CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Parameterized Verification of Distributed Algorithms in Dynamic Graphs

Master's thesis in Computer science and engineering

Maria Kokkou

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Parameterized Verification of Distributed Algorithms in Dynamic Graphs

Maria Kokkou

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Parameterized Verification of Distributed Algorithms in Dynamic Graphs
Maria Kokkou

Supervisor: Nir Piterman, Department of Computer Science and Engineering
Examiner: Wolfgang Ahrendt, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Description of the picture on the cover page (if applicable)

Typeset in LaTeX
Gothenburg, Sweden 2022

Parameterized Verification of Distributed Algorithms in Dynamic Graphs
Maria Kokkou
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Problems set in dynamic graphs have recently been gaining more attention. However, automated verification methods for the proposed solutions to these problems have not yet been studied. We aim in this work to provide some first results in this direction by focusing on distributed algorithms set in dynamic rings.
This work consists of three main parts. We provide a formalization of the mobile agents and algorithms that are used to solve known problems from the distributed computing literature. We then show that under the most common assumptions for mobile agent capabilities, constructing an automated decision procedure for algorithms that solve the distributed computing problems in dynamic rings is undecidable. Finally, we use a different framework for algorithmic verification called "Regular Model Checking". In this part, we provide a method to transform algorithms that solve the problems that we study in dynamic graphs into the components that are needed for the application of regular model checking techniques.

Keywords: Parameterized verification, distributed computing, mobile agents, dynamic graphs, thesis.

Acknowledgements

First and foremost, I would like to thank my supervisor, Nir Piterman, for agreeing to supervise this project and for his continued guidance during my thesis work. Thank you for all your feedback, ideas and time. I would also like to thank Wolfgang Ahrendt, my examiner, for his very detailed comments at the midpoint of this work, that gave me a new perspective and motivated me to continue.
Finally, I would like to thank my friends for their patience, support and understanding throughout the past years. Athanasios, Amalia and George, thank you.

Maria Kokkou, Gothenburg, January 2022

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 The Problem . . . 1
1.2 Contributions . . . 2
1.3 Thesis Organization . . . 2

2 Background **5**
2.1 Mobile Agents . . . 5
2.2 Dynamic Networks . . . 6
2.3 Exploration . . . 6
2.4 Gathering . . . 7
2.5 Black Hole Search . . . 7
2.6 Connection Between Exploration, Gathering and Black Hole Search . . . 8
2.6.1 Exploration and Gathering . . . 8
2.6.2 Exploration and Black Hole Search . . . 9
2.7 Parameterised Verification of Multi-Agent Systems . . . 10
2.8 Model Checking . . . 10
2.9 The Non-Halting Problem for a 2-Counter Machine . . . 12

3 Model **15**
3.1 Network . . . 15
3.2 Agents . . . 15
3.3 Problem Assumptions . . . 16
3.3.1 Exploration . . . 16
3.3.2 Gathering . . . 17
3.3.3 Black Hole Search . . . 17

4 Formal Model of Agents **19**
4.1 Sensors . . . 20
4.2 Variables . . . 20
4.3 Conditions . . . 21
4.4 Moves . . . 21
4.5 Updates . . . 21
4.6 Formalization of an Agent . . . 21

4.7 Semantics 25

5 From Distinct to Sequential Identities 27

6 Undecidability 31
6.1 Agents That Only Have Vision 32
6.2 Agents That Have Vision and One Bit of Information 39

7 Regular Model Checking 43
7.1 Algorithms with agents that use counting variables 43
7.2 Motivation for RMC 43
7.3 Moving in a Ring 44
7.4 Constructing the Necessary Components for RMC 51
7.4.1 Alphabet Definition in the General Case 52
7.4.2 Transducer Construction in the General Case 52
7.4.3 Set of Initial Configurations in the General Case 53
7.4.4 Exploration, Gathering and BHS as Verification Problems . . 54
7.5 More than one agents in the ring 54
7.6 Example: Global Transducer for Multiple Agents 56

8 Conclusion 59
8.1 Summary 59
8.2 Discussion 60

Bibliography 63

List of Figures

2.1 The set of initial configurations for an agent in a line graph. 11
2.2 The transducer describing the transition relation of an agent moving to the right in a line graph. 12

4.1 An agent that solves the Exploration problem in rings of known size with agents that do not have chirality 22
4.2 The "Avanguard" agent that solves the BHS problem in dynamic rings when the agents can only communicate by seeing which other agents are in the same node as them (vision model) following the algorithm in [1]. 23
4.3 The "Retroguard" agent that solves the BHS problem in dynamic rings when the agents can only communicate by seeing which other agents are in the same node as them (vision model) following the algorithm in [1]. 24
4.4 The "Leader" agent that solves the BHS problem in dynamic rings when the agents can only communicate by seeing which other agents are in the same node as them (vision model) following the algorithm in [1]. 25

5.1 Algorithm that allows initially co-located agents to assume sequential identities. 28

7.1 An automaton representing an agent that moves counterclockwise in a ring. 45
7.2 An automaton representing an agent that moves clockwise in a ring. . 46
7.3 A ring $\mathcal{R}$ that consists of four nodes and contains an agent initially placed on node v_2. 46
7.4 Transducer for an agent being in state Init and moving counterclockwise 49
7.5 Transducer for an agent being in state Fwd and moving counterclockwise. 50
7.11 Transducer for an agent switching from state Init to state Bounce. . . 50
7.12 Transducer for an agent switching from state Init to state Fwd. . . . 50
7.13 Transducer for an agent switching from state Fwd to state Terminate. 50
7.6 Transducer for an agent being in state Bounce and moving clockwise. 51
7.7 Transducer for an agent being in state Init when c_1 becomes true. . . 51
7.8 Transducer for an agent being in state Init when c_3 becomes true. . . 51
7.9 Transducer for an agent being in state Fwd when c_2 becomes true. . . 52

7.10 Transducer for an agent being in state Bounce when c_2 becomes true. 52
7.14 Transducer for an agent switching from state Init to state Bounce. . . 52
7.15 The set of initial configurations for some agent A 54
7.16 Some agent A, solving the problem defined in this section 56
7.17 Transducer for an agent being in state Init and moving clockwise. . . 57
7.18 Transducer for an agent detecting that condition c_1 became *true*. . . . 57
7.19 Transducer for an agent switching from state Init to state Term. . . . 57

List of Tables

3.1 *Possibility results for Exploration with explicit termination. In the table, we use n to denote the size of the ring, "f2f" to denote "face-to-face communication" meaning that agents on the same node can read the contents of each other's memory, "vision" to refer to the model of communication in which every agent can only detect the presence and number of other agents on the node it currently occupies, "edge cross detection" means that agents crossing the same edge at the same time in opposite directions can detect each other and, finally, IDs refer to distinct but secret numeric agent identities.* 16

6.1 *Summary of our undecidability results. All agents in all cases have distinct but unknown identities. In the cases where the agents do not begin from the same node (i.e., scattered) we assume there is also a marked node somewhere in the ring.* 33

7.1 Encoding for agent configuration in state Init 47
7.2 Encoding for agent configuration in state Bounce 47
7.3 Encoding for agent configuration in state Fwd 48

8.1 *Summary of our undecidability results. All agents in all cases have distinct but unknown identities. In the cases where the agents do not begin from the same node (i.e., scattered) we assume there is also a marked node somewhere in the ring. Unless indicated otherwise in the table, the model of communication between agents is "vision", that is, each agent can only communicate by seeing how many other agents are located on the node it currently occupies. "One bit of information", means that agents located at the same node can get one bit of information, corresponding to the disjunction of the signalling variables of all co-located agents.* 60

1

Introduction

1.1 The Problem

The number of devices able to connect to a network nowadays, such as devices with internet connectivity, is huge. However, the need for distributed computing methods to solve problems while using the available computation power remains high. One of the methods used in the distributed computing literature is to use mobile entities, such as "mobile agents" or "robots". Those entities can move between the nodes of a graph, gather information about the network and solve potential problems without the nodes communicating directly with each other. Even more recently, research has turned towards the study of problems on dynamic graphs [2]. That is, on graphs that consist of a fixed set of edges and nodes but not all the edges are always present. In those graphs, mobile entities are still required to solve some problem, but this time there is the added difficulty of the graph changing while the agents are moving. Solving problems in dynamic graphs is at least as hard as solving the same problem in the static version of a graph. In the area of distributed computing, problems in dynamic graphs have not yet been studied as much as in static graphs, but for the problems that we are interested in as part of this thesis, it has been shown that more mobile agents with more capabilities are needed in a dynamic setting.

Parameterised verification refers to proving the correctness of a system specification regardless of the components. In the context of this thesis, only the graph is given as a parameter and the initial goal of this thesis was to construct an automated procedure that could decide whether a distributed algorithm for mobile agents that operate in a dynamic ring is correct. We decided to focus on algorithms that solve three well studied problems from the area of distributed computing. The problems we selected are "Exploration", "Gathering" and "Black Hole Search" and the reason those problems were chosen is twofold. First, those problems are connected (see Chapter 2.6) and as a consequence we expect the algorithms that solve the problems to have some common elements. Briefly, the way the problems are connected, is that "Black Hole Search" can be viewed as a special case of "Exploration" and "Gathering" usually depends on the agents exploring the graph as a first step. The second reason, is that all three problems have been studied in the same type of dynamic graphs, called "1-interval connected dynamic rings" (see Chapter 2). While determining the agent capabilities that we would need to represent the systems that would be automatically verified, the focus of this work changed to determining the agent capabilities that make the problem undecidable. In addition to the undecidability results, we also provide a way to use a framework called "Regular Model

Checking" for the verification of algorithms that solve "Exploration", "Gathering" and "Black Hole Search". Regular Model Checking is an incomplete method, however, it has been shown to terminate under specific conditions. In this work, we provide a method to transform any algorithm that solves one of the three problems into the components that are needed for Regular Model Checking techniques to be applied.

1.2 Contributions

Our contributions in this thesis are the following:

- We provide an encoding capable of representing mobile agents and algorithms that solve Exploration, Gathering or Black Hole Search (described in Chapter 2) as automata.
- We formalize the local agent configuration and the global configuration for the automata of the previous bullet point.
- We derive undecidability results for agents that have at least two variables that can count up to an upper bound that depends on the size of the ring, start from the same node or from different nodes, have distinct identities and are able to receive one bit of information from other agents located at a common node at the same time.
- We derive undecidability results for agents that have one variable that can count up to an upper bound that depends on the size of the ring, start from the same node or from different nodes, have distinct identities and are able to receive one bit of information from other agents located at a common node at the same time.
- We derive undecidability results for agents that have at least two variables that can count up to an upper bound that depends on the size of the ring, start from the same node or from different nodes, have distinct identities and are only able to see if another agent is located at a common node at the same time but cannot exchange any other information.
- We derive undecidability results for agents that have one variable that can count up to an upper bound that depends on the size of the ring, start from the same node or from different nodes, have distinct identities and are only able to see if another agent is located at a common node at the same time but cannot exchange any other information.
- We provide a way to transform the agent and algorithm encoding we proposed, to the components required to apply regular model checking techniques for the verification of algorithms solving Exploration, Gathering or Black Hole Search in dynamic rings.

1.3 Thesis Organization

In Chapter 2, we give an overview of the most important notions that are needed to understand the parameterised verification problem for the algorithmic problems (Exploration, Gathering and Black Hole Search) that we study. More specifically,

we give a detailed explanation of the three algorithmic problems that we use and some of the important results in both static and in dynamic graphs about those problems. Furthermore, we show that the three problems that we study are actually closely connected to each other. This result is later used to show that undecidability results for Exploration are also undecidability results for the other two problems. We describe what a dynamic ring is in the context of this thesis and what the mobile agents that we use are. In the same chapter we summarize some of the work on parameterised verification of multi agent systems in static graphs, that is closely connected to our problem. Finally, we give a definition of 2-counter machines and we introduce regular model checking. Then we introduce the non-halting problem for a 2-counter machine that we later use to obtain undecidability results for the problems that we study.
In Chapter 3, we present the specific model that we use throughout this thesis. Like in Chapter 2, we talk about mobile agents, dynamic rings and our three algorithmic problems. However, this time we present the specific assumptions we make about the problems, the specific capabilities of the agents and the specific properties of the networks that we use in the rest of this thesis.
In Chapter 4, we present a method to model mobile agents that are able to solve the problems we discuss in this thesis as automata. In our encoding of the agents, any algorithm that the agent executes is encoded directly in the automaton. As an example, we show the representation we propose for two known algorithms, one for Exploration and one for Black Hole Search, in dynamic rings into automata using our encoding.
In Chapter 5, we show that if each mobile agent in a system is equipped with a unique but secret numerical identity, the agents can undertake different tasks. We consider both the case of initially co-located and of initially scattered agents. We then use the results of this chapter in Chapter 6.
In Chapter 6, we present some undecidability results for the problems we study. As part of the undecidability results, we prove that the type of algorithms used in the literature for the problems we study leads to undecidability.
Because of the undecidability results, in Chapter 7, we move to regular model checking which is an incomplete method that has been used for the verification of parameterized systems before, and we present a way to transform the agents and algorithms for the problems we study into the components that are needed for the application of known regular model checking techniques.
Finally, in Chapter 8 we summarize our results and discuss the open problems that arise from this work.

2
Background

In this chapter we describe the problems that we will address in the following chapters in more detail. Furthermore, we give a short presentation about the techniques we are going to use. As part of the thesis we examine three problems that we also refer to as "tasks". The problems that we examine as part of the thesis are called "Exploration", "Gathering" and "Black Hole Search". All three problems have been extensively studied in the distributed computing literature. Finally, we talk about the connections between those three problems.

2.1 Mobile Agents

The mobile agents that we refer to in this work, are entities that have the ability to move from a node to a neighbouring one in a graph, make computations, remember information that they gathered in previous rounds and in some cases are able to communicate with other agents. A recent survey can be found in [3]. The communication between agents, when it is possible, can be by either *endogenous* or *exogenous* methods. We call "endogenous" those communication methods that depend on the agents exchanging information directly. An example of such a communication method is called "face-to-face" (or f2f) communication and refers to agents that are able to read the contents of other agents' memory, provided they are on the same node at the same time (e.g., [4]). By "exogenous" communication methods we denote any communication method in which agents leave information on the graph (either in nodes or in edges) that other agents can read when they are located on the node or edge that contains the information at a later time. Two such communication methods are for example in [5, 6]. As part of this work, we consider both the case in which agents are initially on the same node and the case of initially scattered agents. Moreover, the agents may have distinct IDs or be completely indistinguishable and they may have a common sense of orientation (called "chirality") or not. Finally, in some cases the agents operate in "Look - Compute - Move" rounds, introduced in [7]. This means that as part of a single round, each agent

- Gathers information about its environment, such as whether other agents are located in the same node. (Look Phase)
- Based on the information the agents gathered during the Look Phase and potentially on information that was gathered in previous rounds, the agent computes its next move. (Compute Phase)
- Finally, the agent attempts to move to the node it computed in the previous

phase or remain in its current node if that was the result of the computation. (Move Phase)

2.2 Dynamic Networks

Computer networks are often represented as graphs. However, a major part of the distributed computing literature considers static graphs when modelling a problem. Taking into consideration the conditions of real world systems it becomes evident that such a representation might not always be accurate. New devices might connect to an existing network and links connecting nodes in the network can potentially fail. In order to better represent those possibilities as well, distributed computing literature has recently started turning to using dynamic networks [2, 8, 9, 1]. A dynamic graph is a sequence of static graphs. More specifically, a dynamic graph $\mathcal{G}$ can be written as $\mathcal{G} = (V, E) = G_0(V, E_0); G_1(V, E_1), \dots$, where V is the set of nodes and $E_i \subseteq E$. Although the general definition of a dynamic graph allows almost any type of change between two consecutive time units, in this work we consider a much more restricted type of dynamic graph. More precisely, we study problems in dynamic graphs called "1-interval connecting". In 1-interval connecting graphs (like in [8]), at most one edge is missing in each static graph, that is, $|E|-1 \leq |E_i| \leq |E|$. Furthermore, the missing edge is chosen by an adversary, therefore, it does not follow a pattern.
The biggest challenge, as well as the biggest advantage, of using dynamic graphs lies in the uncertainty of the changes of the graph. Therefore, although there has been some work in studying problems in dynamic graphs that change in a predetermined way, we only consider the case in which the changes of the network are unknown in advance. The first type of setting is called "Postmortem" (introduced in [10]) and the latter is called "Live" (also introduced in [10]).

2.3 Exploration

In the Exploration problem, the goal is for an agent or a team of agents to visit each node of an unknown graph in a systematic manner. A recent survey of graph exploration with mobile agents is given in [11]. In the case of exploration by a team of mobile agents, the problem has been considered for both the case of initially co-located agents and of agents starting from different nodes. The Exploration problem was first studied by Shannon in [12]. A lower bound on the memory complexity needed for graph exploration was given in [13] and an upper bound was given a few years later in [14]. As far as the time complexity is concerned, the fastest algorithm that can solve the Exploration problem in arbitrary but labelled graphs needs $m + O(n)$ moves as was shown in [15], where m is the number of edges and n the number of nodes. The problem has been studied in various topologies such as trees [16], grids [17], tori [17] and other common interconnection graphs [18]. As far as collaborative exploration (i.e., exploration by a team of agents) is concerned, the goal is to reduce the time needed for exploration as compared to when only one agent is available. Finding an optimal strategy for a group of agents

that minimizes the maximum steps taken by any agent for a given graph is known to be NP-complete [19], therefore research has mainly focused on graphs that are unknown to the agents in the beginning. In the case of distributed exploration, each agent explores the graph using the same algorithm and the agents can communicate with each other by writing on the nodes they visit. The simplest strategy that the agents can use in a distributed setting is a distributed version of the DFS algorithm presented in [20].
In the context of dynamic graphs, a team of mobile agents moves throughout a graph with the goal of visiting each node of the graph at least once [21, 8]. This problem has two versions. In "Terminating Exploration" the algorithm finishes when each node has been visited by at least one agent and there is at least one agent that knows that the graph has been explored. In the second version of the problem, called "Perpetual Exploration", each node has to be visited by some agent infinitely often.

2.4 Gathering

In the Gathering problem, a team of mobile entities (i.e., agents or robots) initially located in arbitrary but distinct locations in a graph or in the plain, meet at the same location after a number of moves. The special case in which the team consists of only two robots is called the Rendezvous problem. Two recent surveys are presented in [22, 23]. The Gathering problem has been studied in various different topologies such as rings [24, 25], trees [25, 26] and grids [26, 24]. Furthermore, the problem has been studied in synchronous, semi-synchronous and asynchronous settings [24]. The agents operating in the graph may have varying degrees of visibility ranging from strong global visibility to weak local visibility or even to being completely oblivious of other agents. Finally, some fault-tolerance results have been given in [27], where the agents start from different nodes and must gather in a graph that has faulty edges that destroy the agents that cross them.
In the context of dynamic graphs, a number of mobile agents are initially scattered (i.e., placed on different nodes) on a graph [9]. The goal is for the agents to manage to meet on some node of the graph after a finite number of moves. The Gathering problem is not solvable in dynamic graphs, however, it is possible to solve a relaxed version. In the relaxed version, all agents reach two neighbouring nodes (i.e., nodes connected by an edge) after a finite number of moves.

2.5 Black Hole Search

The Black Hole Search problem was introduced in [28]. A black hole (BH) is a malicious entity that destroys any agent that visits it without leaving any trace. The Black Hole Search Problem (BHS) assumes the existence of either one or multiple black holes in a network and the goal is to determine the location of the black hole or the black holes using a team of mobile agents. In order to solve the problem, at least one agent needs to survive knowing the location of the black hole. The agents can only infer the position of a black hole if it is located in the last unexplored node

of the network or due to the loss of other agents in it. Therefore, the number of surviving agents after the location of the black hole can be smaller than the initial number of agents in the network. Since an agent lost in a black hole does not leave any trace, various communication methods are used between the agents in order to determine the position of the black hole.
The Black Hole Search problem has been studied with respect to the synchronicity of the network (synchronous, asynchronous), the initial location of the agents (co-located, scattered), the type of graph the agents operate in and the communication method used. Four communication methods are commonly used: whiteboards (the agents can leave messages for other agents at the nodes they visit [18, 29, 6, 30]), pure token (the agents can place tokens at the nodes they visit [31, 5]), enhanced token (the agents can place tokens on the nodes or edges they visit [32, 33]) and timeout mechanisms ([34, 35]). The BHS problem was originally studied in rings and it has since been studied in various other topologies in [36, 4, 37, 18], as well as arbitrary networks, for example in [38, 34]. Finally, the computational complexity of searching for a black hole, as well as hardness and approximation results are found in [39, 40, 41].
In the context of dynamic rings, the problem of locating a black hole has been solved under the assumption of 1-interval connected rings in [1]. In that paper, the authors give algorithms that solve the problem in oriented rings using three agents for both the case of initially co-located agents and in the case of initially scattered agents. Multiple algorithms are presented in the paper and the authors provide tight bounds under numerous assumptions such as whether the agents have chirality, whether the agents know the size of the ring, the initial positions of the agents and whether the agents are anonymous or they have distinct identities.

2.6 Connection Between Exploration, Gathering and Black Hole Search

2.6.1 Exploration and Gathering

The connection between these two problems is based on a detailed review of known algorithms in both static and dynamic rings. Although there is no formal proof that the two problems are connected, we have found that all algorithms for agents that operate in dynamic rings and the algorithms we have found for agents operating in static rings include the exploration of the ring before gathering is achieved.
Although the goals of the two problems are different, many algorithms that solve the Gathering problem, depend on the agents also exploring the ring. This happens in cases where the agents do not have any information about the network and the only way to meet is to perform some computation based on the common information that they gather after an exploration phase. In the case of Gathering in dynamic rings, all the known algorithms (presented in [9]) begin by an Exploration phase. Although it is possible that the agents gather during the exploration phase, it is not guaranteed and the algorithms use various protocols to elect a node or an edge as a meeting point after the exploration phase is complete.

In static rings, an exploration phase might or might not be needed depending on the information the agents have about the network and the available communication mechanisms. The simplest case of agents that operate in a ring and that need to gather in some unknown in advance node, is for the agents not to have any external communication devices or knowledge, but for the nodes of the ring to have unique numerical IDs. In this case, exploration and gathering are the same, since the agents can only meet by traversing the ring once and then determining the node of the meeting based on some pre-decided condition (e.g., gather on the node with the lowest ID). This case is mentioned in [42]. If the agents do not have enough memory to read the node labels then gathering is not possible if the agents do not have some initial knowledge or an external communication method.

The initial knowledge two agents trying to meet may have, is the size of the ring, the distance to the other agent or the knowledge of the other agent's ID. The last case is the simplest since the agents solve the problem by having one agent wait and the other agent explore the ring until it finds it. The remaining cases are described in [43] and when lower bounds for the relevant scenarios are discussed, the authors provide algorithms that satisfy those lower bounds and depend on exploration.

In the case of an external communication method, the agents may have one or more tokens or the nodes may be equipped with whiteboards the agents can write on. Once again, in [43] the authors provide lower bounds on the number of moves the agents need to make and present time optimal algorithms that solve gathering by also exploring the ring.

2.6.2 Exploration and Black Hole Search

As we mentioned in Chapter 2, the Black Hole Search problem was introduced in [28]. The problem consists of finding a method for a team of mobile agents that operate in a graph to determine the location of a malicious entity which destroys any agent that visits it. A few years after the introduction of the problem, the same authors gathered the results that had been obtained until then in [44]. In the same survey, the authors introduced the black hole search problem as a special case of the exploration problem. We give here the intuition behind this result.

The main idea that we will use is that if the agents do not visit some nodes, the BH can be located in one of those nodes. Let $\mathcal{A}$ be an algorithm that solves BHS. Let us suppose that $\mathcal{A}$ is structured so that at least two nodes are never visited by any agent. For any ring, there is a configuration in which the BH is located in one of the two unvisited nodes. Furthermore, the BH only influences the node it resides in, by destroying the agents that visit it without leaving any trace. Therefore, any correct algorithm needs to visit at least one more node to solve the problem. Either the BH is located in that node, hence the BH is found or the BH is located on the last unexplored node. Therefore, all the nodes of the graph must be visited in order to solve BHS, since the BH can always be placed in the last unexplored node for any algorithm and any starting configuration, and the agents cannot infer its position if more than one nodes are unexplored. Formalizing this intuition even in the case of unconscious exploration is not trivial because it is hard to formalize the fact that an agent cannot identify a BH, unless it visits the node it is located at.

2.7 Parameterised Verification of Multi-Agent Systems

The goal of "Parameterised Verification" is to prove the correctness of a system regardless of its specific components. In this thesis, we only consider the size of the ring to be given as a parameter, however, the number of mobile agents operating in the graph is also often considered a critical system parameter. The closest paper to this project is "Parameterised Verification of Autonomous Mobile-Agents in Static but Unknown Environments" by Sasha Rubin [45]. In this paper, the author establishes a framework to model and automatically verify that autonomous mobile agents correctly perform their tasks. In this work, the author reduces the parameterised verification problem to classic questions in monadic second order logic and in automata theory. However, as the title suggests, this work studies a static environment, whereas we will study a dynamic environment. Another closely related paper is "Parameterised verification for multi-agent systems" by Kouvaros and Lomuscio [46]. In this work the authors study the problem of verifying role-based multi-agent systems, where the number of components cannot be determined at design time. Furthermore, they provide semantics about the representation of the system and of the ways the agents can interact with each other and with their environment.
A central part of studying distributed algorithms has always been centered around facing system failures. However, the most common ways those potential problems are studied are delay of messages (i.e., the system being asynchronous) or complete loss of messages. More recently, the interest of the distributed computing community has turned to dynamic networks, which model more accurately possible communication failures that are caused by the network itself. A recent survey of those results can be found in [2]. Formal methods for mobile agents have also been widely studied, however not in dynamic graphs. Some important results are surveyed in [47].
Solving a parameterised verification problem in dynamic environments in general is undecidable, even if the environment is very simple, such as a ring and there is only a single robot in the graph [48, 49]. However, it is still interesting to determine what restrictions on the robot or the environment will result in a decidable parameterised verification problem. Finding those restrictions is still an open problem that we study in this thesis.

2.8 Model Checking

Model Checking introduced in [50, 51] is a technique that facilitates the verification of various models. A recent introductory text can be found in [52]. Model Checking can be used on both finite and infinite models (under some constraints) as well as on parameterised models (e.g., [53]). Furthermore, it has also been used in the verification of distributed algorithms in [54, 55].
Regular Model Checking (RMC), which we will use in this thesis, is one of the frameworks used for algorithmic verification of parameterized or infinite state systems. RMC was first introduced in [56]. In model checking, in order to verify specific

properties (e.g., liveness) we usually compute reachability relations between states. In infinite state systems there is no upper bound on the number of transitions from an initial configuration to a reachable configuration, so the reachability problem is in general undecidable. RMC is an incomplete method (since the general problem is undecidable) but it has been proven in [56] that it can terminate under specific conditions. Three techniques that are used as part of RMC are "acceleration", "abstraction" and "widening" and an overview of them is given in [57].

The RMC framework needs the four following components in order to represent an algorithm $\mathcal{A}$ that attempts to solve a problem in a given system $\mathcal{S}$, where by *system* we denote a graph and the agents that operate on the graph.

- A finite alphabet Σ that is used to represent the valid states of $\mathcal{A}$ in $\mathcal{S}$.
- A configuration of $\mathcal{S}$ using the alphabet Σ, where each configuration is a word over Σ^*.
- A transition relation (called a "transducer") that describes the valid successive configurations of $\mathcal{S}$ according to $\mathcal{A}$, over the alphabet $\Sigma \times \Sigma$.
- A set of all the possible initial configurations of $\mathcal{S}$ over Σ.

A more formal definition of those components is given in Chapter 7. In addition to those components, RMC requires the definition of a *verification problem*. The types of verification problems usually considered in RMC are that of verification of safety properties and verification of liveness properties. Both "safety" and "liveness" require to solve a *reachability* problem for a set of configurations from the initial configurations. The above mentioned techniques ("acceleration", "abstraction" and "widening") are used over the four components in order to check if the properties of the specified verification problem are satisfied. In Chapter 7, we express Exploration, Gathering and BHS as verification problems to be checked by RMC.

As an example, consider a mobile agent that is moving in a line graph. Suppose that in each step, the agent can only move one step to the right. A configuration of the system consists of the size of the graph and of the current position of the agent in the graph. That is, a configuration is a word over the alphabet $\{e, a\}$, where e represents an *empty* node (i.e., a node not occupied by a mobile agent) and a represents a node occupied by an agent. For instance, a line graph consisting of four nodes in which the leftmost node is occupied by an agent, is given by the word $aeee$. The set of all possible initial configurations is given by the regular expression e^*ae^* (Figure 2.1) and the transition relation is given by the transducer given in Figure 2.2. We give a detailed description of how the transducer is constructed and read in Chapter 7.

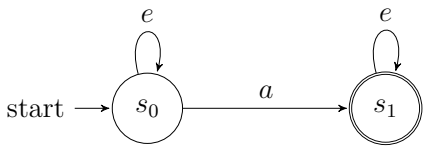

Figure 2.1: The set of initial configurations for an agent in a line graph.

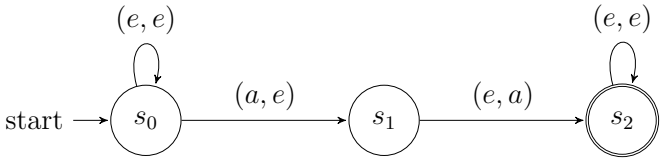

Figure 2.2: The transducer describing the transition relation of an agent moving to the right in a line graph.

2.9 The Non-Halting Problem for a 2-Counter Machine

In this section we first give a definition for a 2-counter machine and then we describe the non-halting problem for those 2-counter machines. In Chapter 6, we will use the non-halting problem for 2-counter machines in order to derive undecidability results for the problem of constructing an automated decision procedure for algorithms that solve distributed Exploration in dynamic rings.
A 2-counter machine (2CM) [58] is a machine that has two registers (call them register x and register y) and can only perform the following computations.

- Increment or decrement x by one.
- Check if x is equal to 0.
- Increment or decrement y by one.
- Check if y is equal to 0.

Formally, a 2CM is a 6-tuple $M = (Q, \Sigma, \delta, I, F, K)$ where:

- Q is the set of states.
- Σ is the input alphabet, including the empty symbol ϵ. In our case, we only consider input free 2-counter machines, so $\Sigma = \{\epsilon\}$
- $\delta : Q \times \Sigma \times \{\text{zero}, \neg\text{zero}\}^2 \to Q$ is the transition function.
- $I \in Q$ is the starting state.
- $F \subseteq Q$ is the set of accepting states.
- $K : Q \times \Sigma \times \{\text{zero}, \neg\text{zero}\}^2 \to \{-1, 0, 1\}^2$ is the counter updating function. It is always the case that $K(q, zero, a) \in \{0, 1\} \times \{-1, 0, 1\}$ and $K(q, a, zero) \in \{-1, 0, 1\} \times \{0, 1\}$. For $K(q, \alpha, \beta) = (a, b)$ we denote $K_1(q, \alpha, \beta)$ for a and $K_2(q, \alpha, \beta)$ for b.

A computation step of a 2CM is determined by the current state of the machine, the input symbol and by the set of counters that contain zero. The action at that step consists of independently altering the value of each counter by adding $0, 1$ or -1 to the previous value of the counter. More formally, a description of the configuration of a 2CM is given by the triple

$$(q, c_1, c_2) \text{ where } q \in Q,\ c_1 \in \{zero, \neg zero\} \text{ and } c_2 \in \{zero, \neg zero\}$$

and by the pair

$$(v_1, v_2) \text{ where } v_1 \in \mathbb{N} \text{ and } v_2 \in \mathbb{N}$$

so that

$$v_i = 0 \text{ iff } c_i = zero \text{ for } i = 1, 2$$

We say that a configuration $C' = \langle (q', c_1', c_2'), (v_1', v_2') \rangle$ is the *successor* of a configuration $C = \langle (q, c_1, c_2), (v_1, v_2) \rangle$ if

- There is a valid transition $\delta(q, c_1, c_2) \to q'$.
- $v_1' = v_1 + K_1(q, c_1, c_2)$
- $v_2' = v_2 + K_2(q, c_1, c_2)$

A computation or run of a 2CM is a series of successive configurations.
The non-halting problem of a 2-counter machine is, given a 2CM M that only uses the aforementioned operations, to decide whether the machine does not halt. We say that the machine halts if it enters a specific state called "halting state". This problem has been proven to be undecidable in [58].

3

Model

In this chapter, we describe in more detail the specific model we use. More precisely, we describe the type of dynamic graphs we consider, as well as the mobile agent capabilities. Furthermore, we talk about the assumptions we make for each of the algorithmic problems we study (i.e., Exploration, Gathering and Black Hole Search).

3.1 Network

The type of graph that we consider as part of this thesis is a ring, that is, a finite graph in which each node has degree exactly two. The systems that we consider are synchronous, which means that each agent move from a node to a neighbouring one, takes one time unit. Each node has two ports, each port corresponding to one of the edges that are incident to the node, and the ports are consistently labelled in all nodes of the ring. That is, if an agent starting from some node always enters a node from port a and leaves the node from port b it will traverse the whole ring. In some cases, the ring contains one marked node that the agents can distinguish. Finally, we only consider 1-interval connecting rings, which means that at most one edge may be missing at any time and as a result, the ring is always connected.

3.2 Agents

The mobile agents that we consider, operate in Look-Compute-Move rounds, as we described in Chapter 2. Briefly, during the "Look" phase, an agent located at node v observes v and checks if another agent is located on v or in one of the ports leading to the nodes connected to v by an edge. In the "Compute" phase the agent makes computations based on the values of its variables and the results obtained during the "Look" phase and, finally, it attempts to move during the "Move" phase. An agent can "see" when other agents are located on the node it occupies (i.e., the agent has "local visibility") but it cannot see the location of agents on other nodes of the ring. Additionally, an agent is not aware of traversing an edge at the same time as another agent moving in the same or in the opposite direction. The agents that we study are equipped with unlimited memory.

3.3 Problem Assumptions

In this section we list the assumptions under which the problems we study have already been solved. We then use these assumptions and the impossibility results in [8, 9, 1] as a guide in obtaining useful undecidability results in Chapter 6.

3.3.1 Exploration

For Exploration [8] we focus on the setting where either there is a marked node or the agents know the size of the ring. Furthermore, we assume that the agents have distinct IDs and a common sense of orientation but the only form of communication they have is seeing how many agents occupy their current node. In our model, the agent can see the number of other agents but not their identities. For example, let an agent A occupy node v at time $t, t+1$ and $t+2$. At $t+1$ an agent, A_2, moves to v and at $t+2$, A_2 leaves v but another agent, A_3, enters v. In this setting, A cannot distinguish between A_2 and A_3 but knows that one more agent occupies v at both $t+1$ and $t+2$. We only consider the fully synchronous case. According to the results of [8], the exploration problem in dynamic rings is not solvable, even with partial termination[1], by any number of mobile agents if the agents are anonymous, that is, if the agents do not have distinct identities, if there is no distinctly marked node in the ring or if the agents do not know the size of the ring. A summary of the results for which Exploration in dynamic rings with explicit termination is possible is given in Table 3.1.

Table 3.1: *Possibility results for Exploration with explicit termination. In the table, we use n to denote the size of the ring, "f2f" to denote "face-to-face communication" meaning that agents on the same node can read the contents of each other's memory, "vision" to refer to the model of communication in which every agent can only detect the presence and number of other agents on the node it currently occupies, "edge cross detection" means that agents crossing the same edge at the same time in opposite directions can detect each other and, finally, IDs refer to distinct but secret numeric agent identities.*

	# of Agents	Initial Positions	Communication	Other Capabilities
[8]	2	co-located or scattered	vision	known n, memory
[8]	2	co-located or scattered	vision	landmark
[59]	3	co-located or scattered	f2f	IDs, memory
[60]	3	scattered	f2f	IDs, edge cross detection, $O(\log n)$ memory

In the undecidability proofs in Chapter 6 we only consider agents that have capabilities that make the problem solvable with respect to the impossibility and possibility results of [8, 59] and [60].

[1] In *Partial Termination* at least one agent enters a terminal state and stops moving after a finite number of steps.

3.3.2 Gathering

We know that Gathering in dynamic rings is not possible and we therefore consider the relaxed version in which the problem is solved if either all agents gather in the same node or if all the agents gather in two neighbouring nodes. In the known algorithms in [9] the agents must either know the size of the graph or the number of agents in order to solve the problem. To be more precise, if the agents have chirality (i.e., a common sense of orientation) they need to know the number of agents, otherwise they need to know the size of the graph. It is always assumed that the agents and the nodes are anonymous and that the agents cannot communicate with each other. Finally, the agents must either be able to detect when they are crossing the same edge as another agent at the same time or all nodes that contain an agent at the beginning of the algorithm must be identically marked.

3.3.3 Black Hole Search

The known algorithms for Black Hole Search in dynamic rings need at least three agents to operate in the ring and the agents must know the size of the ring. Algorithms and tight bounds are presented in [1] for various models with initially co-located agents. Furthermore, in the given algorithms the agents have distinct and visible IDs. Finally, both the case of endogenous (i.e., vision and face-to-face) and of exogenous (i.e., tokens and whiteboards) communication methods are considered in [1] but in this work we only consider the more restricted communication models of vision and face to face communication.

4

Formal Model of Agents

In this chapter we give a detailed description of how we can model mobile agents that are able to solve the type of problems that we study in this thesis (i.e., Exploration, Gathering and Black Hole Search in dynamic rings) when the size of the environment the agents operate in, is given as a parameter.
The idea of coding an algorithm within the agent automata, as we will do in this chapter, was introduced in [61] and was more recently also used in [45]. Using this method, an agent is modeled as a set of instructions which tell the agent how and when to move along an edge (if the edge is present) and how to use the information it collects from its environment and its interaction with other agents in order to complete a task. Our motivation for using this particular method, comes from the fact that a similar method, resulting in a different framework, was used for the static graphs case in [45]. We thought that a reasonable first step would be to check whether a different encoding of the algorithms within agents, also based on the model of [61], would work in a dynamic environment. In addition to the difference in the environment (i.e., static vs dynamic) the difference between the resulting models between [45] and this work comes from the fact that the agents used in dynamic graphs in the literature, so far operate in "Look-Compute-Move" rounds instead of the single action per round model that is widely used for problems in static graphs.
According to the recent survey in [3] and the algorithms that solve the problems that we study in this thesis [8, 9, 1], a mobile agent is capable of moving, gathering information from its environment, communicating with other agents and performing computations. Our formalization of an agent is with respect to those capabilities. More precisely, we represent all information that comes from outside the agent (i.e., either from the agent's environment or from the agent's interaction with other agents) using "sensors" and all information that is a result of the agent's computations as "variables". For the most common communication methods (whiteboards, tokens, vision, wireless, beeping and face to face communication), including the communication methods used in the algorithms that we consider, sensors can represent the information the agents gather. Therefore, we provide a way to formalize both the external information the agent acquires during the execution of an algorithm and the results of internal computations the agent performs. Finally, in the representation of our agents we add a parameter, which we consider to be some initial but not predetermined input to the agent, that the agent can use as part of the computations it performs. In our case, we only use the parameter as the number of nodes in the ring. In addition to sensors, moves and variables we also use *conditions* to define an agent. As we have already mentioned, the agents use the sensors and variables in order to perform computations. The conditions that we define are

the mathematical expressions of sensors and variables that determine the agent's transitions.

4.1 Sensors

Each agent is equipped with sensors that allow it to interact with its environment and gather information. For example, a sensor may allow an agent A that moves to node v at time t, to detect whether another agent that reached v at some time $t' < t$ is still at v at t. Furthermore, another sensor may allow A to detect whether an edge e, incident to its current node is present at some time unit. Several methods of communication between agents have been presented in the distributed computing literature, such as face-to-face communication, tokens and whiteboards (see also Chapter 2). All the possible methods of communication are modelled as sensors. The sensors that we use are further divided into two subcategories: "Pre-Move" and "Post-Move" sensors.

- **Pre-Move Sensors:** This set of sensors is updated before the agent calculates its next move. Such sensors may detect the agent's position, the existence of other agents on the same node, the existence of other agents on the same edge, etc. In general, information that concerns the environment of the agent comes from this type of sensors.
- **Post-Move Sensors:** This set of sensors is updated after the end of a move. Post-Move sensors may hold information like the success or failure of a move from a node v to a neighbouring node v', the existence of an edge or whether a port was free or occupied by another agent. In general, this type of sensors provide information about the result of the move.

We refer to any information that cannot be calculated by the agent locally as information detected by a sensor. Finally, we will use the symbol $D(sens)$ to denote the product of the domains of the sensors. We will use $D(sens)$ to refer to the values of all the sensors the agent is equipped with, at any given time.

4.2 Variables

Agents may or may not have memory that they can use to remember data that they have gathered about their environment or about the execution of the algorithm. For example, variables may be used to count the number of nodes an agent has traversed, the number of rounds it has been waiting on a node or the number of other agents on its current node. In general, we use variables to store new knowledge the agent has acquired about the network and that the agent will need to use again. Furthermore, the agent can change the value of the variables on its own, but it cannot receive new values as input. Additionally, we will use the symbol $D(vars)$ to denote the product of the domain of the variables. We will use $D(vars)$ to be able to refer to the values of all the variables used by the agent.

4.3 Conditions

An agent chooses its next step based on the values of its variables and sensors. We use *conditions* to formalize the information either computed by the agent (i.e., through variables) or gathered from the environment (i.e., through sensors) that is relevant to the execution of the algorithm. In other words, Conditions are predicates evaluating to true or false, whose terms are `Variables` and `Sensors`.

4.4 Moves

In the kind of problems that we consider in this thesis, the agents can move from a node to a neighboring one (i.e., to a node connected by an edge to the agent's current node). The edges incident to a node may be numbered, in which case the set of Moves contains as many possible moves as the degree of the graph. In special cases of graphs, such as rings or lines, the set of Moves only contains the two possible directions, left and right. Finally, the set of Moves can contain the possibility to not perform a move in a round, depending on whether the algorithm permits that.

4.5 Updates

The values of variables, pre-move sensors and post-move sensors are updated during the execution of the algorithm. More specifically, the value of any variable $v_i \in$ `Variables` is updated by the agent after a transition. The value of a pre-move sensor is given to the agent as input before a move and the value of a post-move sensor is given to the agent as input after a move.

4.6 Formalization of an Agent

An agent is a 10-tuple:

$(Q, \delta, \texttt{I}, \texttt{F}, \texttt{Moves}, \texttt{Conditions}, \texttt{Variables}, \texttt{PreM-Sensors}, \texttt{PostM-Sensors}, \texttt{N})$

where:

- Q is a finite set of states.
- $\delta = \texttt{Q} \times \texttt{Conditions} \rightarrow \texttt{Q} \times \texttt{Moves} \times \texttt{VarUpdates}$ is the transition relation, where `VarUpdates` refers to the updates of variables $v_i \in$ `Variables`.
- $I \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states.
- `Moves, Conditions, PreM-Sensosrs, PostM-Sensors` and `Variables` are the sets described above.
- `N` is the number of nodes of the graph.

As an example, the mobile agent in Figure 4.1 solves the Exploration Problem in a ring of known size, when the agents do not have chirality, using Algorithm 1 from [8]. To make the presentation clearer, the expressions on the loops and the expressions on the transitions between two states are mutually exclusive and complete, so we omit re-writing the conjunction of the negation of the expressions of Conditions that

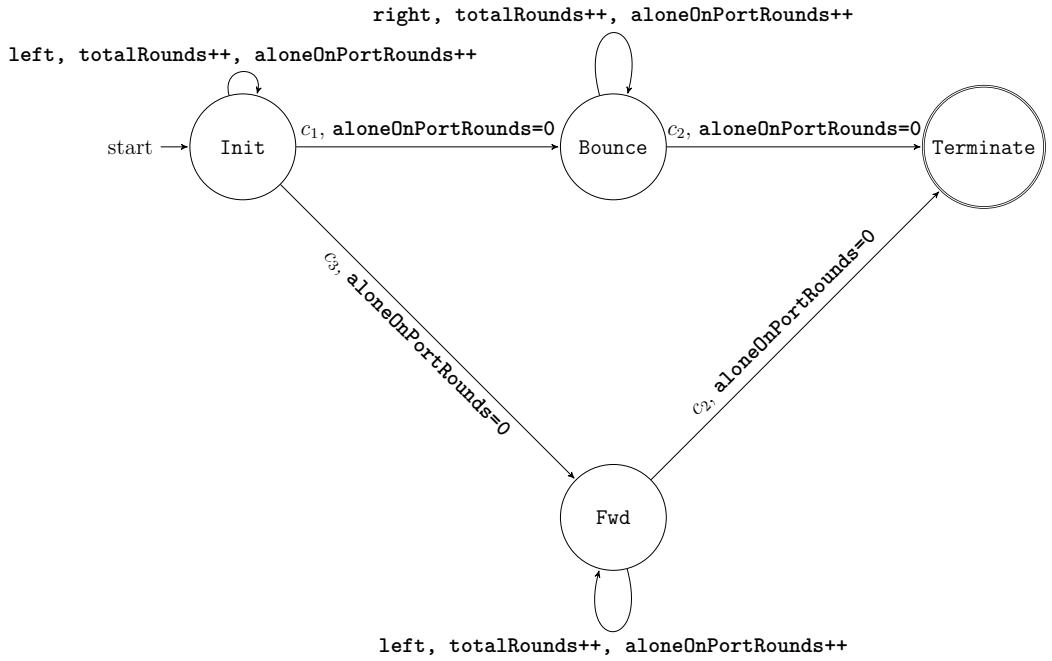

Figure 4.1: An agent that solves the Exploration problem in rings of known size with agents that do not have chirality

are on the transitions between two different states and on the loops. The conditions, are specified below.
We use the following symbols in the automaton:

- Pre-Move Sensors: the agent is on a port (`onPort`), the agent is alone on a port (`aloneOnPort`)
- Post-Move Sensors: a move failed (`moveFailed`)
- Variables: the number of consecutive rounds the agent has been waiting on a port (`aloneOnPortRounds`), the total number of rounds (`totalRounds`). Both variables are initialized to 0, that is, `aloneOnPortRounds = 0` and `totalRounds = 0`.
- Conditions:
  c_1: $(((\text{totalRounds} \geq 2N-4) \wedge (\text{waitOnPort} == N-1)) \vee (\text{totalRounds} \geq 3N-6)) \vee (\text{onPort} \wedge\neg \text{aloneOnPort} \wedge\neg \text{moveFailed})$
  c_2: $\text{totalRounds} \geq 3N-6$
  c_3: $(\text{onPort} \wedge\neg \text{aloneOnPort} \wedge \text{moveFailed}) \vee (\text{totalRounds} \geq 3N-6)$
- Moves = {`left, right, none`}
- Q = {`Init, Forward, Bounce, Terminate`}
- I = `Init` (the initial state)
- F = {`Terminate`} (the final state)
- δ = `Q` × `Conditions` → `Q` × `Moves` × `VarUpdates`
- N = The number of nodes

A second example is presented in Figures 4.2, 4.3 and 4.4. In this example, we give a representation of the agents that solve the Black Hole Search problem in dynamic

rings. The algorithm that we model is introduced in [1] and uses three types of agents which are able to cooperatively locate a black hole in a ring after a finite number of moves. The agents are able to see each other's identity and in the paper are called "*avanguard*", "*retroguard*" and "*leader*" and we also use this naming here to differentiate the agents.
We begin by defining the "*Avanguard*" agent.

- $Q = \{\texttt{Init, Return, NewNode, Move}\}$
- $I = \{\texttt{Init}\}$
- $F = \emptyset$
- $Moves = \{\texttt{right, left}\}$
- Pre-Move Sensors: $\emptyset$
- Post-Move Sensors: $\emptyset$
- Variables: the number of distinct visited nodes since the agent last switched direction (`nodesSinceDirSwitch`)
- Conditions:
  c_1: nodesSinceDirSwitch > 0

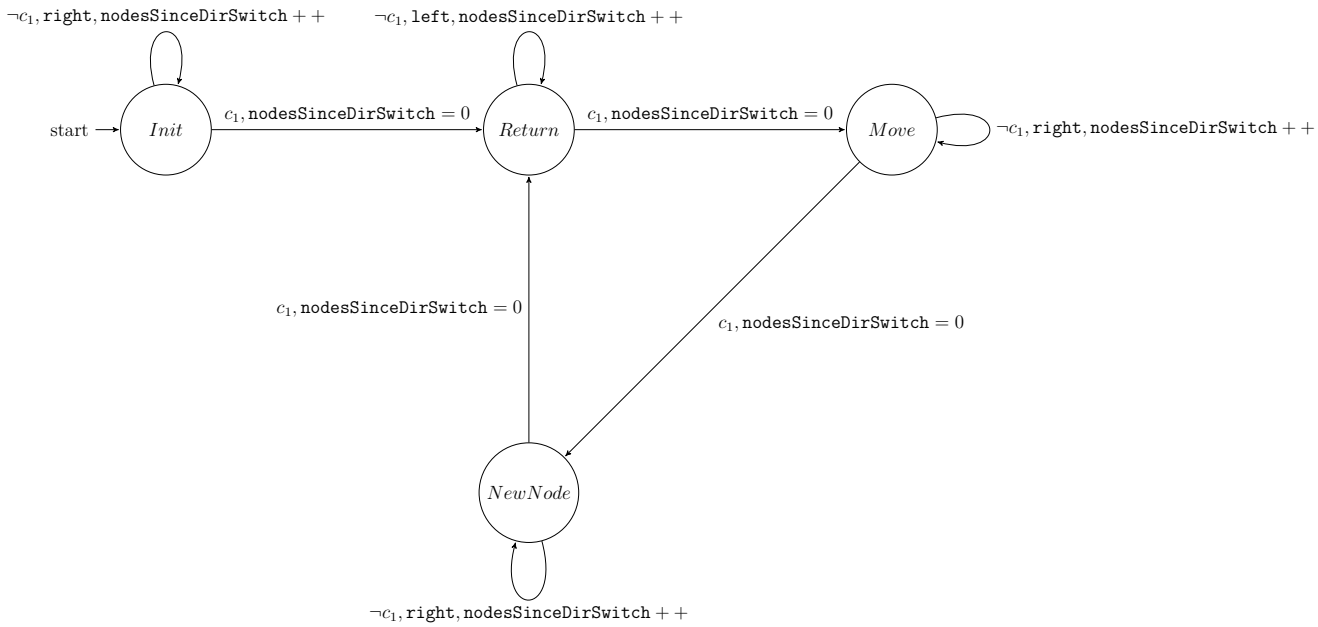

Figure 4.2: The "Avanguard" agent that solves the BHS problem in dynamic rings when the agents can only communicate by seeing which other agents are in the same node as them (vision model) following the algorithm in [1].

Next, we define the "*Retroguard*" agent from the same paper.

- $Q = \{\texttt{Init, Bounce, Return}\}$
- $I = \{\texttt{Init}\}$
- $F = \emptyset$
- $Moves = \{\texttt{right, left}\}$
- Pre-Move Sensors: the retroguard detects that the "*leader*" is located on the same node (`seesLeader`).
- Post-Move Sensors: $\emptyset$

- Variables: the number of distinct visited nodes since the agent last switched direction (`nodesSinceDirSwitch`), the number of counterclockwise nodes starting from the initial node that the retroguard has visited, increased by one after each meeting with the leader (`meetingsWithLeader`)
- Conditions:

  c_1: nodesSinceDirSwitch > meetingsWithLeader

  c_2: seesLeader

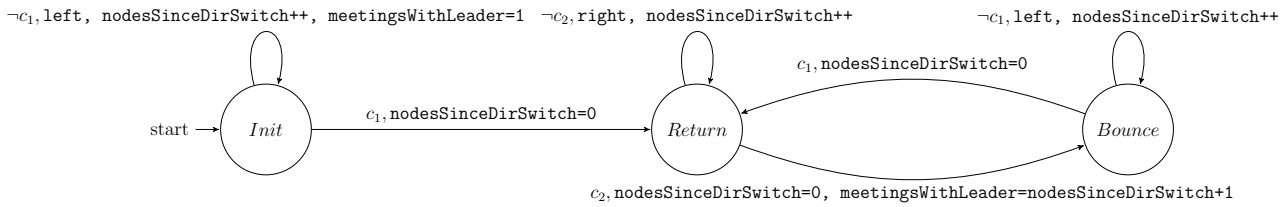

Figure 4.3: The "Retroguard" agent that solves the BHS problem in dynamic rings when the agents can only communicate by seeing which other agents are in the same node as them (vision model) following the algorithm in [1].

Finally, the "*Leader*" agent is defined as:

- $Q = \{$`Init, Move, Cautious, TerminateA, TerminateR`$\}$
- $I = \{$`Init`$\}$
- $F = \{$`TerminateA, TerminateR`$\}$
- $Moves = \{$`wait, right, left`$\}$
- Pre-Move Sensors: the avanguard is on the expected node (`metAvanguard`), the avanguard failed to return to the current node even though the edge is present (`AvanguardFailedReport`), the retroguard did not return within the expected time even though it cannot be blocked by a missing edge (`RetroguardFailedReport`).
- Post-Move Sensors: $\emptyset$
- Variables: the number of distinct visited nodes since the agent last switched direction (`nodesSinceDirSwitch`)
- Conditions:

  c_1: nodesSinceDirSwitch > 0

  c_2: RetroguardFailedReport

  c_3: AvanguardFailedReport

  c_4: metAvanguard

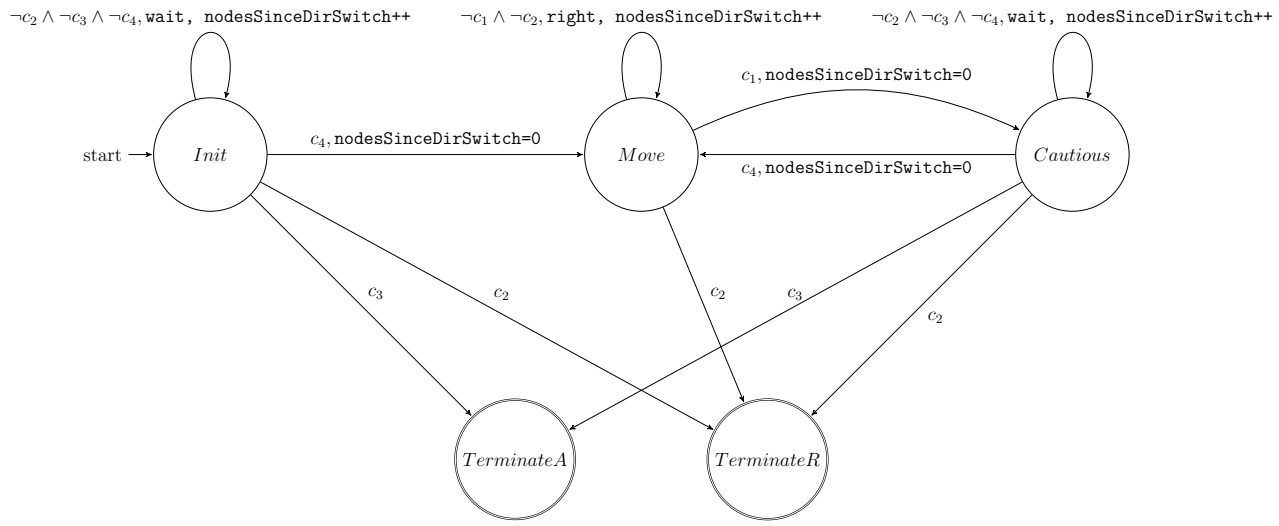

Figure 4.4: The "Leader" agent that solves the BHS problem in dynamic rings when the agents can only communicate by seeing which other agents are in the same node as them (vision model) following the algorithm in [1].

4.7 Semantics

An agent
$A = (Q, \delta, \texttt{I}, \texttt{F}, \texttt{Moves}, \texttt{Conditions}, \texttt{Variables}, \texttt{PreM-Sensors}, \texttt{PostM-Sensors}, \texttt{N})$
walks on a graph G, where G can be any graph. A configuration is described by

1. The position of the agents on the graph, or equivalently the nodes that contain one or more agents.
2. The position of the agents on the node, or equivalently if an agent is placed at a port or in the node.
3. The values of the local variables of each agent.
4. The values of the sensors of each agent.

Using this information, an agent configuration *Conf* is therefore formally defined by a tuple $\langle q, v, s \rangle \in Q \times \texttt{D(vars)} \times \texttt{D(sens)}$.
An agent configuration is initial if q is the initial state, I. From an agent configuration *Conf* we can reach an adjacent configuration *Conf'* if the agent has executed exactly one transition. The agents operate in "Look, Compute, Move" rounds. During the "Look" phase the agent gathers information from its environment by updating the values of the pre-move sensors. In the "Compute" phase the agent uses the values of its local Variables and of its Sensors and it calculates the values of the Conditions. Finally, using all the calculated values, the agent either moves or attempts to move to a neighbouring node during the "Move" phase. Let us now suppose that an agent is in some configuration *Conf*.
Look Phase: In *Conf*, the agent is in some node v of the graph, its local variables have some values and the post-move sensors have either the values that came from the previous move or they are just initialized, if this is the first move. The agent then updates its pre-move sensors.
Compute Phase: Based on the pre-move sensors, the variables and the post-move

sensors, the agent obtains a valuation for the conditions.
Move Phase: Next, based on the value of the conditions and the current state, the agent chooses a move. An agent can only move from a node to another node connected by an edge at each time unit. That is, an agent that occupies a node v at time t can only be at one of the nodes $v-1$, v and $v+1$ at $t+1$. Finally, the agent either attempts to move to a new node or it chooses to remain at its current node. Notice that even if an agent chooses to move, the move might fail. After the move the agent reaches a new state and it updates its post-move sensors and its variables. Now the agent has reached a different configuration, *Conf'*.
Formally, an agent configuration, *Conf*, is defined by the values of the pre-move sensors (`PreM-Sensors`), the move of the agent (`left, right` or `none`) and the values of the post-move sensors (`PostM-Sensors`). The valuation of a condition depends on the values of variables and sensors, and the transition function is given by $\delta = \mathtt{Q} \times \mathtt{Conditions} \rightarrow \mathtt{Q} \times \mathtt{Moves} \times \mathtt{VarUpdates}$. Therefore, an agent configuration, *Conf'*, is the (`PreM-Sensors, move, PostM-Sensors`$'$) successor of *Conf*, if there is a condition c, such that all of the following hold:

1. $c(pre, v, post)$ is *true*
2. $\delta(q, v) = (q', m, update)$
3. $Conf' = \langle q', v', post' \rangle$
4. $v' = \text{Update}(v)$

A global configuration, $Conf_G$, depends on the location of each agent on the graph, as well as the local configurations of the agents. Therefore,

$$Conf_G = \langle (v_1, Conf_1), \ldots, (v_n, Conf_n) \rangle$$

such that v_i is a node in the graph and $Conf_i$ is the configuration of agent$_i$. We call two configurations $Conf_G = \langle (v_1, Conf_1), \ldots, (v_n, Conf_n) \rangle$ and $Conf'_G = \langle (v'_1, Conf'_1), \ldots, (v'_n, Conf'_n) \rangle$ *successors* if the following hold:

1. For each agent, `PreM-Sensors`$_i$ are the pre-move sensor values. In this case we will only consider the case in which the pre-move sensors can give information concerning the location of the agent on the node (i.e., in a port or in the node) and whether or not other agents are on the same node.
2. For each agent, $Conf_i{}'$ is the (`PreM-Sensors`$_i$`, move`$_i$`, PostM-Sensors`$'_i$) successor of *Conf*
3. For each agent, `PostM-Sensors`$'_i$ is the post sensor value for agent$_i$ after taking move `move`$_i$.

Automata like the ones in Figures 4.1 and 4.2 - 4.4 solve the problems we consider when they operates in ring graphs. So we restrict G to be a ring. Furthermore, for any of the problems, more than one automaton needs to opearate in the ring in order to solve the problem (see Lemma 3). We do not make any assumption about the initial positions of the agents. Therefore, in the model presented here, the agents may start from different nodes or be initially co-located.

5

From Distinct to Sequential Identities

In this chapter, we show that a group of mobile agents with unique identities operating in a dynamic ring are able to perform different tasks if they have distinct identities, even if the agents cannot learn the other agents' identities and the initial identities are not predetermined. We begin by presenting an algorithm that assigns sequential identities to agents that are initially placed on the same node and we prove the correctness of our algorithm. Then we prove the same property for agents that are initially placed on different nodes.

Since the agents operate in a distributed setting, all agents follow the same algorithm. Each agent A_i is equipped with two capabilities: it can "see" how many other agents occupy the same node as A_i at any time unit, using sensors, and it can signal one Boolean value to all other co-located agents at each step. As far as signalling is concerned, the disjunction of the co-located agents' Boolean values is given as input to all agents in the same node at each step. In other words, at each time unit, each agent has access to one bit of information, which represents the disjunction of the Boolean value of the signalling variable of each co-located agent. We will prove that the one bit of information that is given as input to the agents, is enough for the agents to be able to assign different tasks to themselves.

Below we give an algorithm that allows co-located agents to assume different tasks based on their identities and know when all other agents on the node have also finished calculating their task. Briefly, the algorithm begins with k initially co-located agents equipped with distinct IDs placed in a common node. Each agent waits and counts a number of time units equal to its original ID. When the number of time units becomes equal to the ID of an agent, say A_{ID}, A_{ID} assigns the next available sequential ID to itself, notifies the remaining agents at the node and all agents locally increase the value of the next available ID by one. When the value of the next available ID becomes equal to k, the agents know that a sequential ID has been assigned to each agent and the algorithm terminates.

The agent or the part of the agent that executes the algorithm is defined as follows:

- $Q = \{$`have orig ID`, `update next seq ID`, `have seq ID`, `other agent got seq ID`, `all agents have seq IDs`$\}$
- $I =$ `have orig ID`
- $F = \{$`all agents have seq IDs`$\}$
- $Moves = \emptyset$
- Pre-Move Sensors = checks if any agent on the node is signalling by having set its signalling variable to *true* (`isSomeAgentSignalling`)

- Post-Move Sensors = $\emptyset$
- Variables = the initial numeric ID of the agent (`id`), the next available sequential ID (`sid`), the number of agents in the system (`k`), the current round (`r`), whether the agent is signalling (`signalling`)
- Conditions =
  c_1 : r < id
  c_2 : isSomeAgentSignalling
  c_3 : sid < k

Before the execution of the algorithm, the variables are initialized in the following way:

- `id`: Can be any natural number, as long as no other agent is assigned the same value.
- `sid`: It is initialized to 1.
- `k`: It is initialized to the number of agents in the system.
- `r`: It is initialized to 1.
- `signalling`: It is initialized to *false*.

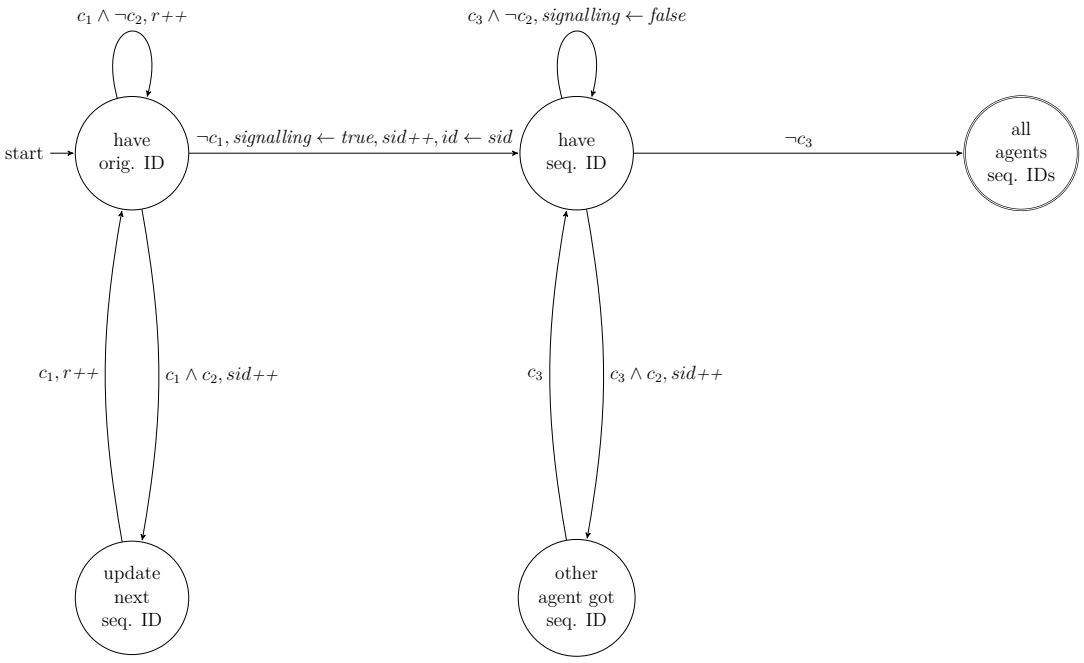

Figure 5.1: Algorithm that allows initially co-located agents to assume sequential identities.

Lemma 1. *Algorithm 5.1 takes a group of k mobile agents that have unique numeric identities and assigns distinct but sequential identities in* $\{1, \ldots, k\}$ *to the agents.*

Proof. The algorithm can be split into four parts, each corresponding to one set of actions:

- Initialization: During the initialization of variables phase, the variables of the agents get assigned common values. The agents do not share memory so assigning values to the variables is local and cannot cause synchronization problems. Furthermore, the agents have enough memory to count the number

of agents on the current node, by our definition, and in the model that we consider the agents can see other agents that reside on the same node during the *Look* phase. Consequently the initial input of all agents is the same.

- States `have orig. ID` and `update next seq. ID`: Those states represent a *while* loop. The agents that do not exit the loop (i.e., by moving to state `have seq. ID`), count the number of rounds, until as many rounds as the initial value of their ID have passed. The IDs of the agents are unique, therefore at most one agent can exit this *while* loop at each round. The agents that do not exit the loop, observe whether at least one agent in the node is signalling. Since all agents are on the same node and operate in synchronous rounds, their input during this step is the same. Updating variables is once again a local action and is a result of the common observation. Consequently, all agents that are still in the while loop have the same values in their local variables.
- Transition from `have orig. ID` to `have seq. ID`: At most one agent, the one that has just exited the while loop (i.e., the one for which c_1 becomes false), can be executing this transition at any time unit. The ID of that agent is equal to the number of the current round r and since each ID is unique, the agent that can execute this part of the algorithm in any round is also unique. Let us call that agent A_s. Agent A_s assigns *sid* to its local ID variable. The *sid* was increased every time another agent set its signalling variable to *true*, detected by c_2. Since the only time the value of the *sid* is used is when an agent assigns it to itself after exiting the while loop between `have orig. ID` and `update next seq. ID`, the current value of the *sid* variable has not been used by another agent. A_s then increases its local value of *sid* and sets its signalling variable to *true*. Since all other agents increase the local value of *sid* when an agent is signalling, local values of the *sid* variables remain synchronized and no more than one agent can assign the same value to its ID.
- States `have seq. ID` and `other agent got seq. ID`: Those two states also represent a while loop. Agent A_s keeps its *sid* variable synchronized with the other agents and waits until all agents have an ID in $\{1,\ldots,k\}$, detected by *sid* becoming equal to the number of agents in the system, k. Therefore, all agents finish executing the algorithm simultaneously when c_3 becomes false and having distinct IDs in $\{1,\ldots,k\}$.

□

Lemma 2. *A group of agents with distinct IDs in a dynamic ring can perform different roles when they meet based on their identities, even if the identities are not predetermined, each agent initially only knows its own identity and face-to-face communication is not possible, assuming the values of the agents' IDs come from an ordered set, each ID is distinct and each agent has enough memory to count the number of agents on the node.*

Proof. In any configuration, the agents can either be initially co-located or initially scattered in the ring.
Let us first consider the case where all agents are initially placed on the same node. As we have shown in Lemma 1, Algorithm 5.1 correctly allows the agents to assume different roles. The assumptions of Algorithm 5.1 match the assumption of this

lemma. Since there exists at least one algorithm that allows initially co-located agents to assume different tasks, the lemma is proved for the case of initially co-located agents.
Let us now consider the case of initially scattered agents. In this case, when agents enter a common node v, this happens in different time units, since all agents are initially placed on different nodes. Therefore, each agent can assign a distinct task to itself by counting how many agents are already on any node it enters. If more than one agents enter the node simultaneously, there are two cases. Either the agents enter the node while moving in the same direction, or they enter the node from different directions. If the agents enter while moving in the same direction, they must have also met in a previous node and already have a way to assign different tasks among themselves. If the agents enter the node while moving in opposite directions, they can use that difference to decide who should compute a new task first among themselves. This can be done in the following way. Let A_{cw} be the agent that entered the node while moving clockwise and let A_{ccw} be the agent that entered the node while moving counterclockwise. Furthermore, suppose that a group of m agents, $\mathcal{A}$, reached the node at some previous step. A_{cw} (resp. A_{ccw}) cannot immediately differentiate between A_{ccw} (resp. A_{cw}) and the agents of $\mathcal{A}$. Out of the two new agents at v, A_{cw} and A_{ccw}, the agent that was moving clockwise, A_{cw}, signals first that it just entered v and changes its identity to $m+1$. In the next step, A_{ccw} observes that A_{cw} signalled, and also signals that it just entered v. A_{ccw} then changes its ID to $m+2$. If no agent entered a node moving clockwise, A_{ccw} detects that there was no signal from another agent and instead of $m+2$, changes its ID to $m+1$. Thus, the lemma is also proved for the case of initially scattered agents.
□

6
Undecidability

When designing this thesis we considered the problem of constructing an automated verification procedure for algorithms that solve the terminating exploration problem in dynamic rings, using mobile agents. However, while studying that problem using the agent capabilities that are commonly assumed in the known distributed computing literature, we discovered that the verification problem becomes undecidable under many of the assumptions. In this chapter, we prove our undecidability results and we show that by combining our undecidability results with the known impossibility results from the distributed computing literature we can define a set of mobile agent capabilities for which the decidability of the verification problem is still open.

As we showed in Chapter 2.6, Exploration is connected to both Black Hole Search and Gathering. In the case of Black Hole Search, if Exploration is not possible the problem cannot be solved. In the case of Gathering we only know that all currently known algorithms for agents that operate in dynamic rings depend on the agents exploring the ring. Due to those results, we check undecidability for Exploration in dynamic rings and the results we obtain also hold for BHS and Gathering under the same assumptions.
Let $\mathcal{R}$ be the set of all ring graphs. Formally, for every $n \in \mathbb{N}$, where n is the size of the graph, there is a graph $R_n = (V_n, E_n, \Sigma, \lambda_n)$ in $\mathcal{R}$ where $\Sigma = \{l, r\}$, $V_n = |v|$ is the set of nodes, $E_n = \{(v_i, v_j) | v_i \in V, v_j \in V \text{ and } v_i \neq v_j\}$ is the set of edges and the label λ_n of each edge that is of the form $(v_i, v_{i+1 \bmod n})$ is r and of each edge of the form $(v_{i-1 \bmod n}, v_i)$ is l. We will prove that if an algorithm requires the agents to be able to use even one counting variable, the problem is undecidable. The proof will be by reduction from the non-halting problem of 2-counter machines (2CM). An input free 2-counter machine is a deterministic program manipulating two integer counters using commands that can increment a counter by one, decrement a counter by one and check whether a counter is equal to zero. We will first show that the exploration problem cannot be solved in dynamic rings by only one agent, so at least two agents are needed by any correct algorithm.

Lemma 3. *Any problem that requires agents to move in order to complete a task (like Exploration, Gathering or BHS) in dynamic rings cannot be solved by a single agent, even if that agent has unlimited memory and knows the size of the ring.*

Proof. Let the agent be initially located on some node v. The agent cannot remain at v indefinitely, otherwise the ring is not explored. Let the agent first attempt

to move clockwise (resp. counterclockwise). The adversary then removes the edge incident to v that leads to its clockwise (resp. counterclockwise) neighbour. Call that edge e_r (resp. e_l). The agent can continue attempting to move clockwise (resp. counterclockwise), choose to remain at v or move counterclockwise (resp. clockwise). In the first case, the adversary keeps e_r (resp. e_l) removed and the agent does not move and therefore does not explore the ring. In the second case the ring is also not explored. Therefore, the agent will eventually need to attempt to move counterclockwise (resp. clockwise) using edge e_l (resp. e_r). However, when the agent attempts to move, the adversary can remove e_l (resp. e_r). Therefore, it is always possible for the adversary to remove the edge the agent attempts to use and not allow the agent to leave its initial node, thus not allowing the agent to explore the ring. As a result, one agent cannot leave its initial node in a dynamic ring. □

In the following lemmas, we address various cases using different numbers of agents operating in a dynamic ring and different agent capabilities and we prove that the problem of deciding if a given algorithm is correct becomes undecidable under many of the most common assumptions in [8, 59, 60, 9, 1]. We split our results into two parts. The first part, corresponds to Lemmas 4 - 7. In this part, we assume that the agents only have "vision" which is the weakest communication mechanism. This means that agents can only see how many other agents are located at the node they currently occupy but they cannot exchange any other information with those agents. In the second part, we focus on decreasing the number of agents that need to operate in the ring to prove that the problem is undecidable. To do that, we assume that this time the agents are able to exchange one bit of information at each round in addition to being able to detect how many other agents are located on a common node. The lemmas of this part, Lemmas 8 - 11, also hold for any other communication mechanism, provided that in that mechanism the agents are also able to exchange at least one bit of information. For example, such communication mechanisms are tokens/pebbles and face to face communication. Also, Lemmas 8 - 11 do not depend on agents moving, so they also hold for agents in static rings. We summarize the undecidability results that we will prove later in this chapter in Table 6.1.

6.1 Agents That Only Have Vision

In the following lemmas the agents cannot signal information to each other without moving, since their only communication capability is vision. Therefore, the agents must leave their current node in order to exchange information. However, since the ring is dynamic, when an agent attempts to leave a node to signal information, a missing edge can prevent the agent from moving. Hence, in our proofs we need to differentiate between an agent not moving as part of the protocol and an agent failing to move due to a missing edge.

Lemma 4. *If an algorithm for solving the exploration problem in dynamic rings depends on at least four co-located agents, each agent is equipped with two ring-size dependent variables, a distinct ID and has chirality, the automated verification*

Table 6.1: *Summary of our undecidability results. All agents in all cases have distinct but unknown identities. In the cases where the agents do not begin from the same node (i.e., scattered) we assume there is also a marked node somewhere in the ring.*

Lemma	# of Agents	Initial Positions	# of "Ring Size Dependent" Variables	Communication
4	4	co-located	2	Vision
5	4	scattered	2	Vision
6	6	co-located	1	Vision
7	6	scattered	1	Vision
8	2	co-located	2	One bit & Vision
9	3	scattered	2	One bit & Vision
10	3	co-located	1	One bit & Vision
11	5	scattered	1	One bit & Vision

problem is undecidable.

Proof. The agents are initially placed at the same node and have distinct identities. Therefore, the four agents can form two pairs, which is possible as we have shown in Lemma 2. We say that agents A_1 and A_2 form pair P_1 and agents A_3 and A_4 form pair P_2. Furthermore, since the agents are on the same node and have chirality, we can make P_1 pick as initial direction the clockwise direction and P_2 pick counterclockwise as its initial direction. Each agent uses one variable to count the computation steps and the other variable to simulate the register the agent follows. We say that the agents that belong to the same pair each simulate one of the registers of the 2CM. That is, A_1 and A_3 simulate register x and A_2 and A_4 simulate register y. The agents then operate in the following manner. All agents count the number of computation steps. When register x is decremented or incremented, A_1 and A_3 increment or decrement their variable respectively. When register y is decremented or incremented, A_2 and A_4 increment or decrement their variable respectively. When it is checked whether register x (resp. y) is equal to 0, A_1 and A_3 (resp. A_2 and A_4) check whether their variable is equal to zero. Then, if the variable is not equal to 0, A_1 and A_3 (resp. A_2 and A_4) remain on the current node for two time units, t and $t+1$. Otherwise, if the variable is equal to 0, A_1 and A_3 (resp. A_2 and A_4) move one step to their chosen direction at t and one step back to the initial node at $t+1$. The remaining agents, in this case A_2 and A_4 (resp. A_1 and A_3), always attempt to move one step towards their chosen direction at t and return to the initial node at $t+1$. This way, A_2 and A_4 (resp. A_1 and A_3) always know if A_1 and A_3 (resp. A_2 and A_4) were blocked by a missing edge and as a result they were blocked from signaling the value of their variable. A missing edge is detected in the following way:

- *One of the agents, say A_1, checks the value of its variable and the value is 0.* In this case, both A_1 and A_2 attempt to move one step to their chosen

direction and they both detect that the edge is missing.

- *One of the agents, say A_1, checks the value of its variable and the value is not 0.* In this case, A_1 does not move in the following step and it detects that A_2 did not move either. Since A_2 always moves when A_1 checks if its variable is equal to 0, A_1 infers that the edge is missing. A_2 also knows that the edge is missing since its move failed.
- *An agent that moved to a neighbouring node did not return after one time unit.* In this case, the agent that tried to move learns that the move failed and the other agent that belongs to the pair infers that the edge is missing.

After a pair of agents detects that an edge is missing, the pair that detected the missing edge waits for the edge to return, before continuing the simulation of the 2CM. If the edge does not return, then the remaining pair finishes the simulation of the computation and two agents are enough to explore the ring. If the edge returns, then at least one of the pairs simulates a computation step at each round and eventually, one of the pairs finishes the simulation and begins exploring. Finally, if the 2CM halts before making n computation steps, the agents cannot count up to the number necessary for the completion of the algorithm and we make the agents stay in place in an infinite loop. Otherwise, the agents follow the computation of the 2CM for n computation steps and then explore the ring. Since the size of the ring can be any natural number, it cannot be guaranteed that the 2CM will not halt and exploration will be solved, thus, the problem is undecidable. □

Lemma 5. *If an algorithm for solving the exploration problem in dynamic rings depends on at least four initially scattered agents, each agent is equipped with two ring-size dependent variables, a distinct ID, has chirality and there is a marked node in the ring the automated verification problem is undecidable.*

Proof. The agents begin from different nodes in the ring and have chirality. Therefore, they can attempt to reach the marked node by all moving in the same direction, say counterclockwise. The following cases are possible:

1. *All agents reach the marked node.* This case is the same as the one described in Lemma 4 and we have proved it is undecidable.
2. *Three agents reach the marked node and one agent is blocked by a missing edge.* In this case two agents form a pair and the remaining agent in the marked node, waits. This time, the agents reach the marked node in different time units. We call A_1 the first agent to reach the marked node, A_2 the second agent to reach the marked node and A_3 the third agent to reach the marked node. Each agent that enters the marked node can see how many other agents are also in the node and the agents are able to assign different tasks to themselves by counting the number of agents that are already on the marked node. Let the first two agents that reach the marked node (A_1 and A_2) form a pair and use their distinct IDs to each simulate one of the registers of the 2CM. Say A_1 and A_2 have formed a pair, A_3 is in the marked node, A_4 is blocked by a missing edge, A_1 simulates register x and A_2 simulates register y. The pair now begins simulating the 2CM in the following way. Each agent uses one variable to count the number of computation steps and the other variable to simulate its assigned register. Checking if one of the registers is equal to zero is

done by a back and forth move, as we described in the proof of Lemma 4. The first pair to be formed chooses counterclockwise (i.e., the same direction they moved when searching for the marked node) as the direction the agents use to signal that a variable is zero. Otherwise, an adversary can block both A_4 and the pair by removing the edge between the marked node and its clockwise neighbour. If the pair is blocked while simulating the 2CM, A_4 must have been unblocked and it will eventually reach A_3. If A_4 reaches A_3, A_4 and A_3 form a pair and simulate the 2CM, choosing clockwise (i.e., the opposite direction of the first pair) as the direction they move to signal or check whether a variable is equal to zero. Furthermore, since all agents are now in the same node and have formed pairs we can use the same argumentation as in the proof of Lemma 4. That is, the adversary can only block one of the two pairs at each move, therefore, one of the pairs always simulates a step of the 2CM and we have already shown in Lemma 4 that the problem is undecidable in this case.

3. *Two agents reach the marked node and two agents are blocked by a missing edge.* Two pairs have been formed. The pair in the marked node (call it P_1) begins the simulation and chooses counterclockwise (towards the initial direction) as the direction for checking if a variable is zero. The blocked pair (call it P_2), also begins the simulation of the 2CM and also chooses counterclockwise as its direction. By having both pairs choose the same direction as the direction used for signaling and checking whether a variable is equal to zero, the adversary cannot block both pairs by removing one edge. If P_1 becomes blocked by a missing edge when checking for a zero, P_2 must have been unblocked. While the two pairs are not on the same node, either P_1 simulates the 2CM, P_2 simulates the 2CM or both pairs simulate the 2CM.
4. *One agent reaches the marked node and three agents are blocked by a missing edge.* In order to avoid being blocked by a missing edge the agents perform the following moves. Two of the three agents on the marked node form a pair, P_1, and choose counterclockwise (the same direction as the initial) as the direction they use to signal or check a zero. P_1 begins simulating the 2CM. The remaining agent (call it A_3) that is blocked by a missing edge, changes direction and moves until it finds the marked node. At each rounds, the adversary can either block P_1 from simulating the 2CM or A_3 from reaching the marked node. Thus, either the 2CM is able to continue the simulation of the 2CM or A_3 and the agent on the marked node meet and form another pair, P_2. That pair chooses counterclockwise (the same direction as the initial) as the direction they use to signal or check a zero and the adversary cannot block both pairs from simulating the 2CM by removing an edge.
5. *All agents are blocked by a missing edge.* The agents have all managed to meet and they can continue the simulation using the method in Lemma 4.

Since the agents are able to distinguish the marked node, see if an edge is missing and count the number of agents on a node, they are able to infer in which of the above cases they are and perform the corresponding actions. Therefore, in every case the problem is undecidable for four initially scattered agents that have distinct IDs, chirality, two variables that depend on the size of the graph and move in a dynamic ring. □

Lemma 6. *If an algorithm for solving the exploration problem in dynamic rings depends on at least six co-located agents, each agent is equipped with one ring-size dependent variable, a distinct ID and has chirality the automated verification problem is undecidable.*

Proof. The agents are initially placed at the same node and have distinct identities, so they can form teams and decide on different roles, due to Lemma 2. Like in Lemma 4, we say that agents A_1 and A_2 form pair P_1 and choose the clockwise direction for signaling and that agents A_3 and A_4 form pair P_2 and choose the counterclockwise direction for signaling. The remaining agents, A_5 and A_6, only leave the initial node once, when signalling. A_5 moves in the same direction as P_1 and A_6 moves in the same direction as P_2. One agent from each pair simulates one of the registers of the 2CM. That is, A_1 and A_3 simulate register x and A_2 and A_4 simulate register y. Finally, A_5 counts the number of computation steps of P_1 and A_6 counts the number of computation steps of P_2, by incrementing their respective variable by one once every three time units if their corresponding pair is not blocked. The proof of this lemma follows the proof of Lemma 4 with the following difference. This time, each round lasts for three time units, instead of two. The first time unit is reserved for A_5 and A_6 to signal whether n simulation steps have passed. During the first time unit reserved for the counting agents, the agents that have formed pairs only observe A_5's and A_6's move (or lack of move) without simulating the 2CM. In the following two time units the pairs simulate the 2CM as described in the proof of Lemma 4. When A_5's (resp. A_6's) count reaches n computation steps, it signals it to the agents of P_1 (resp. P_2). To signal, A_5 (resp. A_6) moves one step clockwise (resp. counterclockwise) and then returns to the node. Since during that time unit no other agent moves, the agents that have formed pairs, infer that the agent that moved is a counting agent. There are three cases for each pair. We will use P to refer to either pair and A_c to refer to the counting agent corresponding to P.

1. The agents of P and A_c are on the initial node. This is possible when the edge P uses to signal a zero was not missing in the previous round or if P did not perform a check for zero. In this case, when A_c signals that n computation steps have been simulated, all agents of P know.
2. A_c and one of the agents of P are on the initial node. This is possible when one of the agents of P (say A_x) checks if the other agent's (say A_y) variable (say z) is zero at computation step $n-1$ and $z \neq 0$. In this case, A_x moves from the initial node, v_i, to the designated neighbour, v_{neigh}. Let the edge between v_i and v_{neigh} disappear before A_x has returned to v_i. When A_c signals that n computation steps have been simulated in the next round, only three of the agents are on the node and A_x does not learn that n computation steps have been simulated. Either the edge between v_i and v_{neigh} returns or it does not. In the first case, A_x learns that n computation steps have been simulated by a signalling move during the first time unit of the next round from A_c. In the latter case, A_x does not learn that the simulation is complete but the other pair is enough to explore the ring and it is not blocked by a missing edge since the missing edge is the one between v_i and v_{neigh}.
3. Only A_c is on the initial node. This is possible when one of the agents in a pair (say A_x) checks if the other agent's (say A_y) variable (say z) is zero at

computation step $n-1$ and $z=0$. In this case, both A_x and A_y move from the initial node, v_i to the designated neighbour, v_{neigh}. Let the edge between v_i and v_{neigh} disappear before A_x and A_y have returned to v_i. When A_c signals that n computation steps have been simulated, A_x and A_y are not on v_i and consequently do not learn that n computation steps have been simulated. Either the edge between v_i and v_{neigh} returns or it does not. In the first case, A_x and A_y learn that n computation steps have been simulated by a signalling move during the first time unit of the next round from A_c. In the latter case, A_x and A_y do not learn that the simulation is complete but the other pair is enough to explore the ring and it is not blocked by a missing edge since the missing edge is the one between v_i and v_{neigh}.

Since at most one edge is missing in each time unit and the adversary cannot block both pairs by removing the same edge, if one of the pairs, P, is in Case 2 or in Case 3 the other pair, P' has to be in Case 1. So, in every case the problem is undecidable. □

Lemma 7. *If an algorithm for solving the exploration problem in dynamic rings depends on at least six initially scattered agents, each agent is equipped with one ring-size dependent variable, a distinct ID, has chirality and there is one marked node in the ring the automated verification problem is undecidable.*

Proof. The proof in this case follows the same reasoning as the proof of Lemma 5, with the difference that this time, the goal is for the agents to form teams of three agents instead of pairs. Since there is a marked node and the agents have chirality all agents can start moving in the same direction, say counterclockwise, trying to reach the marked node. If one or two agents are blocked by a missing edge, they continue attempting to move in the same direction, by waiting for the edge to return. If three agents are blocked by a missing edge they form a team and begin simulating the 2CM, using the original direction, counterclockwise, for signalling a zero. The following cases are possible:

1. *All agents are on the marked node.* The agents can form two teams and simulate the 2CM following the proof of Lemma 6.
2. *Three agents are blocked by a missing edge and three agents reach the marked node.* In this case, the agents can form teams of three, where each team consists of one agent simulating register x, one agent simulating register y and one agent counting the number of computation steps of the 2CM. The two teams must choose a common direction as the direction for signalling a zero, otherwise the adversary can block both teams from simulating the 2CM by removing one edge. Let the agents choose the same direction as the initial direction of movement, in this case counterclockwise. The two teams can independently simulate the 2CM following the method described in the proof of Lemma 6. Since the adversary can only block one of the pairs in each round, eventually at least one of the pairs finishes the simulation phase.
3. *The majority of agents are blocked by a missing edge.* The first three agents that are blocked by a missing edge form a team and begin attempting to simulate the 2CM, using counterclockwise as the direction for signaling a zero. If another agent, A, reaches the team already simulating the 2CM, A changes

direction and attempts to reach the marked node. If A is blocked at any step, the team that was originally blocked must have been unblocked and it can simulate the 2CM. Either the team will finish the simulation, or the agent(s) that reach the team and change direction reach the marked node. Notice that this means that it is not possible for all agents to be blocked by a missing edge, so it does not need to be a separate case. Therefore, either the team that is not on the marked node finishes the simulation or the agents that are not part of that team form another team in a different node and simulate the 2CM using counterclockwise (i.e., the initial direction) for signaling a zero or reach the marked node and simulate the 2CM using counterclockwise (i.e., the initial direction) for signaling a zero. In all cases, the adversary can block at most one team by removing an edge and eventually, one of the teams will finish the simulation phase.

4. *The majority of agents reach the marked node.* In this case, the first three agents that reach the marked node form a team, choose counterclockwise (that is, the initial direction) as the direction for signaling a zero and begin simulating the 2CM. The remaining agents always attempt to move counterclockwise to reach the marked node, since at least three agents are needed for a group of agents to stop and form a team. Therefore, either the team of agents on the marked node will complete the simulation phase without being blocked or the team will be blocked but the rest of the agents will become unblocked and eventually reach the marked node. In the latter case, a second team is formed in the marked node and the simulation continues in the same way as in the proof of Lemma 6.

Consequently, in every case the problem is undecidable. □

In addition to the results of Lemmas 4 - 7, there are some impossibility results in [8] that further restrict the assumptions that can be made for an algorithm that solves the terminating exploration problem in dynamic graphs. Namely, any number of agents operating in a dynamic ring that has no knowledge of the size of the ring, does not depend on a marked node and uses agents that do not have distinct IDs, cannot solve the terminating exploration problem in a dynamic ring. Combining those impossibility results with our undecidability results, further restricts the possible assumptions that would make the automated verification problem decidable.

Theorem 1. *Every automated decision procedure for the correctness of exploration algorithms in a dynamic ring, where agents have at least one variable that can count to the size of the ring, must depend on the number of agents being less than six, specific initial conditions, or very restricted forms of communication.*

Proof. In Lemmas 4 - 7 we study the automated verification problem for a very restricted method of communication between the agents. That is, any agent, A_i, operating in a dynamic ring is only able to see how many other agents are located on the same node, v, as A_i but does not have any other information about any of the moves of the other agents before reaching v. Let us first consider the case in which all agents are initially placed on the same node. As shown in Lemma 6, even six agents that can only "see" each other and use one variable to count, without

exchanging any additional information, make the exploration problem undecidable. Similarly, the presence of six initially *scattered* agents that can use a variable to count and can "see" other co-located agents, also make the exploration algorithms undecidable. Furthermore, due to Lemma 3, we know that one agent is not enough to solve exploration in dynamic rings. Taking the impossibility results from [8] into account as well, algorithms that may be decidable by an automated decision procedure, can:

- Have between two and five agents operate in the dynamic ring, due to Lemmas 3, 6 and 7.
- Have distinct but unknown IDs for the agents from [8] and Lemmas 6 and 7.
- Assume there is a marked node in the ring from [8] and Lemmas 6 and 7.
- Have no restrictions on the initial positions of the agents from Lemmas 6 and 7.
- Have at most one ring size dependent variable per agent from Lemmas 6 and 7.

□

6.2 Agents That Have Vision and One Bit of Information

The main methodology difference in the following lemmas is that the agents can now transfer limited information to other agents located on the same node. More precisely, each agent is able to signal one bit of information in the form of a Boolean value, and all other co-located agents receive the value corresponding to the disjunction of the values being signalled by the agents on the same node. Hence, the agents are able to exchange sufficient information to simulate the 2CM without needing to move to a different node and as a result they cannot be blocked by a missing edge at any step of the simulation.

Lemma 8. *If an algorithm for solving the exploration problem in dynamic rings depends on at least two co-located agents, each agent is equipped with two ring-size dependent variables, a distinct ID and can transmit one bit of information in each round the automated verification problem is undecidable.*

Proof. The agents are initially placed at the same node and have distinct identities. Therefore, the two agents can assume different roles as we have shown in Lemma 2. One agent, say A_s, *simulates* the 2CM and the other agent, say A_c, *counts* the number of simulation steps performed by A_s. Let the two variables of each agent be v_1 and v_2 and the two counters of the 2CM be x and y. Agent A_s uses one of its variables, say v_1, to simulate one of the 2CM's counters, say x. Similarly, A_s uses its other variable, v_2, to simulate the remaining counter of the 2CM, y. The simulation is performed in the following way:

- When register x is incremented (resp. decremented) by 1, A_s increments (resp. decrements) v_1 by one.
- When register y is incremented (resp. decremented) by 1, A_s increments (resp. decrements) v_2 by one.

- When it is checked if register x (resp. y) is equal to 0, A_s checks if v_1 (resp. v_2) is equal to zero.

The simulation of each computation step of the 2CM (i.e., incrementing a variable, decrementing a variable or checking if a variable is equal to zero) takes one time unit. The counting agent, A_c, uses one of its variables, say v_1 to count the number of computation steps and does not use its second variable. If A_s finishes the simulation of the computation of the 2CM, it signals it to A_c by setting the value of the Boolean variable that the two agents use to communicate to *true*. Similarly, if A_c counts n computation steps it signals it to A_s by setting the value of the Boolean variable to *true*. If the 2CM halts before making n computation steps, the agents cannot count up to the number necessary for the completion of the algorithm and we make the agents stay in place in an infinite loop. Otherwise, the agents follow the computation of the 2CM for n computation steps and then solve exploration in the ring. Since the size of the ring can be any natural number, it cannot be guaranteed that the 2CM will not halt and exploration will be solved, thus, the problem is undecidable. □

Lemma 9. *If an algorithm for solving the exploration problem in dynamic rings depends on at least three initially scattered agents, each agent is equipped with two ring-size dependent variables, a distinct ID, can transmit one bit of information in each round has chirality and there is one marked node in the ring the automated verification problem is undecidable.*

Proof. The agents all start from different and unknown nodes and there is a marked node at the ring. First all agents begin moving in the same direction (say clockwise) until they reach the marked node. The goal is for at least two agents to meet on a node and then follow the simulation of the 2CM described in Lemma 8. Either all agents eventually reach the marked node or a number of agents are blocked by a missing edge. When an agent (call it A_1) is blocked by a missing edge, it waits on its current node, v, until the missing edge returns or until another agent reaches v. Let us first consider the case of at least one more agent, say A_2, reaching v before the edge returns. A_1 and A_2 entered v at different times so they have a way to break the symmetry between them and assign different roles to themselves. The two agents simulate the 2CM in the same way as in Lemma 8 with one agent (say A_1) using both its variables to simulate the counters of the 2CM and the other agent (say A_2) using one variable to count the computation steps. In the case where the missing edge reappears at some later time but before any other agent reaches v, A_1 continues moving clockwise and it either manages to reach the marked node or it is again blocked by a missing edge and the same argumentation holds. Finally, if the edge does not return and no other agent reaches v, the remaining two agents must have both reached the marked node and stopped moving. In all cases, at least two of the three agents eventually meet either at the marked node or when they encounter a missing edge. After two agents meet, the agents are able to assign different tasks to themselves (due to reaching a common node at different time units) and follow the method described in Lemma 8 to simulate a run of the 2CM. □

Lemma 10. *If an algorithm for solving the exploration problem in dynamic rings depends on at least three co-located agents, each agent is equipped with one ring-size*

dependent variable, a distinct ID and can transmit one bit of information in each round the automated verification problem is undecidable.

Proof. The proof of this lemma is similar to the proof of Lemma 8, with the difference that instead of simulating both counters of the 2CM using one agent, this time we use two agents. Let A_1, A_2 and A_c be the three agents in the ring. A_1 and A_2 each use their variable to simulate one of the counters of the 2CM. Let A_1 simulate counter x and A_2 simulate counter y. We refer to A_c as the "counting agent" and we use it to count the number of simulation rounds that have passed. Each simulation round lasts three time units:

- In the first time unit, t, agent A_c signals if n computation rounds have passed and if so, the simulation stops.
- In the second time unit, $t+1$, A_1 simulates counter x and signals if its variable's value is zero to A_2.
- In the third time unit, $t+2$, A_2 simulates counter y and signals if its variable's value is zero to A_1.

When register x (resp. y) is incremented (resp. decremented), agent A_1 (resp. A_2) increments (resp. decrements) its variable. When it is checked if register x (resp. y) is equal to zero, A_1 (resp. A_2) checks if the value of its variable is zero. If it is, A_1 (resp. A_2) signals it to A_2 (resp. A_1) by setting its Boolean variable to true for one round. In the next round, A_1 (resp. A_2) resets its Boolean variable to zero. □

Lemma 11. *If an algorithm for solving the exploration problem in dynamic rings depends on at least five initially scattered agents, each agent is equipped with one ring-size dependent variable, a distinct ID, can transmit one bit of information in each round, has chirality and there is one marked node in the ring the automated verification problem is undecidable.*

Proof. Since the agents have chirality, they begin moving in the same direction, say clockwise, until they reach the marked node or until they are blocked by a missing edge. The goal is for at least three agents to meet at a common node and then follow the procedure described in Lemma 10. Following the same argumentation as in Lemma 9, either at least three agents manage to reach the marked node, or at least three agents are blocked by a missing edge. In either case, at least three agents manage to reach a common node. Furthermore, due to Lemma 2 the agents can assume different tasks based on their identities, therefore, the at least three agents that meet on a node can simulate the 2CM following the procedure presented in Lemma 10 and the problem is undecidable. □

Theorem 2. *Every automated decision procedure for the correctness of exploration algorithms in a ring, where agents have at least one variable that can count to the size of the ring and where agents in the same node can exchange one Boolean value, must depend on the number of agents being less than five, specific initial conditions, or very restricted forms of communication.*

Proof. This theorem follows the same argumentation as Theorem 1. In Lemmas 8 - 11 we increase the communication capabilities between the agents, by assuming the agents are able to send and receive one bit of information, in addition to knowing

the number of other agents on the node. By adding this communication capability, two things happen:

1. The initial position of the agents (i.e., co-located or scattered) affects the number of agents that make the automated verification problem undecidable.
2. In the case of initially co-located agents, the automated verification problem is also undecidable in static rings under the same assumptions.

This time the restrictions to possibly get a decidable version of the automated verification problem for terminating exploration algorithms in dynamic graphs, become:

- Have two agents operating in the dynamic ring, due to Lemmas 3 and 10, if the agents are initially co-located.
- Have between two and four agents operating in the dynamic ring, due to Lemmas 3 and 11, if the agents are initially scattered.
- Have distinct but unknown IDs for the agents from [8] and Lemmas 10 and 11.
- Assume there is a marked node in the ring from [8] and Lemmas 10 and 11.
- Have no restrictions on the initial positions of the agents from Lemmas 10 and 11.
- Have at most one ring size dependent variable per agent from Lemmas 10 and 11.

□

7

Regular Model Checking

In this chapter, we provide a way to use regular model checking (RMC) to check whether an algorithm in correct. As we described in Chapter 2, RMC is an incomplete method, but it has been proven to terminate under specific conditions for parameterized systems. We begin this section by discussing why model checking cannot be used for the existing algorithms for the problems we consider, since the known algorithms depend on the size of the ring. Next, we talk about regular model checking and we provide a way to transform the formalization of an agent from Section 4.6 to the form that is needed to use RMC techniques. We begin by showing a specific example of this transformation and then we move to describing the transformation for the general case.

7.1 Algorithms with agents that use counting variables

Similarly to the undecidability results of the previous chapter the presence of the ring size dependent variables is still a problem for deciding if an algorithm solves Exploration in dynamic rings using model checking. Namely, the unbounded variables are a problem because in order to represent all the possible configurations we would need a state for each possible value of the variables' values. Since we need to be able to represent the algorithm for any size of graph, the presence of even one counting variable prevents us from representing the algorithm when using model checking. On the other hand, *regular* model checking was introduced specifically for analyzing a subset of parameterized and infinite-state systems. Therefore, the size of the environment being given as a parameter to the algorithms we consider is not prohibitive to further studying RMC methods.

7.2 Motivation for RMC

Regular Model Checking is a framework that is used for the verification of infinite and parameterized systems. Since RMC is a model checking technique, its goal is to automatically verify system properties. As we described in Chapter 2, RMC represents sets of states using finite state automata and transition relations using finite state transducers. In general, system configurations can be represented as words or tree structures in the context of RMC. However, in this thesis we choose to only represent system configurations as words.

The motivation for presenting a way to transform algorithms that solve Exploration, Gathering and BHS into the components that are needed for the application of RMC techniques, is to provide a possible way for automated verification in view of the undecidability results of the previous chapter. Even though RMC is not guaranteed to terminate since the problems we discuss are undecidable, it has been shown that RMC methods terminate sufficiently often (Chapter 2). Even though we do not discuss specific RMC algorithms in this thesis, we show how to construct all the components that are needed for the application of RMC algorithms such as the ones presented in [57].

7.3 Moving in a Ring

As we showed in Section 4.7, in order to represent the global configuration we need:

1. The position of the agent in the graph (in this case, a ring).
2. The internal configuration of the agent. That is, the values of the agent's pre-move sensors, the move direction (if any) and the values of the agent's post-move sensors.

Let us begin by the representation of an agent moving in a ring. At any given time, an agent might be moving counterclockwise, moving clockwise or not moving. Let Σ be a finite alphabet of symbols. We can represent an agent moving clockwise in a ring with $\Sigma = \{e, o\}$ where e represents a node that does not contain an agent (i.e., the node is empty) and o indicates that a node is occupied by an agent. We will construct an automaton $\mathcal{A}_l = \{Q, q_0, t, F\}$ that accepts the language of valid left moves of an agent in a ring. In this case, $Q = \{q_0, \ldots, q_5\}$ is the set of states, q_0 is the initial state, $F = \{q_4, q_5\} \subseteq Q$ is the set of accepting states and $t := Q \times (\Sigma \times \Sigma) \mapsto Q$ is the transition relation. We use $L(\mathcal{A}_l)$ to denote the language of $\mathcal{A}_l$. The words that are accepted by $\mathcal{A}_l$ are of the form (x, y), where $x = a_1 \ldots a_n$ and $y = b_1 \ldots b_n$ and $(a_1, b_1) \ldots (a_n, b_n) \in L(\mathcal{A}_l)$. This representation allows us to simulate a transition between two configurations of a given system, in this case, of an agent moving counterclockwise in a ring. In the context of regular model checking (RMC) an automaton of the form of $\mathcal{A}_l$ is called a transducer [57]. More generally, a transducer is a way to represent the transitions of a system from some configuration *Conf* to a successive configuration *Conf'*. The automaton $\mathcal{A}_l$ is represented in Figure 7.1.

Similarly, we can also create another automaton, $\mathcal{A}_r$ that represents an agent moving clockwise in a ring. That automaton is depicted in Figure 7.2.

Let us now briefly discuss how a transducer of that form is read. As an example we will use the transducer of Figure 7.1, which describes an agent moving counterclockwise in a ring. Suppose we have a ring graph $\mathcal{R}$ that consists of four nodes and an agent is initially placed on node v_2. This initial configuration is shown in Figure 7.3. Equivalently, we can describe that configuration with the sequence "*eeoe*", where e represents an empty node, o represents a node occupied by an agent and each letter of the sequence corresponds to one node of $\mathcal{R}$. In this example, we consider the case of an agent moving counterclockwise in a ring. After each step, the agent moves to the counterclockwise neighbouring node, so from v_i to $v_{i+1 \mod 4}$. The corresponding sequences to the moves of the agent are:

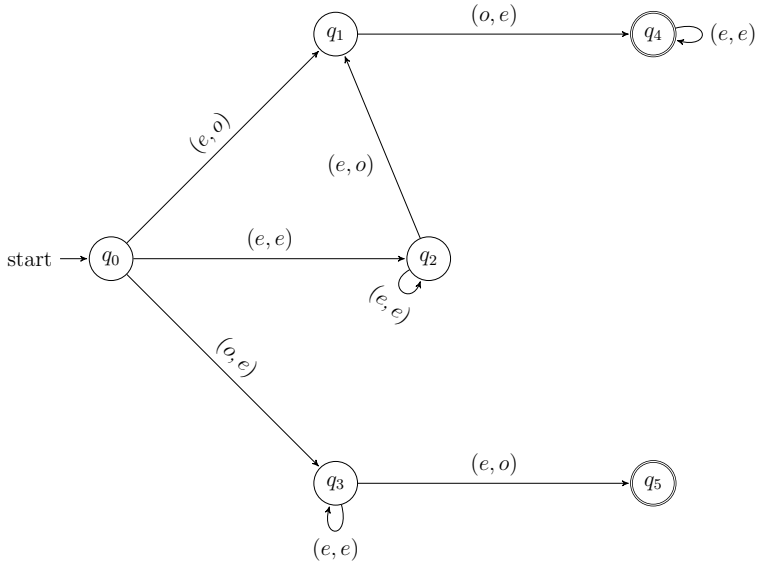

Figure 7.1: An automaton representing an agent that moves counterclockwise in a ring.

$$eeoe \rightarrow eoee \rightarrow oeee \rightarrow eeeo \rightarrow eeoe$$

A transducer describes the move from one configuration to the next. So in our example, the transducer in Figure 7.1 should be able to represent any move from a node to its counterclockwise neighbour. For example, let us consider the move of the agent from v_2 to v_1. The corresponding sequences are *eeoe* and *eoee*. Using the definition of the language accepted by a transducer in the beginning of this section, $x = eeoe$ (an agent is on v_2), $y = eoee$ (an agent is on v_1) and the corresponding word of the language that is accepted by the transducer is $(e,e)(e,o)(o,e)(e,e)$ where each pair is of the form (x_i, y_i). By looking at the transducer in Figure 7.1, the word indeed belongs to the language of an agent moving counterclockwise in a ring. On the contrary, words describing an agent moving clockwise in the ring such as $(e,e)(o,e)(e,o)(e,e)$ or an agent moving counterclockwise but for more than one nodes in a single step such as $(e,e)(e,o)(e,e)(o,e)$ are correctly not accepted by the transducer in Figure 7.1. Furthermore, the size of the ring between any two successive configurations cannot change since the word would contain a pair of the form $(\_,e), (\_,o), (e,\_)$ or $(o,\_)$ which are not part of the language and therefore can never reach an accepting state.

Transducer for Exploration in dynamic rings

The problems we consider in this thesis are more complex than an agent simply moving in the ring and as a result the transducer describing the legal moves that are defined by an algorithm is also more complex. In this section, we will construct the transducer corresponding to the algorithm described by the agent of Figure 4.1, which solves the Exploration Problem in dynamic rings of known size for agents that do not have chirality. As part of the algorithm, the agent can move left or move

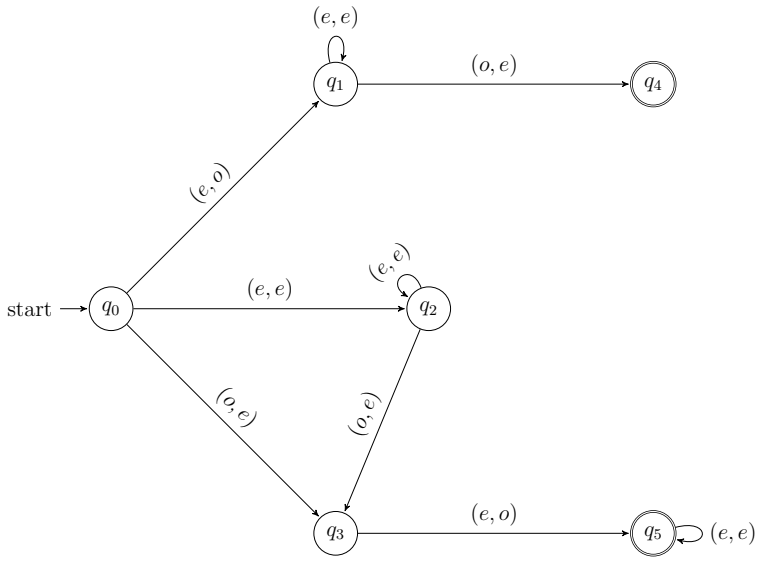

Figure 7.2: An automaton representing an agent that moves clockwise in a ring.

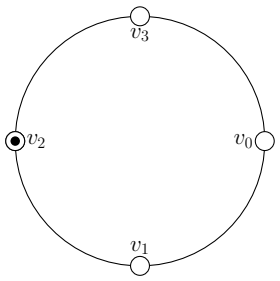

Figure 7.3: A ring $\mathcal{R}$ that consists of four nodes and contains an agent initially placed on node v_2.

right depending on the valuation of its conditions. That agent consists of four states (Init, Fwd, Bounce and Terminate), so we split the description of the transducer to parts corresponding to those states.

1. *Init*: In this state, the agent checks the values of conditions c_1 and c_3. If one of those conditions is true the agent must transition to a different state, otherwise, the agent must move on step to the left. Since we can only use one letter to describe the state of the agent, we will look at all possible valuations of the conditions and encode each valid valuation with one letter in the alphabet we will later define for the transducer.

Table 7.1: Encoding for agent configuration in state Init

c_1	c_2	c_3		
0	0	0	→	move left
0	0	1	→	do not move, switch to state Fwd
0	1	0	→	move left
0	1	1	→	do not move, switch to state Fwd
1	0	0	→	do not move, switch to state Bounce
1	0	1	→	not valid
1	1	0	→	do not move, switch to state Bounce
1	1	1	→	not valid

When an agent does not make a left move, two things happen: (1) a condition becomes *true* and (2) the agent changes its state. We will use the following letters to describe the possible actions of the agent:

- I_l The agent is in state Init, c_1 is false and c_3 is false which implies that the agent moves one step to the left.
- I_{c_1} The agent is in state Init, c_1 is true and c_3 is false which implies that the agent needs switch to state Bounce without moving.
- I_{c_3} The agent is in state Init, c_1 is false and c_3 is true which implies that the agent needs to switch to state Fwd without moving.

2. *Bounce:* Similarly to Init, we look at the possible valuations of the conditions and encode each valid combination in a letter which we will then add to the alphabet of the transducer.

Table 7.2: Encoding for agent configuration in state Bounce

c_1	c_2	c_3		
0	0	0	→	move right
0	0	1	→	move right
0	1	0	→	do not move, switch to state Terminate
0	1	1	→	do not move, switch to state Terminate
1	0	0	→	move right
1	0	1	→	move right
1	1	0	→	do not move, switch to state Terminate
1	1	1	→	do not move, switch to state Terminate

We use the following letters to describe the possible actions of the agent:

- B_r The agent is in state Bounce and c_2 is false which implies that the agent moves one step to the right.
- B_{c_2} The agent is in state Bounce and c_2 is true which implies that the agent does not move and switches to state Terminate.

3. *Fwd:* Once again, we need to look at the possible valuations of the conditions and encode each valid combination in a letter which we will then add to the alphabet of the transducer.

Table 7.3: Encoding for agent configuration in state Fwd

c_1	c_2	c_3		
0	0	0	$\rightarrow$	move left
0	0	1	$\rightarrow$	move left
0	1	0	$\rightarrow$	do not move, switch to state Terminate
0	1	1	$\rightarrow$	do not move, switch to state Terminate
1	0	0	$\rightarrow$	move left
1	0	1	$\rightarrow$	move left
1	1	0	$\rightarrow$	do not move, switch to state Terminate
1	1	1	$\rightarrow$	do not move, switch to state Terminate

We will use the following letters to describe the possible actions of the agent:

- F_l The agent is in state Fwd and c_2 is false which implies that the agent moves one step to the left.
- F_{c_2} The agent is in state Fwd and c_2 is true which implies that the agent does not move and switches to state Terminate.

4. *Terminate:* Finally, for the last state, the agent does not perform any action, so we only need to represent that the agent entered that state. We will only use T to describe that action of the agent.

In addition to the possibilities described above, we need to also represent the case of a node not containing an agent. We will do this with e. So the alphabet of our transducer is $\Sigma = \{I_l, I_{c_1}, I_{c_3}, B_r, B_{c_2}, F_l, F_{c_2}, T, e\}$.

Representing a move of the agent can be done using the automata of Figures 7.1 and 7.2 from the previous section. In this case too, we represent the ring by a sequence of letters. We use e to represent a node that does not contain an agent. However, this time, the agent configuration contains more information in addition to the position of the agent. Therefore, instead of representing a node occupied by an agent with just an o (like in Figures 7.1 and 7.2), we represent the agent with $I_l, I_{c_1}, I_{c_3}, B_r, B_{c_2}, F_l, F_{c_2}$ or T, depending on its state and the valuation of its conditions. Therefore, the part of the transducer that describes an agent moving counterclockwise and is in state "Init" becomes the one in Figure 7.4.

Similarly, the part of the transducer that describes an agent that is in state "Fwd" and is moving counterclockwise is in Figure 7.5.

The part of the transducer describing an agent that is in state "Bounce" and is moving counterclockwise is the one in Figure 7.6.

Since there is no other valid state-move combination in the agent, we do not define an agent moving clockwise in states Init or Fwd and we do not define an agent moving

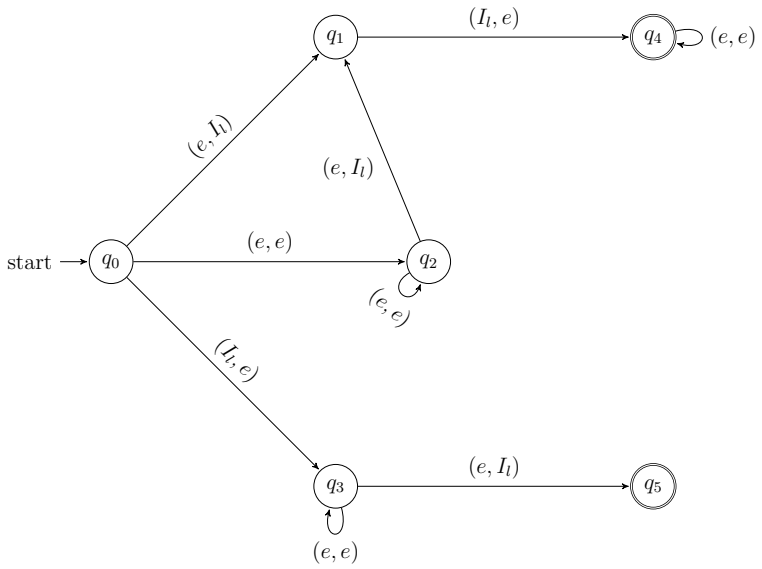

Figure 7.4: Transducer for an agent being in state Init and moving counterclockwise

counterclockwise in Bounce. At this point, we have defined the moves of an agent and we need to add the parts of the transducer that correspond to conditions becoming true and to changes of the agent's state. The regular expression of a condition (say c_1) becoming true when an agent is in state Init is $(e, e)^*(I_l, I_{c_1})(e, e)^*$ (Figure 7.7). This means that after a number of left moves the agent does not move (hence I_l and I_{c_1} are in the same position in the word in two successive configurations) and c_1 is true. Similarly, the regular expression for an agent that is:

- In Init and c_3 becomes true is $(e, e)^*(I_l, I_{c_3})(e, e)^*$. See Figure 7.8.
- In Fwd and c_2 becomes true is $(e, e)^*(F_l, F_{c_2})(e, e)^*$. See Figure 7.9.
- In Bounce and c_2 becomes true is $(e, e)^*(B_r, B_{c_2})(e, e)^*$. See Figure 7.10.

Finally, to show that an agent switches state and which transitions between states are valid, we use the following regular expressions.

- $(e, e)^*(I_{c_1}, B_r)(e, e)^*$: According to the algorithm, when an agent is in Init, if c_1 becomes true the agent should not move and switch to state Bounce. Hence, when c_1 becomes true in state Init in configuration *Conf*, the agent transitions to Bounce in *Conf′* without moving to a different node, where *Conf′* is the successor of *Conf*. This is represented by the pair (I_{c_1}, B_r). Notice that we use B_r here because according to our definition of the table for Bounce, when $c_1 = 1, c_2 = 0$ and $c_3 = 0$, the corresponding encoding is B_r. See Figure 7.11.
- $(e, e)^*(I_{c_3}, F_l)(e, e)^*$: According to the algorithm, when an agent is in Init, and c_3 becomes true the agent should not move and switch to state Fwd. Using the same reasoning as in the previous case, this is represented by the pair (I_{c_3}, F_l). We use F_l here because according to our definition of the table for Fwd, when $c_1 = 0, c_2 = 0$ and $c_3 = 1$, the corresponding encoding is F_l. See Figure 7.12.
- $(e, e)^*(F_{c_2}, T)(e, e)^*$: According to the algorithm, when an agent is in Fwd, and c_2 becomes true the agent should not move and switch to state Terminate. Using the same reasoning as in the first case, this is represented by the pair

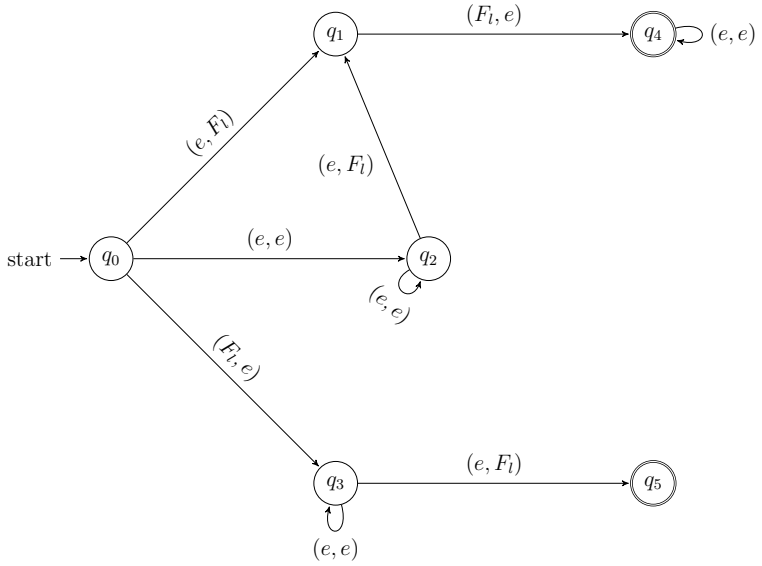

Figure 7.5: Transducer for an agent being in state Fwd and moving counterclockwise.

(F_{c_2}, T). See Figure 7.9.

- $(e, e)^*(B_{c_2}, T)(e, e)^*$. According to the algorithm, when an agent is in Bounce, and c_2 becomes true the agent should not move and switch to state Terminate. Using the same reasoning as in the first case, this is represented by the pair (B_{c_2}, T). See Figure 7.14.

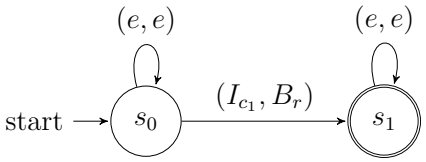

Figure 7.11: Transducer for an agent switching from state Init to state Bounce.

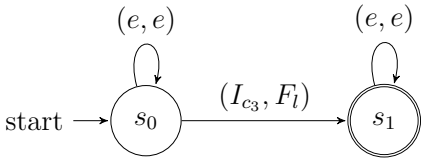

Figure 7.12: Transducer for an agent switching from state Init to state Fwd.

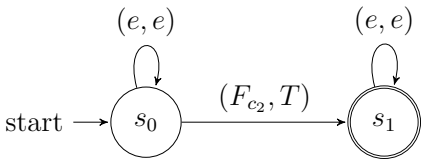

Figure 7.13: Transducer for an agent switching from state Fwd to state Terminate.

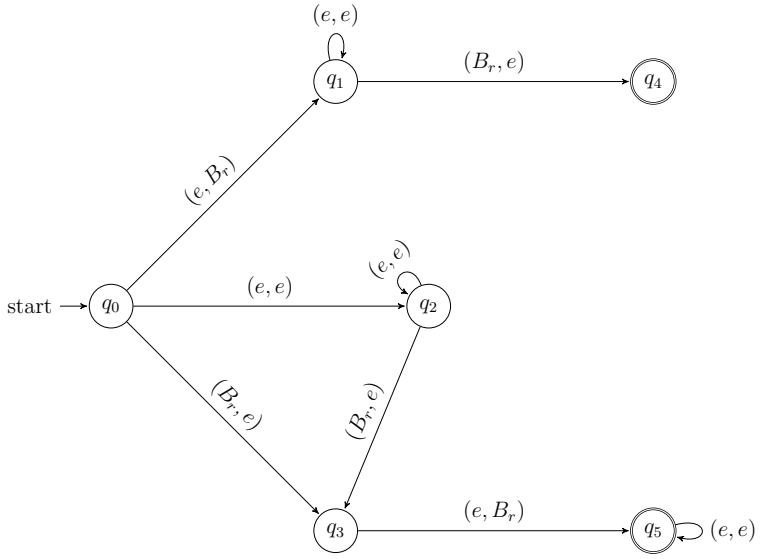

Figure 7.6: Transducer for an agent being in state Bounce and moving clockwise.

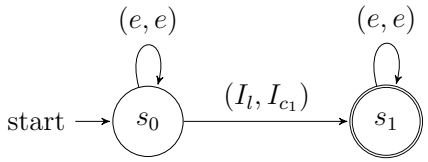

Figure 7.7: Transducer for an agent being in state Init when c_1 becomes true.

The transducer describing all possible transitions defined by the algorithm encoded in the agent in Figure 4.1 that explores a dynamic ring of known size using agents that do not have chirality, is the union of the automata presented in Figures 7.4 - 7.14.

7.4 Constructing the Necessary Components for RMC

In a more general case, the construction of the transducer follows the structure of the agent encoding of the algorithm that we described in Section 4.6 ("Formalization of an Agent"). In that agent configuration, an agent is of the form $\mathcal{A} = \{Q, \delta, \texttt{I}, \texttt{F}, \texttt{Moves}, \texttt{Conditions}, \texttt{Variables}, \texttt{PreM-Sensors}, \texttt{PostM-Sensors}, \texttt{N}\}$. During each round, the agent gets an input that is a combination of the variable

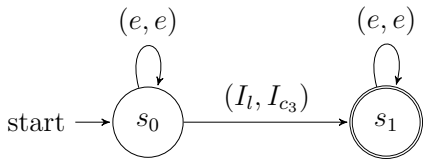

Figure 7.8: Transducer for an agent being in state Init when c_3 becomes true.

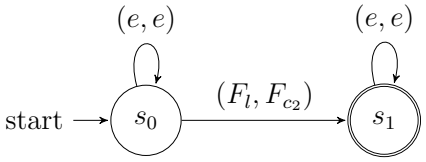

Figure 7.9: Transducer for an agent being in state Fwd when c_2 becomes true.

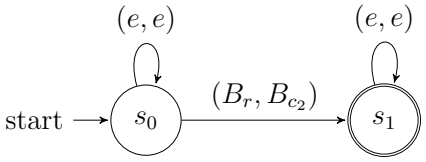

Figure 7.10: Transducer for an agent being in state Bounce when c_2 becomes true.

values and the sensor values, makes a number of computations based on that input and that computation leads to true or false values for a subset of the conditions it checks. According to the valuation of the conditions the agent either moves (clockwise, counterclockwise, no move/waiting phase), attempts to move but fails, or transitions to a different, internal, state. The construction of the transducer that we will show consists of constructing several automata that we call "sub-transducers", the union of which produces the transducer.

7.4.1 Alphabet Definition in the General Case

The first step, towards the construction of the transducer is defining its alphabet, Σ. We split the alphabet definition to a number of parts equal to the number of states in the agent. For each state, construct a table with the possible valuations of all conditions of the agent. After constructing the table, encode each action (i.e., move or wait phase) of the agent that corresponds to a valid combination of condition values for the agent in that state with one letter, l_{move}, and add l_{move} to the alphabet of the transducer. That it, $\Sigma = \Sigma \cup \{l_{move}\}, \forall m, q \in$ `Moves, Q` for which there exists a valid transition $\delta(q, v) = (q', m, updates)$ of the agent. Furthermore, add one letter, l_{cond}, in the alphabet for each valid condition or combination of conditions becoming true within the state. That is $\Sigma = \Sigma \cup \{l_{cond}\}$.

7.4.2 Transducer Construction in the General Case

After the alphabet is complete, we begin building the sub-transducers. We split this procedure into the following parts, for each state of the agent.

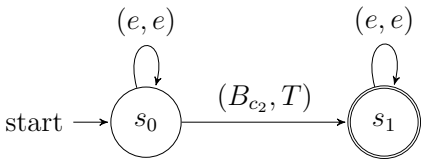

Figure 7.14: Transducer for an agent switching from state Init to state Bounce.

1. *Condition checks.* A condition becoming true either does not affect the agent in the current state and as a result does not need to be recognized by a sub-transducer or it leads to a change in the agent's configuration through a transition. The latter possibility either means that an agent should change its move or that the agent changes state and is described in the following cases. The format of a sub-transducer representing that a condition becomes true is $(e, e)^*(x, c)(e, e)^*$, where x is a letter of the transducer's alphabet corresponding to the agent's previous state and c is the letter of the transducer's alphabet corresponding to the condition that becomes true.
2. *Agent actions.* As we have mentioned in a previous step, we consider an agent action to be either a move or a wait phase. A move can be either clockwise or counterclockwise, since we only consider ring graphs as part of this thesis. We only construct sub-transducers for the moves that are possible within the given state. The sub-transducer that corresponds to a move in each valid direction within a state, is of the form of the automata in Figures 7.1 and 7.2, but "o" is replaced by the letter of the transducer's alphabet which corresponds to that action and agent state. Let us now consider the case in which an agent changes directions within the same state. An agent continues performing the same action (i.e., moving in direction *dir* or waiting) until a condition changes. The following things happen when an agent changes direction within a state. (1) The agent is moving in a direction d or waiting at a node (that we also symbolize with d for brevity), (2) a condition or set of conditions, c, becomes true and (3) the direction changes. This is reflected by constructing two sub-transducers: one that accepts $(e, e)^*(d, c)(e, e)^*$ and one that accepts $(e, e)^*(c, d')(e, e)^*$. Finally, an agent that is only waiting (symbolized as an example by w) at a state, is accepted by a sub-transducer of the form $(e, e)^*(w, w)(e, e)^*$.
3. *Internal change of state.* An agent only changes state, if some condition, c, is fulfilled. When an agent transitions from some state s to some state s' because a condition c became true, two sub-transducers need to be added. That is, a sub-transducer that accepts $(e, e)^*(s, c)(e, e)^*$ and one that accepts $(e, e)^*(c, s')(e, e)^*$.

Finally, the transducer describing all successive configurations of the system, is given by the union of all the sub-transducers.

7.4.3 Set of Initial Configurations in the General Case

In addition to the alphabet and the transducer, we need to provide a way to represent the set of initial configurations. Before the execution of an algorithm begins, internally the agent is in its initial state and the conditions that the agent checks before making a move or an internal change of state are all set to false. There is already a symbol in the alphabet of the transducer that corresponds to the agent being internally in its initial state and the conditions being false. Let that symbol be s. Furthermore, we have already defined e to denote an "empty" node, not occupied by an agent, in the alphabet of the transducer. The regular expression corresponding to the set of initial configurations for some agent A is se^* which is

depicted in Figure 7.15.

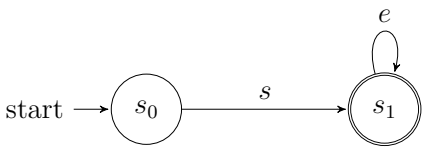

Figure 7.15: The set of initial configurations for some agent A

7.4.4 Exploration, Gathering and BHS as Verification Problems

The "Gathering" problem is a reachability problem. More precisely, in any correct algorithm there should exist a reachable state in which either all agents in the system are located at a common node or at two neighbouring nodes.
For problems similar to Exploration, we need to add memory that keeps information obtained from the transducer. In particular, each time an agent visits a node we flip a bit that corresponds to that node from 0 to 1. If the node is already visited by some other agent, the bit remains 1. At the end of the algorithm we check if all the nodes have been visited, or equivalently, we check if all the bits are set to 1. If indeed all the nodes are visited, exploration is achieved.
Finally, the agents can learn the position of a black hole in different ways, depending on the kind of communication methods that are used. In the case of endogenous communication, the agents can only discover a black hole by having at least two agents, A_1 and A_2, meet at a node v and then have only one of the agents, say A_1, move to a previously unexplored node at time t. Either A_1 returns to v at $t + 1$ or A_2 infers that the BH is located at the node A_1 visited. Notice that in the cases of endogenous communication mechanisms, the agents cannot exchange any information without meeting and an agent that is lost in the BH does not leave any information behind. Therefore, the agents cannot learn the position of the BH unless they use the described technique, known as "cautious walk" in the literature. Therefore, for an algorithm that solves BHS to be correct we need to check that before some agent disappears at some step s it met with another agent at $s - 1$. This is a reachability problem, like Gathering, which also requires some memory, like Exploration. We use memory to record that two agents met in a node, by flipping a bit from 0 to 1 when an agent meets with another agent, indicating that the execution of "cautious walk" has started. The reachability problem then is to reach a state in which an agent is alone on a node, after meeting with another agent in the exactly previous step, or equivalently, by reaching a state in which an agent is alone on a node and the bit that indicates that the agent is performing cautious walk is set to 1.

7.5 More than one agents in the ring

The description of the transducers we have given so far describes the possible transitions of one agent that operates in a dynamic ring and attempts to accomplish

a given task. However, as we showed in Lemma 3, one agent in a dynamic ring cannot even leave its initial node. Therefore, all the problems in dynamic rings that require agents to move, require at least two mobile agents to operate in the ring. For the algorithms that we study in dynamic rings, the lower bounds for agents are two agents for Exploration (according to [8]), three agents for BHS (according to [1]) and Gathering by the definition of the problem refers to at least two agents.
At the beginning of the "Moving in a Ring" section, we defined the language that is accepted by the transducer to be of the form (x, y), where $x = a_1 \dots a_n$ and $y = b_1 \dots b_n$ and x and y are two successive configurations. For k agents, where k can be any integer but needs to be fixed in each algorithm, we define x to be of the form

$$x = \langle (a_{11}a_{12} \dots a_{1n}), (a_{21}a_{22} \dots a_{2n}), \dots, (a_{k1}a_{k2} \dots a_{kn}) \rangle$$

Similarly, we define y to be

$$y = \langle (b_{11}b_{12} \dots b_{1n}), (b_{21}b_{22} \dots b_{2n}), \dots, (b_{k1}b_{k2} \dots b_{kn}) \rangle$$

For convenience, we think of words that belong to the language that is accepted by the global transducer as having the following form:

$$\left\langle \begin{array}{l} \Big[(a_{11}, b_{11}), (a_{12}, b_{12}), \dots, (a_{1n}, b_{1n})\Big], \\ \Big[(a_{21}, b_{21}), (a_{22}, b_{22}), \dots, (a_{2n}, b_{2n})\Big], \\ \qquad\qquad \vdots \\ \Big[(a_{k1}, b_{k1}), (a_{k2}, b_{k2}), \dots, (a_{kn}, b_{kn})\Big] \end{array} \right\rangle$$

In this type of words, each line i corresponds to one transition of the i-th agent in a ring. The transition function for the global transducer is $t := Q \times (\Sigma \times \Sigma)^k \mapsto Q$ which for each agent $i \in \{1, \dots, k\}$ becomes $t_i := Q_i \times (\Sigma_i \times \Sigma_i) \mapsto Q_i$, where t is the transition function of the transducer defined in a previous section ("Moving in a Ring") and Q_i and Σ_i are respectively the set of states and the alphabet corresponding to the local transducer T_i of agent i. We have already defined in the previous section the local transducer of an agent to be the union of the sub-transducers of valid *condition checks*, *agent actions* and *internal changes of state* for each agent. The global transducer that describes the moves of all agents in the system is the intersection of all the local transducers. Notice that the local transducers of the agents can either be the same for each agent in the ring (like in the case of the agents in Figure 4.1) or they can be different for each agent or for subsets of agents (as is the case for the agents in Figure 4.2 - 4.4). In the case where more than one agents have the same local transducer, we change the alphabet corresponding to the local transducer of each agent, to be unique to that agent. The transducer for the global configuration is defined as a relation over $(\Sigma \times \Sigma)^k$, where k is the number of agents operating in the ring. That is, for k agents and m sub-transducers for each agent,

$$T_{global} = \bigcap_{i=1}^{k} \bigcup_{j=1}^{m} T_j(\Sigma_i \times \Sigma_i)$$

Similarly, the set of initial configurations when there are more than one agents in the ring, is the union of the set of initial configurations for each agent in the ring.

7.6 Example: Global Transducer for Multiple Agents

Let us now look at an example for a problem that uses multiple agents such as the ones we defined in Chapter 4. In order to make the example simpler, instead of using complex agents that solve one of the three algorithmic problems we have been studying, we consider the following problem: "Have two agents move clockwise in a ring. If both agents are blocked by the same missing edge terminate". We call the two agents in the ring A_1 and A_2.
We split the description of the global transducer to the two parts indicated by the two boxes in the following description of the transducer:

$$T_{global} = \boxed{\bigcap_{i=1}^{k} \boxed{\bigcup_{j=1}^{m} T_j(\Sigma_i \times \Sigma_i)}}$$

Let us first discuss the part of the global transducer enclosed in the smaller box. This part refers to the union of the sub-transducers that form the local transducer of one agent. This is the procedure we described in the Section "Transducer for Exploration in Dynamic Graphs". The agents that solve the problem we defined are of the following form:

- $Q = \{\texttt{Init}, \texttt{Term}\}$
- $I = \texttt{Init}$
- $F = \{\texttt{Term}\}$
- $Moves = \{\texttt{right}\}$
- Pre-Move Sensors = checks if more than one agents are in the node (`metAgent`)
- Post-Move Sensors = checks if the edge leading to the clockwise neighbouring node was missing (`blocked`)
- Variables = $\emptyset$
- Conditions =
  c_1 : $\texttt{metAgent} \wedge \texttt{blocked}$

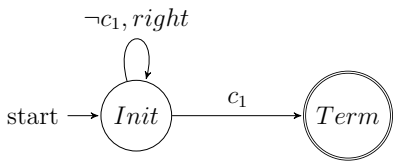

Figure 7.16: Some agent A, solving the problem defined in this section

Following the steps in Section 7.4, we can derive the local transducer for each agent. That is the local transducers comprise of three parts:

- A sub-transducer describing the clockwise move of an agent in a ring.

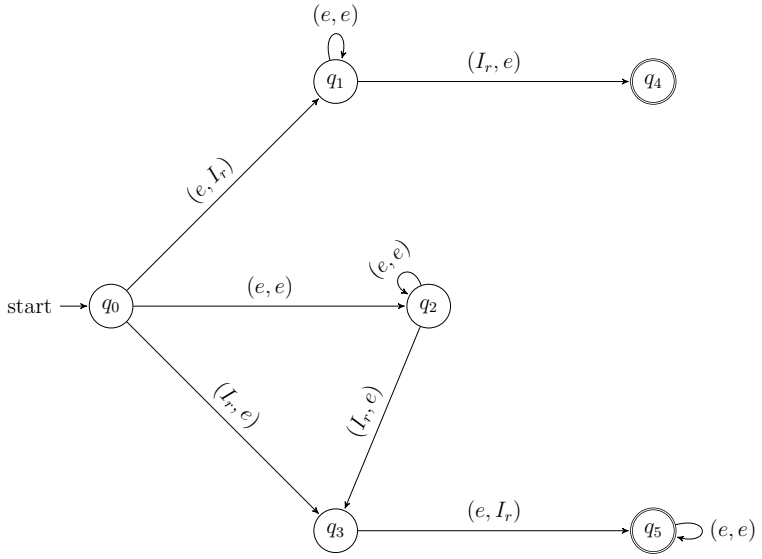

Figure 7.17: Transducer for an agent being in state Init and moving clockwise.

- A sub-transducer describing the condition (c_1) between two states of the agent (*Init* and *Term*) becoming *true*.

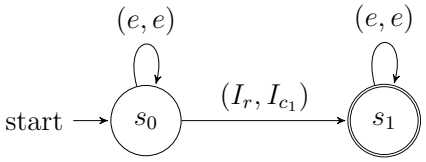

Figure 7.18: Transducer for an agent detecting that condition c_1 became *true*.

- A sub-transducer describing the internal transition of the agent from a state (*Init*) to another (*Term*).

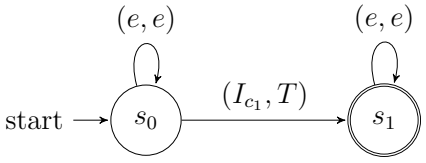

Figure 7.19: Transducer for an agent switching from state Init to state Term.

The part of the transducer enclosed in the smaller box is completed by taking the union of the three sub-transducers.
Let us now move to the larger box of the global transducer definition. In this case, the algorithm needs two agents operating in the ring. Therefore, $k = 2$ and we need to make one more local transducer which is also the union of the automata in Figures 7.17 - 7.19, and differentiate the alphabets. That is, we define the alphabet of the local transducer for A_1 to be $\Sigma_{A_1} = \{e, I_{r_1}, I_{c_{11}}, T_1\}$ and for A_2 to be $\Sigma_{A_2} = \{e, I_{r_2}, I_{c_{12}}, T_2\}$. Finally, to obtain the global transducer, we take the intersection of the two local transducers.

8

Conclusion

We begin this chapter by summarizing our results. Then, we discuss *how* our results fit with the known impossibility results. Finally, we conclude by presenting some open problems that arise from this work.

8.1 Summary

In this thesis we study the problem of parameterized verification of distributed algorithms in dynamic graphs, for three well known problems from the distributed computing literature. Namely, the problems we are interested in are "Exploration", "Gathering" and "Black Hole Search". We began this work by studying the known algorithms for those three problems in *1-interval connecting* dynamic rings and seeing how those algorithms work. We also worked on learning the minimum agent capabilities to solve the problems, as well as the agent capabilities that have been used in the literature. Using this information, we defined a type of automaton that is capable of encoding this type of algorithms and agents. The automata we defined

- Use *sensors* in order to gather information from their environment, such as whether any other agents are located on the current node.
- Use *variables* in order to record information they compute, such as the number of moves they have made.
- Can check a number of predefined *conditions* in order to decide the next action to be performed.

Although "Exploration" is the most well known and the easiest of the three problems, we chose those problems because they are connected. Our initial assumptions were that:

- Under certain conditions, the parameterized verification problem for any given algorithm that solves "Exploration" in dynamic graphs is decidable.
- For the more complex problems (that is, first "Gathering" and then "Black Hole Search"), as the agent capabilities increase, so would the undecidability results.

However, while attempting to determine the characteristics mobile agents can have so that the problem remains solvable we proved that those agent capabilities already make the Exploration problem undecidable. Thus, we chose to focus more on the undecidability results which are summarized in Table 8.1.

Due to those undecidability results, in Chapter 7 we focus on finding a way to use regular model checking. RMC, being an incomplete method that has been used before in the context of parameterized verification, turned out to be more suitable

Table 8.1: *Summary of our undecidability results. All agents in all cases have distinct but unknown identities. In the cases where the agents do not begin from the same node (i.e., scattered) we assume there is also a marked node somewhere in the ring. Unless indicated otherwise in the table, the model of communication between agents is "vision", that is, each agent can only communicate by seeing how many other agents are located on the node it currently occupies. "One bit of information", means that agents located at the same node can get one bit of information, corresponding to the disjunction of the signalling variables of all co-located agents.*

# of Agents	Initial Positions	# of Ring Size Dependent Variables	One bit of information
2	co-located	2	✓
3	scattered	2	✓
3	co-located	1	✓
5	scattered	1	✓
4	co-located	2	X
4	scattered	2	X
6	co-located	1	X
6	scattered	1	X

to the three problems that we study. We provide a way to transform the type of algorithms and the agents that are needed to solve the three distributed computing problems that we defined to the components needed to use regular model checking techniques. More precisely, our results concerning RMC are split into the following parts.

1. Defining the alphabet of any given system.
2. Describing the procedure of deriving the transducer. The construction of the transducer is further split into three stages: the movement of the agent, the condition checks and the internal change of state of the agent.
3. Defining the set of the possible initial configurations.
4. Describing Exploration, Gathering and BHS as reachability problems.
5. Finally, since none of those problems have local solutions and at least two mobile agents need to operate in an 1-interval connected dynamic ring to guarantee that at least one agent moves, we generalized the components to describe multiple agents operating in a ring.

8.2 Discussion

As part of this work, we derived some undecidability results that we combine in Theorems 1 and 2 with impossiblity results from [8]. This combination helps us restrict the agent capabilities for which Exploration or Exploration based problems such as Gathering and BHS might be decidable. To be more specific, in the case of agents that only have vision and no other communication methods, we have learned that any possibly decidable algorithm can only have between two and five agents with one ring size dependent variable (Theorem 1). This corresponds to more limited memory than what is used in the known algorithms in [8]. However, those algorithms

only need two agents. It is therefore interesting to study whether algorithms that use more agents with less memory could be decidable.
The second direction that we explored was related to more complex communication mechanisms. Our goal was to add as little communication capabilities as possible, as a first step. Indeed, even one bit of information exchanged between the agents suffices to further restrict the possibly decidable cases, with respect to the number of agents with one ring size dependent variable. In Theorem 2 we show that in the case of initially co-located agents, two agents with one ring size dependent variable are the only case that an algorithm might be decidable. On the other hand, if agents are initially scattered, the automated verification problem may only be decidable for algorithms that use between two and four agents with one ring size dependent variable.
The most natural and interesting next direction would be to discover whether the remaining cases, that is, the cases that we have so far been referring to as "potentially decidable" in Theorems 1 and 2 are actually decidable or undecidable. From a distributed computing perspective, it is worth focusing more on algorithms where the memory of agents is limited, since there are no lower bounds on the amount of agent memory that would make Exploration, Gathering or BHS impossible to solve. Another interesting direction for the automated algorithm verification problem, would be to consider different communication mechanisms (such as tokens) for agents with less memory and study whether the problem would then become decidable. Finally, it would be worth studying the problem in other types of 1-interval connecting dynamic graphs.

Bibliography

[1] Giuseppe Antonio Di Luna, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Black hole search in dynamic rings. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 987–997. IEEE, 2021.

[2] Giuseppe Antonio Di Luna. Mobile agents on dynamic graphs. In *Distributed Computing by Mobile Entities*, pages 549–584. Springer, 2019.

[3] Shantanu Das and Nicola Santoro. Moving and computing models: Agents. In *Distributed Computing by Mobile Entities*, pages 15–34. Springer, 2019.

[4] Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Searching for a black hole in tree networks. In *International Conference On Principles Of Distributed Systems*, pages 67–80. Springer, 2004.

[5] Paola Flocchini, David Ilcinkas, and Nicola Santoro. Ping pong in dangerous graphs: Optimal black hole search with pebbles. *Algorithmica*, 62(3-4):1006–1033, 2012.

[6] Stefan Dobrev, Paola Flocchini, and Nicola Santoro. Improved bounds for optimal black hole search with a network map. In *International Colloquium on Structural Information and Communication Complexity*, pages 111–122. Springer, 2004.

[7] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous oblivious robots with limited visibility. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 247–258. Springer, 2001.

[8] G Di Luna, Stefan Dobrev, Paola Flocchini, and Nicola Santoro. Distributed exploration of dynamic rings. *Distributed Computing*, 33(1):41–67, 2020.

[9] Giuseppe Antonio Di Luna, Paola Flocchini, Linda Pagli, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Gathering in dynamic rings. *Theoretical Computer Science*, 811:79–98, 2020.

[10] Nicola Santoro. Time to change: On distributed computing in dynamic networks (keynote). In *19th International Conference on Principles of Distributed Systems (OPODIS 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[11] Shantanu Das. Graph explorations with mobile agents. In *Distributed Computing by Mobile Entities*, pages 403–422. Springer, 2019.

[12] C Shannon. Presentabilitytion of a maze solving machine. In *Trans. 8th Conf. Cybernetics: Circular, Causal and Feedback Mechanisms in Biological and Social Systems (New York, 1951)*, pages 169–181, 1951.

[13] Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005.

[14] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008.

[15] Petrişor Panaite and Andrzej Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33(2):281–295, 1999.

[16] Krzysztof Diks, Pierre Fraigniaud, Evangelos Kranakis, and Andrzej Pelc. Tree exploration with little memory. *Journal of Algorithms*, 51(1):38–63, 2004.

[17] Hanane Becha and Paola Flocchini. Optimal construction of sense of direction in a torus by a mobile agent. *International Journal of Foundations of Computer Science*, 18(03):529–546, 2007.

[18] Stefan Dobrev, Paola Flocchini, R Královič, P Ružička, Giuseppe Prencipe, and Nicola Santoro. Black hole search in common interconnection networks. *Networks: An International Journal*, 47(2):61–71, 2006.

[19] Pierre Fraigniaud, Leszek Gasieniec, Dariusz R Kowalski, and Andrzej Pelc. Collective tree exploration. *Networks: An International Journal*, 48(3):166–177, 2006.

[20] Shantanu Das, Paola Flocchini, Shay Kutten, Amiya Nayak, and Nicola Santoro. Map construction of unknown graphs by multiple agents. *Theoretical Computer Science*, 385(1-3):34–48, 2007.

[21] David Ilcinkas and Ahmed M Wade. Exploration of the t-interval-connected dynamic graphs: the case of the ring. *Theory of Computing Systems*, 62(5):1144–1160, 2018.

[22] Himadri Sekhar Mondal and Sruti Gan Chaudhuri. A survey on gathering of distributed mobile agent in discrete domain. In *Proceedings of the Global AI Congress 2019*, pages 601–608. Springer, 2020.

[23] Paola Flocchini. Gathering. In *Distributed Computing by Mobile Entities*, pages 63–82. Springer, 2019.

[24] François Bonnet, Alessia Milani, Maria Potop-Butucaru, and Sébastien Tixeuil. Asynchronous exclusive perpetual grid exploration without sense of direction. In *International Conference On Principles Of Distributed Systems*, pages 251–265. Springer, 2011.

[25] Gabriele Di Stefano and Alfredo Navarra. Optimal gathering of oblivious robots in anonymous graphs. In *International Colloquium on Structural Information and Communication Complexity*, pages 213–224. Springer, 2013.

[26] Sumitro Bhaumik and Sruti Gan Chaudhuri. Gathering of asynchronous mobile robots in a tree. In *2015 Applications and Innovations in Mobile Computing (AIMoC)*, pages 97–102. IEEE, 2015.

[27] Jérémie Chalopin, Shantanu Das, and Nicola Santoro. Rendezvous of mobile agents in unknown graphs with faulty links. In *International Symposium on Distributed Computing*, pages 108–122. Springer, 2007.

[28] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Mobile search for a black hole in an anonymous ring. In *International Symposium on Distributed Computing*, pages 166–179. Springer, 2001.

[29] Balasingham Balamohan, Paola Flocchini, Ali Miri, and Nicola Santoro. Time optimal algorithms for black hole search in rings. *Discrete Mathematics, Algorithms and Applications*, 3(04):457–471, 2011.

[30] Paola Flocchini, Matthew Kellett, Peter C Mason, and Nicola Santoro. Searching for black holes in subways. *Theory of Computing Systems*, 50(1):158–184, 2012.

[31] Paola Flocchini, David Ilcinkas, and Nicola Santoro. Ping pong in dangerous graphs: Optimal black hole search with pure tokens. In *International Symposium on Distributed Computing*, pages 227–241. Springer, 2008.

[32] Wei Shi. Black hole search with tokens in interconnected networks. In *Symposium on Self-Stabilizing Systems*, pages 670–682. Springer, 2009.

[33] Stefan Dobrev, Paola Flocchini, Rastislav Královič, and Nicola Santoro. Exploring an unknown dangerous graph using tokens. *Theoretical Computer Science*, 472:28–45, 2013.

[34] Colin Cooper, Ralf Klasing, and Tomasz Radzik. Searching for black-hole faults in a network using multiple agents. In *International Conference On Principles Of Distributed Systems*, pages 320–332. Springer, 2006.

[35] Colin Cooper, Ralf Klasing, and Tomasz Radzik. Locating and repairing faults in a network with mobile agents. *Theoretical Computer Science*, 411(14-15):1638–1647, 2010.

[36] Jérémie Chalopin, Shantanu Das, Arnaud Labourel, and Euripides Markou. Black hole search with finite automata scattered in a synchronous torus. In *International Symposium on Distributed Computing*, pages 432–446. Springer, 2011.

[37] Stefan Dobrev, Paola Flocchini, Rastislav Kralovic, Giuseppe Prencipe, Peter Ruzicka, and Nicola Santoro. Black hole search by mobile agents in hypercubes and related networks. In *OPODIS*, volume 3, pages 169–180, 2002.

[38] Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Searching for a black hole in arbitrary networks: optimal mobile agent protocols. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 153–162. ACM, 2002.

[39] Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71(2, 3):229–242, 2006.

[40] Ralf Klasing, Euripides Markou, Tomasz Radzik, and Fabiano Sarracco. Hardness and approximation results for black hole search in arbitrary networks. *Theoretical Computer Science*, 384(2-3):201–221, 2007.

[41] Ralf Klasing, Euripides Markou, Tomasz Radzik, and Fabiano Sarracco. Approximation bounds for black hole search problems. *Networks: An International Journal*, 52(4):216–226, 2008.

[42] Andrzej Pelc. Deterministic rendezvous algorithms. In *Distributed Computing by Mobile Entities*, pages 423–454. Springer, 2019.

[43] Evangelos Kranakis, Danny Krizanc, and Euripides Markou. The mobile agent rendezvous problem in the ring. *Synthesis Lectures on Distributed Computing Theory*, 1(1):1–122, 2010.

[44] Paola Flocchini and Nicola Santoro. Distributed security algorithms by mobile agents. In *International Conference on Distributed Computing and Networking*, pages 1–14. Springer, 2006.

[45] Sasha Rubin. Parameterised verification of autonomous mobile-agents in static but unknown environments. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 199–208, 2015.

[46] Panagiotis Kouvaros and Alessio Lomuscio. Parameterised verification for multi-agent systems. *Artificial Intelligence*, 234:152–189, 2016.

[47] Maria Potop-Butucaru, Nathalie Sznajder, Sébastien Tixeuil, and Xavier Urbain. Formal methods for mobile robots. In *Distributed Computing by Mobile Entities*, pages 278–313. Springer, 2019.

[48] E Allen Emerson and Kedar S Namjoshi. On reasoning about rings. *International Journal of Foundations of Computer Science*, 14(04):527–549, 2003.

[49] Ichiro Suzuki. Proving properties of a ring of finite-state machines. *Information Processing Letters*, 28(4):213–214, 1988.

[50] E Allen Emerson and Edmund M Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *International Colloquium on Automata, Languages, and Programming*, pages 169–181. Springer, 1980.

[51] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on programming*, pages 337–351. Springer, 1982.

[52] Edmund M Clarke, Thomas A Henzinger, and Helmut Veith. Introduction to model checking. In *Handbook of Model Checking*, pages 1–26. Springer, 2018.

[53] E Allen Emerson and Vineet Kahlon. Parameterized model checking of ring-based message passing systems. In *International Workshop on Computer Science Logic*, pages 325–339. Springer, 2004.

[54] Benjamin Aminof, Sasha Rubin, Ilina Stoilkovska, Josef Widder, and Florian Zuleger. Parameterized model checking of synchronous distributed algorithms by abstraction. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 1–24. Springer, 2018.

[55] Annu John, Igor Konnov, Ulrich Schmid, Helmut Veith, and Josef Widder. Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *2013 Formal Methods in Computer-Aided Design*, pages 201–209. IEEE, 2013.

[56] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *International Conference on Computer Aided Verification*, pages 403–418. Springer, 2000.

[57] Parosh Aziz Abdulla. Regular model checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 14(2):109–118, 2012.

[58] Marvin Minsky. Computation: Finite and infinite machines prentice hall. *Inc., Engelwood Cliffs, NJ*, 1967.

[59] Archak Das, Kaustav Bose, and Buddhadeb Sau. Exploring a dynamic ring without landmark. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pages 320–334. Springer, 2021.

[60] Subhrangsu Mandal, Anisur Rahaman Molla, and William K Moses. Live exploration with mobile robots in a dynamic ring, revisited. In *International*

Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, pages 92–107. Springer, 2020.

[61] Roderick Bloem and Joost Engelfriet. Monadic second order logic and node relations on graphs and trees. In *Structures in Logic and Computer Science*, pages 144–161. Springer, 1997.