Gathering of Six Robots on Anonymous Symmetric Rings

Gianlorenzo D'Angelo[1], Gabriele Di Stefano[1], and Alfredo Navarra[2]

[1] Dipartimento di Ingegneria Elettrica e dell'Informazione, Università degli Studi dell'Aquila, Italy. gianlorenzo.dangelo@univaq.it gabriele.distefano@univaq.it
[2] Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Italy. navarra@dmi.unipg.it

Abstract. The paper deals with a recent model of robot-based computing which makes use of identical, memoryless mobile robots placed on nodes of anonymous graphs. The robots operate in Look-Compute-Move cycles; in one cycle, a robot takes a snapshot of the current configuration (Look), takes a decision whether to stay idle or to move to one of its adjacent nodes (Compute), and in the latter case makes an instantaneous move to this neighbor (Move). Cycles are performed asynchronously for each robot.
In particular, we consider the case of only six robots placed on the nodes of an anonymous ring in such a way they constitute a symmetric placement with respect to one single axis of symmetry, and we ask whether there exists a strategy that allows the robots to gather at one single node. This is in fact the first case left open after a series of papers [3, 6–8] dealing with the gathering of oblivious robots on anonymous rings. As long as the gathering is feasible, we provide a new distributed approach that guarantees a positive answer to the posed question. Despite the very special case considered, the provided strategy turns out to be very interesting as it neither completely falls into symmetry-breaking nor into symmetry-preserving techniques.

1 Introduction

We study one of the most fundamental problems of self-organization of mobile entities, known in the literature as the *gathering* problem. Robots, initially situated at different locations, have to gather at the same location (not determined in advance) and remain in it. We consider the case of an anonymous ring in which neither nodes nor links have any labels. Initially, some of the nodes of the ring are occupied by robots and there is at most one robot in each node. Robots operate in Look-Compute-Move cycles. In each cycle, a robot takes a snapshot of the current global configuration (Look), then, based on the perceived configuration, takes a decision to stay idle or to move to one of its adjacent nodes (Compute), and in the latter case makes an instantaneous move to this neighbor (Move). Cycles are performed asynchronously for each robot. This means that the time between Look, Compute, and Move operations is finite but unbounded,

and is decided by the adversary for each robot. The only constraint is that moves are instantaneous, and hence any robot performing a Look operation sees all other robots at nodes of the ring and not on edges. However, a robot r may perform a Look operation at some time t, perceiving robots at some nodes, then Compute a target neighbor at some time $t' > t$, and Move to this neighbor at some later time $t'' > t'$, at which some robots are in different nodes from those previously perceived by r at time t because in the meantime they performed their Move operations. Hence, robots may move based on significantly outdated perceptions. We stress that robots are memoryless (oblivious), i.e., they do not have any memory of past observations. Thus, the target node (which is either the current position of the robot or one of its neighbors) is decided by the robot during a Compute operation solely on the basis of the location of other robots perceived in the previous Look operation. Robots are anonymous and execute the same deterministic algorithm. They cannot leave any marks at visited nodes, nor send any messages to other robots.

We remark that the Look operation provides the robots with the entire ring configuration. Moreover, it is assumed that the robots have the ability to perceive whether there is one or more robots located at a given node of the ring. This capability of robots is important and well-studied in the literature under the name of *multiplicity detection* [3, 6, 7, 2, 4, 5, 9]. In fact, without this capability, many computational problems (such as the gathering problem considered herein) are impossible to solve for all non-trivial starting configurations.

1.1 Related Work and Our Results

Under the Look-Compute-Move model, the gathering problem on rings was initially studied in [7], where certain configurations were shown to be gatherable by means of symmetry-breaking techniques, but the question of the general-case solution was posed as an open problem. In particular, it has been proved that the gathering is not feasible in configurations with only two robots, in *periodic* configurations (invariable under non-trivial rotation) or in those with an axis of symmetry of type edge-edge. A configuration is called *symmetric* if the ring has a geometrical *axis of symmetry*, which reflects single robots into single robots, multiplicities into multiplicities, and empty nodes into empty nodes. A symmetric configuration is not periodic if and only if it has exactly one axis of symmetry [7]. A symmetric configuration with an axis of symmetry has an *edge-edge symmetry* if the axis goes through (the middles of) two edges; it has a *node-edge symmetry* if the axis goes through one node and one edge; it has a *node-node symmetry* if the axis goes through two nodes; it has a *robot-on-axis symmetry* if there is at least one node on the axis of symmetry occupied by a robot. For an odd number, all the gatherable configurations have been solved. For an even number of robots greater than two, if the initial configuration is not periodic, the feasibility of the gathering has been solved, except for some types of symmetric configurations. In [6], the attention has been devoted to these left open symmetric cases. The new proposed technique was based on preserving symmetry rather than breaking it, and the problem was solved when the number of robots is greater than

18. This left open the case of an even number of robots between 4 and 18, as the case of just 2 robots is not gatherable [7]. The case of 4 robots has been solved in [3, 8]. Moreover, in [3] all the cases of $2k$ robots with $k \geq 2$ have been addressed when the initial axis of symmetry is of type robot-on-axis. Hence, the first case left open concerns 6 robots with an initial axis of symmetry of type node-edge, or node-node. In this paper, we address the problem of 6 robots and provide a distributed algorithm that gathers the robots when starting from any symmetric configuration of type node-edge, or node-node.

2 Definitions and Notation

We consider an n-node anonymous ring without orientation. Initially, exactly six nodes of the ring are occupied by robots. During a Look operation, a robot perceives the relative locations on the ring of multiplicities and single robots. We remind that a multiplicity occurs when more than one robot occupy the same location.

The current configuration of the system can be described in terms of the view of a robot r which is performing the Look operation at the current moment. We denote a configuration seen by r as a tuple $Q(r) = (q_0, q_1, \ldots, q_j)$, $j \leq 5$, which represents the sequence of the numbers of free consecutive nodes interleaved by robots when traversing the ring either in clockwise or in anti-clockwise direction, starting from r. When comparing two configurations, we say that they are equal regardless the traversing orientation. Formally, given two configurations $Q = (q_0, q_1, \ldots, q_j)$ and $Q' = (q'_0, q'_1, \ldots, q'_j)$, we have $Q = Q'$ if and only if $q_0 = q'_0$, $q_1 = q'_1$, ..., and $q_j = q'_j$ or $q_0 = q'_j$, $q_1 = q'_{j-1}$, ..., and $q_j = q'_0$. For instance, in the configuration of Fig. 1, node x can see the configuration $Q(x) = (1, 2, 1, 3, 1, 2)$ or $Q(x) = (2, 1, 3, 1, 2, 1)$. In our notation, a multiplicity is represented as $q_i = -1$ for some $0 \leq i \leq j$, disregarding the number of robots in the multiplicity. We also assume that the initial configuration is symmetric and not periodic. In this paper, we are interested only in node-edge and node-node symmetries without robots on axis as the other cases are either solved or not gatherable.

We can then represent a symmetric configuration independently from the robot view as in Fig. 1. In detail, without multiplicities, the ring is divided by the robots into 6 intervals: A, B, C, B', C', and D with a, b, c, b, c, and d free nodes, respectively. In the case of node-edge symmetry, A is the interval where the axis passes through a node and D is the interval where the axis passes through an edge; in the case of node-node symmetry, A and D are the intervals such that either $a < d$ or $a = d$ and $b < c$; the case where $a = d$ and $b = c$ cannot occur as it generates two axis of symmetry. Note that, in the case of node-node symmetry, a and d are both odd, while, in the case of node-edge symmetry, a is odd and d is even. The axis of symmetry passing trough intervals A and D is denoted as $\overrightarrow{DA}$ when directed from D to A. The direction of the axis distinguishes A and D. When the direction is not specified or it is clear by the context, we denote it by $\overline{DA}$ or $\overline{AD}$. We denote as: x (x', resp.) the robots

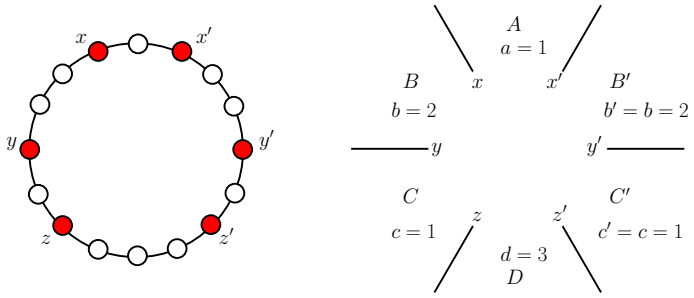

Fig. 1. A symmetric configuration and its representation.

between A and B (B', resp.); y (y', resp.) the robots between B and C (B' and C', resp.); z (z', resp.) the robots between C (C', resp.) and D, see Fig. 1.

A robot $r \in \{x, y, z, x', y', z'\}$ can perform only two moves: it moves *up* ($r\uparrow$) if it goes towards A; it moves *down* ($r\downarrow$) if it goes towards D. Note that, in general a robot could not be able to distinguish intervals A and D. However, we show that in our algorithm this is not the case and hence a robot r is always allowed to distinguish between moves $r\uparrow$ and $r\downarrow$.

3 Resolution Algorithm

The main idea of the algorithm is to perform moves $x\uparrow$, $x'\uparrow$, $y\uparrow$ and $y'\uparrow$, with the aim of preserving the symmetry and gathering in the middle node of interval A, where the axis is directed. In some special cases, it may happen that the axis of symmetry changes at run time. Before multiplicities are created, the algorithm in a symmetric configuration allows only two robots to move in order to create a new symmetric configuration.

In the general case, the algorithm compares b and d, and performs a pair of moves such that if $b > d$, then b is enlarged, while, if $b < d$, then b is reduced. In this way, the axis of symmetry and its direction do not change. In fact, in order to obtain a new axis of symmetry between $\overline{BC'}$ or $\overline{CB'}$ after one move, b must be equal to d. When $b > d$, $x\uparrow$ and $x'\uparrow$ are performed, while, when $b < d$, $y\uparrow$ and $y'\uparrow$ are performed. In both cases, (apart for some special cases) the ordering between b and d is maintained in the new configuration. Eventually, either one multiplicity is created at the middle node of the original interval A by means of robots x and x', or two symmetric multiplicities are created on the positions originally occupied by x and x' by means of the moves of y and y', respectively. In the second case, the two multiplicities will move up again to the middle node of the original interval A by allowing at most 4 robots to move all together. Once such a multiplicity has been created, the remaining robots join it, and conclude the gathering. In the special case of $b = d$, which can only happen in the initial configuration, the algorithm tries to break this equality by enlarging or reducing d by means of either $z\uparrow$ and $z'\uparrow$ (when $C > 0$) or $z\downarrow$ and $z'\downarrow$ (when $C = 0$

and $D > 0$). The special cases when $C = D = 0$ require specific arguments. The remainder of the section is structured as follows: in the next subsection we give the algorithm for the general case, then we describe the algorithm in some special cases, namely when a multiplicity is created on the middle node of interval A, when 2 non-symmetric multiplicities are created, and when $n = 7$.

3.1 General case

In this section we describe the algorithm as it is performed by a single robot. In general, the algorithm allows only two types of configurations: those which are symmetric and those which differ from a symmetric one only by one move. As already observed, we do consider only node-edge and node-node symmetries, also excluding possible robots-on-axis symmetries. In the Look phase, a robot r obtains the tuple $Q(r)$ and performs Procedure GATHER, see Fig. 2. First of all, it checks whether there is a multiplicity containing more than two robots, this is realized by counting the number of elements in $Q(r)$, as a multiplicity is always counted as one interval of dimension -1. If the number of elements is less than five or two non-symmetric multiplicities have been created in a ring of more than 7 nodes, then Procedure MULTIPLICITY is invoked, see Fig. 5. Otherwise, the algorithm checks whether the ring is composed of seven nodes (that is $\sum_i q_i = 1$) in which case, Procedure SEVEN is invoked, see Fig. 6. Excluding these special cases, the main activity of the algorithm is performed by means of Procedure MOVING and Function IDENTIFICATION, see Fig.s 3 and 4, respectively.

Function IDENTIFICATION computes the values a, b, c and d of the symmetric configuration that might be the one in input, or the one before the last move. The function also returns the identity of r among x, y and z, and the boolean variable `move` which indicates whether r is allowed to move or not. Due to symmetry arguments, a robot cannot distinguish of being, for instance, x or x'. However, if the current configuration is at one step from a symmetry of interest, the robot can recognize whether it is the one which has already performed the move, or if it has to perform it in order to re-establish the desired symmetry. Finally, by means of Procedure MOVING, the Move phase of the robot is realized, if it is allowed to do it.

We now describe in details Procedure MOVING and Function IDENTIFICATION, while the special cases addressed by Procedures MULTIPLICITY and SEVEN will be described in Subsection 3.2.

Moving algorithm Procedure MOVING takes as input the intervals describing a symmetric configuration and the identity of the robot which is performing its Move phase. When $b < d$ or $b > d$, robot r moves up if its identity is either x (code line 2) or y (code line 5), respectively. When $b = d$, if $c > 0$, r moves up if it is identified as z (code line 8). Otherwise (that is, when $b = d$ and $c = 0$), in order to avoid to create two multiplicities at the extremes of interval D, robot r moves down if it is identified as z and $d > 0$ (code line 11). The last case considers when $b = d = c = 0$. In this case, it results $a > 1$ as otherwise the

```
Procedure: GATHER
Input: (q_0, q_1, ..., q_j), j ≤ 5
1  if j < 5 OR (∑_i q_i > 1 AND there are 2 multiplicities AND the configuration
   is not symmetric) then
2      MULTIPLICITY((q_0, q_1, ..., q_j));
3  else
4      if ∑_i q_i = 1 then SEVEN;
5      else
6          (a, b, c, d, r, move) := IDENTIFICATION((q_0, q_1, ..., q_5));
7          if a = −1 then MULTIPLICITY((q_0, q_1, ..., q_5));
8          else
9              if move then MOVING(a, b, c, d, r);
```

Fig. 2. General algorithm executed each time a robot wakes up.

ring would be composed of seven nodes. Then, r moves up if it is identified as x (code line 13).

Identification Function IDENTIFICATION implements the correct "positioning" of a robot with respect to the perceived configuration. To this aim, it makes use of Function SYMMETRIC that checks whether an input configuration is symmetric, and in the positive case, it returns the role of the robot in such a configuration and the values of a, b, c and d obtained by rotating k times its view. The behavior of SYMMETRIC will be described later in this section, while the pseudo-code can be found in [1]. We now focus on Function IDENTIFICATION. At code lines 2–3 of Function IDENTIFICATION, the robot checks whether the perceived configuration is symmetric and, in the positive case, it sets the variable `move` to true. If the configuration is not symmetric, it must be at one step from a symmetric one. Indeed, it is at one step from the symmetric configuration before the last move, and from the symmetric configuration obtained by the move symmetric to the last one. In some special cases, it may also happen that the current configuration is at one step from other symmetric configurations, but we know how to distinguish the "good" one. In detail, at code lines 4–15 the function checks if the robot r is allowed to move of one node from the current configuration $Q(r)$. The configuration $Q^i(r)$ of r after a move $i \in \{-1, 1\}$ is computed at code line 5 by adding i to q_0 and subtracting i to q_5. If $Q^i(r)$ is symmetric (code line 6), then, given the role of r, the algorithm retrieves the symmetric configuration $\bar{Q}^i(r)$ that should have been occurred before the moves of r and the symmetric robot r' (lines 7–13). In the pseudo-code, variables $\alpha[i]$, $\beta[i]$, $\gamma[i]$, $\delta[i]$, ($a[i]$, $b[i]$, $c[i]$, $d[i]$, resp.) and $r[i]$, $i \in \{-1, 1\}$, denote the values of a, b, c, d, and r related to configuration $Q^i(r)$ ($\bar{Q}^i(r)$, resp.). Variable `dir` $\in \{1, -1\}$ indicates the direction where node r is moving when passing from $Q(r)$ to $Q^i(r)$ that is, if `dir` $= 1$, then r is moving up, otherwise it is moving down. In order to check whether $\bar{Q}^i(r)$ is an admissible configuration, the function simulates one step of

Procedure: MOVING
Input : a, b, c, d, and $r \in \{x, y, z\}$

```
1  if b > d then
2  |  if r = x then x↑ ;
3  else
4  |  if b < d then
5  |  |  if r = y then y↑ ;
6  |  else
7  |  |  if c > 0 then
8  |  |  |  if r = z then z↑ ;
9  |  |  else
10 |  |  |  if d > 0 then
11 |  |  |  |  if r = z then z↓ ;
12 |  |  |  else
13 |  |  |  |  if  r = x then x↑ ;
```

Fig. 3. Algorithm performing the Move phase of a robot.

the moving algorithm, code line 15. If the tested move was allowed, then variable `move`[i] is set to true. If the robot is allowed to do exactly one move i, then code lines 16–19 return the values $a[i]$, $b[i]$, $c[i]$, $d[i]$, and $r[i]$. If the robot is allowed to do both the tested moves, then code lines 20–23 return the values $a[i]$, $b[i]$, $c[i]$, $d[i]$ and $r[i]$ where i is the move such that the identity of $r[i]$ is either x or z. In fact, there might be only two cases where the robot can perform the two opposite moves (see the correctness proof of the algorithm in [1]). In the first case, r can behave as x or y (i.e., $r[i] = x$ and $r[-i] = y$ for some $i \in \{-1, 1\}$). In the second case it always behaves as z (i.e., $r[1] = r[-1] = z$). In the former case, we force r to behave always as x. In the latter case, r can move to any direction, indifferently. These situations might lead to the change of the original axis of symmetry. However, such a change can occur only once.

Note that, Function IDENTIFICATION works correctly also when multiplicities occur and the configuration is only at one step from symmetry. However, if a robot r belongs to a multiplicity formed by x and y, then we require r to provide its view in the form $(-1, c, d, c, b, a)$. This can always be obtained because b is either 0 or -1 while $d > 0$ since before creating the multiplicities the moves $y\uparrow$ have been performed, i.e. $b < d$. Actually, in the case two non-symmetric multiplicities occur, the right moves will be determined by Procedure MULTIPLICITY that either will bring the configuration to a symmetric one or only at one step from symmetry. At that point, Function IDENTIFICATION is again invoked. This alternation will continue until one multiplicity containing more than 3 robots does occur.

Symmetric algorithm Function SYMMETRIC (provided in [1]) works as follows. It takes as input a configuration $Q(r) = (q_0, q_1, \ldots, q_5)$ and checks whether it is

```
Function: IDENTIFICATION
Input  : (q_0, q_1, ..., q_5)
Output: a, b, c, d, r ∈ {x, y, z}, move ∈ {true, false}
1  move := false; move[1] := false; move[−1] := false; D[1] =↑; D[−1] =↓;
2  (sym, a, b, c, d, k, r) := SYMMETRIC((q_0, q_1, ..., q_5));
3  if sym then move := true else
4      for i in {1, −1} do
5          (sym, α[i], β[i], γ[i], δ[i], k, r[i]) := SYMMETRIC((q_0 + i, q_1, ..., q_5 − i));
6          if sym then
7              (a[i], b[i], c[i], d[i]) := (α[i], β[i], γ[i], δ[i]); dir := −i(−1)^k;
8              case r[i] = x
9                  a[i] := α[i] + 2dir; b[i] := β[i] − dir;
10             case r[i] = y
11                 b[i] := β[i] − dir; c[i] := γ[i] + dir;
12             case r[i] = z
13                 c[i] := γ[i] + dir; d[i] := δ[i] − 2dir;
14             if (a[i], b[i], c[i], d[i]) is not periodic, with less than 3 multiplicities,
               and no edge-edge symmetry then
15                 if by simulating MOVING(a[i], b[i], c[i], d[i], r[i]), r[i] is allowed to
                   move with direction D[dir] then move[i] := true;
16 if move[1] ≠ move[−1] then
17     move := true;
18     if move[1] then (a, b, c, d, r) := (a[1], b[1], c[1], d[1], r[1]);
19     else (a, b, c, d, r) := (a[−1], b[−1], c[−1], d[−1], r[−1]);
20 if move[1] = move[−1] = true then
21     move := true;
22     if r[1] = x then (a, b, c, d, r) := (a[1], b[1], c[1], d[1], r[1]);
23     else (a, b, c, d, r) := (a[−1], b[−1], c[−1], d[−1], r[−1]);
```

Fig. 4. Algorithm for the identification of a robot.

symmetric. In the positive case, SYMMETRIC returns the values of a, b, c and d, an integer k (to be explained next), and the role of $r \in \{x, y, z\}$ in $Q(r)$. To this aim, SYMMETRIC rotates, for each $j \in \{0, 1, \ldots, 5\}$, the position of $q_j \in \{0, 1, \ldots, 5\}$ by i positions and checks whether the rotated configuration is symmetric. First, it checks whether there are two pairs of equal intervals $q_{1+i \bmod 6} = q_{5+i \bmod 6}$ and $q_{2+i \bmod 6} = q_{4+i \bmod 6}$. In the positive case, value i is stored in k, and if q_i is odd, three cases may arise: the configuration has a node-edge symmetry if $q_{3+i \bmod 6}$ is even; if $q_{3+i \bmod 6}$ is odd, the configuration has a node-node symmetry if $q_i < q_{3+i \bmod 6}$ or $q_i = q_{3+i \bmod 6}$ and $q_{1+i \bmod 6} < q_{2+i \bmod 6}$; the configuration is not symmetric.

3.2 Multiplicities and seven nodes case

In the case only one multiplicity is created at the middle node of the current interval A, then the gathering is almost completed as this node will be reached by

all the other robots by means of Procedure MULTIPLICITY, see Fig. 5, code lines 4–5. This procedure is also invoked when two multiplicities have been created, and the configuration requires at least two robots (belonging to the same multiplicity) to move in order to re-establish the symmetry, i.e., the configuration is at more than one step from symmetry, code lines 1–3.

Procedure: MULTIPLICITY
Input: $(q_0, q_1, \ldots, q_j)$, $1 < j \leq 5$

```
1 if there are two multiplicities then
2     if r belongs to the multiplicity closer than the other to a single robot then
3         move r towards the other multiplicity along the path free from single
          robots;
4 else
5     if r does not belong to any multiplicity AND between r and the multiplicity
      there is no other robot then
6         move r towards the multiplicity along a shortest path;
```

Fig. 5. Algorithm used for some configurations with multiplicities.

When the input ring is made of only seven nodes, the gathering problem requires suitable arguments. In fact, the first move must be necessarily $y\uparrow$, as any other one could lead to deadlock. The algorithm shown in Fig. 6 solves the case of six robots on a ring of seven nodes, i.e. when $a = 1$, and $b = c = d = 0$. Actually, Procedure SEVEN brings the robots to constitute a configuration with a multiplicity of more than 2 robots, then the gathering is finalized by means of Procedure MULTIPLICITY. The correctness of the algorithm is provided by Theorem 1. We will show that in this case the gathering node may vary with respect to the possible occurring execution. In particular, it can be any node except the ones originally occupied by z and z'. Moreover, the allowed moves may bring the configuration to asymmetric situations at more than one step from symmetry.

Theorem 1. *Procedure* GATHER *solves the gathering problem when the initial configuration is given by 6 robots on a ring of 7 nodes by means of Procedures* SEVEN *and* MULTIPLICITY.

Proof. The main idea behind Procedure SEVEN is to create an interval of two free nodes delimited by two single robots. Once this has been realized, the remaining 4 robots, occupying the remaining 3 nodes can detect the central node among these 3 nodes as gathering node, and move there (code line 11). Once the configuration moves to have only one multiplicity placed at the gathering node with more than three robots, Procedure GATHER invokes Procedure MULTIPLICITY, hence finalizing the gathering. Let us show then, how the interval of two free nodes is obtained. For describing the evolving of the configuration we always refer to

Procedure: SEVEN
Input: Ring of 7 nodes with 6 robots

```
1  if 6 nodes are occupied then y↑;
2  else
3      if more than 3 nodes are occupied AND two consecutive free nodes do not
       occur then
4          if robot r is not in a multiplicity AND (between r and a multiplicity
           there is a sequence of nodes S with
5          S given by only one free node OR
6          S given by one free node two single robots OR
7          (S given by one free node and one single robot AND the configuration is
           symmetric)) then
8              move r onto the localized free node;
9      else
10         if more than 3 nodes are occupied AND two consecutive free nodes are
           bounded by two single robots, r and r' then
11             move any robot but r and r' towards the middle of the three nodes
               between r and r' opposite to the interval of two consecutive free
               nodes;
```

Fig. 6. Algorithm invoked by Procedure GATHER for solving the case of 6 robots on a 7 nodes ring.

robots with their initial roles according to Figure 1. After code line 1 of the Procedure SEVEN, either two or one multiplicity is created. In the first case, the algorithm moves the remaining two single robots towards the multiplicities of one node by means of code line 8, allowed by code line 5. As there are no other possible moves, the required interval of two free nodes is created, eventually. In the second case, there is only one multiplicity and two non-consecutive free nodes. For the ease of discussion, we consider $y\uparrow \not\equiv y'\uparrow$ as the computed move, i.e. we consider the two symmetric moves of robots with role y as two distinguished moves.[3] Now the execution depends on the delays of the robots and the possible pending move $y'\uparrow$. In this case, the axis of symmetry changes to $\overrightarrow{C'B}$, with $b = -1$. If $y'\uparrow$ is not pending, then the only possible moves, according to code line 5, are $x'\uparrow$ and $z\uparrow$ (note that such moves would be $y\uparrow$ and $y'\uparrow$ with respect to the new axis of symmetry). Once both the two steps have been performed, the configuration is still symmetric, and the possible moves allowed from the current view, according to code line 7, are $z'\downarrow$ and $y'\uparrow$. If they are both realized, then the required interval of two free nodes is created, and the gathering node will be the one originally occupied by x. If only one move is realized, say $z'\downarrow$, then the only move allowed afterwards by code line 6 is $y'\downarrow$. Again the interval of two free nodes is created, but now the gathering node will be the one originally occupied

[3] Indeed, from the robot's perspective, roles y and y' are indistinguishable because of the symmetry.

by y. The remaining case to be analyzed is when after the first step $y\uparrow$, the symmetric move $y'\uparrow$ is pending. In this case, the execution depends on whether x' performs its Look operation before or afterwards the move $y'\uparrow$, while $x\uparrow$ is always computed sooner or later. If x' performs its Look operation afterwards $y'\uparrow$, then it does not move because it is part of a multiplicity and the only other moves allowed by the algorithm before creating an interval of two free nodes are $z\uparrow$ and $z'\uparrow$. If x' performs its Look operation before $y'\uparrow$, then $x'\uparrow$ as well as $z\uparrow$ are performed, eventually. If y' moves before z' performs its Look operation, then $z'\uparrow$ will be computed by code line either 5 or 6, and the gathering node will be the one originally free. If y' moves after z', then $z'\downarrow$ will be computed by code line 7, and the gathering node will be the one originally occupied by x. □

4 Correctness for the general algorithm

In this section we provide the correctness proof for the proposed algorithm. Actually, the proof of the next lemma can be found in [1].

Lemma 1. *Each time a robot r performs the Look operation, if the input ring has more than seven nodes and there are no multiplicities, r can recognize one unique robot or a couple of symmetric robots allowed to move.*

From the above results, the main contribution of the paper follows:

Theorem 2. *Algorithm* GATHER *solves the gathering problem starting from all initial configurations of* 6 *robots on a ring having exactly one axis of symmetry, provided that the axis is not of type edge-edge nor robot-on-axis.*

Proof. From Theorem 1, if the input ring has seven nodes, the gathering is feasible. From Lemma 1, we have that starting from a symmetric configuration, this always evolves by either increasing or decreasing intervals B and B'. In the first case, a multiplicity corresponding to $a = -1$ will be created, eventually. Then, by means of Procedure MULTIPLICITY the gathering correctly terminates. In the second case, two symmetric multiplicities will be created, obtaining $b = b' = -1$. As described in Section 3.1, all the robots belonging to the multiplicities, will behave as y or y', hence allowing at most four robots to move concurrently. The used technique to gather the two multiplicities into one is similar to "*Phase 3: gathering two multiplicities using guards*" provided in [6], hence it does not require further arguments. Once a single multiplicity containing more than 2 robots occurs, Procedure MULTIPLICITY correctly terminates the gathering. □

5 Conclusion

We have considered the basic gathering problem of six robots placed on anonymous rings. We positively answer to the previously open question whether it is possible to perform the gathering when the placement of the robots implies a symmetric configuration of type node-node or node-edge, without robots on the

axis. The proposed algorithm makes use of new techniques that sometimes do not fall neither into symmetry-breaking nor into symmetry-preserving approaches. The very special case of a seven nodes ring already exploits new properties of the robots' view in order to decide movements. We believe that our approaches can provide useful ideas for further applications in robot-based computing. In particular, the gathering problem of $2i$ robots, $4 \leq i \leq 9$, placed on anonymous rings in symmetric configurations of type node-node or node-edge, without robots on the axis remains open. However, our technique provides some evidences that allow us to claim the following conjecture:

Conjecture 1. The gathering problem given by configurations with $2i$ robots, $2 \leq i \leq 9$, is solvable on rings with $n > 6$ nodes having exactly one axis of symmetry, provided that the axis is not of type edge-edge.

References

1. Haba, K., Izumi, T., Katayama, Y., Inuzuka, N., Wada, K.: On gathering problem in a ring for 2n autonomous mobile robots. In: 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), poster. (2008)
2. Klasing, R., Kosowski, A., Navarra, A.: Taking advantage of symmetries: Gathering of many asynchronous oblivious robots on a ring. Theor. Comput. Sci. **411** (2010) 3235–3246
3. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theor. Comput. Sci. **390** (2008) 27–39
4. Koren, M.: Gathering small number of mobile asynchronous robots on ring. Zeszyty Naukowe Wydzialu ETI Politechniki Gdanskiej. Technologie Informacyjne **18** (2010) 325–331
5. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Gathering of asynchronous robots with limited visibility. Theor. Comput. Sci. **337** (2005) 147–168
6. Izumi, T., Izumi, T., Kamei, S., Ooshita, F.: Randomized gathering of mobile robots with local-multiplicity detection. In: 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS). Volume 5873 of Lecture Notes in Computer Science. (2009) 384–398
7. Izumi, T., Izumi, T., Kamei, S., Ooshita, F.: Mobile robots gathering algorithm with local weak multiplicity in rings. In: 17th International Colloquium on Structural Information and Communication Complexity (SIROCCO). Volume 6058 of Lecture Notes in Computer Science. (2010) 101–113
8. Prencipe, G.: Impossibility of gathering by a set of autonomous mobile robots. Theor. Comput. Sci. **384** (2007) 222–231
9. D'Angelo, G., Di Stefano, G., Navarra, A.: Gathering of six robots on anonymous symmetric rings. Technical Report R.11-112, Dipartimento di Ingegneria Elettrica e dell'Informazione, Università dell'Aquila (2011)