuil. On the synthesis of mobile robots algorithms: The case of ring gathering. In *Proc. of 16th Int. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS'14)*, volume 8756 of *LNCS*, pages 237–251. Springer, 2014.