Self-Stabilizing Robots in Highly Dynamic Environments*

Marjorie Bournat† Ajoy K. Datta‡ Swan Dubois†

Abstract

This paper deals with the classical problem of exploring a ring by a cohort of synchronous robots. We focus on the perpetual version of this problem in which it is required that each node of the ring is visited by a robot infinitely often.

The challenge in this paper is twofold. First, we assume that the robots evolve in a highly dynamic ring, *i.e.,* edges may appear and disappear unpredictably without any recurrence nor periodicity assumption. The only assumption we made is that each node is infinitely often reachable from any other node. Second, we aim at providing a self-stabilizing algorithm to the robots, *i.e.,* the algorithm must guarantee an eventual correct behavior regardless of the initial state and positions of the robots.

Our main contribution is to show that this problem is deterministically solvable in this harsh environment by providing a self-stabilizing algorithm for three robots.

†UPMC Sorbonne Universités, CNRS, Inria, LIP6 UMR 7606, France

‡University of Nevada, Las Vegas, United States

*This work has been partially supported by the ANR project ESTATE and was initiated while the second author was visiting professor at UPMC Sorbonne Universités.

1 Introduction

We consider a cohort of autonomous and synchronous robots that are equipped with motion actuators and sensors, but that are otherwise unable to communicate [23]. They evolve in a *discrete environment*, where the space is partitioned into a finite number of locations, represented by a graph, where the nodes represent the possible locations of robots and the edges the possibility for a robot to move from one location to another. Refer to [21] for a survey of results in this model. One fundamental problem is the *exploration* of graphs by robots. Basically, each node of the graph has to be visited by at least one robot. There exist several variants of this problem depending on whether the robots are required to stop once they completed the exploration of the graph or not.

Typically, the environment of the robots is modeled by a *static* undirected connected graph where vertices are possible locations of robots and edges represent the moving abilities of the robots. Clearly, such modeling is not suitable for dynamic environments that we use in this paper. Numerous models dealing with topological changes over time have been proposed in the past few decades. There have been some attempts to unifying them as well. The *evolving graphs* were introduced in [25]. They proposed modeling the time as a sequence of discrete time instants and the dynamicity of the system by a sequence of static graphs, one for each instant of time. More recently, another graph model, called *Time-Varying Graphs* (TVG), has been introduced in [4]. In contrast with evolving graphs, TVGs allow systems evolving in continuous time. Also in [4], TVGs are ordered into classes based on mainly two features: the quality of connectivity of the graph and the possibility/impossibility to perform tasks.

As in other distributed systems, *fault-tolerance* is a central issue in robot networks. Indeed, it is desirable that the misbehavior of some robots does not prevent the whole system to reach its objective. *Self-stabilization* [8, 10, 24] is a versatile technique to tolerate *transient* (*i.e.,* of finite duration) faults. After the occurrence of a catastrophic failure that may take the system to some arbitrary global state, self-stabilization guarantees recovery to a correct behavior in finite time without external (*i.e.,* human) intervention. In the context of robot networks, that implies that the algorithm must guarantee an eventual correct behavior regardless of the initial state and positions of the robots.

Our objective in this paper is to study the feasibility of the exploration of a highly dynamic graph by a cohort of self-stabilizing deterministic robots.

Related Work. Since the seminal work of Shannon [22], exploration of graphs by a cohort of robots has been extensively studied. There exist mainly three variants of the problem: (*i*) *exploration with stop*, where robots are required to detect the end of the exploration, then stop moving (*e.g.,* [12]); (*ii*) *exploration with return*, where robots must come back to their initial location once the exploration completed (*e.g.,* [9]); and (*iii*) *perpetual exploration*, where each node has to be infinitely often visited by some robots (*e.g.,* [1]). Even if we restrict ourselves to deterministic approaches, there exist numerous solutions to these problems depending on the topology of the graphs to explore (*e.g.,* ring-shaped [12], line-shaped [14], tree-shaped [13], or arbitrary network [5]), and the assumptions made on robots (*e.g.,* limited range of visibility [6], common sense of orientation [2], *etc.*). But, most of the above work considered only static graphs.

Recently, some work dealt with the exploration of dynamic graphs. The first two papers [15, 17] focused on the exploration (with stop) of so-called periodically varying graphs (*i.e.,* the presence of each edge of the graph is totally periodic). The papers [18, 16, 7] considered another restriction on dynamicity by considering T-interval-connected graphs (*i.e.,* the graph is always connected and there exists a stability of this connectivity in any interval of time of length T [20]). However, there exist no exploration algorithms for highly dynamic graphs, *i.e.,* graphs where edges may appear and disappear unpredictably without any recurrence, periodicity, or stability assumption and where the only assumption made is that each node is infinitely often reachable from any other node.

To the best of our knowledge, there exist no self-stabilizing algorithm for exploration either in a static or a dynamic environment. Note that there exist solutions in static graphs to other problems (*e.g.,* naming and leader election [3]).

Our Contribution. The main contribution of this paper is to give a positive answer to the open question whether self-stabilizing deterministic exploration of highly dynamic graphs is possible or not. We answer that question by providing a self-stabilizing algorithm to perpetually explore any highly dynamic ring with three deterministic synchronous robots. This is the first exploration algorithm that deals with highly dynamic graphs. This is also the first self-stabilizing algorithm for exploration.

Organization of the paper. This paper is organized as follows. In Section 2, we present the formal model and state the assumptions made. In Section 3, we describe our algorithm. Section 4 contains the proof sketch of our algorithm.

2 Model

In this section, we propose an extension of the classical model of robot networks in static graphs introduced by [19] to the context of dynamic graphs.

Dynamic graphs. In this paper, we consider the model of *evolving graphs* introduced in [25]. We hence consider the time as discretized and mapped to $\mathbb{N}$. An evolving graph $\mathcal{G}$ is an ordered sequence $\{G_1, G_2, \ldots\}$ of subgraphs of a given static graph $G = (V, E)$. In the following, we restrict ourselves to bidirectional graphs. For any $i \geq 0$, we have $G_i = (V, E_i)$ and we say that the edges of E_i are *present* in $\mathcal{G}$ at time i. The *underlying graph* of $\mathcal{G}$, denoted $U_\mathcal{G}$, is the static graph gathering all edges that are present at least once in $\mathcal{G}$ (*i.e.,* $U_\mathcal{G} = (V, E_\mathcal{G})$ with $E_\mathcal{G} = \bigcup_{i=0}^{\infty} E_i$). An *eventual missing edge* is an edge of $E_\mathcal{G}$ such that there exists a time after which this edge is never present in $\mathcal{G}$. A *recurrent edge* is an edge of $E_\mathcal{G}$ that is not eventually missing. The *eventual underlying graph* of $\mathcal{G}$, denoted $U_\mathcal{G}^\omega$, is the static graph gathering all recurrent edges of $\mathcal{G}$ (*i.e.,* $U_\mathcal{G}^\omega = (V, E_\mathcal{G}^\omega)$ where $E_\mathcal{G}^\omega$ is the set of recurrent edges of $\mathcal{G}$). In this paper, we chose to make minimal assumptions on the dynamicity of our graph since we restrict ourselves on *connected-over-time* evolving graphs. The only constraint we impose on evolving graphs of this class is that their eventual underlying graph is connected [11] (intuitively, that means that any node is infinitely often reachable from any other one). For the sake of the proof, we also consider the weaker class of *edge-recurrent* evolving graphs where the eventual underlying graph is connected and matches to the underlying graph. In the following, we consider only connected-over-time evolving graphs whose underlying graph is an anonymous and unoriented ring of arbitrary size. Although the ring is unoriented, to simplify the presentation and discussion, in this paper, we, as external observers, distinguish between the clockwise and the counter-clockwise (global) direction in the ring.

Robots. We consider systems of autonomous mobile entities called robots moving in a discrete and dynamic environment modeled by an evolving graph $\mathcal{G} = \{(V, E_1), (V, E_2) \ldots\}$, V being a set of nodes representing the set of locations where robots may be, E_i being the set of bidirectional edges representing connections through which robots may move from a location to another one at time i. Robots are uniform (they execute the same algorithm), identified (each of them has a distinct identifier), have a persistent memory but are unable to directly communicate with one another by any means. Robots are endowed with local strong multiplicity detection (*i.e.,* they are able to detect the exact number of robots located on their current node). They have no a priori knowledge about the ring they explore (size, diameter, dynamicity. . .). Finally, each robot has its own stable chirality (*i.e.,* each robot is able to locally label the two ports of its current node with *left* and *right* consistently over the ring and time but two different robots may not agree on this labeling). We assume that each robot has a variable *dir* that stores a direction (either *left* or *right*). At any time, we say that a robot points to *left* (resp. *right*) if its *dir* variable is equal to this (local) direction. We say that a robot considers the clockwise (resp., counter-clockwise) direction if the (local) direction pointed to by this robot corresponds to the (global) direction seen by an external observer.

Execution. A configuration γ of the system captures the position (*i.e.,* the node where the robot is currently located) and the state (*i.e.,* the value of every variable of the robot) of each robot at a given time. Given an evolving graph $\mathcal{G} = \{G_1, G_2, \ldots\}$, an algorithm $\mathcal{A}$, and an initial configuration γ_0, the execution $\mathcal{E}$ of $\mathcal{A}$ on $\mathcal{G}$ starting from γ_0 is the infinite sequence $(G_0, \gamma_0), (G_1, \gamma_1), (G_2, \gamma_2), \ldots$ where, for any $i \geq 0$, the configuration γ_{i+1} is the result of the execution of a synchronous round by all robots from (G_i, γ_i) as explained below.

The round that transitions the system from (G_i, γ_i) to (G_{i+1}, γ_{i+1}) is composed of three atomic and synchronous phases: Look, Compute, Move. During the Look phase, each robot gathers information about its environment in G_i. More precisely, each robot updates the value of the following local predicates: (*i*) $NumberOfRobotsOnNode()$ returns the exact number of robots present at the node of the robot; (*ii*) $ExistsEdgeOnCurrentDirection()$ returns true if an edge is present at the direction currently pointed by the robot, false otherwise; (*iii*) $ExistsEdgeOnOppositeDirection()$ returns true if an edge is present in the direction opposite to the one currently pointed by the robot, false otherwise; (*iv*) $ExistsAdjacentEdge()$ returns true if an edge adjacent to the current node of the robot is present, false otherwise. During the Compute phase, each robot executes the algorithm $\mathcal{A}$ that may modify some of its variables (in particular *dir*) depending on of its current state and the values of the predicates updated during the Look phase. Finally, the Move phase consists of moving each robot trough one edge in the direction it points to if there exists an edge in that direction, otherwise, *i.e.,* if the edge is missing at that time, the robot remains at its current node. Note that the i^{th} round is entirely executed on G_i and that the transition from G_i to G_{i+1} occurs only at the end of this round. We say that a robot is *edge-activated* during a round if there exists at least one edge adjacent to its location during that round.

Self-Stabilization. Intuitively, a self-stabilizing algorithm is able to recover in a finite time a correct behavior from any arbitrary initial configuration (that captures the effect of an arbitrary transient fault in the system). More formally, an algorithm $\mathcal{A}$ is *self-stabilizing* for a problem on a class of evolving graphs $\mathcal{C}$ if and only if it ensures that, for any configuration γ_0, the execution of $\mathcal{A}$ on any $\mathcal{G} \in \mathcal{C}$ starting from γ_0 contains a configuration γ_i such that the execution of $\mathcal{A}$ on $\mathcal{G}$ starting from γ_i satisfies the specification of the problem. Note that, in the context of robot networks, this definition implies that robots must tolerate both arbitrary initialization of their variables and arbitrary initial positions (in particular, robots may be stacked in the initial configuration).

Perpetual Exploration. Given an evolving graph $\mathcal{G}$, a perpetual exploration algorithm guarantees that every node of $\mathcal{G}$ is infinitely often visited by at least one robot (*i.e.,* a robot is infinitely often located at every node of $\mathcal{G}$). Note that this specification does not require that every robot visits infinitely often every node of $\mathcal{G}$.

3 Exploring a Highly Dynamic Ring with Three Robots

In this section, we present our self-stabilizing deterministic algorithm for the perpetual exploration of any connected-over-time ring with three robots. In this context, the difficulty to complete the exploration is twofold. First, in connected-over-time graphs, robots must deal with the possible existence of some eventual missing edge (without the guarantee that such edge always exists). Note that, in the case of a ring, there is at most one eventual missing edge in any execution (otherwise, we have a contradiction with the connected-over-time property). Second, robots have to handle the arbitrary initialization of the system (corruption of variables and arbitrary position of robots).

Principle of the algorithm. The main idea behind our algorithm is that a robot does not change its direction (arbitrarily initialized) while it is isolated. This allows robots to perpetually explore connected-over-time rings with no eventual missing edge regardless of the initial direction of the robots.

Obviously, this idea is no longer sufficient when there exists an eventual missing edge since, in this case, at least two robots will eventually be stuck (*i.e.,* they point to an eventual missing edge that they are never able to cross) forever at one end of the eventual missing edge. When two (or more) robots are located at the same node, we say that they form a tower. In this case, our algorithm succeed (as we explain below) to ensure that at least one robot leaves the tower in a finite time. In this way, we obtain that, in a finite time, a robot is stuck at each end of the eventual missing edge. These two robots located at two ends of the eventual missing edge play the role of "sentinels" while the third one (we call it a "visitor") visits other nodes of the ring in the following way. The "visitor" keeps its direction until it meets one of these "sentinels", they then switch their roles: After the meeting, the "visitor" still maintains the same direction (becoming thus a "sentinel") while the "sentinel" robot changes its direction (becoming thus a "visitor" until reaching the other "sentinel").

In fact, robots are never aware if they are actually stuck at an eventual missing edge or are just temporarily stuck on an edge that will reappear in a finite time. That is why it is important that the robots keep consider their directions and try to move forward while there is no meeting in order to track a possible eventual missing edge. Our algorithm only guarantees a convergence in a finite time towards a configuration where a robot plays the role of "sentinel" at each end of the eventual missing edge if such an edge exists. Note that, in the case where there is no eventual missing edge, this mechanism does not prevent the correct exploration of the ring since it is impossible for a robot to be stuck forever.

Our algorithm easily deals with the initial corruption of its variables. Indeed, we use variables only to save some information about the environment of the robots in the previous rounds and we update them at each round. Thus, their arbitrary initial value is erased in a finite time. The main difficulty to achieve self-stabilization is to deal with the arbitrary initial position of robots. In particular, the robots may initially form towers. In the worst case, all robots of a tower may be stuck at an eventual missing edge and be in the same state. They are then unable to start the "sentinels"/"visitor" scheme explained above. Our algorithm needs to "break" such a tower in a finite time (*i.e.,* one robot must leave the node where the tower is located). In other words, we tackle a classical problem of symmetry breaking. We succeed by providing each robot with a function that returns, in a finite number of invocations, different global directions to two robots of the tower based on the private identifier of the robot and without any communication among the robots. More precisely, this is done thanks to a transformation of the robot identifier: each bit of the binary representation of the identifier is duplicated and we add the bits "01" at the end of the sequence of these duplicated bits. Then, at each invocation of the function, a robot reads the next bit of this transformed identifier. If the robot reads zero, it try to move to its left. Otherwise, it try to move to its right. Doing so, in a finite number of invocation of this function, at least one robot leaves the tower. If necessary, we repeat this "tower breaking" scheme until we are able to start the "sentinels"/"visitor" scheme.

The main difficulty in designing this algorithm is to ensure that these two mechanisms ("sentinels"/"visitor" and "tower breaking") do not interfere with each other and prevent the correct exploration. We solve this problem by adding some waiting at good time, especially before starting the procedure of tower breaking by identifier to ensure that robots do not prematurely turn back and "forget" to explore some parts of the ring.

Formal presentation of the algorithm. Before presenting formally our algorithm, we need to introduce the set of constants (*i.e.,* variables assumed to be not corruptible) and the set of variables of each robot. We also introduce three auxiliary functions.

As stated in the model, each robot has an unique identifier. We denote it by id and represent it in binary as $b_0 b_1 \dots b_{|id|-1}$. We define, for the purpose of the "breaking tower" scheme, the constant $TransformedIdentifier$ by its binary representation $b_0 b_0 b_1 b_1 \dots b_{|id|-1} b_{|id|-1} 01$ (each bit of id is duplicated and we add the two bits 01 at the end). We store the length of the binary representation of $TransformedIdentifier$ in the constant ℓ and we denote its ith bit by $TransformedIdentifier[i]$ for any $0 \leq i \leq \ell - 1$.

In addition to the variable dir defined in the model, each robot has the following three variables: (i) the variable $i \in \mathbb{N}$ corresponds to an index to store the position of the last bit read from $TransformedIdentifier$; (ii) the variable $NumberRobotsPreviousEdgeActivation \in \mathbb{N}$ stores the number of robots that were present at the node of the robot during the look step of the last round where it was edge-activated; and (iii) the variable $HasMovedPrevious\text{-}\ EdgeActivation \in \{true, false\}$ indicates if the robot has crossed an edge during its last edge-activation.

Our algorithm makes use of a function Update that updates the value of the two last variables according to the current environment of the robot each time it is edge-activated. We provide the pseudo-code of this function in Algorithm 1. Note that this function also allows us to deal with the initial corruption of the two last variables since it resets them in the first round where the robot is edge-activated.

We already stated that, whenever robots are stuck forming a tower, they make use of a function to "break" the tower in a finite time. The pseudo-code of this function GiveDirection appears in Algorithm 2. It assigns the value $left$ or $right$ to the variable dir of the robot depending on

Algorithm 1 Function Update

```
function Update
    if ExistsAdjacentEdge() then
        NumberRobotsPreviousEdgeActivation ← NumberOfRobotsOnNode()
        HasMovedPreviousEdgeActivation ← ExistsEdgeOnCurrentDirection()
    end if
end function
```

the the ith bit of the value of $TransformedIdentifier$. The variable i is incremented modulo ℓ (that implicitly resets this variable when it is corrupted) to ensure that successive calls to GiveDirection will consider each bit of $TransformedIdentifier$ in a round-robin way. As shown in the next section, this function guarantees that, if two robots are stuck together in a tower and invoke repeatedly their own function GiveDirection, then two distinct global directions are given in finite time to the two robots regardless of their chirality. This property allows the algorithm to "break" the tower since at least one robot is then able to leave the node where the tower is located.

Finally, we define the function OppositeDirection that simply affects the value $left$ (resp. $right$) to the variable dir when $dir = right$ (resp. $dir = left$).

There are two types of configurations in which the robots may change the direction they consider. So, our algorithm needs to identify them. We do so by defining a predicate that characterizes each of these configurations.

The first one, called $WeAreStuckInTheSameDirection()$, is dedicated to the detection of configurations in which the robot must invoke the "tower breaking" mechanism. Namely, the robot is stuck since at least one edge-activation with at least another robot and the edge in the direction opposite to the one considered by the robot is present. More formally, this predicate is defined as follows:

$$\begin{aligned} &WeAreStuckInTheSameDirection() \equiv \\ &\qquad (NumberOfRobotsOnNode() > 1) \\ &\quad \wedge\ (NumberOfRobotsOnNode() = NumberRobotsPreviousEdgeActivation) \\ &\quad \wedge\ \neg ExistsEdgeOnCurrentDirection() \\ &\quad \wedge\ ExistsEdgeOnOppositeDirection() \\ &\quad \wedge\ \neg HasMovedPreviousEdgeActivation \end{aligned}$$

The second predicate, called $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$, is designed to detect configurations in which the robot must transition from the "sentinel" to the "visitor" role in the "sentinel"/"visitor" scheme. More precisely, such configuration is characterized by the fact that the robot is edge-activated, stuck during its previous edge-activation, and there are strictly more robots located at its node than at its previous edge-activation. More formally, this predicate is defined as follows:

$$\begin{aligned} &IWasStuckOnMyNodeAndNowWeAreMoreRobots() \equiv \\ &\qquad (NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation) \\ &\quad \wedge\ \neg HasMovedPreviousEdgeActivation \\ &\quad \wedge\ ExistsAdjacentEdge() \end{aligned}$$

Now, we are ready to present the pseudo-code of the core of our algorithm (see Algorithm 3). The basic idea of the algorithm is the following. The function GiveDirection is invoked when $WeAreStuckInTheSameDirection()$ is true (to try to "break" the tower after the appropriate waiting), while the function OppositeDirection is called when $IWasStuckOnMyNodeAndNowWe$-

Algorithm 2 Function GiveDirection

```
function GiveDirection
    i ← i + 1 (mod ℓ)
    if TransformedIdentifier[i] = 0 then
        dir ← left
    else
        dir ← right
    end if
end function
```

Algorithm 3 Self-stabilizing perpetual exploration

```
if WeAreStuckInTheSameDirection() then
    GiveDirection
end if
if IWasStuckOnMyNodeAndNowWeAreMoreRobots() then
    OppositeDirection
end if
Update
```

$AreMoreRobots()$ is true (to implement the "sentinel"/"visitor" scheme). Afterwards, the function Update is called (to update the state of the robot according to its environment).

4 Proof Sketch

Preliminaries. First, we introduce some definitions and preliminary results that are extensively used in the proof.

We saw previously that the notion of tower is central in our algorithm. Intuitively, a tower captures the simultaneous presence of all robots of a given set on a node at each time of a given interval. We require either the set of robots or the time interval of each tower to be maximal. Note that the tower is not required to be on the same node at each time of the interval (robots of the tower may move together without leaving the tower).

We distinguish two kinds of towers according to the agreement of their robots on the global direction to consider at each time there exists an adjacent edge to their current location (excluded the last one). If they agreed, the robots form a long-lived tower while they form a short-lived tower in the contrary case. This implies that a short-lived tower is broken as soon as the robots forming the tower are edge-activated, while the robots of a long-lived tower move together at each edge activation of the tower (excluded the last one).

Definition 4.1 (Tower)**.** *A tower T is a couple (S, θ), where S is a set of robots ($|S| > 1$) and $\theta = [t_s, t_e]$ is an interval of $\mathbb{N}$, such that all the robots of S are located at a same node at each instant of time t in θ and S or θ are maximal for this property. Moreover, if the robots of S move during a round $t \in [t_s, t_e[$, they are required to traverse the same edge.*

Definition 4.2 (Long-lived tower)**.** *A long-lived tower $T = (S, [t_s, t_e])$ is a tower such that there is at least one edge-activation of all robots of S in the time interval $[t_s, t_e[$.*

Definition 4.3 (Short-lived tower)**.** *A short-lived tower T is a tower that is not a long-lived tower.*

For $k > 1$, a long-lived (resp., a short-lived) tower $T = (S, \theta)$ with $|S| = k$ is called a k-long-lived (resp., a k-short-lived) tower.

As their are only three robots on our system, and that in each round they consider a global direction, we can make the following observation.

Observation 4.1. *There are at least two robots having the same global direction at each instant time.*

In the remainder of this section, we consider an execution $\mathcal{E}$ of Algorithm 3 executed by three robots r_1, r_2, and r_3 on a connected-over-time ring $\mathcal{G}$ of size $n \in \mathbb{N}^*$ starting from an arbitrary configuration.

For the sake of clarity, the value of a variable or a predicate *name* of a given robot r at the end of the Look phase of a given round t is denoted by the notation $name(r, t)$.

We say that a robot r has a coherent state at time t, if during the Look phase of round t, the value of its variable $NumberRobotsPreviousEdgeActivation(r, t)$ corresponds to the value of its predicate $NumberOfRobotsOnNode()$ at its previous edge-activation and the value of its variable $HasMovedPreviousEdgeActivation(r, t)$ corresponds to the value of its predicate $ExistsEdgeOn$-$CurrentDirection()$ at its previous edge-activation. The following lemma states that, for each robot, there exists a suffix of the execution in which the robot is coherent.

Lemma 4.1. *For any robot, there exists a time from which its state is always coherent.*

Proof. Consider a robot r performing algorithm 3.

As $\mathcal{G}$ belongs to the class of connected-over-time rings, at least one adjacent edge to each node of $\mathcal{G}$ is infinitely often present in the system. As the robots of the system are synchronous, from the previous observation we can conclude that they are infinitely often edge-activated.

Variables can be updated only during Compute phases of rounds. If r is edge-activated at time t, then during the Compute phase of time t, the function UPDATE updates respectively its variables $NumberRobotsPreviousEdgeActivation$ and $HasMovedPreviousEdgeActivation$ with the current values of its predicates $NumberOfRobotsOnNode()$ and $ExistsEdgeOnCurrentDirection()$.

Moreover, each time r is not edge-activated, the values of its variables $NumberRobotsPrevious$-$EdgeActivation$ and $HasMovedPreviousEdgeActivation$ are not updated.

Consider a time t' when r is edge-activated. Assume that t' is not the first time when r is edge-activated. Then as the variables are only updated during the Compute phases of rounds, during the Look phase of time t' the values of the variables $NumberRobotsPreviousEdgeActivation$ and $HasMovedPreviousEdgeActivation$ of r correspond respectively to the values of the predicates $NumberOfRobotsOnNode()$ and $HasMovedPreviousEdgeActivation()$ of r at the previous edge-activation.

Call t_1 the first time when r is edge-activated. By the three above arguments we can conclude that from time $t_1 + 1$, r is in a coherent state. ☐

Let t_1, t_2, and t_3 be respectively the time at which the robot r_1, r_2, and r_3, respectively are in a coherent state. Let $t_{max} = max\{t_1, t_2, t_3\}$. From Lemma 4.1, the three robots are in a coherent state from t_{max}. In the remaining of the proof, we focus on the suffix of the execution after t_{max}.

The two following lemmas show that, regardless of the chirality of the robots and the initial values of their variables i, a finite number of synchronous invocations of the function GIVEDIRECTION by two robots of a tower returns them a distinct global direction. To prove that, we need to take a close look at properties granted by the transformed identifiers of the robots.

Lemma 4.2. *Let tl_1 and tl_2 be two transformed identifiers, such that $tl_1 \neq tl_2$. Let i and j be two integers such that $i \in [0, |tl_1| - 1]$ and $j \in [0, |tl_2| - 1]$. If $tl_1[i] = tl_2[j]$, then there exists an integer k such that $tl_1[(i + k) \pmod{|tl_1|}] \neq tl_2[(j + k) \pmod{|tl_2|}]$.*

Proof. Consider two non-transformed identifiers $ntl1$ and ntl_2. Consider tl_1 and tl_2 their respective transformed identifiers. ntl_1 and ntl_2 are distinct, so tl_1 and tl_2 are distinct by definition of the transformed identifier. Take two integers i and j such that i is in $[0, |tl_1| - 1]$ and j is in $[0, |tl_2| - 1]$.

We want to prove that if $tl_1[i]$ equals $tl_2[j]$ then it exists an integer k such that $tl_1[(i + k) \pmod{|tl_1|}]$ is not equal to $tl_2[(j + k) \pmod{|tl_2|}]$.

By contradiction we assume that such a k does not exist. This means that for all k in $\mathbb{N}$, $tl_1[(i + k) \pmod{|tl_1|}]$ equals $tl_2[(j + k) \pmod{|tl_2|}]$.

The construction of the transformed identifier is made by duplicating each bit of the non-transformed identifier concatenated with the pair of bits "01". Thus we have $|tl_1| = 2 \times |ntl_1| + 2$, and $|tl_2| = 2 \times |ntl_2| + 2$.

Note $\{b_1 b_1 \dots b_{|ntl_1|} b_{|ntl_1|} 01\}$ the binary representation of tl_1. Similarly note $\{b'_1 b'_1 \dots b'_{|ntl_2|} b'_{|ntl_2|} 01\}$ the binary representation of tl_2. Call final pair, the pair of bits "01" during each transformed identifier.

Consider the integer h such that $tl_1[(i + h) \pmod{|tl_1|}]$ corresponds to the 0 of the final pair of tl_1.

Either the labels tl_1 and tl_2 have the same size or one is greater than the other one.

Case 1: $|\mathbf{tl_1}| = |\mathbf{tl_2}|$.

By assumption we have $tl_2[(j+h) \pmod{|tl_1|}]$ equals to 0. Moreover $tl_1[(i+h+1) \pmod{|tl_1|}]$ corresponds to the 1 of the final pair of tl_1, thus by assumption $tl_2[(j + h + 1) \pmod{|tl_1|}]$ is equal to 1. We can conclude that $tl_2[(j + h) \pmod{|tl_1|}]$ corresponds either to the second bit b'_p of a pair of bits $b'_p b'_p$ where each bit is equal to 0, with p is an integer in $[1, |ntl_1 - 1|]$, and such that b'_{p+1} equals to 1, or to the 0 of the final pair of tl_2.

Case 1.1: $\mathbf{tl_2}[(\mathbf{j} + \mathbf{h}) \pmod{|\mathbf{tl_1}|}]$ corresponds to the second bit $\mathbf{b'_p}$ of a pair of bits $\mathbf{b'_p b'_p}$ where each bit is equal to 0.

In this case we know that $tl_2[(j + h + 1) \pmod{|tl_1|}]$ corresponds to the first bit b'_{p+1} of a pair of bits $b'_{p+1} b'_{p+1}$ where each bit is equal to 1 (with p an integer in $[1, |ntl_1 - 1|]$). As to construct the transformed identifier each bit of the non-transformed identifier is duplicated and at the end the final pair is added, the only odd sequence of bits containing only bits of value equals to 1 must include the bit 1 of the final pair. All the other sequences of bits containing only bits of value equals to 1 but not containing the 1 of the final pair are even. Thus here the sequence of bits $\{b'_{p+1} b'_{p+1} \dots b'_{p+1+q} b'_{p+1+q}\}$ with q an integer in $[0, |ntl_1| - p - 1]$ and such that all the bits of this sequence are equal to 1, is an even sequence. However the sequence $\{b_0 \dots b_z b_z\}$ with b_0 corresponding to the 1 of the final pair of tl_1, and with z an integer in $[0, |ntl_1|]$ and such that all the bits of this sequence are equal to 1, is an odd sequence. Thus there exists an integer $y = 1 + min\{p + 1 + q, z\}$ such that b'_y is not equal to b_y, which is in contradiction with the fact that for all integers k in $\mathbb{N}$, $tl_1[(i+k) \pmod{|tl_1|}]$ equals $tl_2[(j+k) \pmod{|tl_2|}]$.

Case 1.2: $\mathbf{tl_2}[(\mathbf{j} + \mathbf{h}) \pmod{|\mathbf{tl_1}|}]$ corresponds to the 0 of the final pair of $\mathbf{tl_2}$.

In this case, as by assumption we have $|tl_1|$ equals to $|tl_2|$, then we have i equals to j. Moreover we have for all k in $\mathbb{N}$, $tl_1[(i + k) \pmod{|tl_1|}]$ equals $tl_2[(j + k) \pmod{|tl_1|}]$.

Thus here we now have for all k in $\mathbb{N}$, $tl_1[(i+k) \pmod{|tl_1|}]$ equals $tl_2[(i+k) \pmod{|tl_2|}]$, with $|tl_1|$ equals to $|tl_2|$. This implies that tl_1 is equal to tl_2 which is in contradiction with the fact that the two transformed identifiers are distinct.

Case 2: $|\mathbf{tl_1}| \neq |\mathbf{tl_2}|$.

Without lost of generality, assume that $|tl_1|$ is strictly less than $|tl_2|$.

Similarly as previously in order to have $tl_1[(i+h) \pmod{|tl_1|}]$ equals to $tl_2[(j+h) \pmod{|tl_2|}]$, $tl_2[(j+h) \pmod{|tl_2|}]$ must either corresponds to the second bit b'_p of a pair of bits $b'_p b'_p$ where each bit is equal to 0, with p is an integer in $[1, |ntl_2 - 1|]$, and such that b'_{p+1} equals to 1, or to the 0 of the final pair of tl_2.

Case 2.1: $\mathbf{tl_2}[(\mathbf{j}+\mathbf{h}) \pmod{|\mathbf{tl_2}|}]$ **corresponds to the second bit** $\mathbf{b'_p}$ **of a pair of bits** $\mathbf{b'_p b'_p}$ **where each bit is equal to 0.**

We can use the same argument than the one used for the case 1.1 with the odd and even sequences of bits to lead to a contradiction with the fact that for all integers k in $\mathbb{N}$, $tl_1[(i+k) \pmod{|tl_1|}]$ equals $tl_2[(j+k) \pmod{|tl_2|}]$.

Case 2.2: $\mathbf{tl_2}[(\mathbf{j}+\mathbf{h}) \pmod{|\mathbf{tl_2}|}]$ **corresponds to the 0 of the final pair of** $\mathbf{tl_2}$**.**

In this case, as tl_2 and tl_1 have different size, the next time we read the 0 of the final pair of tl_1 the bit considered in tl_2 will correspond a bit b'_p with p in $[1, |ntl_2| - 1]$. We are then in a case identical to the case 2.1 which leads to a contradiction.

These arguments prove the lemma. □

Lemma 4.3. *Let* tl_1 *and* tl_2 *be two transformed identifiers, such that* $tl_1 \neq tl_2$*. Let* i *and* j *be two integers such that* $i \in [0, |tl_1| - 1]$ *and* $j \in [0, |tl_2| - 1]$*. If* $tl_1[i] \neq tl_2[j]$*, then there exists an integer* k *such that* $tl_1[(i+k) \pmod{|tl_1|}] = tl_2[(j+k) \pmod{|tl_2|}]$*.*

Proof. Here we assume that for all k in $\mathbb{N}$, $tl_1[(i+k) \pmod{|tl_1|}]$ is not equal to $tl_2[(j+k) \pmod{|tl_2|}]$. With similar arguments than the one used for the proof of lemma 4.2 we obtain contradictions leading to the correctness of this lemma. □

Technical lemmas on towers. We are now able to state a set of lemmas that show some interesting technical properties of towers under specific assumptions during the execution of our algorithm. These properties are extensively used in the main proof of our algorithm.

Lemma 4.4. *The robots of a long-lived tower* $T = (S, [t_s, t_e])$ *consider a same global direction at each time between the Look phase of round* t_s *and the Look phase of round* t_e *included.*

Proof. Consider a long-lived tower $T = (S, [t_s, t_e])$.

By definition of a long-lived tower we know that there exists at least a time in $[t_s, t_e[$ at which the robots of S are edge-activated.

Call t_{act} the first time in $[t_s, t_e]$ at which the robots of S are edge-activated.

From the Look phase of time $\mathbf{t_s}$ **to the Look phase of time** $\mathbf{t_{act}}$ **the robots of S consider a same global direction.**

By contradiction assume that there exists a time t between the Look phase of time t_s and the Look phase of time t_{act} at which the robots of S consider opposite global directions.

Call S_1 ($|S_1| \geq 1$) the set of robots of S considering the clockwise direction at time t, and S_2 ($|S_2| \geq 1$) the set of robots of S considering the counter clockwise direction at time t.

During $[t_s, t_{act}[$ the robots of S are not edge-activated. When a robot r is not edge activated, its respective values of the predicates $ExistsEdgeOnOppositeDirection()$ and $ExistsAdjacentEdge()$ are false. Thus the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of r are false, implying that it does not change the direction it considers (as no instructions permitting to change the direction are executed).

At time $t_s - 1$ the robots of S are necessarily edge-activated. Indeed, by definition of a long-lived tower during the Look phase of time $t_s - 1$ either the robots of S are not at a same node, or they are at a same node but have to cross different edges during the Move phase of time $t_s - 1$, otherwise the tower T does not start at time t_s. During the Look phase of time t_s the robots of S are on a same node. So some of the robots of S had moved during round $t_s - 1$. If some robots of S had not moved during the round $t_s - 1$, they had been edge-activated for the other robots of S to join them. And for the robots of S that had moved during time $t_s - 1$ they had crossed an adjacent edge to their location, so they were edge-activated.

From the two previous paragraphs and as the robots can change their global directions only during the Compute phase of each round, we can conclude that if the robots of S are considering opposed global directions at a time t this implies that the robots of S_1 and of S_2 were considering opposed global directions during the Move phase of the round $t_s - 1$.

S_1 and S_2 were thus both moving during time $t_s - 1$. Indeed, if both of the sets were not moving during the Move phase of round $t_s - 1$ then the two sets would not meet at time t_s. Moreover if only one set of robots was not moving during the Move phase of round $t_s - 1$ this implies that the adjacent edge e to the location of this set of robots in the direction considered by the robots of this set was missing during round $t_s - 1$. Assume without lost of generality that it is the set S_1 that was not moving during the Move phase of round $t_s - 1$. The two sets of robots are considering global opposite directions, thus if e is missing this implies that S_2 cannot join S_1 as to join S_1 at time t_s the robots of S_2 have to cross e during the Move phase of round $t_s - 1$. Thus the robots of S_1 and S_2 were moving during time $t_s - 1$. This implies that during the call to the function UPDATE of round $t_s - 1$ the variables $HasMovedPreviousEdgeActivation$ of the robots of S_1 and S_2 are set to true.

Moreover as seen previously the robots can change their global directions only during the Compute phase of each round when they are edge-activated. Thus when the robots wake up at time t_{act} their respective values of variables $HasMovedPreviousEdgeActivation$ are true, so their predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false. Thus the robots do not change the directions they were considering. The robots of S_1 still consider the clockwise direction while the robots of S_2 still consider the counter clockwise direction. As at time t_{act} the robots of S are edge-activated, the set S_1 and S_2 separate them during the Move phase of time t_{act}, which leads to a contradiction with the fact that the robots of S form a long-lived tower.

Thus the robots of S are considering a same global direction between the Look phase of time t_s and the Look phase of time t_{act}.

From time $\mathbf{t_{act}}$ to the Look phase of time $\mathbf{t_e}$ the robots of S consider a same global direction.

Call t_{act_bis} the first time in $]t_{act}, t_e]$ when the robots of S are edge-activated. At time t_e the robots of S are at a same node, however at time $t_e + 1$ they are either at different nodes, or they are at a same node but have crossed different edges during the Move phase of time t_e, thus at time t_e the robots of S are necessarily edge-activated. So t_{act_bis} exists.

First as at time t_{act} the robots of S are edge-activated, they have to consider a same global direction during the Move phase of time t_{act}, otherwise they separate them. As the directions considered by the robots can be changed only during the Compute phase of each round when they are edge-activated during times between $]t_{act}, t_{act\_bis}[$ the robots still consider a same global direction.

As seen previously the robots can change their global directions only during the Compute phase of each round when they are edge-activated thus during the Look phase of time t_{act_bis}, the robots still consider a same global direction. Thus in the case where t_{act_bis} is equal to t_e the property is proved. Moreover if t_{act_bis} is not equal to t_e, after the Compute phase of time t_{act_bis} the robots cannot consider opposed global directions, otherwise T is broken, which is a contradiction with the fact that T last from time t_s to time t_e. Then by recurrence we can prove that the robots possess a same global direction in $[t_s, t_e]$.

This prove the lemma. ☐

The following lemma is used to prove, in combination with Lemmas 4.2 and 4.3, the "tower breaking" mechanism since it proves that robots of a long-lived tower synchronously invoke their GiveDirection function after their first edge-activation.

Lemma 4.5. *For any long-lived tower $T = (S, [t_s, t_e])$, any (r_i, r_j) in S^2, and any t less or equal to t_e, we have $WeAreStuckInTheSameDirection()(r_i, t) = WeAreStuckInTheSameDirection()(r_j, t)$ if all robots of S have been edge-activated between t_s (included) and t (not included).*

Proof. Consider a long-lived tower $T = (S, [t_s, t_e])$.

Consider two of the robots r_i and r_j of S.

Call t_{act} the first time in $[t_s, t_e[$ where the robots of S are edge-activated. By definition of a long-lived tower, this time exists.

By contradiction, assume that there exists a time $t > t_{act}$ such that $WeAreStuckInTheSameDirection()(r_i, t) \neq WeAreStuckInTheSameDirection()(r_j, t)$.

The predicate $WeAreStuckInTheSameDirection()$ is a boolean, thus it has only two values: true or false. As by assumption $WeAreStuckInTheSameDirection()(r_i, t) \neq WeAreStuckInTheSameDirection()(r_j, t)$, this implies that this predicate is true for one of the robots among r_i, r_j, while it is false for the other one.

Without lost of generality assume that $WeAreStuckInTheSameDirection()(r_i, t)$ is true while $WeAreStuckInTheSameDirection()(r_j, t)$ is false.

By definition of a long-lived tower and according to lemma 4.4, we know that from time t_s to the end of the Look phase of time t_e all the robots of S are on a same node and consider a same global direction. This implies that the values of the predicates $NumberOfRobotsOnNode()$, $ExistsEdgeOnCurrentDirection()$, $ExistsEdgeOnOppositeDirection()$ and $ExistsAdjacentEdge()$ of all the robots of S are identical from time t_s to the end of the Look phase of time t_e.

Consider a time $t' \in [t_s, t[$ at which the robots of S are edge-activated. t' exists as t is strictly greater than t_{act}. As at time t' the robots of S are edge-activated, during the call to the function UPDATE of the round t', the robots of S update their variables $NumberRobotsPreviousEdgeActivation$ and $HasMovedPreviousEdgeActivation$, respectively with the values of their predicates $NumberOfRobotsOnNode()$ and $ExistsEdgeOnCurrentDirection()$. Moreover as the robots of S possess the same values of predicates from time t_s to the end of the Look phase of round t_e, after the Compute phase of the round t', all the robots of S possess the same values of variables $NumberRobotsPreviousEdgeActivation$ and $HasMovedPreviousEdgeActivation$.

For each time t" in $]t', t_e[$ when the robots wake up, if they are not edge-activated, then no robots change the values of their variables (as the function UPDATE is only executed when the robots are edge-activated). Moreover for each time t" in $]t', t_e[$ when the robots of S wake up, if they are edge-activated, then they update their values of variables with the values of their predicates. However, as seen previously all the robots of S possess the same values of predicates from time t_s to the end of the Look phase of time t_e. Therefore for all time in $]t', t_e[$ all the robots of S possess the same values of variables $NumberRobotsPreviousEdgeActivation$ and $HasMovedPreviousEdgeActivation$. Moreover as the variables can change only during the Compute phase of each round, the variables of the robots of S are also identical during the Look phase of round t_e.

Besides, the predicate $WeAreStuckInTheSameDirection()$ depends only on the values of the variables $NumberRobotsPreviousEdgeActivation$, and $HasMovedPreviousEdgeActivation$, and on the values of the predicates $NumberOfRobotsOnNode()$, $ExistsEdgeOnCurrentDirection()$, and $ExistsEdgeOnOppositeDirection()$. As seen previously all these values are identical for all the robots of S from time $t'+1$ until the end of the Look phase of time t_e, thus for all $t" \in]t', t_e]$, we have $WeAreStuckInTheSameDirection()(r_i, t") = WeAreStuckInTheSameDirection()(r_j, t")$. As t is included in $]t', t_e]$ there is a contradiction.

This proves the lemma. □

Lemma 4.6. *If there exists an eventual missing edge, then all long-lived towers have a finite duration.*

Proof. Assume that there exists a time $t_{missing} \in \tau$ and exists an edge e of $\mathcal{G}$ such that for all t greater or equal to $t_{missing}$, e is missing.

Consider the execution after time $t_{missing}$.

Call u and v the two adjacent nodes of e, such that if e was present in $\mathcal{G}$ a robot on node u would have to cross e in the clockwise direction to be located on v. As $\mathcal{G}$ is a ring each node possesses two adjacent edges. Call e' the other adjacent edge of u.

By contradiction assume that there exists a long-lived tower $T = (S, \theta)$ such that $\theta = [t_s, +\infty[$. Exactly 3 robots are executing our algorithm, thus here $|S|$ is either equals to 2 or 3.

First we prove that in the case where t_e is equal to $+\infty$ then it exists a robot of S such that its predicate $WeAreStuckInTheSameDirection()$ is infinitely often true.

By contradiction assume that for each robot r_i of S, it exists a time t_i in θ such that for all time t greater or equal to t_i its predicate $WeAreStuckInTheSameDirection()$ is false. Set $t_{false} = max\{t_{missing}, \{t_i\}_{r_i \in S}\}$ ($t_{false} \in \theta$) the maximum of all the times greater than $t_{missing}$ after which the predicates $WeAreStuckInTheSameDirection()$ of all the robots of S are false.

We recall that by lemma 4.4 all the robots of S are considering a same global direction from time t_s to the Look phase of time t_e.

Case 1: $|\mathbf{S}| = 3$.

Call $t_{act} \geq t_{false}$ ($t_{act} \in \theta$), the first time where the robots of S are edge-activated. As $\mathcal{G}$ belongs to the class of connected-over-time rings, at least one adjacent edge to each node appears infinitely often, thus t_{act} exists. From time t_s to $+\infty$ the three robots of the system form a 3-long-lived tower. They are thus on a same node from time t_s. Therefore from time t_s the predicates $NumberOfRobotsOnNode()$ of the robots of S are equal to 3. During the call to the function UPDATE of round t_{act}, as the robots of S are edge-activated they update their variables $NumberRobotsPreviousEdgeActivation$ with the value of their predicates $NumberOfRobotsOnNode()$ which is equal to 3. The variables $NumberRobotsPrevious$-$EdgeActivation$ of the robots of S are updated when the robots are edge-activated. Like for time t_{act} when the robots are edge-activated their variables $NumberRobotsPreviousEdge$-$Activation$ are filled with the value 3. Moreover when the robots are not edge-activated they conserve the same value than the one they had the last time they were edge-activated. This implies that from time $t_{act}+1$ the values of the variables $NumberRobotsPreviousEdgeActivation$ of the robots of S are equal to 3. Therefore from time $t_{act}+1$ the predicates $IWasStuckOn$-$MyNodeAndNowWeAreMoreRobots()$ of the robots of S are false, as the condition $Number$-$OfRobotsOnNode() > NumberRobotsPreviousEdgeActivation$ is false. Moreover by assumption we know that the predicates $WeAreStuckInTheSameDirection()$ of all the robots of S are false from time t_{false} to time t_e. This implies that the predicates $WeAreStuckInThe$-$SameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the robots of S are false from time $t_{act}+1$. So from time $t_{act}+1$ the robots of S are always considering the same global direction.

Without lost of generality assume that from time $t_{act}+1$ the robots of S are considering the clockwise direction. By definition of a connected-over-time ring, all the edges of $\mathcal{G}$ except e are infinitely often present in the system. So there exists infinitely often an edge in the clockwise direction to the current location of the 3-long-lived tower. Therefore as the robots of S consider the clockwise direction from time $t_{act}+1$ they reach the node u in finite time. However e is missing forever, thus the robots of T are not able to traverse e. Call t_{first} the first time the robots of S are edge-activated on node u. As said previously e' is infinitely often present in the system, so t_{first} exists. During the call to the function UPDATE of round t_{first}, the values of the variables $HasMovedPreviousEdgeActivation$ of the three robots are set to false. The next time the robots of T are edge-activated, e' is present while e is missing and their variables $HasMovedPreviousEdgeActivation$ are false, so the predicates $WeAreStuckInTheSameDirection()$ of the robots of S are true. This leads to a contradiction with the fact that the predicates $WeAreStuckInTheSameDirection()$ of all the robots of S are false from time t_{false}.

Case 2: $|\mathbf{S}| = 2$.

Assume without lost of generality that the 2-long-lived tower is formed of the robots r_1 and r_2.

While forming T the robots of S can meet r_3 or not.

Case 2.1: The 2-long-lived tower does not meet $\mathbf{r_3}$.

By similar arguments than the one used for the case 1 we prove that there is a contradiction.

Case 2.2: The 2-long-lived tower meets r_3.

At each instant time, each robot considers a direction. There is no state in our algorithm where a robot can consider no direction. During the Move phase of time i if a robot r is located on a node where an adjacent edge is present in the same direction than the one considered by r, then r moves during the Move phase of time i. Moreover there are only two possible directions (the clockwise and the counter clockwise direction). Thus if at a time t' strictly greater than t_{false} the robots of S meet r_3 it is either because the two entities (the tower and r_3) were moving during the Move phase of time $t'-1$ while considering two opposed global directions or because the two entities were considering the same global direction and that one of the entity could not move (an edge was missing in its direction) during the Move phase of the round $t'-1$.

Call t'_{act} the first round after t' where the robots are edge-activated.

At time $t'-1$ the robots of S are necessarily edge-activated to be able to meet at time t'.

Case 2.2.1: The meeting occurs because the two entities were moving in two opposite global directions.

In this case during the call to the function UPDATE of time $t'-1$ the variables $HasMovedPreviousEdgeActivation$ of the three robots are set to true. The variables are only updated during the Compute phase of rounds where the robots are edge-activated. Thus from time $t'-1$ to the Look phase of time t'_{act} the variables $HasMovedPreviousEdgeActivation$ are still true. So at time t'_{act} the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the three robots are false, as their variables $HasMovedPreviousEdgeActivation$ are true. Thus the two entities conserve the global direction they were considering during the Move phase of round $t'-1$. And so during the Move of the round t'_{act} the two entities are considering different global directions.

Case 2.2.2: The meeting occurs because an entity was moving and the other was stuck.

In this case, during the Compute phase of time $t'-1$ the variable $HasMovedPreviousEdgeActivation$ of each robot of the entity that has moved is set to true, while the variable $HasMovedPreviousEdgeActivation$ of each robot of the entity that has not moved is set to false. The variables are only updated during the Compute phase of rounds where the robots are edge-activated. Thus at time t'_{act} each robot of the entity that has moved during the Move phase of time $t'-1$ has its predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ to false, as its variable $HasMovedPreviousEdgeActivation$ is true. However the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of each of the robots of the other entity is true. Thus each robot of this last entity considers a direction opposed to the one considered during the Move phase of the round $t'-1$. So during the Move of the round t'_{act} the two entities are considering different global directions.

Thus in the two cases of meeting during the Move phase of time t'_{act} the two entities are considering two different global directions. Thus they separate during the Move phase of round t'_{act} as an edge exits at this time. After time t'_{act}, as long as r_3 is alone on its node it does not change the direction it considers as its predicates

$WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false. As long as the robots of S do not meet r_3, their predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false. Moreover, as by assumption, from time t_{false} the predicates $WeAreStuckInTheSameDirection()$ of the robots of S are false, this implies that the robots of S keep consider the same global direction as long as they do not meet again r_3. Besides as e is missing after time $t_{missing}$ and as the robots of S and r_3 have already meet each other, and as they separate themselves going in different global directions, and as they keep consider their respective directions as long as they do not meet again, there is no way for a meeting between this two entities to happen again.

Thus here we are in a case similar to the one described case 2.1. Therefore by using similar arguments we can conclude that this case leads to a contradiction.

These arguments permit to state that in the case where there exists an eventual missing edge and that t_e is equal to $+\infty$ then it exists a robot of S such that its predicate $WeAreStuckInTheSameDirection()$ is infinitely often true.

As t_e is equal to $+\infty$ and that all the edges except e are infinitely often present in $\mathcal{E}$, the robots of S are infinitely often edge-activated. According to lemma 4.5 after the first time where the robots of S are edge-activated, they all consider the same value for their predicates $WeAreStuckInTheSameDirection()$. Call t_{true} the first time the robots of S are edge-activated. After t_{true} all the robots of S have their predicates $WeAreStuckInTheSameDirection()$ infinitely often true. Thus after time t_{true} all the robots of S call the function GiveDirection infinitely often and at the same instant time.

Thus for the robots to keep forming T, if the robots have the same chirality, they need to consider the same value of bit each time the function GiveDirection is called, and if the robots have not the same chirality they need to consider different values of bit each time the function GiveDirection is called. However according to lemma 4.2 and to lemma 4.3 this cannot happen infinitely often. Thus there exists a time where the robots of T when executing the function GiveDirection consider bits that lead them to consider different global direction. Thus the tower T is broken. This leads to a contradiction with the fact that θ equals $]t_s, +\infty[$.

In conclusion we can say that if there exists an eventual missing edge, then all long-lived towers have a finite duration. □

Lemma 4.7. *Every execution containing only configurations without any long-lived tower cannot reach a configuration with a 3-short-lived tower.*

Proof. Consider an execution $\mathcal{E}$ composed of configurations that do not contain long-lived towers.

We want to prove that for all t in τ it is not possible to have a configuration containing a 3-short-lived tower in $\mathcal{E}$.

By contradiction assume that there exists a configuration containing a 3-short-lived tower in $\mathcal{E}$.

For a 3-short-lived tower to be formed, the three robots must be on a same node at the same time. Assume that the three robots meet on a node v at time t_{meet} for the first time.

Call u and w the two adjacent nodes of v in $\mathcal{G}$. To go on node v from node u, a robot needs to cross an edge in the clockwise direction. And to go on node v from node w, a robot needs to cross an edge in the counter clockwise direction.

Every robot performing our algorithm consider a direction at each instant time. This implies that if a robot is on a node x considering a direction and that there exists an adjacent edge to x in the same direction than the one considered by the robot then the robot crosses this edge.

Moreover as $\mathcal{G}$ is based on a ring, only two edges are adjacent to each node.

Besides a robot can cross at most one edge per round.

These three arguments prove that for a robot to be on node v during the Look phase of time t_{meet} it can:

- Be on v during the Look phase of time $t_{meet} - 1$. In this case it cannot be able to move during the Move phase of round $t_{meet} - 1$, otherwise it is not on node v during the Look phase of time t_{meet}. The only way for the robot to be on node v during the Look phase of time $t_{meet} - 1$ and to be again on this node during the Look of time t_{meet} is to consider during the Move phase of time $t_{meet} - 1$ a global direction such that there is no adjacent edge to v in this global direction at time $t_{meet} - 1$.

- Be on node u during the Look phase of time $t_{meet} - 1$ considering the clockwise direction during the Move phase of round $t_{meet} - 1$. The edge linking u and v must be present in $\mathcal{E}$ at time $t_{meet} - 1$.

- Be on node w during the Look phase of time $t_{meet} - 1$ considering the counter clockwise direction during the Move phase of round $t_{meet} - 1$. The edge linking w and v must be present in $\mathcal{E}$ at time $t_{meet} - 1$.

Case 1: A robot is on node v during the Look phase of time $\mathbf{t_{meet} - 1}$.

Without lost of generality assume that this is the robot r_1 that is on node v during the Look phase of time $t_{meet} - 1$. As said previously, if r_1 is still on node v during the Look phase of time t_{meet} this implies that it considers a global direction such that there is no adjacent edge to v in that direction at time $t_{meet} - 1$. Without lost of generality assume that r_1 is considering the counter clockwise direction after the Compute phase of round $t_{meet} - 1$. Thus the edge linking v to u is missing during round $t_{meet} - 1$. Thus if a 3-short-lived tower is formed at time t_{meet} the two other robots must be either on node v or on node w during the Look phase of time $t_{meet} - 1$ and must consider the counter clockwise direction during the Move phase of round $t_{meet} - 1$.

It is not possible for both r_2 and r_3 to be on node v during the Look phase of time $t_{meet} - 1$ as by assumption the 3-short-lived tower is formed for the first time at time t_{meet}. So either one of the robots among r_2 and r_3 is on node v or both r_2 and r_3 are on node w during the Look phase of time $t_{meet} - 1$.

Case 1.1: During the Look phase of time $\mathbf{t_{meet} - 1}$ a robot is on node v with $\mathbf{r_1}$, while an other robot is on node w.

Assume without lost of generality that it is r_2 that is with r_1 on node v during the Look phase of time $t_{meet} - 1$. Therefore it is r_3 that is on node w during the Look phase of round $t_{meet} - 1$. For the 3-short-lived tower to be formed at time t_{meet}, r_3 must consider the counter clockwise direction and the edge linking w to v must be present at time $t_{meet} - 1$. Like r_1, r_2 must consider the counter clockwise direction during the Move phase of time $t_{meet} - 1$, otherwise it moves to node w. Thus r_1 and r_2 are on node v

during the Look phase of time $t_{meet}-1$ and are still on node v during the Look phase of time t_{meet}. However as the edge linking w to v is present at time $t_{meet}-1$, r_1 and r_2 are edge-activated at time $t_{meet}-1$. They are thus involved in a 2-long-lived tower (see definition 4.2), which leads to a contradiction with the fact that there is no configuration containing a long-lived tower in $\mathcal{E}$.

Case 1.2: $\mathbf{r_2}$ and $\mathbf{r_3}$ are on node w.

For the 3-short-lived tower to be formed at time t_{meet} on node v, both r_2 and r_3 must consider the counter clockwise direction during the Move phase of time $t_{meet}-1$. Moreover the edge linking w to v must be present at time $t_{meet}-1$. Thus r_2 and r_3 are on a same node during the Look phase of time $t_{meet}-1$ and they are also on the same node during the Look phase of time t_{meet}. Besides they are edge-activated at time $t_{meet}-1$. This implies that r_2 and r_3 are involved in a 2-long-lived tower which leads to a contradiction with the fact that there is no configuration containing a long-lived tower in $\mathcal{E}$.

Case 2: A robot is on node w during the Look phase of time $\mathbf{t_{meet}-1}$.

Without lost of generality assume that this is r_1 that is on node w during the Look phase of time $t_{meet}-1$. As seen previously no robot can be on node v otherwise this leads to a contradiction. Moreover the three robots cannot be one the same node at time $t_{meet}-1$ otherwise we have a contradiction with the fact that t_{meet} corresponds to the first time in $\mathcal{E}$ where a 3-short-lived tower is formed. Thus during the Look phase of time $t_{meet}-1$ either one of the robots among r_2 and r_3 is on node u while the other one is on node w or both are on node u.

Case 2.1: During the Look phase of time $\mathbf{t_{meet}-1}$ a robot is on node w with $\mathbf{r_1}$, while an other robot is on node u.

Assume that during the Look phase of time $t_{meet}-1$ this is r_2 that is on node w, while r_3 is on node u. As the 3-short-lived tower must be formed at time t_{meet} on node v the edge linking w to v must be present at time $t_{meet}-1$ and the two robots r_1 and r_2 must consider the counter clockwise direction during the Move phase of time $t_{meet}-1$. Thus here r_1 and r_2 are forming a 2-long-lived tower, which leads to a contradiction.

Case 2.2: $\mathbf{r_2}$ and $\mathbf{r_3}$ are on node u during the Look phase of time $\mathbf{t_{meet}-1}$.

Similarly in the case where r_3 and r_2 are both on node u, during the Look phase of time $t_{meet}-1$ the edge linking u to v must be present and r_2 and r_3 must consider the clockwise direction. Thus r_2 and r_3 are forming a 2-long-lived tower, which leads to a contradiction.

Case 3: A robot is on node u during the Look phase of time $\mathbf{t_{meet}-1}$.

This case has been treated while treating the case 2.

All the possible scenarios lead to contradictions. Thus we can conclude that every execution starting from a configuration without a long-lived tower cannot contain a 3-short-lived tower. □

Lemma 4.8. *Every execution starting from a configuration without a 3-long-lived tower cannot reach a configuration with a 3-long-lived tower.*

Proof. Consider an execution $\mathcal{E}$ starting from a configuration $\mathcal{C}$ which does not contain a 3-long-lived tower. We recall that exactly 3 robots execute our algorithm. Thus if a configuration contains a 3-long-lived tower the three robots of the system are involved in it, and only one 3-long-lived tower exists per configuration.

We want to prove that for all t in τ it is not possible to have a configuration containing a 3-long-lived tower in $\mathcal{E}$.

By contradiction assume that it is possible to have a configuration containing a 3-long-lived tower in $\mathcal{E}$. Call $\mathcal{C}'$ the first configuration of $\mathcal{E}$ containing a 3-long-lived tower. Name this 3-long-lived tower T. Assume that T starts at time t in τ. Call t_{act} ($t_{act} \geq t$) the first time the 3 robots of T are edge-activated. By definition of a long-lived tower, the time t_{act} exists.

For the robots to form a 3-long-lived tower they must be on a same node from the Look phase of time t until at least the Look phase of time $t_{act} + 1$ and they must consider a same global direction during the Move phase of round t_{act} (See definition 4.2). Otherwise the robots break the tower during the Move phase of round t_{act} and this would lead to a contradiction with the fact that T is a 3-long-lived tower.

By lemma 4.7 we know that if there is no long-lived tower in $\mathcal{E}$ it is not possible to have 3 robots on a same node at the same time. This implies that a meeting between the three robots can only happens between a single robot and a 2-long-lived tower. Moreover the meeting between this two entities can happen either because the two entities (the 2-long-lived tower and the single robot) were moving during the phase $t-1$ while considering two global opposite directions or because the two entities were considering the same global direction and that one of the entity was not able to move during the round $t-1$.

Based on the arguments of the case 2.2 of the proof of lemma 4.6, we know that after the Compute phase of round t_{act} the two entities are considering two global opposite directions. This leads to a contradiction with the fact that T is a 3-long-lived tower.

Thus we can conclude that every execution starting from a configuration without a 3-long-lived tower cannot contain a 3-long-lived tower. □

Lemma 4.9. *Let γ be a configuration such that all but one robots consider the same global direction. Then starting from γ, no execution without any long-lived towers can reach a configuration where all robots consider the same global direction.*

Proof. Consider that $\mathcal{E}$ does not contain long-lived tower.

Consider that $\mathcal{E}$ starts from a configuration $\mathcal{C}$ where two robots are considering a global direction opposed to the one considered by the third robot of the system.

We want to prove that $\mathcal{E}$ cannot contain a configuration in which the three robots of the system are considering the same global direction.

We process by contradiction. Call $\mathcal{C}'$ the first configuration of $\mathcal{E}$ such that the three robots of the system are considering a same global direction. Assume that $\mathcal{C}'$ happens at time t. This implies that the three robots possess the same global direction during the Look phase of time t.

During the Look phase of time $t-1$ the robots are still considering different global directions, otherwise there is a contradiction with the fact that $\mathcal{C}'$ is the first configuration of $\mathcal{E}$ where the three robots are considering the same global direction.

Call $\mathcal{C}$" the configuration at time $t-1$.

By assumption there are no long-lived towers in $\mathcal{E}$, and moreover by lemma 4.7 we know that in an execution where there are no long-lived towers when a meeting happens it does not involved three robots. Thus the configurations of $\mathcal{E}$ contain either three isolated robots or one 2-short-lived tower and one isolated robot.

At least one robot must change the global direction it considers during the Compute phase of round $t-1$ for the three robots of the system to consider the same global direction during the Look phase of time t. Thus in $\mathcal{C}$" some of the three robots are not isolated. Indeed, if the three robots are isolated then their predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false and thus the robots do not change their directions (and therefore the three robots do not consider the same global direction during the Look phase of time t). Thus there is necessarily a 2-short-lived tower in $\mathcal{C}$". Assume without lost of generality that this 2-short-lived tower is composed of the robots r_1 and r_2.

By definition of a 2-short-lived tower once the 2 robots involved in it are edge-activated, they separate them. As the robots performing our algorithm consider at each instant time a direction. If the two robots separate themselves when they are edge-activated this implies that they necessarily consider two different global directions.

Thus during the Move phase of time $t-1$ the robots r_1 and r_2 are considering two global opposite directions. As the robots executing algorithm 3 can change their directions only during the Compute phase of a round, this implies that during the Look phase of time t, r_1 and r_2 are considering two different global directions. Therefore there is a contradiction with the fact that the three robots of the system consider the same global direction in $\mathcal{C}'$.

This proves the lemma. ☐

Lemma 4.10. *Consider an execution containing no 3-long-lived towers. If a 2-long-lived tower $T=(S,[t_s,t_e])$ is located at a node u at round t_e, then the robot that does not belong to S cannot be located at node u during the Look phase of round t_e. Moreover during the Look phase of round t_e+1, one robot of S located at u considers a global direction opposite to the one considered by the other robot of S (which is not on u).*

Proof. Consider an execution without 3-long-lived towers.

Consider a 2-long-lived tower $T=(S,[t_s,t_e])$. Assume that r_1 and r_2 are composing T.

By definition of a long-lived tower, we know that r_1 and r_2 are at least one time edge-activated between $[t_s,t_e[$. Call t_{act} the first time in $[t_s,t_e[$ when the robots of S are edge-activated. According to lemma 4.5 from time $t_{act}+1$ to the end of the Look phase of round t_e all the robots of S possess the same value for their predicates $WeAreStuckInTheSameDirection()$.

Note that at time t_e the robots of T are necessarily edge-activated. Indeed, if this is not the case this implies that at time t_e there is no adjacent edge to the location where T is, and thus the robots of S cannot move during the Move phase of round t_e. Thus they cannot break the tower, which leads to a contradiction with the fact that T is broken at time t_e.

During the Compute phase of time $\mathbf{t_e}$ the robots $\mathbf{r_1}$ and $\mathbf{r_2}$ are executing the function GiveDirection.

To prove this statement we process by contradiction. If the two robots are not executing the function GiveDirection at time t_e this implies that their predicates $WeAreStuckInThe$-$SameDirection()$ are false. Thus at the end of the Look phase of time t_e either the two robots of S have their predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNode$-$AndNowWeAreMoreRobots()$ false, or they have both their predicates $IWasStuckOnMy$-$NodeAndNowWeAreMoreRobots()$ to true, or one of the robot of S has its predicate $IWas$-

$StuckOnMyNodeAndNowWeAreMoreRobots()$ to true while the other robot of S has its predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ to false.

Case 1: The predicates WeAreStuckInTheSameDirection() and IWasStuckOnMyNodeAndNowWeAreMoreRobots() of the robots of S are false at the end of the Look phase of time t_e.

In this case, the two robots keep consider the same global direction during the Move phase of time t_e (as no instructions implying a change of direction is executed). So there is a contradiction with the fact that at time t_e T is broken.

Case 2: The predicates IWasStuckOnMyNodeAndNowWeAreMoreRobots() of the robots of S are true at the end of the Look phase of time t_e.

In this case, the two robots change the direction they consider. However r_1 and r_2 are forming a 2-long-lived tower thus by lemma 4.4 this implies that these two robots are considering the same global direction from time t_s to the end of the Look phase of time t_e. Thus during the Look phase of time t_e, r_1 and r_2 are considering the same global direction. So if the two robots change their global directions during the Compute phase of time t_e, they still consider a same global direction during the Move phase of time t_e. So the robots are still involved in a 2-long-lived tower during the Look phase of time $t_e + 1$ which is a contradiction with the fact that at time t_e T is broken.

Case 3: At the end of the Look phase of time t_e the predicate IWasStuckOnMyNodeAndNowWeAreMoreRobots() of one of the robots of S is true, while it is false for the other robot of S.

This case cannot happen. Indeed, during the call to the function UPDATE of time t_{act} the robots r_1 and r_2 have their values of variables $NumberRobotsPreviousEdgeActivation$ and $HasMovedPreviousEdgeActivation$ that are respectively filled with the values of their predicates $NumberOfRobotsOnNode()$ and $ExistsEdgeOnCurrentDirection()$. r_1 and r_2 are forming a 2-long-lived tower therefore they are on the same node and are considering a same global direction from time t_s to the end of the Look phase of time t_e thus their respective values of their respective predicates are equal. This is in particular true for their predicates $ExistsAdjacentEdge()$. Moreover when the robots are not edge-activated their variables are not updated. This implies that the next time the two robots are edge-activated, call this time t_{act_bis}, they wake up with the same values of variables. Moreover during the call of the function UPDATE of t_{act_bis} the values of the variables of r_1 and r_2 are filled with the same values (for the same arguments than the one used at time t_{act}). By recurrence we can conclude that from the call to the function UPDATE at time t_{act} to the Look phase of time t_e the robots of S possess the same values of variables.

Thus here we know that from time t_s to the end of the Look phase of time t_e the values of the predicates $ExistsAdjacentEdge()$ of the robots of S are identical, and from time $t_{act} + 1$ to the end of the Look phase of time t_e the respective values of the variables $HasMovedPreviousEdgeActivation$ and $NumberRobotsPreviousEdgeActivation$ of the robots of S are also identical.

As during the Look phase of time t_e the robots r_1 and r_2 are on a same node, the values of their predicate $NumberOfRobotsOnNode()$ are equal. Thus during the Look phase of time t_e it is not possible that one of the robots of S considers the condition "$NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation$" equals to true while the other one consider it equals to false.

From the two previous paragraph we can conclude that the two robots of S have necessarily the same value of predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$.

In conclusion we know that at time t_e r_1 and r_2 are executing the function *GiveDriection*. As during the Look phase of time $t_e + 1$ r_1 and r_2 are not forming a 2-long-lived tower anymore, this implies that after executing the function GiveDirection the two robots consider two different global directions. Moreover the robots are on a node u during the Look phase of time t_e. To execute the function GiveDirection the condition $\neg ExistsEdgeOnCurrentDirection() \wedge ExistsEdgeOnOppositeDirection()$ must be true. Thus during time t_e one of the adjacent edge of u is missing while the other one is present. As after the execution of the function GiveDirection the two robots are considering two global opposite directions, one robot is able to move during the Move phase of round t_e while the other one cannot. Thus one of the robot of S is still on node u during the Look phase of time t_{e+1} and it considers a global direction opposite to the one considered by the other robot of S.

$\mathbf{r_3}$ cannot be on node u during the Look phase of time $\mathbf{t_e}$.

To prove this statement we also process by contradiction. Assume that r_3 is on node u during the Look phase of time t_e. This implies that the three robots meet at a certain time between $[t_s, t_e[$. Moreover note that the three robots must be edge-activated to be able to meet. Call t_{e_act} the last time of $[t_s, t_e[$ at which the robots r_1 and r_2 are edge-activated. By definition of a long-lived tower, t_{e_act} exists.

There are no 3-long-lived towers in $\mathcal{E}$. Therefore either the three robots meet at time $t_{e\_act} + 1$ and were not on a same node during the Look phase of time t_{e_act}, or they meet at time $t_{e\_act} + 1$ and were on a same node at time t_{e_act} but were considering different global directions during the Move phase of time t_{e_act}, otherwise there is a contradiction with the fact that there is no 3-long-lived towers in $\mathcal{E}$.

Case 2.1: The three robots meet at time $\mathbf{t_{e\_act} + 1}$ and were not on a same node during the Look phase of time $\mathbf{t_{e\_act}}$.

During the Look phase of time t_{e_act} the robot r_3 is not on the same node as the robots r_1 and r_2. This implies that the values of the predicates $NumberOfRobotsOnNode()$ of r_1 and r_2 are equal to 2. At time t_{e_act} the robots of S are edge-activated thus during the call to the function Update at round t_{e_act} the values of their respective variables $NumberRobotsPreviousEdgeActivation$ are updated with the values of their predicates $NumberOfRobotsOnNode()$, thus are updated to 2. Moreover the values of the variables are only updated during the Compute phase of rounds where the robots are edge-activated. By definition of t_{e_act} the next time after t_{e_act} when the robots are edge-activated is time t_e. Thus at the end of the Look phase of time t_e the variables $NumberRobotsPreviousEdgeActivation$ of the robots of S are still equal to 2. Besides from time $t_{e\_act} + 1$ to the Look phase of time t_e the three robots are not edge-activated thus they stay on the same node. Thus during the Look phase of time t_e the robots r_1 and r_2 wake up with the r_3 on the same node as them, thus their predicates $NumberOfRobotsOnNode()$ have thus a value of 3. As proved previously T is broken at time t_e because r_1 and r_2 execute the function GiveDirection. However they can execute this function only if the condition "$NumberOfRobotsOnNode() = NumberRobotsPreviousEdgeActivation$" is true. During the Look phase of time t_e this condition is not true, thus here r_1 and r_2 cannot execute the function GiveDirection during the Compute phase of time t_e and thus they cannot separate them during the Move phase of time t_e, which is a contradiction with the fact that the tower T breaks at time t_e.

Case 2.2: The three robots meet at time $\mathbf{t_{e\_act}+1}$ and were on a same node during the Look phase of time $\mathbf{t_{e\_act}}$ but considering opposite global directions during the Move phase of time $\mathbf{t_{e\_act}}$.

At time t_{e_act} the three robots are on a same node, however during the Move phase of time t_{e_act} the robots consider different global directions. As they are again on a same node at time $t_{e\_act}+1$ and that they are edge-activated at time t_{e_act}, this implies that the two adjacent edges to the location where the three robots are during the Look phase of time t_{e_act} are present. Therefore during the Move phase of time t_{e_act} the robots are able to move, and thus during the call to the function Update of round t_{e_act} the variables $HasMovedPreviousEdgeActivation$ of the three robots are set to true. Moreover the values of the variables are only updated during the Compute phase of rounds where the robots are edge-activated. By definition of t_{e_act} the next time after t_{e_act} when the robots are edge-activated is time t_e. Thus during the Look phase of time t_e the robots of S have their variables $HasMovedPreviousEdgeActivation$ to true. As proved previously T is broken because r_1 and r_2 execute the function GiveDirection. However they can execute this function only if their variables $HasMovedPreviousEdgeActivation$ are false. During the Look phase of time t_e the variables $HasMovedPreviousEdgeActivation$ of r_1 and r_2 are not false, thus they cannot execute the function GiveDirection during the Compute phase of time t_e and thus they cannot separate them during the Move phase of time t_e, which is a contradiction with the fact that the tower T breaks at time t_e.

Therefore r_3 cannot be on node u during the Look phase of time t_e.

This prove the lemma. □

The next two lemmas show that the whole ring is visited between two consecutive 2-long-lived towers if these two towers satisfy some properties. They are used in the proof of the "sentinels"/"visitor" scheme.

Lemma 4.11. *Consider an execution $\mathcal{E}$ without any 3-long-lived tower but containing a 2-long-lived tower $T = (S, [t_s, t_e])$. If there exists another 2-long-lived tower $T' = (S', [t'_s, t'_e])$ after T in $\mathcal{E}$ and if T' is the first 2-long-lived tower in $\mathcal{E}$ such that $t'_s > t_e + 1$, then all the edges of $\mathcal{G}$ have been crossed by at least one robot between time t_e and time t'_s.*

Proof. Consider an execution $\mathcal{E}$ starting from a configuration without a 3-long-lived tower. By lemma 4.8 we know that it is not possible to have a 3-long-lived tower during the whole execution.

Consider two 2-long-lived towers T and T' ($T \neq T'$), such that T' is the next 2-long-lived tower after T in $\mathcal{E}$. Assume that T starts at time t_s and ends at time t_e, and T' starts at time t'_s and ends at time t'_e, with $t'_s > t_e + 1$.

We want to prove that during $[t_e, t'_s]$ each edge of $\mathcal{G}$ is crossed by at least one robot.

By contradiction assume that there exists an edge e that is not crossed by no robot during time $[t_e, t'_s]$.

Assume that T is composed of the robots r_1 and r_2. Assume that during the Look phase of time t_e the tower T was located on a node u_0, thus by lemma 4.10, during the Look phase of time $t_e + 1$ one robot among r_1 and r_2 is on u_0 considering a global direction opposed to the one considered by the other robot of S.

Without lost of generality, assume that it is r_1 that is on node u_0 at during the Look phase of time $t_e + 1$ and that it considers the counter clockwise direction while r_2 considers the clockwise direction.

Call u_1 the adjacent node of u_0 in the clockwise direction.

As during the Look phase of time $t_e + 1$ r_2 considers the clockwise direction, and as the variables are only updated during the Compute phase of rounds, this implies that during the Move phase of round t_e, r_2 was considering the clockwise direction. Moreover as during the Look phase of time $t_e + 1$ only one robot among the robots of S is present on u_0, this implies that r_2 succeeds to move during the Move phase of round t_e. Thus during the Look phase of time $t_e + 1$ r_2 is on node u_1. Note that the edge linking u_0 to u_1 has been crossed by r_2 during the Move phase of round t_e.

Note $\{u_0, u_1, \ldots, u_k, \ldots, u_{n-1}\}$ the nodes of $\mathcal{G}$ in the clockwise direction from the node u_0, with k an integer such that $2 \leq k \leq n - 2$.

From the two previous paragraphs we can conclude that the edge e permits to go from a node u_i to a node $u_{(i+1) \pmod n}$ considering the clockwise direction, with i an integer such that $1 \leq i \leq (n - 1)$.

By lemma 4.10 we know that r_3 is not on node u_0 during the Look phase of time t_e.

During the Look phase of time $t_e + 1$ the robot r_3 considers either the clockwise or the counter clockwise direction.

Case 1: During the Look phase of time $\mathbf{t_e + 1}$ $\mathbf{r_3}$ considers the clockwise direction.

As the variables are only updated during the Compute phase of rounds, during the Move phase of round t_e r_3 considers the clockwise direction.

For the same reason during the Move phase of time t_e r_1 considers the counter clockwise direction. Moreover during the Look phase of time t_e, r_1 is on node u_0. However during the Look phase of round $t_e + 1$ r_1 is still on node u_0, this implies that the edge linking u_{n-1} to u_0 is missing at time t_e.

As during the Look phase of time t_e r_3 is not on node u_0, and as r_3 considers the clockwise direction during the Move phase of time t_e, and as the edge linking u_{n-1} to u_0 is missing at time t_e, then r_3 cannot be on u_O during the Look phase of time $t_e + 1$.

During the Look phase of time t_{e+1}, r_3 is thus on a node among $\{u_1, \ldots, u_k, \ldots, u_{n-1}\}$ and r_1 is alone on u_0.

As e is not crossed by no robot, and that during the Look phase of time $t_e + 1$ r_3 is considering the clockwise direction we consider two cases. First we consider the case where r_3 is on a node among $\{u_1, \ldots, u_i$, and secondly we consider the case where r_3 is on a node among $\{u_{(i+1) \pmod n}, \ldots, u_{n-1}\}$.

By the definition of a long-lived tower and according to lemma 4.4, for two robots to be involved in a 2-long-lived tower they need to wake up on a same node at a certain round i

and consider the same global direction from time i until the time when the tower breaks.

To have two robots on the same node, a meeting must happen.

Case 1.1: During the Look phase of time $\mathbf{t_e + 1}$ $\mathbf{r_3}$ is on a node among $\{\mathbf{u_1, \dots, u_i}\}$. As long as the robots are alone on their respective nodes, their predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false (as the condition "$NumberOfRobotsOnNode() > 1$" and the condition "$NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation$" cannot be true), thus they cannot change their direction. Thus as long as there is no meeting the robots r_2 and r_3 consider the clockwise direction while r_1 considers the counter clockwise direction.

As during the Look phase of time t_e, r_2 is on the node u_0, r_3 is on a node among $\{u_1, \dots, u_i\}$, as r_2 and r_3 are both considering the clockwise direction during the Move phase of time t_e, as r_1 is on node u_0 during the Look phase of time t_e and during the Look phase of time $t_e + 1$ considering the counter clockwise direction during the Move phase of time t_e, as the robots do not change their directions as long as they are alone on their respective nodes, and as no robot can cross e, the first meeting happens between r_2 and r_3 because r_3 was stuck on a node and r_2 was moving. Call $t_e + 1 \leq t_{meet} \leq t'_s$, the time at which occurs the first meeting after the tower T breaks. For a meeting to happen at time t_{meet} between r_2 and r_3, the two robots are necessarily edge-activated. And as seen previously r_2 was thus moving during the Move phase of round $t_{meet} - 1$, while r_3 was not moving during the Move phase of round $t_{meet} - 1$. As r_2 and r_3 are edge-activated during the round $t_{meet} - 1$, their variables $HasMovedPreviousEdgeActivation$ and $NumberRobotsPreviousEdgeActivation$ are updated. For r_2 its variable $HasMovedPreviousEdgeActivation$ is updated to true. For r_3 its variable $HasMovedPreviousEdgeActivation$ is updated to false, and its variable $NumberRobotsPreviousEdgeActivation$ is updated to 1.

Call t_{act} ($t_{act} \geq t_{meet}$) the first time after t_{meet} when the robots r_2 and r_3 are edge-activated. The variables are only updated during the Compute phase of rounds where the robots are edge-activated. Therefore at the end of the Look phase of time t_{act} the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of r_2 are false, as its variable $HasMovedPreviousEdgeActivation$ is true, so after the Compute phase of round t_{act} r_2 still considers the clockwise direction. However at the end of the Look phase of round t_{act} the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of r_3 is true, thus it considers the counter clockwise direction after the Compute phase of time t_{act}. Thus r_2 and r_3 are considering opposite global directions during the Move of round t_{act}, they are thus not able to form a 2-long-lived tower. Moreover r_1 cannot help in forming a tower (and thus neither a 2-long-lived tower) as it is considering the counter clockwise direction and it is on a node among $\{u_{(i+1) \pmod n} \dots u_0\}$ during the Look phase of time t_{act}.

Moreover at time t_{act} we are in a symmetrical situation compared to the situation that happened at time t_e. Indeed during the Move phase of round t_{act} a robot is considering the clockwise direction (r_2) while the two other robots are considering the counter clockwise direction. Two robots (r_2 and r_3) are on a same node during the Look phase of time t_{act} but are not on a same node during the Look phase of time $t_{act} + 1$. The third robot of the system (r_1) is not on the same node as r_2 and r_3 during the Look phase of time t_{act}. Moreover if we call u'_0 the node where r_2 is located during the Look phase of time $t_{act}+1$. Then we can denote by $\{u'_0, u'_1, \dots, u'_p, \dots u'_{n-1}\}$ the nodes of $\mathcal{G}$ in the counter clockwise direction from u'_0, with p an integer such that $2 \leq p' \leq n-2$. The edge(s) between u'_0 and the position where r_3 is located during the Look phase of time $t_{act}+1$ have been visited during the Move phase of time t_e. During the Move phase of time t_{act} r_2 and r_3 are considering two opposite global directions, and separate them as they are edge-activated. Thus during the Look phase of time t_e+1, r_3 is either on node u'_1 (if only one of the robots among r_2 and r_3 has moved during the Move phase of round t_{act}) or on node u'_2 (if both r_2 and r_3 have moved during the Move phase of round t_{act}). Thus the edge e is an edge permitting to go from u'_j to $u'_{(j+1) \pmod n}$ considering the counter clockwise direction, with either $1 \leq j \leq (n-1)$ or $2 \leq j \leq (n-1)$. Besides r_1 is on a node among $\{u'_1, \dots, u'_j\}$ considering the counter clockwise direction during the Look phase of time t_{act} (as it was on a node among $\{u_{(i+1) \pmod n}, \dots, u_0\}$ between the Look phase of time t_e and the Look phase of time t_{act} always considering the counter clockwise direction, where $u_{(i+1) \pmod n}$ equals uj', and u_0 is a node among $\{u'_1, \dots, u'_j\}$).

Thus using symmetric arguments and then recurrence we can say that when r_3 is on a node among $\{u_1, \dots, u_i$, all the meetings involving two robots do not lead to the formation of a 2-long-lived tower. Thus there is a contradiction with the fact that T' can be formed without crossing e.

Case 1.2: During the Look phase of time $\mathbf{t_e+1}$ $\mathbf{r_3}$ is on a node among $\{\mathbf{u_{i+1}, \dots, u_{n-1}}\}$.

When r_3 is on a node among $u_{i+1}, \dots u_{n-1}\}$, as no robots can cross e and that r_3 and r_2 are considering the clockwise direction while r_1 is considering the counter clockwise direction, and that during the Look phase of round t_e+1 r_2 is on node u_1 and r_1 is on node u_0, then the first meeting that happens after T breaks is either between r_3 and r_1 or between r_3 and r_2.

Case 1.2.1: The first meeting after time $\mathbf{t_e}$ happens between $\mathbf{r_3}$ and $\mathbf{r_1}$.

First note that r_1 and r_3 cannot meet on where r_2 is located.

The two robots r_1 and r_3 are considering two opposite global directions during the Move phase of round t_e. Moreover they keep consider these directions as long as they are isolated. Thus, as the variables dir are updated during the Compute phase of rounds, when r_1 and r_3 meet during the Look phase of time t'_{meet} they are considering two opposite global directions, implying that both were moving during the Move phase of time $t'_{meet}-1$. As the robots meet during the Look phase of time $t'_{meet}-1$, the robots r_1 and r_3 were edge-activated during time $t'_{meet}-1$. Thus during the call to the function UPDATE of time $t'_{meet}-1$ the variables $HasMovedPreviousEdgeActivation$ of r_1 and r_3 are set to true. Moreover the values of the variables are only updates during the Compute phase of rounds where the robots are edge-activated. Thus during the Look phase of time t'_{act}, where $t'_{act} \geq t'_{meet}$ is the first time after t'_{meet} when the robots r_1 and r_3 are edge-activated, their variables $HasMovedPreviousEdgeActivation$ are still true. Thus at the end of the Look phase of time t'_{act} the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of these two robots are false, thus they still consider the same direction. So they are not able to form a 2-long-live tower. As the robots r_1 and r_3 are edge-activated at time t'_{act} they separate themselves during the move phase of time t'_{act}. Moreover r_2 cannot meet r_1 or r_3 at a time between time t'_{meet} and t'_{act} as it is considering the clockwise direction and that it is on a node among $\{u_1\}\ldots,\{u_i\}$.

Besides at time t'_{act} we are in a situation similar to the one that we described at time t_e.

Thus using identical arguments that the one used when r_3 is on a node among $\{u_1,\ldots,u_k$ and meet for the first time after T breaks r_2, we can conclude that all the meetings involving two robots do not lead to the formation of a 2-long-lived tower. Thus there is a contradiction with the fact that T' can be formed without crossing e.

Case 1.2.2: The first meeting after time $\mathbf{t_e}$ happens between $\mathbf{r_3}$ and $\mathbf{r_2}$.

r_3 is on a node among $u_{i+1},\ldots,u_{n-1}\}$, considering the clockwise direction. We know that the three robots of the system keep consider their respective direction as long as there is no meeting. As during the Look phase of time t_e+1, r_2 is on node u_1 considering the clockwise direction, and that r_2 does not change its direction until the meeting, and that e cannot be crossed, some adjacent edges to the positions where r_3 is located must be present in the clockwise direction for r_3 to meet r_2. However r_1 is considering the counter clockwise direction and it is on node u_0 during the Look phase of time t_e+1 thus as r_3 and r_1 do not meet first, there exists an edge which is crossed by r_1 and r_3 at the same time t_{cross} but in reverse direction. Thus the robot r_1 and r_3 switch their position during the Move phase of time t_{cross}. At time t_{cross} we are in a situation similar to the one that we described at time t_e. Using similar arguments we can thus conclude that that T' can be formed without crossing e.

Thus when r_3 is considering the clockwise direction whatever its location during the Look phase of time t_e+1, we cannot form the 2-long-lived tower T' if e is not crossed.

Case 2: During the Look phase of time $\mathbf{t_e}+1$ $\mathbf{r_3}$ considers the counter clockwise direction.

We know that during the Look phase of time t_e, r_3 is not on node u_0. Moreover as the variable *dir* is only updated during the Compute phases of rounds, r_3 consider the counter clockwise during the Move phase of time t_e. We consider 2 different cases described below.

Case 2.1: During the Look phase of time $\mathbf{t_e}$ $\mathbf{r_3}$ is on a node among $\{\mathbf{u_{i+1}},\ldots,\mathbf{u_{n-1}}\}$.

Here we are in a situation symmetrical to the one described Case 1.1. So using symmetrical arguments we can conclude that T' cannot be formed if e is not crossed.

Case 2.2: During the Look phase of time $\mathbf{t_e}$ $\mathbf{r_3}$ is on a node among $\{\mathbf{u_1},\ldots,\mathbf{u_i}\}$.

As during the Move phase of time t_e r_3 considers the counter clockwise direction if there exists an adjacent edge to its current direction in the counter clockwise direction, then r_3 moves.

As proved previously at time t_e the edge linking node u_0 to u_1 is present.

Thus here we consider two cases, the case where r_3 is on node u_1 during the Look phase of time t_e and thus on node u_0 during the Look phase of time t_e+1, and the case where r_3 is on a node among $\{u_1,\ldots,u_i$ during the Look phase of time t_e+1.

Case 2.2.1: $\mathbf{r_3}$ is on node u_0 during the Look phase of time $\mathbf{t_e + 1}$.

If during the Look phase of time $t_e + 1$ r_3 is on node u_0 then the first time t_{act}" (t_{act}" $\geq t_e + 1$) after time $t_e + 1$ the robots r_1 and r_3 are edge-activated they must separate themselves, otherwise r_1 and r_3 are involved in a 2-long-lived tower which is a contradiction with the fact that $t'_s > t_e + 1$. Moreover as during the Look phase of time t_e r_3 is not on node u_0, if during the Look phase of time $t_e + 1$ it is on node u_0 this implies that it has moved during the Move phase of round t_e. So its variable $HasMovedPreviousEdgeActivation$ is set to true during the call to the function Update of time t_e. As the values of the variables are updated only during the Compute phases of rounds where the robots are edge-activated, during the Look of phase of time $t_e + 1$, the variable $HasMovedPreviousEdgeActivation$ of r_3 is still true. Therefore it does not change its moving direction after the Compute phase of round t_{act}". Therefore if r_3 is on node u_0 during the Look of phase $t_e + 1$, the next time the robots are edge-activated, r_1 has to change its moving direction. As at time t_{act}" r_1 and r_3 are edge-activated, and as during the Move phase of this round they consider two opposite global directions, they separate them. So during the Look phase of time t_{act}" $+ 1$ r_1 and r_3 are not on the same node. At time t_{act}" we are thus in a situation identical to the one described in case 1.1. So we can use similar arguments to show that T' cannot be formed if e is not crossed.

Case 2.2.2: $\mathbf{r_3}$ is on node among $\{\mathbf{u_1}, \ldots, \mathbf{u_i}\}$ during Look phase of time $\mathbf{t_e + 1}$.

This case is symmetrical to the case 1.2, thus using symmetrical arguments we show that there is a contradiction with the fact that T' can be formed without crossing e.

All the cases have been treated, and prove the lemma. □

Lemma 4.12. *Consider that there are no 3-long-lived towers in $\mathcal{E}$, and let $T_i = (S_i, [t_{s\_i}, t_{e\_i}])$ be the i^{th} 2-long-lived tower of $\mathcal{E}$ (with $i \geq 2$). If $T_{i+1} = (S_{i+1}, [t_{s\_i+1}, t_{e\_i+1}])$ exists such that $t_{s\_i+1} = t_{e\_i} + 1$, then all the edges of $\mathcal{G}$ have been crossed by at least one robot between time $t_{s\_i} - 1$ and time t_{s_i+1}.*

Proof. Assume that $\mathcal{E}$ does not contain 3-long-lived towers.

Consider $T_{first} = (S_{first}, [t_{sfirst}, t_{efirst}])$ the first 2-long-lived tower of $\mathcal{E}$. Consider a 2-long-lived tower $T_i = (S_i, [t_{s\_i}, t_{e\_i}])$ of $\mathcal{E}$, with $i \geq 2$ and such that T_i corresponds to the i^{th} 2-long-lived tower of $\mathcal{E}$.

First note that once a 2-long-lived tower $T = (S, [t_s, t_e])$ is broken, the next 2-long-lived tower in $\mathcal{E}$ can only appear at time $t \geq t_e + 1$. Indeed, there are only three robots in the system. Moreover by lemma 4.10 we know that during the Look phase of time t_e the robot that is not involved in the tower T cannot be on the same node as the robots of S. Besides if r_3 is on a same node as the robots of S during $[t_{3s}, t_{3e}] \subseteq [t_s, t_e]$ and if during $[t_{3s}, t_{3e}[$ the robots are at least one time edge-activated then the three robots are forming a 3-long-lived tower, which leads to a contradiction with the fact that there is no 3-long-lived towers in $\mathcal{E}$. This implies that a 2-long-lived tower other than T cannot be present in $\mathcal{E}$ during $[t_s, t_e]$. Therefore, the next 2-long-lived tower of $\mathcal{E}$ can only appear from time $t_e + 1$ included.

During the Look phase of time $\mathbf{t_{s\_i}}$ the robot not involved in $\mathbf{T_i}$ is considering a global direction opposed to the one considered by the robots of S_i, and it is on a node different from the one where the robots of S_i are located.

To prove this statement, we analyze how T_i is constructed.

Case 1: Construction of the 2-long-lived tower $\mathbf{T_i = (S_i, [t_{s\_i}, t_{e\_i}])}$ such that $\mathbf{t_{s\_i} = t_{e\_i-1} + 1}$.

Assume that T_{i-1} is composed of the robots r_1 and r_2 and that they are on a node u of $\mathcal{G}$ during the Look phase of time t_{e_i-1}.

According to lemma 4.10 r_3 is not on u during the Look phase of time t_{e_i-1}, and moreover during the Look phase of time $t_{e\_i-1} + 1$ one robot of S_{i-1} is located at u considering a global direction opposed to the one considered by the other robot of S_{i-1}. Assume that at time $t_{e\_i-1} + 1$ this is r_1 that is still on node u, and that it is considering the counter clockwise direction while r_2 is considering the clockwise direction.

Call u_1 the node of $\mathcal{G}$ adjacent to u and such that a robot on node u must cross an edge in the clockwise direction to go on node u_1. Call u_2 the node of $\mathcal{G}$ adjacent to u_1 and such that a robot on node u_1 must cross an edge in the clockwise direction to go on node u_2.

Do the robots $\mathbf{r_1}$ and $\mathbf{r_2}$ can be involved in $\mathbf{T_i}$?

If r_1 and r_2 form T_i, this implies that they are together on the same node during the Look phase of time $t_{e\_i-1} + 1$. By assumption r_1 is on node u during the Look phase of time $t_{e\_i-1} + 1$. However according to lemma 4.10, only one robot of S_{i-1} is on node u during the Look phase of time $t_{e\_i-1} + 1$. Thus r_2 cannot be on node u during the Look phase of time t_{e_i-1}. Therefore r_1 and r_2 cannot form T_i.

Do the robots $\mathbf{r_2}$ and $\mathbf{r_3}$ can be involved in $\mathbf{T_i}$?

As seen previously, r_2 cannot be on node u during the Look phase of time $t_{e\_i-1} + 1$. As r_2 is on node u during the Look phase of time t_{e_i-1}, this implies that r_2 has moved during the Move phase of time t_{e_i-1}. The variables *dir* are modified only during the Compute phases of rounds. During the Look phase of time $t_{e\_i-1} + 1$, r_2 is considering the clockwise direction, thus it considers the clockwise direction during the Move phase of time t_{e_i-1}. Therefore at time t_{e_i-1} it exists an edge linking node u and node u_1, and r_2 is on node u_1 during the Look phase of time $t_{e\_i-1} + 1$. If r_3 and r_2 form a 2-long-lived tower at time $t_{e\_i-1} + 1$ it implies that these two robots are on the same node during the Look of time $t_{e\_i-1} + 1$. Thus r_3 has to be on node u_1 during the Look phase of time $t_{e\_i-1} + 1$. As every robot can only cross at most an edge per round, for the robot r_3 to be on node u_1 during the Look phase of $t_{e\_i-1} + 1$ it has to be either on node u_1 or on an adjacent node of u_1 during the Look phase of time t_{e_i-1}. As by lemma 4.10 r_3 is not on node u at time t_{e_i-1}, the only way for r_3 to be on node u_1 during the Look phase of round $t_{e\_i-1} + 1$ is to be during the Look phase of time t_{e_i-1} either on node u_1 or on node u_2. If r_3 is on node u_1 during the Look phase of time $te\_i - 1$ as the edge linking u to u_1 is present at time t_{e_i-1}, r_3 has to consider the clockwise direction and the edge linking node u_1 to node u_2 must be missing (otherwise r_3 moves and thus it is not on node u_1 during the Look phase of time $t_{e\_i-1} + 1$). Similarly if r_3 is on node u_2 during the Look phase of time t_{e_i-1} it has to consider the counter clockwise direction and the edge linking node u_2 to node u_1 must be present at time t_{e_i-1}, otherwise r_3 cannot be on node u_1 during the Look phase of time $t_{e\_i-1} + 1$.

Assume first that during the Look phase of time t_{e_i-1} r_3 is on node u_1 and the edge linking node u_1 to node u_2 is missing. As during the Look phase of time

t_{e_i-1} r_2 and r_3 are not on a same node but that they are on a same node during the Look phase of time $t_{e\_i-1}+1$, this implies that they are edge-activated at time t_{e_i-1}. Thus during the call to the function UPDATE of time t_{e_i-1} the variable $HasMovedPreviousEdgeActivation$ of r_2 is set to true while the variable $HasMovedPreviousEdgeActivation$ of r_3 is set to false. Moreover during the call to the function UPDATE of time t_{e_i-1} the variable $NumberRobotsPreviousEdgeActivation$ of r_3 is set to 1 as the two other robots of the system are not on the same node as it during the Look phase of time t_{e_i-1}. Call t_{act} ($t_{act} \geq t_{e\_i-1}+1$) the first time after $t_{e\_i-1}+1$ at which r_2 and r_3 are edge-activated. The variables of a robot are only updated during Compute phases of rounds where this robot is edge-activated. Thus at the end of the Look phase of time $t_{e\_i-1}+1$ the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of r_2 are false as the value of its variable $HasMovedPreviousEdgeActivation$ is true, thus it still considers the clockwise direction while the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of r_3 is true, thus it changes its direction to consider the counter clockwise direction. Thus the robots r_2 and r_3 separate themselves during the Move phase of time t_{act}. Therefore they are not involved in a 2-long-lived tower.

Assume secondly that r_3 is on u_2 during the Look phase of time t_{e_i-1} considering the counter clockwise direction during the move phase of time t_{e_i-1} and that the edge linking u_1 and u_2 is present at time t_{e_i-1}. As seen previously during time t_{e_i-1} the two robots are edge-activated. The two robots move during the Move phase of time t_{e_i-1}. Thus during the call to the function UPDATE of time t_{e_i-1} the variables $HasMovedPreviousEdgeActivation$ of r_2 and r_3 are set to true. Moreover, also as seen previously the first time greater or equal to t_{e_i-1} when the robots r_2 and r_3 are edge-activated, their variables have the same values as after the Compute phase of time t_{e_i-1}. Thus the first time greater or equal to t_{e_i-1} when the robots r_2 and r_3 are edge-activated their predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false. Thus they keep consider their respective global directions, which are two opposite global directions. Thus the robots r_2 and r_3 separate themselves during the Move phase of the first time greater or equal to t_{e_i-1} where they are edge-activated. Therefore they are not involved in a 2-long-lived tower.

Thus whatever the position of r_3 during the Look phase of time t_{e_i-1}, the robots r_2 and r_3 cannot be involved in T_i.

Do the robots $\mathbf{r_1}$ and $\mathbf{r_3}$ can be involved in $\mathbf{T_i}$?

r_1 stays on node u from time t_{e_i-1} to the Look phase of time $t_{e\_i-1}+1$ while considering the counter clockwise direction and while being edge-activated (as the edge linking node u to node u_1 is present in the system at time t_{e_i-1}), proving that the adjacent edge to u in the counter clockwise direction is missing at time t_{e_i-1}. By lemma 4.10 r_3 is not on node u during the Look phase of time t_{e_i-1}, and as the adjacent edge in the counter clockwise direction of u is missing at time t_{e_i-1}, if r_3 is on node u during the Look phase of time $t_{e\_i-1}+1$ it has to be located on node u_1 during the Look phase of time t_{e_i-1} and it has to consider the counter clockwise direction during the Move phase of this time. During the Look phase of

time $t_{e\_i-1}+1$ both r_1 and r_3 are thus considering the counter clockwise direction. During the call to the function UPDATE of time t_{e_i-1} the variable *HasMovedPreviousEdgeActivation* of r_1 is set to false, while the variable *HasMovedPreviousEdgeActivation* of r_3 is set to true. Moreover during the call to the function UPDATE of time t_{e_i-1} the variable *NumberRobotsPreviousEdgeActivation* of r_1 is set to 2, as it is on node u with the robot r_2 during the Look phase of time t_{e_i-1}, and as by lemma 4.10 r_3 cannot be on node u during the Look phase of this time.
Call t_{act} ($t_{act} \geq t_{e\_i-1}+1$) the first time after $t_{e\_i-1}+1$ at which r_1 and r_3 are edge-activated. The variables of a robot are only updated during Compute phases of rounds where this robot is edge-activated. Thus during the Look phase of time t_{act} the variables of r_1 and r_3 have the same values as after the Compute phase of time t_{e_i-1}.
When the robots are edge-activated, either both adjacent edges of u are present or only one. Thus during t_{act} either the adjacent edge in the counter clockwise direction of u is present or not.
Assume that the adjacent edge in the counter clockwise direction of u is present during t_{act}. The predicates *WeAreStuckInTheSameDirection*() and *IWasStuckOnMyNodeAndNowWeAreMoreRobots*() of r_3 are false as its variable *HasMovedPreviousEdgeActivation* is true. Thus r_3 still consider the counter clockwise direction during the Move phase of time t_{act}. For the robot r_1 as during the Look phase of time t_{act} its variable *dir* indicates the counter clockwise direction and as the edge on the counter clockwise direction of its current location is present at time t_{act}, its predicate *ExistsEdgeOnCurrentDirection*() is true, thus its predicate *WeAreStuckInTheSameDirection*() is false at time t_{act}. Moreover as by lemma 4.10 r_2 is not on node u during the Look phase of time $t_{e\_i-1}+1$, and as for robots to meet at a time t they need to be edge-activated at time $t-1$, then during the Look phase of time t_{act} r_2 cannot be on node u, thus the predicate *NumberOfRobotsOnNode*() of r_1 is equal to 2 at time t_{act}. Thus the condition "*NumberOfRobotsOnNode*() $>$ *NumberRobotsPreviousEdgeActivation*" is not true for r_1 at the end of the Look phase of time t_{act}, thus its predicate *IWasStuckOnMyNodeAndNowWeAreMoreRobots*() is false. Thus the predicates *WeAreStuckInTheSameDirection*() and *IWasStuckOnMyNodeAndNowWeAreMoreRobots*() of r_1 are false at the end of the Look phase of time t_{act}. Thus r_1 considers the counter clockwise direction during the Move phase of time t_{act}. Thus the two robots consider the same global direction during the Move phase of time $t_{e\_i-1}+1$. Therefore they are involved in a 2-long-lived tower.
Assume that the adjacent edge in the counter clockwise direction of u is not present during t_{act}. The predicates *WeAreStuckInTheSameDirection*() and *IWasStuckOnMyNodeAndNowWeAreMoreRobots*() of r_3 are false as its variable *HasMovedPreviousEdgeActivation* is true. Thus r_3 considers the counter clockwise direction during the Move phase of time t_{act}. As by assumption at time t_{act} r_1 and r_3 are edge-activated, if the adjacent edge in the counter clockwise direction of u is missing at time t_{act} then this implies that the edge linking node u to node u_1 is present a this time. For the same reasons as previously the predicate *NumberOfRobotsOnNode*() of r_1 is equal to 2. Thus all the conditions of the the predicate *WeAreStuckInThe-*

$SameDirection()$ of r_1 are true. Therefore during the Compute phase of round t_{act} r_1 execute the function GIVEDIRECTION. If the function modifies the variable dir of r_1 such that it considers the clockwise direction during the Move phase of time t_{act} then r_1 and r_3 separate them during the Move phase of this time, so they are not involved in a 2-long-lived tower. However if the function modifies the variable dir of r_1 such that it considers the counter clockwise direction during the Move phase of time t_{act} then r_1 and r_3 are involved in T_i.

In the two cases where the 2-long-lived tower in formed at time $t_{e\_i-1}+1$, note that by lemma 4.10 r_2 cannot be on node u during the Look phase of time $t_{e\_i-1}+1$. As r_2 is on node u during the Look phase of time t_{e_i-1}, this implies that r_2 has moved during the Move phase of time t_{e_i-1}. During the Look phase of time $t_{e\_i-1}+1$, r_2 is considering the clockwise direction. As the variable dir is modified only during the Compute phases of rounds, then during the Move phase of time t_{e_i-1} r_2 considers the clockwise direction. Therefore at time t_{e_i-1} it exists an edge linking node u and node u_1, and r_2 is on node u_1 during the Look phase of time $t_{e\_i-1}+1$.

Thus T_i is necessarily formed of r_1 and r_3. Note moreover that during the Look phase of time $t_{e\_i-1}+1$ the robot r_2 is on node u_1 considering a global direction opposed to the one considered by the robots of T_i. Besides the edge linking u to u_1 has been crossed by r_3 and by r_2 during the Move phase of time t_{e_i-1} which is equal to time $t_{s\_i}+1$.

Case 2: Construction of the 2-long-lived tower $\mathbf{T_i = (S_i, [t_{s\_i}, t_{e\_i}])}$ such that $\mathbf{t_{s\_i} > t_{e\_i-1} + 1}$.

As the next 2-long-lived tower of T_{i-1} in $\mathcal{E}$ starts at time t_{s_i}, and as by assumption $t_{s\_i} > t_{e\_i-1}+1$, then from time $t_{e\_i-1}+1$ to time $t_{s\_i}-1$ the robots cannot form 2-long-lived towers. In this case by lemma 4.7 the robots cannot form 3-short-lived towers. Moreover, by assumption the execution does not contain 3-long-lived towers. This implies that from time $t_{e\_i-1}+1$ to time $t_{s\_i}-1$ the robots are only either isolated or forming 2-short-lived towers.

At time t_{s_i} two robots are forming a 2-long-lived tower. This implies that during the Look phase of round $t_{s\_i}-1$ the two robots were not on a same node. However during the Look phase of time t_{s_i} they are on a same node. This implies that at time $t_{s\_i}-1$ the robots involved in T_i are edge-activated.

Call t_{act_bis} ($t_{act\_bis} \geq t_{s\_i}$) the first time after t_{s_i} where the two robots involved in the tower formed at time t_{s_i} are edge-activated.

Call t_{l_act} the last time in $[t_{e\_i-1}+1, t_{s\_i}-1[$ such that at least two robots of the system are edge-activated.

Property 1: To obtain a 2-long-lived tower at time $\mathbf{t_{s\_i}}$ there must exist a 2-short-lived tower formed at time $\mathbf{t_{l\_act}+1}$.

We prove this statement by contradiction. Assume that at time $t_{l\_act}+1$ there is no 2-short-lived tower in $\mathcal{E}$. This implies that the three robots are isolated during the Look phase of time $t_{l\_act}+1$. For two robots to form a 2-short-lived tower at time t they have to meet at a time t, and to meet at time t the two robots have to be edge-activated. As at time $t_{l\_act}+1$ the robots are isolated, and by definition of t_{l_act}, then from time $t_{l\_act}+1$ to the Look phase of time $t_{s\_i}-1$ the robots are isolated.

To have a 2-long-lived tower at time t_{s_i} a meeting must happen between robots at time t_{s_i}. As proved previously using lemma 4.7 we know that at most two robots

can meet at each instant time in $[t_{e\_i-1}+1, t_{s\_i}-1]$. The meeting can then occurs at time t_{s_i} between two robots considering either two opposite global directions or the same global direction.
If the meeting at time t_{s_i} happens between two robots considering reverse global directions during the Move phase of round $t_{s\_i}-1$ then the two robots have moved during this Move phase. Indeed, if one robot is stuck (there is no adjacent edge to the current location of the robot in the direction it considers) as the other robot considers the opposite global direction it cannot join it. So here the two robots are moving during the Move phase of round $t_{s\_i}-1$. From time t_{s_i} included to time t_{act_bis} excluded the robots of the tower are not edge-activated, thus their predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false and thus they still consider their own global direction. Moreover when the robots are not edge-activated, their values of variables do not change. Thus when they wake-up at time t_{act_bis}, the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the two robots are false (as their values of variables $HasMovedPreviousEdgeActivation$ was set to true during the call to the function Update of round $t_{s\_i}-1$ and as these values have not changed since this time). So they still conserve their reverse global directions. So they separate during the move phase of time t_{act_bis}, and thus no 2-long-lived tower has been created at time t_{s_i}.
If the meeting happens at time t_{s_i} between two robots r_1 and r_2 considering the same global direction, then this implies that one of the robot is stuck on a node. Assume that it is r_1 that does not move during the Move phase of time $t_{s\_i}-1$. As the variables of a robot are only updated during the Compute phases of rounds where this robot is edge-activated, the values of the variables of r_1 and r_2 during the Look phase of time t_{act_bis} are identical to the one during the Compute phase of round $t_{s\_i}-1$. As proved previously during the Look phase of time $t_{s\_i}-1$ the robots are isolated. Therefore during the call to the function Update of round $t_{s\_i}-1$ the variables $NumberRobotsPreviousEdgeActivation$ of r_1 and r_2 are set to 1. Thus for the two robots r_1 and r_2 the condition "$NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation$" is true at the end of the Look phase of time t_{act_bis}. Moreover as during the Move phase of time $t_{s\_i}-1$ the robot r_1 has not moved the value of its variable $HasMovedPreviousEdgeActivation$ is set to false during the call to the function Update of time $t_{s\_i}-1$, while the one of the robot r_2 is set to true. Thus at the end of the Look phase of time t_{act_bis}, the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false for the robot r_2 which thus conserves its direction after the Compute phase of time t_{act_bis} while the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of robot r_1 is true, thus it considers a reverse direction. So after the Compute phase of time t_{act_bis} the robots r_1 and r_2 are considering two opposite global directions and thus they separate themselves. So there is no 2-long-lived tower formed at time t_{s_i}.
So we prove that to have a 2-long-lived tower from a situation where the robots are either isolated or forming 2-short-lived tower, if we want to obtain a 2-long-lived tower at a time t_{s_i}, then a 2-short-lived tower must be formed at time $t_{l\_act}+1$,

where t_{l_act} is the last time in $[t_{e\_i-1}+1, t_{s\_i}-1[$ such that at least two robots of the system are edge-activated.

Property 2: To obtain a 2-long-lived tower at time $\mathbf{t_{s\_i}}$ the 2-short-lived tower formed at time $\mathbf{t_{l\_act}+1}$ must be on a node such that one of its adjacent edge is missing at time $\mathbf{t_{l\_act}+1}$.

We prove this statement by contradiction. Consider a 2-short-lived tower T_{short} formed at time $t_{l\_act}+1$ on a node v. As t_{l_act} corresponds to the last time in $[t_{e\_i-1}+1, t_{s\_i}-1[$ such that at least two robots of the system are edge-activated, and as robots of a 2-short-lived tower can separate them only during the Move phases of rounds where they are edge activated, then the robots of T_{short} are still on node v during the Look phase of time $t_{s\_i}-1$. Assume by contradiction that at time $t_{s\_i}-1$ the two adjacent edges of v are present in the system. Assume that the tower T_{short} is composed of the robots r_1 and r_2.

By definition of a 2-short-lived tower we know that during the Move phase of time $t_{s\_i}-1$ r_1 and r_2 separate them. As the robots executing our algorithm consider at each instant time a direction, for the robot to separate them, they have to consider two opposite global directions. Thus during the Move phase of time $t_{s\_i}-1$ r_1 and r_2 consider two opposite global directions. If the two adjacent edges to v are present at time $t_{s\_i}-1$ then r_1 and r_2 move during the Move phase of time $t_{s\_i}-1$.

At time t_{s_i} two robots must be again together on a same node (to form the 2-long-lived tower T_i). Without lost of generality assume that it is r_1 that is on a same node with an other robot at time t_{s_i}. Call r_{meet} the robot on the same node than r_1 during the Look phase of round t_{s_i}. And assume that they meet on a node w.

As r_1 has moved, during the Look phase of round $t_{s\_i}-1$, then its variable *HasMovedPreviousEdgeActivation* has been set to true during the call to the function UPDATE of time $t_{s\_i}-1$. As the variables of a robot are only updated during the Compute phases of rounds where this robot is edge-activated, during the Look phase of time t_{act_bis} the variable *HasMovedPreviousEdgeActivation* of r_1 is still true. Thus at the end of the Look phase of time t_{act_bis} the predicates *WeAreStuckInTheSameDirection*() and *IWasStuckOnMyNodeAndNowWeAreMoreRobots*() of r_1 are false, and so it keeps consider the same direction as the one it considers during the Move phase of time $t_{s\_i}-1$. Without lost of generality assume that this direction is the clockwise direction.

w corresponds thus to the adjacent edge of v in the clockwise direction.

As r_1 arrives on node w considering the clockwise direction, as there is no 2-long-lived tower at time $t_{s\_i}-1$, and as each robot can cross at most one edge per round, then during the Move phase of time $t_{s\_i}-1$ r_1 is the only robot which crosses the edge linking node v to node w. Thus for r_{meet} to be on node w during the Look phase of time t_{s_i}, during the Move phase of round $t_{s\_i}-1$ either r_{meet} has moved in the counter clockwise direction, or it was stuck on v.

In the first case, during the call to the function UPDATE of round $t_{s\_i}-1$ the variable *HasMovedPreviousEdgeActivation* of r_{meet} is set to true. As during the Look phase of round t_{act_bis} its variable *HasMovedPreviousEdgeActivation* is still true, then at the end of the Look phase of this time its predicates *WeAreStuckInTheSameDirection*() and *IWasStuckOnMyNodeAndNowWeAreMoreRobots*() are

false, and so it keeps consider the counter clockwise direction.
In the second case, as r_1 crosses the edge linking node v to node w during the Move phase of time $t_{s\_i}-1$, this implies that if r_{meet} is stuck on node v, then it is considering the clockwise direction during the Move phase of round $t_{s\_i}-1$, and the adjacent edge of w in the clockwise direction is missing at time $t_{s\_i}-1$. As during the Look phase of time $t_{s\_i}-1$, r_1 and r_2 are on node v, this implies that r_{meet} is alone on node w. Thus during the call to the function UPDATE of round $t_{s\_i}-1$ the value of the variable $NumberRobotsPreviousEdgeActivation$ of r_{meet} is set to 1. Moreover as r_{meet} has not moved during the round $t_{s\_i}-1$ during the call to the function UPDATE the value of its variable $HasMovedPreviousEdgeActivation$ is set to false. As during the Look phase of round t_{act_bis} the values of the variables of r_{meet} have not change since the Compute phase of time $t_{s\_i}-1$ this implies that at the end of the Look phase of round t_{act_bis} the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the robot r_{meet} is true, and so it changes its direction. Thus it considers the counter clockwise direction during the Move phase of round t_{act_bis}.
Thus whatever the sens of direction considered by r_{meet} during the Look phase of time $t_{s\_i}-1$, during the Move phase of time t_{act_bis} the robots r_1 and r_{meet} separate them. The tower formed at time t_{s_i} is thus a 2-short-lived tower and not a 2-long-lived tower.
We conclude that at time $t_{l\_act}+1$ an adjacent edge of v must be missing.
Remark:
Note that as r_1 and r_2 are considering two opposite global directions during the Move phase of time $t_{s\_i}-1$, and as one adjacent edge on v is missing at time $t_{s\_i}-1$, this is necessarily r_3 that meets one of the robots r_1 or r_2 to form the 2-long-lived tower at time t_{s_i}. Moreover using similar pitch as the one used in case 1, we can say that T_i is necessarily composed of the robot r_3 and of the robot of T_{short} that stays on node v during the Move phase of time $t_{s\_i}-1$.

From the two properties enunciated we can say that there is only two ways to have a 2-long-lived tower from a configuration where the robots are either isolated or forming 2-short-lived tower.

Indeed, the 2-short-lived tower (named T_{short}) that starts at time $t_{l\_act}+1$ on node v can be formed either by two robots r_1 and r_2 considering during the Move phase of time t_{l_act} reverse global directions, or by two robots r_1 and r_2 considering the same global direction during the Move phase of time t_{l_act}.

Recall that from time $t_{e\_i-1}+1$ to time $t_{s\_i}-1$ the robots are either isolated or forming 2-short-lived towers. Moreover during the move phase of time t_{e_i-1} the tower T_{i-1} breaks. This implies that during the Move phase of time t_{e_i-1} two robots are considering two opposite global directions. As the variable dir is modified only during the Compute phases of rounds, during the Look phase of time $t_{e\_i-1}+1$ two robots are considering the same global direction while the other robot of the system considers the opposite global direction. Thus by lemma 4.9 we know that between time $t_{e\_i-1}+1$ to time $t_{s\_i}-1$, it is not possible to have the three robots considering the same global direction.

Case 2.1: The 2-short-lived tower is formed at time $t_{l\_act}+1$ on node v by two robots considering during the Move phase of time t_{l_act} two reverse global directions.

As the variables *dir* are updated during the Compute phases of rounds, during the Look phase of time $t_{l\_act}+1$, r_1 and r_2 still consider two opposite global directions. Assume without lost of generality that during the Look phase of time $t_{l\_act}+1$ r_1 is considering the clockwise direction while r_2 is considering the counter clockwise direction. We know by definition of a 2-short-lived tower that the robots r_1 and r_2 separate them during the Move phase of time $t_{s\_i}-1$. As the variable *dir* of a robot is modified only during the Compute phases of rounds where this robot is edge-activated, during the Move phase of round $t_{s\_i}-1$ r_1 is still considering the clockwise direction while r_2 is still considering the counter clockwise direction. Moreover as seen previously (in property 2) it must miss an adjacent edge of node v during time $t_{s\_i}-1$ if a 2-long-lived tower starts at time t_i. Assume without lost of generality that it is the adjacent edge to v in the clockwise direction that is missing. Thus r_1 is still on node v during the Look phase of time t_{s_i}. Moreover by the remark of property 2, we know that T_i is necessarily composed of r_1 and r_3. As there is no 3-long-lived towers from time $t_{e\_i-1}+1$ to time $t_{s\_i}-1$, then r_3 is not on v during the Look phase of time $t_{s\_i}-1$. To form T_i r_3 must be on node v during the Look phase of time t_{s_i}. As the adjacent edge of v in the clockwise direction is missing at time $t_{s\_i}-1$, for r_3 to be on node v during the Look phase of time t_{s_i} it must necessarily be on the adjacent node of v in the counter clockwise during the Look phase of time $t_{s\_i}-1$ and considers the clockwise direction during the Move phase of time $t_{s\_i}-1$. Then during the Look phase of time t_{s_i} r_1 and r_3 consider the same global direction while r_2 considers the opposite global direction. Moreover during the Look phase of time t_{s_i} r_2 is on a node different from v, and more precisely it is on the node adjacent to v in the counter clockwise direction. Besides note that the edge linking v to the adjacent node of v in the counter clockwise direction has been crossed by r_3 and by r_2 during the Move phase of time $t_{s\_i}-1$.

Case 2.2: The 2-short-lived tower is formed at time $\mathbf{t_{l\_act}+1}$ on node v by two robots considering during the Move phase of time $\mathbf{t_{l\_act}}$ the same global direction.

As the variables *dir* are updated during the Compute phases of rounds, during the Look phase of time $t_{l\_act}+1$, r_1 and r_2 still consider a same global direction. Assume without lost of generality that r_1 and r_2 are considering the clockwise direction during the Look phase of time $t_{l\_act}+1$. We know by definition of a 2-short-lived tower that the robots r_1 and r_2 separate them during the Move phase of time $t_{s\_i}-1$. Assume without lost of generality that during the Move phase of round $t_{s\_i}-1$ r_1 still considers the clockwise direction while r_2 considers the counter clockwise direction. We know by lemma 4.9 that during the Move phase of time $t_{act}+1$ r_3 was considering the counter clockwise direction. As there is no 3-long-lived towers from time $t_{e\_i-1}+1$ to time $t_{s\_i}-1$, then r_3 is not on v during the Look phase of time $t_{s\_i}-1$. Moreover as r_1, and r_2 are on node v during the Look phase of time $t_{s\_i}-1$, this implies that r_3 is alone on its node during the Look phase of time $t_{s\_i}-1$. Thus the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of r_3 are false at the end of the Look phase of time t_{s_i} thus it still considers the counter clockwise direction during the Move phase of round $t_{s\_i}-1$. Moreover if we want to have a 2-long-lived tower at time t_{s_i} there must exist a missing adjacent edge to node v at time $t_{s\_i}-1$. If the adjacent edge to v in the clockwise direction is missing at time $t_{s\_i}-1$ then the robot r_3 cannot be on node v at time t_{s_i}. Thus it is necessarily the edge in the counter clockwise direction of v that is missing at time $t_{s\_i}-1$. Thus during the Move phase of time $t_{s\_i}-1$ r_2 cannot move from v, while r_1 moves on the adjacent node of v in the clockwise direction. This implies that during the Look phase of time t_{s_i} r_2 and r_3 are on node v considering the counter clockwise direction while r_1 is on a node different from v, more precisely it is on the adjacent node of v in the clockwise direction and it is considering the clockwise direction. Moreover note, that the edge linking node v to the adjacent node to v in the clockwise direction has been crossed by r_3 and by r_1 during the Move phase of time $t_{s\_i}-1$.

Thus whatever the time at which the tower T_i is build after the tower T_{i-1} (it is possible to have $T_{i-1}=T_{first}$) the robots of T_i consider during the Look phase of time t_{s_i} a global direction opposed to the one considered by the robot not involved in T_i, and moreover this last robot is not on the same node as the robots of T_i during the Look phase of time t_{s_i}. Moreover the edge linking the node where the tower is located during the Look phase of time t_{s_i} to the node where the robot not involved in T_i is located during the Look phase of time t_{s_i} has been crossed during the Move phase of time $t_{s\_i}-1$.

To build a 2-long-lived tower $\mathbf{T_{i+1}=(S_{i+1},[t_{s\_i+1},t_{e\_i+1}])}$ (corresponding to the next 2-long-lived tower after T_i in $\mathcal{E}$) such that $t_{s\_i+1}=t_{e\_i}+1$, all the edges of $\mathcal{G}$ have been crossed between time $t_{s\_i}-1$ and time t_{s_i+1}.

By contradiction assume that T_{i+1} is formed at time t_{s_i+1} but that there exists an edge e such that it is not visited from time ts_i-1 to time t_{s_i+1}.

Assume without lost of generality that T_i is composed of the robots r_1 and r_2 and that during the Look phase of time t_{s_i} these two robots are on a node x_0 and are considering the counter clockwise direction. Thus, by the observations on the the different ways to construct the 2-long-lived tower T_i we know that during the Look phase of time t_{s_i} the robot r_3 is on node x_1 where x_1 is the adjacent node of x_0 in the clockwise direction, and it considers the clockwise direction. Note moreover that during the Move phase of time $t_{s\_i}-1$, the robot r_3 has crossed the edge linking node x_0 to node x_1.

Note $\{x_0,x_1,\ldots,x_k,\ldots,x_{n-1}\}$ the nodes of $\mathcal{G}$ in the clockwise direction from node x_0, with k an integer such that $2\le k\le n-2$. We know that during the Move phase of time $t_{s\_i}-1$ the edge linking node x_0 to x_1 has been crossed thus e is necessarily an edge permitting to go from a node x_i to a node $x_{(i+1) \pmod n}$ considering the clockwise direction, with i an integer such that $1\le i\le(n-1)$.

As seen in case 1 to build a tower $T_{i+1}=(S_{i+1},[t_{s\_i+1},t_{e\_i+1}])$ corresponding to the next 2-long-lived tower after T_i in $\mathcal{E}$, such that $t_{s\_i+1}=t_{e\_i}+1$, during the Look phase of time t_{e_i} all the robots of the system must consider the same global direction, the robots of T_i must be stuck on a node u' while the robot r_3 must be on the adjacent node of u' that possesses an adjacent edge leading to node u'.

As long as the robot r_3 and the robots of T_i do not change their respective directions, as e cannot be crossed, then they cannot meet again.

As long as the robot r_3 is alone on a node, its predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false, thus it keeps consider the clockwise direction.

As long as the robots of T_i are not stuck they continue to consider the same global direction (the counter clockwise direction). As the edge e cannot be crossed, at a time the two robots of S_i are necessarily stuck making their predicates $WeAreStuckInTheSameDirection()$ to true. Here the robots r_1 and r_2 execute the function GiveDirection. We know by assumption that the robots r_1 and r_2 are forming the tower T_i until the formation of the tower T_{i+1}. Moreover we know that to form T_{i+1} during the Look phase of time $t_{s\_i+1} - 1$ the three robots must consider the same global direction. However at the time where the robot of S_i are stuck, the robots of S_i consider the counter clockwise direction while r_3 considers the clockwise direction (as it has not change its direction since time t_{s_i}). Thus the tower T_{i+1} cannot be formed. Thus after the execution of the function GiveDirection the robots of S_i are still forming T_i and thus by definition of a 2-long-lived tower, they have to consider a same global direction. If after the execution of the function GiveDirection they consider the counter clockwise direction then the same scenario starts again (execution of the function GiveDirection). If after the execution of the function GiveDirection the two robots of S_i consider the clockwise direction then they are able to move (as to execute the function GiveDirection an edge in the opposite direction than the one considered before the call to this function must be present).

Here, when the robots of S_i consider the clockwise direction, similarly as previously the robots of T_i continue to consider a same global direction as long as they are not stuck. During times where the robots r_1 and r_2 are considering the clockwise direction they can either be stuck or meet the robot r_3.

When they are stuck then we can use similar arguments as the one used when the robots of S_i are considering the counter clockwise direction and are stuck, to say that the robots r_1 and r_2 execute the function GiveDirection and that after the execution of this function they can either both change their directions or both still consider the same direction. If after the execution of the function GiveDirection the robots of S_i both change their directions then as they do not have meet r_3 since T_i starts, we are again in a case identical to the one that happens at time t_{s_i}. If after the execution of the function GiveDirection the robots of S_i do not change the direction they consider, then they can again be in a situation where they are stuck (in this case they repeat the same scheme, execution of the function GiveDirection, then change of directions or not) or they can meet the robot r_3.

Call t_{meet} the first time after t_{s_i} such that the robots of S_i meet r_3. As the edge e cannot be crossed, as r_3 considers the clockwise direction as long as it is alone on its node, and as during the Look phase of time t_{s_i} the robots of S_i are on node x_0 while r_3 is on node x_1 then the meeting occurs because the two entities (the tower T_i and the robot r_3) are considering the clockwise direction during the Move phase of time $t_{meet} - 1$, and because r_3 is stuck during the Move phase of time $t_{meet} - 1$. Thus the conditions to form T_{i+1} are not verified (as it is the tower T_i that must be stuck, and the robot r_3 that has meet one of the robot of T_i in order to form T_{i+1}). During phase $t_{meet} - 1$ the three robots of the system are edge-activated. If it was not the case, they cannot meet at time t_{meet}. As during the Move phase of time $t_{meet} - 1$ the robots of T_i have moved, during the call to the function Update of round $t_{meet} - 1$ their variables $HasMovedPreviousEdgeActivation$ are set to true. As during the Move phase of time $t_{meet} - 1$ r_3 has not moved, during the call to the function UPDATE of round $t_{meet} - 1$, its variable $HasMovedPreviousEdgeActivation$ is set to false. Moreover during the Look phase of time $t_{meet} - 1$ r_3 is alone on its node, otherwise there is a contradiction with the fact that t_{meet} is the first time after t_{s_i} where the robots meet. Thus during the call to the function UPDATE of round $t_{meet} - 1$ the variable $NumberRobotsPreviousEdgeActivation$ of r_3 is set to 1. Call t_{act} ($t_{act} \geq t_{meet}$) the first time after t_{meet} at which the robots are edge-activated. As the variables of a robot are only updated during Compute phases of rounds where this robot is edge-activated, during the Look phase of time t_{act} the values of the variables of the robots are identical to the one set during the Compute phase of time $t_{meet} - 1$. Thus at the end of the Look phase of time t_{act} the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the robots of S_i are false as their variables $HasMovedPreviousEdgeActivation$ are true. However the predicate $IWasStuck$-$OnMyNodeAndNowWeAreMoreRobots()$ of robot r_3 is true. Thus r_3 considers during the Move phase of time t_{act} a direction opposed to the one it was considering during the Move phase of the round $t_{meet} - 1$. So during the Move phase of the round t_{act} the two entities are considering two different global directions and separate themselves (as they are edge-activated).

During the Look phase of time $t_{act} + 1$ we are in a situation symmetrical to the one that happens at time t_{s_i}. Indeed, during the Look phase of time $t_{act} + 1$ the robots of T_i are considering the clockwise direction while r_3 is considering the counter clockwise direction. Moreover r_3 is on a node on the clockwise of the node where the robots of T_i are located. Thus by using symmetrical arguments we can prove that in this symmetrical situation whatever the direction consider by the robots of T_i and whatever their states (stuck on a node or meeting r_3) it is not possible to form T_{i+1}.

Then using an argument of recurrence on these situations (situation where the robot r_3 is on a node at the counter clockwise of the node where the robots of T_i are located or r_3 is on a node at the clockwise of the node where the robots of T_i are located) we succeed to prove that if e is not crossed then we cannot form the tower T_{i+1}.

This proves the lemma. □

Main lemmas. Upon establishing all the above properties of towers, we are now ready to state the main lemmas of our proof. Each of these three lemmas below shows that after time t_{max} our algorithm performs the perpetual exploration in a self-stabilizing way for a specific subclass of connected-over-time rings.

Lemma 4.13. *Algorithm 3 is a perpetual exploration algorithm for the class of static rings of arbitrary size using three robots.*

Proof. Assume that $\mathcal{G}$ is a static ring. This implies that for all t in τ all the edges of the ring are always present. Thus at each round t the robots are edge-activated. A robot executing our algorithm considers at each round a specific direction. It is not possible for a robot to not consider a direction. This implies that during the Move phase of a round t, if a robot r on a node u considers a global direction such that the adjacent edge of u in this direction is present at time t, then r moves. During the Look phase of round $t + 1$ r is not on node u anymore.

Here as all the edges are always present at each instant time in $\mathcal{E}$, there is necessarily an adjacent edge to the node where a robot is located in the same global direction as the one considered by this robot. Thus at each instant time the predicates $ExistsEdgeOnCurrentDirection()$ of all the robots are true. This implies that during the call of the function UPDATE of each round t (such that $t > t_{max}$), the variables $HasMovedPreviousEdgeActivation$ of all the robots of the system are set to true.

For a robot to change its local direction, at least one of its predicate must be true. The predicate $WeAreStuckInTheSameDirection()$ of a robot can be true only if its variable $HasMovedPrevious$-$EdgeActivation$ is false. Similarly the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of a robot can be true only if its variable $HasMovedPreviousEdgeActivation$ is false. However as proved previously after time t_{max} all the robots of the system always have their variables $HasMovedPreviousEdgeActivation$ set to true. Thus from time $t_{max}+1$ a robot of $\mathcal{G}$ keep consider the same global direction.

As the three robots have a stable direction and consider respectively always the same global direction after t_{max}, as there always exists an adjacent edge to their current location in the global direction they consider, and as $\mathcal{G}$ has a finite size, then from time t_{max} all the robots succeed to visit infinitely often all the nodes of the static ring.

In conclusion we can say that whatever the size of $\mathcal{G}$ (which belongs to the class of connected-over-time rings) such that $\mathcal{G}$ is static, three fully-synchronous robots executing algorithm 3 permit to solve the perpetual exploration problem in $\mathcal{G}$. ☐

Lemma 4.14. *Algorithm 3 is a perpetual exploration algorithm for the class of edge-recurrent but non static rings of arbitrary size using three robots.*

Proof. Assume that $\mathcal{G}$ belongs to the class of edge-recurrent rings. This implies that all the edges of $\mathcal{G}$ are infinitely often present in the system.

We want to prove that the three robots executing our algorithm solve the perpetual exploration problem in $\mathcal{G}$.

By contradiction assume that this is not the case. This means that there exists at least one node w of $\mathcal{G}$ and a time $t_{\neg visited}$ in τ such that for all t greater or equal to $t_{\neg visited}$, w is not visited by any robot.

Consider the execution after time $t_{\neg visited}$.

Case 1: After time $\mathbf{t_{\neg visited}}$ there exists a 3-long-lived tower $\mathbf{T = (S, [t_s, t_e])}$ in $\mathcal{E}$.

Call t_{act} the first time in $[t_s, t_e[$ when the robots of S are edge-activated. By definition of a long-lived tower t_{first_act} exists.

At time t_{act} the three robots are on the same node thus their predicates $NumberOfRobotsOn$-$Node()$ is equal to 3. As at time t_{act} the three robots are edge-activated, during the call to the function UPDATE their variables $NumberRobotsPreviousEdgeActivation$ are updated with the values of their predicates $NumberOfRobotsOnNode()$, thus their variables $HasMovedPreviousEdgeActivation$ are set to 3. As long as the robots of S are forming T, each time they are edge-activated, using the same arguments that the one used at time t_{act} we can say that the variables $HasMovedPreviousEdgeActivation$ of the three robots are true. Moreover the variables of a robot are updated only during the Compute phases of times where this robot is edge-activated. Then from time $t_{act}+1$ to the Look phase of time t_e the variables $NumberRobotsPreviousEdgeActivation$ of the robots of S are equal to 3. So from time $t_{act}+1$ to the Look phase of time t_e the condition "$NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation$" cannot be true for the robots of S, as there are exactly 3 robots in the system. Thus the predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ cannot be true for the robots of T.

Assume that r and r' are two robots of T. Thus r and r' are robots among $\{r_1, r_2, r_3\}$. Call tl_r (respectively $tl_{r'}$) the transformed identifier of r (respectively of r'), and call i_r (respectively $i_{r'}$) the position in tl_r (respectively in tl'_r) considered by r (respectively by r') during the Look phase of time $t_{act}+1$. Call k the smaller integer either such that if r and r' have the same chirality then the bit at the position $((i_r+k) \pmod{|tl_r|})$ of tl_r and the bit at the position $((i_{r'}+k) \pmod{|tl_{r'}|})$ of $tl_{r'}$ are different or if r and r' have a different chirality then the bit at the position $((i_r+k) \pmod{|tl_r|})$ of tl_r and the bit at the position $((i_{r'}+k) \pmod{|tl_{r'}|})$ of $tl_{r'}$ are equal. By lemma 4.2 and lemma 4.3 we know that such a k exists.

By lemma 4.5 we know that from time $t_{act}+1$ the predicate $WeAreStuckInTheSameDirection()$ of all the robots of S are identical.

Between time $t_{act}+1$ and the Look phase of time t_e, the robots of S can have their predicates $WeAreStuckInTheSameDirection()$ either to true or to false. When their predicates $WeAreStuckInTheSameDirection()$ is false, as their predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ cannot be true, then their predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false. When the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ the robots of S are false, the three robots keep consider a same global direction, implying that they cannot break T. They can only break T when their predicates $WeAreStuckInTheSameDirection()$ is true. Moreover when the robots of S have their predicates $WeAreStuckInTheSameDirection()$ to true they execute the function GiveDirection. The function GiveDirection gives a direction to the robot executing it, according to the value of the current bit of its transformed label, and increments the position of the bit the robot considered. If between time $t_{act}+1$ and time t_e the number of times where the predicates $WeAreStuckInTheSameDirection()$ of the robots of T are true is less than k, then this implies that the tower T is infinite ($t_e = +\infty$). Indeed, the robots of T breaks the tower only when the number of times their predicates $WeAreStuckInTheSameDirection()$ is true is equal to k times, by definition of k.

Case 1.1: Between time $\mathbf{t_{act}+1}$ and time $\mathbf{t_e}$ the number of times the predicate WeAreStuckInTheSameDirection() of the robots of S is true is less than k.

Call $t_{no\_predicates} \geq t_{act}+1$ the first time such that for all t greater or equal to $t_{no\_predicates}$, for all robots r_i of T the predicate $WeAreStuckInTheSameDirection()(r_i, t)$ is false. This time exists as k is finite. As seen previously the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ is always false after time t_{act} and thus after time $t_{no\_predicates}$. Moreover by definition of $t_{no\_predicates}$ after this time the predicate $WeAreStuckInTheSameDirection()$ of the robots of T is false. Thus after time $t_{no\_predicates}$ the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the robots of T are false, thus they always consider the same global direction. As $\mathcal{G}$ belongs to the class of edge-recurrent rings, then each edge of $\mathcal{G}$ is infinitely often present in $\mathcal{E}$ and thus this implies that the robots of T sees infinitely often an adjacent edge to their current location in the global direction they consider. Thus the robots of T are infinitely often able to move in the global direction they consider. As from time $t_{no\_predicates}$ this global direction does not change and as $\mathcal{G}$ has a finite size this implies that w is visited after time $t_{\neg visited}$, which leads to a contradiction.

Case 1.2: Between time $\mathbf{t_{act}+1}$ and time $\mathbf{t_e}$ the number of times the predicate WeAreStuckInTheSameDirection() of the robots of S is true is equal to k.

In this case, the robots break the tower T. By lemma 4.8, we know that from time t_e it is not possible to have a 3-long-lived tower anymore in $\mathcal{E}$. We can only have configurations containing a 2-long-lived tower and a single robot, or containing 2-short-lived tower and a single robot, or containing 3 isolated robots. These three cases are treated in case 2 and 3.

Case 2: After time $\mathbf{t_{\neg visited}}$ there exists a 2-long-lived tower $\mathbf{T' = (S', [t'_s, t'_e])}$ in $\mathcal{E}$.

Assume without lost of generality that T' is composed of the robot r_1 and r_2. Call t'_{act} the first time in $[t'_s, t'_e[$ when the robots of T' are edge-activated. By definition of a long-lived tower t'_{act} exists.

Call ℓ_1 (respectively ℓ_2) the transformed identifier of r_1 (respectively of r_2), and call i_1 (respectively i_2) the position in ℓ_1 (respectively in ℓ_2) considered by r_1 (respectively by r_2) during the Look phase of time $t'_{act}+1$. Like, previously we introduce an integer k' corresponding to the smaller integer such that if r_1 and r_2 have the same chirality then the bit at the position $((i_1+k') \pmod{|\ell_1|})$ of ℓ_1 and the bit at the position $((i_2+k') \pmod{|\ell_2|})$ of ℓ_2 are different and if the two robots r_1 and r_2 have a different chirality then the bit at the position $((i_1+k') \pmod{|\ell_1|})$ of ℓ_1 and the bit at the position $((i_2+k') \pmod{|\ell_2|})$ of ℓ_2 are equal. By lemma 4.2 and lemma 4.3 we know that k' exists.

By lemma 4.5 we know that from time $t'_{act}+1$ the predicate $WeAreStuckInTheSameDirection()$ of all the robots of S' are identical.

Case 2.1: Between time $\mathbf{t'_{act}+1}$ and time $\mathbf{t_e}$ the number of times the predicate WeAreStuckInTheSameDirection() of the robots of S is true is less than k.

In this case the tower T' is infinite ($t'_e = +\infty$). Call $t_{not\_stuck} \geq t'_{act}+1$ the first time such that for all t greater or equal to t_{not_stuck}, for all robots r_i of T' the predicate $WeAreStuckInTheSameDirection()(r_i, t)$ is false. This time exists as k' is finite.

After time t'_{act} the robots r_1 and r_2 have the same value of predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$. Indeed each time in $[t'_s, t'_e]$ these two robots are edge-activated during the call to the function UPDATE the robots r_1 and r_2 have their values of variables $NumberRobotsPreviousEdgeActivation$ and $HasMovedPreviousEdgeActivation$ that are respectively filled with the values of their predicates $NumberOfRobotsOnNode()$ and $ExistsEdgeOnCurrentDirection()$. r_1 and r_2 are forming a 2-long-lived tower, therefore by definition of a long-lived tower and according to lemma 4.4, they are on a same node and are considering a same global direction from time t'_s to the Look phase of time t'_e, thus their respective values of predicates are equal from time t'_s to time t'_e. Moreover when the robots are not edge-activated their variables are not updated. Thus from time $t'_{act}+1$ to the Look phase of time t'_e the robots of S have the same values of predicates and variables thus they have the same values of predicates.

This implies that after time t_{not_stuck} either the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the robots of T' are false or their predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are true.

Case 2.1.1: After time $\mathbf{t_{not\_stuck}}$ the robots of $\mathbf{T'}$ have their predicates WeAreStuckInTheSameDirection() and IWasStuckOnMyNodeAndNowWeAreMoreRobots() false.

In this case we can use the same arguments than the one used in case 1.1 to prove that w is visited after time $t_{\neg visited}$.

Case 2.1.2: There exists a time greater or equal to $\mathbf{t_{not\_stuck}}$ at which the predicates IWasStuckOnMyNodeAndNowWeAreMoreRobots() of the robots of $\mathbf{T'}$ are true.

Call t_{meet} the first time greater or equal to t_{not_stuck} such that the predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the robots of T' are true. For the predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the robots of S to be true, a meeting must happen. The meeting can happen because either the two entities (the tower T' and the robot r_3) were considering opposite global directions during the Move phase of time $t_{meet} - 1$, and were able to move during this Move phase, or because the two entities are considering a same global direction during the Move phase of time $t_{meet} - 1$ and one of the entity was stuck during this Move phase.

Call t_{m_act} ($t_{m\_act} \geq t_{meet}$) the first time at which the three robots are edge-activated. As during the Look phase of time t_{meet} the three robots are on a same node for the first time after time t_{not_stuck}, this implies that at time $t_{meet} - 1$ the robots were edge-activated. Thus during the call to the function UPDATE of time $t_{meet} - 1$ the variables of the robots are filled with the values at time $t_{meet} - 1$ of their predicates. As the variables of a robot are only updated during the Compute phases of rounds where this robot is edge-activated, during the Look phase of time t_{m_act} the robots have the same values of variables as after the Compute phase of time $t_{meet} - 1$.

In the case where the meeting happens because the two entities where moving considering global opposite directions then during the Compute phase of time $t_{meet} - 1$ the variables $HasMovedPreviousEdgeActivation$ of the robots are set to true. Thus at the end of the Look phase of time t_{m_act} the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the robots are false, as their variables $HasMovedPreviousEdgeActivation$ are true. Thus the two entities conserve the global direction they were considering during the Move phase of round $tmeet - 1$ (as when the robot are not edge-activated they also conserve their direction). And so during the Move phase of the round t_{m_act} the two entities are considering opposite global directions and are able to separate them, as they are edge-activated at this time.

In the case where the meeting happens because the two entities were considering the same global direction and that one of the entity was not able to move then during the Compute phase of time $t_{meet} - 1$ the variable $HasMovedPreviousEdgeActivation$ of each robot of the entity that has moved is set to true, while it is set to false for each robot of the entity that has not moved. Thus at the end of the Look phase of round t_{m_act} each robot of the entity that has moved at time $t_{meet} - 1$ has their predicates $WeAreStuckInTheSameDirection()$

and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ false, as its variable *HasMovedPreviousEdgeActivation* is true. However the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of each robot of the other entity is true. Thus this last entity considers a direction opposite to the one it was considering during the Move phase of the round $t_{meet} - 1$. So during the Move of the round t_{m_act} the two entities are considering different global directions.

If after time t_{m_act} the robots of T' have their predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ false, then we can use the same arguments than the on used in case 2.1.1 to prove that w is visited after time $t_{\neg visited}$.

If there exists a time greater to t_{m_act} at which the predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the robots of T' are again true, then this implies that they meet again the robot r_3. Call t_{meet_bis} the first time greater or equal to $t_{m\_act}+1$ at which the robots of T' have their predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ to true. During the Look phase of time $t_{m\_act}+1$ the two entities (tower T' and the robot r_3) are considering two opposite global directions. For all times in $]t_{m\_act}+1, t_{meet\_bis}[$, r_3 is alone on the node where it is located (as t_{meet_bis} corresponds to the first time after time $t_{m\_act}+1$ where the three robots are on a same node), therefore its predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false. Thus between time $t_{m\_act}+1$ and time t_{meet_bis} r_3 has conserved the same global direction. Similarly, for all times in $]t_{m\_act}+1, t_{meet\_bis}[$, as r_1 and r_2 are not on the same node as r_3, the condition "$NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation$" is not true for them, and thus their predicates $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false. Moreover t_{m_act} is greater than t_{not_stuck}, by assumption between time $t_{m\_act}+1$ and time t_{meet_bis} the predicates $WeAreStuckInTheSameDirection()$ of the robots of T' are false. Thus between time $t_{m\_act}+1$ and time t_{meet_bis} r_1 and r_2 have conserved the same global direction as the one they considered during the Move phase of time t_{m_act}. Moreover we know that during the Look phase of time t_{act} the robots of T' and r_3 are together on a same node, and during the Move phase of time t_{act} they consider different global directions and are able to separate them as they are edge-activated. From the Move phase of time t_{act} until the Move phase of time $t_{meet\_bis}-1$ the tower and the robot r_3 are considering opposite global directions, then during the Look phase of time t_{meet_bis} they are again on a same node. This implies that all the nodes of $\mathcal{G}$ have been visited between time t_{m_act} and time t_{meet_bis}, which leads to a contradiction with the fact that w is not visited after time $t_{\neg visited}$.

Case 2.2: Between time $t'_{act}+1$ and time t_e the number of times the predicate WeAreStuckInTheSameDirection() of the robots of S is true is equal to k.

In this case, the tower T' breaks at time t'_e.

Case 2.2.1: After time t'_e there is no more 2-long-lived tower in $\mathcal{E}$.

In this case, as there is no more long-lived towers, by lemma 4.7 we know that all the configurations after time t'_e contain either one 2-short-lived tower and one isolated robot, or 3 isolated robots. These two cases are treated in case 3.

Case 2.2.2: After time t'_e there exists in $\mathcal{E}$ an other 2-long-lived tower $T" = (S", [t_s", t_e"])$.

If T" is such that $t_s" > t'_e + 1$ then by lemma 4.11 we know that between time t'_e and time $t_s"$, all the node of $\mathcal{G}$ have been visited (as all the edges of $\mathcal{G}$ have been crossed). Thus there is a contradiction with the fact that w is not visited after time $t_{\neg visited}$.
Consider the case where T" is such that $t_s" = t'_e+1$. Call $T_{first} = (S_{first}, [t_{sfirst}, t_{efirst}])$ the first 2-long-lived tower of $\mathcal{E}$.
If T' is such that $t_s > t_{sfirst}$ then according to lemma 4.12 all the nodes of $\mathcal{G}$ have been visited between time $t'_s - 1$ and time $t_s"$. This leads to a contradiction with the fact that w is not visited after time $t_{\neg visited}$.
If T' is not such that $t_s > t_{sfirst}$ then we can apply on T" the same arguments than the one used on T' (case 2) to prove that either there is a contradiction with the fact that w is not visited after time $t_{\neg visited}$ or to prove that we obtain a configuration from which there is no more 2-long-lived tower in $\mathcal{E}$ (case 2.2.1).

Case 3: After time $\mathbf{t_{\neg visited}}$ all the configurations of $\mathcal{E}$ contain either 3 isolated robots or one 2-short-lived tower and one isolated robot.

Case 3.1: After time $\mathbf{t_{\neg visited}}$ all the configurations of $\mathcal{E}$ contain 3 isolated robots.
Consider the robot r_1 and assume without lost of generality that it considers the clockwise direction during the Look phase of time $t_{\neg visited} + 1$. By assumption after time $t_{\neg visited}$ there is no towers, thus r_1 is alone one the node where it is located at each round $t > t_{\neg visited}$. Thus for all times $t > t_{\neg visited}$ its predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ are false. This implies that after time $t_{\neg visited}$, r_1 always considers the clockwise direction. Moreover as $\mathcal{G}$ is a dynamic graph that belongs to the class of edge-recurrent rings, then all the edges of $\mathcal{G}$ are infinitely often present. Therefore, r_1 is infinitely often able to move in the direction it considers. So as $\mathcal{G}$ has a finite size, this implies that r_1 succeed to visit node w, which leads to a contradiction.

Case 3.2: After time $\mathbf{t_{\neg visited}}$ it exists at some times 2-short-lived tower.

Case 3.2.1: After time $\mathbf{t_{\neg visited}}$ the three robots consider the same global direction.
By assumption there exists a time greater than $t_{\neg visited}$ at which a 2-short-lived tower is formed. By definition of a 2-short-lived tower once the robots that form this tower are edge-activated, they separate them. Call t_{end_tower} the time at which the robots of the 2-short-lived tower are edge-activated. As the robots executing our algorithm consider a direction at each round, the only way for the robots to separate them is to consider two opposite global directions during the Move phase of time t_{end_tower}. As the variables dir are updated only during the Compute phases of rounds, during the Look phase of time t_{end_tower} there are in the system two robots considering the same global direction while the third robot of the system is considering the opposite global direction. The case where the three robots of the system do not consider the same global direction is treated in case 3.2.2.

Case 3.2.2: After time $\mathbf{t_{\neg visited}}$ the three robots consider different global directions.

By assumption we know that there is no long-lived towers in $\mathcal{E}$ thus by lemma 4.9 we know that it is not possible to have again the three robots considering the same global direction.

Thus after time $t_{\neg visited}$, at each instant time a robot is considering the clockwise direction. Call r the robot which is located on node $x \neq w$ and which considers the clockwise direction during the Look phase of a time $t > t_{\neg visited}$.

We describe the situations in which r can be when it is edge-activated. Call t_{r_act} ($t_{r\_act} \geq t$) the first time after t when r is edge-activated. If during the Look phase of time t_{r_act} r is alone on its node, then at the end of the Look phase of time t_{r_act} its predicates *WeAreStuckInTheSameDirection*() and *IWasStuckOnMyNodeAndNowWeAreMoreRobots*() are false thus it considers the clockwise direction during the Move phase of round t_{r_act}. If at time t_{r_act} the adjacent edge to x in the clockwise direction is present then r moves in the clockwise direction. If at time t_{r_act} the adjacent edge to x in the clockwise direction is missing and the adjacent edge to x in the counter clockwise direction is present, then the robot r stays on node x.

If when r is edge-activated at time t_{r_act} it is with an other robot on node x, then we know that this two robots are forming a 2-short-lived tower. By definition of a 2-short-lived tower these two robots separate themselves during the Move phase of time t_{r_act}. As the robots executing our algorithm consider a direction at each round, for the two robots to separate them, they must consider opposite global directions during the Move phase of time t_{r_act}. If at time t_{r_act} the adjacent edge to x in the clockwise direction is present then one of the robot moves in the clockwise direction. If the adjacent edge to x in the clockwise direction is missing and that the adjacent edge in the counter clockwise direction is present, then one of the robots stays on node x.

Thus whatever the situation (during the Look phase of time t_{r_act} r is alone or forming a 2-short-lived tower) if an adjacent edge to x exists in the clockwise direction a robot is able to cross it, and if such an edge does not exists then a robot stays on node x. Moreover as all the edges are infinitely often present in $\mathcal{E}$, at a time $t_{appear} \geq t_{r\_act}$ an adjacent edge to x in the clockwise direction appears. Assume that $t_{appears}$ is the first time after t_{r_act} such that an adjacent edge to x in the clockwise direction exists. Using a recurrence on all the times where the robot(s) on x are edge-activated, we know that during the Move phase of time $t_{appears}$ there is a robot on x considering the clockwise direction. This robot is during the Look phase of time $t_{appears} + 1$ on node x_1 (where x_1 is the adjacent node of x in the clockwise direction) considering the clockwise direction. We can then iterate the same pitch for this robot on node x_1, and so on until reaching node x_k ($k \in \mathbb{N}^*$ and $k \leq (n-1)$) such that when the robot succeed to leave node x_k considering the clockwise direction it reaches node w. Thus there is a contradiction with the fact that w is not visited after time $t_{\neg visited}$.

All the cases has been treated, and all lead to contradictions with the fact that w is not visited after time $t_{\neg visited}$. This proves the lemma. □

Lemma 4.15. *Algorithm 3 is a perpetual exploration algorithm for the class of connected-over-time but not edge-recurrent rings of arbitrary size using three robots.*

Proof. Assume that there exists a time $t_{missing} \in \tau$ and exists an edge e of $\mathcal{G}$ such that for all t greater or equal to $t_{missing}$, e is not present in $\mathcal{E}$.

Call u and v the two adjacent nodes of e, such that if e was present in $\mathcal{G}$ a robot on node u would have to cross e in the clockwise direction to be located on v.

We want to prove that the three robots executing our algorithm solves the perpetual exploration problem in $\mathcal{G}$.

By contradiction assume that this is not the case. This means that there exists at least one node w of $\mathcal{G}$ and a time $t_{\neg visited}$ in τ such that for all t greater or equal to $t_{\neg visited}$, w is not visited anymore by any robot.

Consider the execution after time $t_{exec\_transition} = max\{t_{missing}, t_{\neg visited}\}$.

Case 1: After time $\mathbf{t_{exec\_transition}}$ there exists a 3-long-lived tower in $\mathcal{E}$.

According to lemma 4.6 this 3-long-liver tower is broken in finite time. Moreover once this tower is broken, according to lemma 4.8 it is not possible to have in $\mathcal{E}$ a configuration containing a 3-long-lived tower anymore. Thus after the time when the 3-long-lived tower is broken, there exists only configurations where there are either three isolated robots or tower of 2 robots and an isolated robot.

Case 2: After time $\mathbf{t_{exec\_transition}}$ there exists a 2-long-lived tower in $\mathcal{E}$.

According to lemma 4.6 this 2-long-lived tower is broken in finite time. Once this tower is broken, either there exists in the remainder of $\mathcal{E}$ a configuration containing a 2-long-lived tower T_{second} or not.

In the first case, by lemma 4.6 T_{second} is broken in finite time. This 2-long-lived tower corresponds at least to the second 2-long-lived tower of the execution, thus by lemma 4.11 and lemma 4.12 once T_{second} is broken it is not possible to have in $\mathcal{E}$ a configuration containing a 2-long-lived tower, as e is missing forever. Thus there is no long-lived towers after the breaking of T_{second}, so using lemma 4.7 we can say that in this case the execution is then composed of configurations containing either three isolated robots or one 2-short-lived tower and one isolated robot.

Similarly, in the second case by applying lemma 4.7 we can say that the robots are either isolated or forming 2-short-lived tower.

Case 3: After time $\mathbf{t_{exec\_transition}}$ all the configurations of E contain either 3 isolated robots or one 2-short-lived tower an one isolated robot.

From the cases 1 and 2 we can conclude that whatever the initial configuration that occurs at time $t_{exec\_transition}$ it leads to a configuration $\mathcal{C}_{stationary}$ from which the execution is only composed of configurations where the robots are either isolated or able to form 2-short-lived towers. Set $t_{stationary}$ the time at which $\mathcal{C}_{stationary}$ occurs in the $\mathcal{E}$.

Consider the execution after time $t_{exec\_stationary}$.

Either the three robots are considering the same global direction or not.

Case 3.1: After time $t_{exec\_stationary}$ the three robots are considering the same global direction.

Assume without lost of generality that the three robots consider the clockwise direction. A meeting necessarily happens between two of these robots. By contradiction, assume that this is not the case. $\mathcal{G}$ belongs to the class of connected-over-time rings, and e is an eventual missing edge, thus by definition of a connected-over-time ring, all the other edges are infinitely often present in $\mathcal{E}$. If there is no meeting, this implies that no robot is sufficiently enough time stuck on a node for an other robot to join it. However as e is missing forever, one of the robot succeed in finite time (by definition of connected-over-time rings) to reach node u. As long as this robot is alone on node u, its predicate $NumberOfRobotsOnNode()$ is equal to 1, and thus the conditions "$NumberOfRobotsOnNode() > 1$" or "$NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation$" cannot be true. Thus as long as this robot is alone on node u its predicates are false, and thus it does not change its direction, so it still considers the clockwise direction and therefore stays on node u as e is missing. As the two other robots of the system are also considering the clockwise direction and as there is no meeting by assumption, and as $\mathcal{G}$ belongs to the class of connected-over-time rings, they are able to reach in finite time node u. Thus there is necessarily a meeting between two robots on node u. Which leads to a contradiction with the fact that there is no meetings.

As the three robots are considering the clockwise direction a meeting between two robots happens necessarily because one of the robot was stuck on its node. Assume that the first meeting after $t_{exec\_stationary}$ happens between robots r_1 and r_2 at time t_{meet}. Assume that it is r_1 that was stuck during the Move phase of time $t_{meet} - 1$. Call time t_{act} ($t_{act} \geq t_{meet}$) the first time after t_{meet} when robots r_1 and r_2 are edge-activated. During the Look phase of time $t_{meet} - 1$ r_1 and r_2 were alone on their respective nodes otherwise their is a contradiction with the fact that t_{meet} is the first time after time $t_{exec\_stationary}$ where a meeting occurs. At time t_{meet} the robots r_1 and r_2 meet, thus this implies that at time $t_{meet} - 1$ they are edge-activated. Thus during the call to the function UPDATE the variables $HasMovedPreviousEdgeActivation$ and $NumberRobotsPreviousEdgeActivation$ of r_1 and r_2 are updated with the respective values of their predicates $ExistsEdgeOnCurrentDirection()$ and $NumberOfRobotsOnNode()$. So during the call to the function UPDATE of round $t_{meet} - 1$, the variables $NumberRobotsPreviousEdgeActivation$ of r_1 and r_2 are both set to 1. As by assumption r_1 does not move during the Move phase of time $t_{meet} - 1$, and that r_1 and r_2 meet at time t_{meet} this implies that r_2 moves during the Move phase of time $t_{meet} - 1$. Therefore during the call to the function UPDATE of round $t_{meet} - 1$ the variable $HasMovedPreviousEdgeActivation$ of r_2 is set true while the variable $HasMovedPreviousEdgeActivation$ of r_1 is set to false. Besides, the variables of a robot are updated only during the Compute phases of rounds where this robot is edge-activated. Thus at the end of the Look phase of time t_{act} the predicate $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of r_1 is true thus it changes its moving direction, while r_2 has its predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ to false, thus it does not change its moving direction. Thus during the Look phase of time $t_{act} + 1$, r_1 and r_2 are considering two opposite global directions.

Therefore during the Look phase of time $t_{act}+1$ two robots of the system are considering the same global direction while the other robot of the system is considering the opposite global direction. Moreover as we consider the execution after time $t_{exec\_stationary}$, all the configurations contain either isolated robots or 2-short-lived tower, thus using lemma 4.9 we can conclude that from time $t_{act}+1$ there are always two robots considering the same global direction while the other robot of the system considers the opposite global direction.

Case 3.2: After time $\mathbf{t_{exec\_stationary}}$ the three robots do not consider the same global direction.

Whatever the initial configuration that occurs at time $t_{exec\_transition}$ it leads to a configuration from which the execution is only composed of configurations where the robots are either isolated or able to form 2-short-lived towers and such that two robots are considering a global direction opposed to the one considered by the other robot of the system. Thus if we succeed to prove that the perpetual exploration is solved in this case, we can conclude that the perpetual exploration is solved if there is an eventual missing edge.

Case 3.2.1: w corresponds to node u.

We know by lemma 4.9 that at each instant time at least one robot is considering the clockwise direction in the system. We describe the situations in which this robot considering the clockwise direction can be when it is edge-activated. Call this robot r. Consider that the robot r is on a node $x \neq w$ at time $t > t_{exec\_stationary}$. Call t_{r_act} ($t_{r\_act} \geq t_{exec\_stationary}$) the first time after $t_{exec\_stationary}$ when r is edge-activated. As the variable dir is updated during the Compute phases of rounds where r is edge activated, then during the Look phase of t_{r_act} r still considers the clockwise direction. At time t_{r_act} if r is alone on its node, then its predicate $NumberOfRobotsOnNode()$ is equal to 1. Thus at the end of the Look phase of time t_{r_act} the condition "$NumberOfRobotsOnNode() > 1$" is false and similarly as the variable $NumberRobotsPreviousEdgeActivation$ is always greater or equal to 1, the condition "$NumberOfRobotsOnNode() > NumberRobotsPreviousEdgeActivation$" cannot be true. Thus at the end of the Look phase of time t_{r_act} the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of r are false, thus it does not change the direction it considers, therefore it considers the clockwise direction during the move phase of round t_{r_act}. If at time t_{r_act} the adjacent edge to x in the clockwise direction is present then r moves in the clockwise direction. If the adjacent edge to x in the clockwise direction is missing and that the adjacent edge in the counter clockwise direction is present, then the robot r stays on node x.

If when r is edge-activated at time t_{r_act} it is with an other robot on node x, then we know that this two robots are forming a 2-short-lived tower, thus they separate themselves during the Move phase of time t_{r_act}. Thus during the Move phase of time t_{r_act} one robot is considering the clockwise direction while the other one is considering the counter clockwise direction. If at time t_{r_act} the adjacent edge to x in the clockwise direction is present then one of the robot moves in the clockwise direction. If the adjacent edge to x in the clockwise direction is missing and that the adjacent edge in the counter clockwise direction is present, then one of the robots stays on node x.

Thus whatever the situation (r is alone on x at time t_{r_act}, or it is forming a 2-short-lived tower) if an adjacent edge to x exists in the clockwise direction a robot is able to move, and if such an edge does not exists then a robot considering the clockwise direction stays on node x. Moreover as all the edges except e are infinitely often present in the system, at a time $t_{appear} \geq t_{r\_act}$ an adjacent edge to x in the clockwise direction appears. Assume that t_{appear} is the first time after t_{r_act} such that there exists an adjacent edge to x in the clockwise direction. Using a recurrence on all the times the robots on x are edge-activated, we know that during the Move phase of time $t_{appears}$ there is a robot r' on x considering the clockwise direction. Call x_1 the adjacent node of x in the clockwise direction. Thus during the Look phase of time $t_{appear} + 1$, r' is on node x_1 considering the clockwise direction. We can then iterate this pitch to r' on node x_1. Using similar arguments we know that a robot considering the clockwise direction succeeds to reach the adjacent node x_2 of x_1 in the clockwise direction, and so on until reaching node x_i (i an integer such that $1 \leq i \leq (n-1)$) such that when the robot succeeds to leave node x_i considering the clockwise direction it reaches node w. Thus there is a contradiction with the fact that u is not visited after time $t_{\neg visited}$.

Case 3.2.2: w corresponds to node v.

This situation is symmetrical to the case 3.2.1. Thus using symmetrical arguments to the one used when w corresponds to node u we obtain a contradiction showing that v is visited at a time after $t_{\neg visited}$.

Case 3.2.3: w corresponds to a node different from u and v.

Note $\{v, \ldots, w_{k-1}, w, w_{k+1}, \ldots, u\}$ the nodes of $\mathcal{G}$ in the clockwise direction from node v, with k an integer such that $2 \leq k \leq (n-3)$.

Call R_{vw} the set of all the robots situated on a node among $\{v, \ldots, w_{k-1}\}$ and call R_{wu} the set of all the robots situated on a node among $\{w_{k+1}, \ldots, u\}$. We have $|S_{vw}| + |S_{wu}| = 3$.

If w is not visited after time $t_{\neg visited}$, this means that there is no robot of R_{vw} considering the clockwise direction on node w_{k-1} while the edge linking w_{k-1} to w is present, and there is no robot of R_{wu} considering the counter clockwise direction and located on node w_{k+1} while the edge linking w_{k+1} to w is present.

Without lost of generality we assume that $|R_{vw}|$ is equal to three or two. When R_{vw} contains one or zero robot then R_{wu} contains two or three robots. The case where R_{wu} contains two or three robots is symmetric to the case where R_{vw} contains two or three robots. Thus if we prove that in the case where R_{vw} contain two or three robots, w is visited, using symmetrical arguments we can prove that w is also visited when R_{wu} contains two or three robots.

Case 3.2.3.1: There are 3 robots among nodes $\{\mathbf{v}, \ldots, \mathbf{w_{k-1}}\}$.

We can use similar arguments than the one used for the case 3.2.2 (case where w corresponds to node u) to show that there is a contradiction, as w is reached by a robot after time $t_{\neg visited}$.

Case 3.2.3.2: There are 2 robots among nodes $\{\mathbf{v}, \ldots, \mathbf{w_{k-1}}\}$.

This implies that there is one robot among nodes $\{w_{k+1}, \ldots, u\}$. Assume without lost of generality that the robots r_1 and r_2 belong to R_{vw} while r_3 belongs to

R_{wv}.
As r_3 is on a node among $w_{k+1}, \ldots, u\}$ and as e is missing forever after time $t_{missing}$, if there exists a time after time $t_{exec\_stationary}$ at which r_3 is on a node among $\{v, \ldots, w_{k-1}, w$ this implies that w has been visited by r_3. Thus assume that after time $t_{exec\_stationary}$ r_3 stays on a node among $\{w_{k+1}, \ldots, u\}$. Similarly, if r_1 or r_2 is on a node among $\{w, w_{k+1}, \ldots, u\}$ this implies that w has been visited by at least one of these two robots. Thus we assume that after time $t_{exec\_stationary}$ r_1 and r_2 stay on nodes among $\{v, \ldots, w_{k-1}\}$.
We know by lemma 4.9 that at each instant time two robots are considering the same global direction, and that the third robot of the system consider an opposite global direction.

Case 3.2.3.2.1: $\mathbf{r_1}$ and $\mathbf{r_2}$ consider opposite global directions.

Assume that r_1 considers the clockwise direction while r_2 considers the counter clockwise direction.
As long as r_1 is alone on a node x ($x \in \{v, \ldots, w_{k-1}\}$) considering the clockwise direction, when it is edge activated at time $t_{r_1\_act}$ its predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNow$-$WeAreMoreRobots()$ are false thus it keeps consider the clockwise direction during the Move phase of round $t_{r_1\_act}$. Thus if at time $t_{r_1\_act}$ the adjacent edge to x in the clockwise direction is present then r_1 moves in the clockwise direction. Similarly if at time $t_{r_1\_act}$ the adjacent edge to x in the clockwise direction is missing and the adjacent edge to x in the counter clockwise direction is present, then r_1 stays on node x.
If at time $t_{r_1\_act}$ r_1 is for the first time since $t_{exec\_stationary}$ edge-activated with an other robot on node x, we know that the tower they form corresponds to a 2-short-lived tower. As by assumption after time $_{exec\_stationary}$ r_3 is always on a node among $\{w_{k+1}, \ldots, u\}$, while r_1 and r_2 are always on nodes among $\{v, \ldots, w_{k-1}\}$, the meeting happens necessarily between robots r_1 and r_2. Moreover by assumption before the meeting the two robots consider opposite global directions. This implies that both of these two robots have moved during the Move phase of time $t_{meet} - 1$ (where $t_{meet} \leq t_{r_1\_act}$ corresponds to the last time before $t_{r_1\_act}$ where the robots r_1 and r_2 are on a same node without being necessarily edge-activated). As at time t_{meet} r_1 and r_2 are on a same node for the first time, this implies that during time $t_{meet} - 1$ they are edge-activated. Besides, as r_1 and r_2 are moving during the Move phase of time $t_{meet} - 1$, during the call to the function UPDATE of round $t_{meet} - 1$ the variables $HasMovedPreviousEdgeActivation$ of the two robots are set to true. Moreover the variables of a robot are only updated during the Compute phases of rounds where it is edge-activated. Thus at the end of the Look phase of round $t_{r_1\_act}$ the predicates $WeAreStuckInTheSameDirection()$ and $IWasStuckOnMyNodeAndNowWeAreMoreRobots()$ of the robots are false as their variables $HasMovedPreviousEdgeActivation$ are true. Thus they keep consider their respective directions. If at time $t_{r_1\_act}$ the adjacent edge to x in the clockwise direction is present then r_1 moves in the clockwise direction. If the adjacent edge to x in the clockwise direction is missing and the adjacent

edge to x in the counter clockwise direction is present, then r_1 stays on node x.

Thus whatever the situation if an adjacent edge to x exists in the clockwise direction r_1 is able to move, and if such an edge does not exists then r_1 stays on node x. Moreover as all the edges except e are infinitely often present in the system, at a time greater or equal to $t_{r_1\_act}$ an adjacent edge to x in the clockwise direction appears. The first time after $t_{r_1\_act}$ such an edge appears r_1 is on node x and it is considering the clockwise direction thus it succeeds to move. We can iterate this pitch when r_1 is on node x_1 (the adjacent node of x in the clockwise direction), and then when it is on node x_2 (the adjacent node of x_1 in the clockwise direction), and so on until reaching node w_{k-1}. On node w_{k-1} we repeat the same arguments and show that r_1 stays on node w_{k-1} considering the clockwise direction until an adjacent edge in the clockwise direction to this node exists. Thus r_1 reaches node w. Therefore there is a contradiction with the fact that we cannot reach node w.

Case 3.2.3.2.2: $\mathbf{r_1}$ and $\mathbf{r_2}$ consider the same global direction.

Now assume that the robots r_1 and r_2 consider the same global direction. As e is missing forever, it exists a time t'_{meet} at which the two robots r_1 and r_2 meet because one of them is stuck on a node. As seen previously (in case 3.1) the robot that is stuck changes its direction during the Compute phase of time t_{m_act} (where t_{m_act} is the first time greater or equal to t'_{meet} when the two robots r_1 and r_2 are edge-activated) while the other robot still consider its direction. During the Move phase of time t_{m_act} the two robots separate them (as they consider two opposite global directions and as they are edge-activated at this time). As the variable *dir* is only updated during the Compute phases of rounds, during the Look phase of time $t_{m\_act}+1$ the two robots still consider two opposite global directions and are among nodes $\{v, \dots w_{k-1}, w$. If one of the robots is on w at time t_{m_act+1} then w is visited after time $t_{\neg visited}$, otherwise we can apply the same reasoning than the one described in case 3.2.3.2.1 to show that w is visited after time $t_{\neg visited}$.

Thus whatever the number of robots present in R_{vw} and in R_{wu} the node w is visited.

We can thus conclude that if there exists an eventual missing edge, then the robots explore perpetually $\mathcal{G}$. □

The end of the road. To conclude the proof, it is sufficient to observe that a connected-over-time ring is by definition either static, edge-recurrent but non static, or connected-over-time but not edge-recurrent. As we prove the self-stabilization of our algorithm in these three cases in Lemmas 4.13, 4.14, and 4.15, we can claim the following final result.

Theorem 4.1. *Algorithm 3 is a self-stabilizing perpetual exploration algorithm for the class of connected-over-time rings of arbitrary size using three robots.*

Proof. Consider $\mathcal{G}$ a dynamic graph of any size that belongs to the class of connected-over-time rings.

First of all, note that even if our robots can start in a non coherent state, it exists a time t_{max} from which all the robots of the system are in a coherent state (See lemma 4.1).

From the time where the three robots have coherent state, they succeed to solve the perpetual exploration problem in $\mathcal{G}$.

Indeed, by definition of connected-over-time rings, $\mathcal{G}$ can be either a dynamic ring where eventually one edge is missing while all the other edges are infinitely often present, or it can be a static ring or it can belong to the class of edge-recurrent rings (all the edges are infinitely often present, there is no eventual missing edge).

In the first case, lemma 4.13 shows that the three robots executing algorithm 3 succeed to solve the perpetual exploration problem.

In the second case, the lemma 4.15 states that the three robots performing our algorithm solve the perpetual exploration problem.

In the latter case, it is the lemma 4.14 that states that the three robots executing our algorithm solve the perpetual exploration problem.

Thus whatever the case considered the three robots executing our algorithm succeed to solve the perpetual exploration problem. We conclude that algorithm 3 solves the perpetual exploration problem for dynamic graphs of any size that belong to the class of connected-over-time rings using three fully-synchronous robots. □

5 Conclusion

In this paper, we addressed the open question: "Is it possible to achieve self-stabilization for swarm of robots evolving in highly dynamic graphs?". We answered positively to this question by providing a self-stabilizing algorithm for three synchronous robots that perpetually explore any connected-over-time ring, *i.e.,* any dynamic ring with very weak assumption on connectivity: every node is infinitely often reachable from any another one without any recurrence, periodicity, nor stability assumption.

In addition to the above contributions, our algorithm overcomes the robot networks state-of-the-art in a couple of ways. First, it is the first algorithm dealing with highly dynamic graphs. All previous solutions made some assumptions on periodicity or on all-time connectivity of the graph. Second, it is the first self-stabilizing algorithm for the problem of exploration, either for static or for dynamic graphs.

This work opens an interesting field of research with numerous open questions. First, we should investigate the necessity of every assumption made in this paper. For example, we assumed that robots are synchronous. Is this problem solvable with asynchronous robots? Second, we can investigate the issue of the number of robots. What are the minimal/maximal number of robots to solve the problem? It would be worthwhile to explore other problems in this rather complicated environment, *e.g.,* gathering, leader election, *etc.*. It may also be interesting to consider other classes of dynamic graphs and other classes of faults, *e.g.,* crashes of robots, Byzantine failures, *etc.*.

References

[1] R Baldoni, F. Bonnet, A. Milani, and M. Raynal. On the solvability of anonymous partial grids exploration by mobile robots. In *International Conference on Principles of Distributed*

Systems (OPODIS), pages 428–445, 2008.

[2] L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil. Exclusive perpetual ring exploration without chirality. In *International Symposium on Distributed Computing (DISC)*, pages 312–327, 2010.

[3] L. Blin, M. Potop-Butucaru, and S. Tixeuil. On the self-stabilization of mobile robots in graphs. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 301–314, 2007.

[4] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[5] J. Chalopin, P. Flocchini, B. Mans, and N. Santoro. Network exploration by silent and oblivious robots. In *Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 208–219, 2010.

[6] A. Datta, A. Lamani, L. Larmore, and F. Petit. Ring exploration by oblivious agents with local vision. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 347–356, 2013.

[7] G. Di Luna, S. Dobrev, P. Flocchini, and N. Santoro. Live exploration of dynamic rings. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 570–579, 2016.

[8] E. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communication of the ACM*, 17(11):643–644, 1974.

[9] K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.

[10] S. Dolev. *Self-stabilization*. MIT Press, March 2000.

[11] S. Dubois, M.-H. Kaaouachi, and F. Petit. Enabling minimal dominating set in highly dynamic distributed systems. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 51–66, 2015.

[12] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 105–118, 2007.

[13] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. *Theoretical Computer Science*, 411(14-15):1583–1598, 2010.

[14] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. How many oblivious robots can explore a line. *Information Processing Letters*, 111(20):1027–1031, 2011.

[15] P. Flocchini, B. Mans, and N. Santoro. Exploration of periodically varying graphs. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 534–543, 2009.

[16] D. Ilcinkas, R. Klasing, and A. Mouhamadou Wade. Exploration of constantly connected dynamic graphs based on cactuses. In *Colloquium on Structural Information & Communication Complexity (SIROCCO)*, pages 250–262, 2014.

[17] D. Ilcinkas and A. Mouhamadou Wade. On the power of waiting when exploring public transportation systems. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 451–464, 2011.

[18] D. Ilcinkas and A. Mouhamadou Wade. Exploration of the t-interval-connected dynamic graphs: The case of the ring. In *Colloquium on Structural Information & Communication Complexity (SIROCCO)*, pages 13–23, 2013.

[19] R. Klasing, E. Markou, and A. Pelc. Gathering asynchronous oblivious mobile robots in a ring. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 744–753, 2006.

[20] F Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *Symposium on the Theory of Computing (STOC)*, pages 513–522, 2010.

[21] M. Potop-Butucaru, M. Raynal, and S. Tixeuil. Distributed computing with mobile robots: An introductory survey. In *International Conference on Network-Based Information Systems (NBiS)*, pages 318–324, 2011.

[22] C. Shannon. Presentation of a maze-solving machine. *8th Conference of the Josiah Macy, Jr. Foundation*, pages 173–180, 1951.

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

[24] S. Tixeuil. *Algorithms and Theory of Computation Handbook*, chapter Self-stabilizing Algorithms, pages 26.1–26.45. Chapman & Hall. CRC Press, Taylor & Francis Group, November 2009.

[25] B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.