Systematic Design and Formal Verification of Multi-Agent Systems

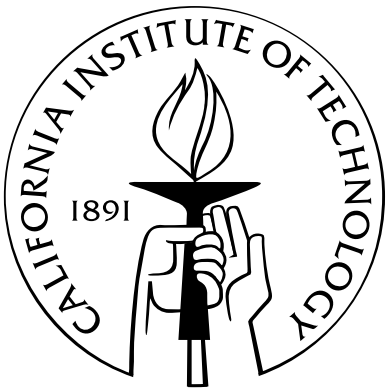
Thesis by

Concetta Pilotto

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2011

(Defended March 28, 2011)

To my family and friends.

Acknowledgements

I would like to thank my advisor, Professor K. Mani Chandy, for providing guidance and assistance throughout the period of my PhD research. I would like to thank Professor John O. Ledyard for serving on my committee and giving me the possibility to present the results of my research work at the *Social and Information Sciences Laboratory* (SISL) seminar series on a regular basis. I would like to thank Professor Richard M. Murray for serving on my committee, proving feedback on my work and giving me the possibility to present some of my work at the *Verification & Validation MURI hands-on workshop*. I would also like to thank Professor Steven H. Low, Professor John C. Doyle, and Dr. Gerard J. Holzmann for serving on my committee.

I would also like to thank all colleagues and friends who made my life at Caltech more enjoyable, including all members of the Infosphere group. They have always provided support and assistance during my Ph.D. studies. A special thanks goes to Jerome White, the colleague with whom I have worked more closely during my graduate studies at Caltech.

Lastly, but not least, I would like to thank the SISL and *Multidisciplinary Research Initiative* (MURI) from the Air Force Office of Scientific Research that supported my PhD research during all these years at Caltech.

Abstract

This thesis presents methodologies for verifying the correctness of multi-agent systems operating in hostile environments. Verification of these systems is challenging because of their inherent concurrency and unreliable communication medium. The problem is exacerbated if the model representing the multi-agent system includes infinite or uncountable data types.

We first consider message-passing multi-agent systems operating over an unreliable communication medium. We assume that messages in transit may be lost, delayed or received out-of-order. We present conditions on the system that reduce the design and verification of a message-passing system to the design and verification of the corresponding shared-state system operating in a friendly environment. Our conditions can be applied both to discrete and continuous agent trajectories.

We apply our results to verify a general class of multi-agent system whose goal is solving a system of linear equations. We discuss this class in detail and show that mobile robot linear pattern-formation schemes are instances of this class. In these protocols, the goal of the team of robots is to reach a given pattern formation.

We present a framework that allows verification of message-passing systems operating over an unreliable communication medium. This framework is implemented as a library of PVS theorem prover meta-theories and is built on top of the timed automata framework. We discuss the applicability of this tool. As an example, we automatically check correctness of the mobile robot linear pattern formation protocols.

We conclude with an analysis of the verification of multi-agent systems operating in hostile environments. Under these more general assumptions, we derive conditions on the agents' protocols and properties of the environment that ensure bounded steady-state system error. We apply these results to message-passing multi-agent systems that allow for lost, delayed, received out-of-order or forged messages, and to multi-agent systems whose goal is tracking time-varying quantities. We show that pattern formation schemes are robust to leaders dynamics, i.e., in these schemes, followers eventually form the pattern defined by the new positions of the leaders.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this Chapter, we introduce the main ideas of this Thesis. Definitions and theory are presented in later Chapters.

1.1 Thesis Contributions

This Thesis provides a theory for a new problem space and gives a mechanism to reason about it using the tools of computer science – automata and mechanical theorem proving.

Theory for a new problem domain

The Thesis introduces new theoretical tools for proving properties of multi-agent systems with continuous state spaces, where states change continuously with time and where agents communicate asynchronously at discrete points over a faulty communication medium. The novel aspects of the problem domain deal with the combination of (a) continuous dynamics of agents and (b) asynchronous communication with discrete messages that may be delayed or lost.

Much of the research on verifying distributed systems in computer science deals with proving that eventually some property will hold; for example, verification theory helps to prove that a collection of agents will eventually sort themselves in increasing order of identifiers. By contrast, this Thesis deals primarily with either (a) convergence properties showing that the actual state trajectory will get arbitrarily close to a desired trajectory, though it may never actually reach it, or (b) bounds on the deviation of the actual state trajectory from the desired one. Research in the computer science literature deals primarily with systems in which only one agent changes its state at a time, while the states of all other agents remain unchanged, and where state changes are discrete. This Thesis presents theory about multi-agent systems in which all agents may change their states concurrently and continuously.

An Example. A simple example of this problem space is a system with N agents, $N > 2$, indexed $0, 1, .., N-1$, where agents 0 and $N-1$, called leaders, move in the space in some arbitrary manner, and the goal of the remaining agents is to form a straight line with equal spacing between agents (see Figure 2.1). Agents move according to some dynamics – for example with constant velocity or constant acceleration or some other control law. All the agents may be moving at the same time. Agents send messages containing their current positions, and these messages may be delayed or lost. When an agent receives messages, it changes its heading based on the information about other agents. If agents 0 and $N-1$ are stationary, a question studied in this Thesis is: "Will the system converge to the desired state?". If agents 0 and $N-1$ are moving, a question is: "What is a bound (if any) between the actual state of the system and the desired state?" "How do message delay and the motion of the leaders impact the bound?"

Nondeterminism and Control. Nondeterminism is an important aspect of the problems studied in this Thesis. "How should systems in which messages are lost be modeled?" For example, if one set of agents can never communicate with another set of agents because all messages between the two sets are lost, then the two sets cannot collaborate, and nothing interesting can be proven about the combined system. Models in which every k-th message gets through are too restrictive. Models in this Thesis use an assumption of "fairness" from temporal logic that a message from one set to the other gets through "eventually". A challenge is to integrate nondeterministic models of communication with agents that change their states continuously. For example, a mobile agent that is moving from one location to another may accelerate for the first half of the movement and decelerate for the second half (see Figure 2.16(b)); or it may move at constant velocity (see Figure 2.16(a)); or, it may choose some other control law. When this agent receives a message from another agent, the receiver cannot determine when the sender sent the message because the agents do not share a clock; the receiver may need to change direction based on the messages it receives. An agent may receive a message while it is traveling from one location to another, and the precise time and location at which the message will be received cannot be predicted. These models are very general and make weak assumptions, so that it is hard to deduce interesting properties; however, this Thesis presents theorems about convergence and stability of such systems.

Time-Varying Goal Configuration. The Thesis presents theory for cases in which the environments where systems operate may or may not change. The goal state of a system depends on the environment; when the environment is unchanging the goal state is constant, and when the environment changes the goal state changes as well. In the simple example of the N moving agents given earlier, the environment is represented by the locations of the leaders – agents 0 and $N-1$. When the leaders move the line between them changes and the follower agents (indexed $1, \ldots, N-2$) follow

the changing line. The requirement that an agent will receive a message eventually (i.e., in some arbitrary, but finite, time) is too weak because the leaders may move arbitrarily far in arbitrarily long time. Therefore, we introduce the concept of an "epoch" – a bounded time interval during which the fairness constraints are satisfied; in other words instead of merely requiring that some condition holds eventually we require that the condition holds within some time interval Δ, and we present theorems for the distance between the actual state and the goal state as a function of Δ.

Worst-Case Analysis. The Thesis analyzes worst cases rather than average cases. A worst case can be thought of in terms of a competitive game between the system and its environment. The environment knows the algorithm used by the system and the state of the system at all times. The environment takes actions to frustrate the system - for instance to maximize the distance between the actual and desired states of the system. For instance, the environment may choose to delay messages, lose messages, and reorder messages, given its knowledge of the system state. The environment must, however, satisfy the constraints imposed by the fairness assumptions.

Formalization of Systems as Automata and Mechanical Theorem Proving

Proofs of the properties can be completely axiomatic so that a theorem-proving program can check proofs mechanically. Alternatively, proofs can skip some detailed steps. In the latter case, people with knowledge of the underlying mathematical domain can check the proof, but programs may be unable to do so. Most proofs in mathematics are not provided at the level of detail that enable current theorem-proving programs to verify them. We present a theory, based on prior work on automata, which allows proofs to be verified mechanically. Automatic verification can be extremely time consuming, and we do not mechanically verify all the proofs in this Thesis. We do mechanically verify some theorems to demonstrate that the theoretical framework can be used for this purpose. A contribution of this Thesis is an automaton – theoretic representation of multi-agent systems with continuous dynamics in which agents communicate with each other through faulty communication media.

1.2 Multi-Agent Systems

In this Section, we informally describe multi-agent systems. A multi-agent system (MAS) is a collection of agents that interact with each other. Agents may cooperate towards some collective task or they may compete, each trying to achieve its individual goal. Agents may be robots, devices, software components, or, even, people.

Multi-agent systems are decentralized systems as they achieve their goal without global coordination. In multi-agent systems, the interactions between agents are local. Agents usually interact

with a subset of the system. Agents in the system may execute their actions concurrently and at different rates.

Many applications may be modeled as multi-agent systems. Homeland security [54], health monitoring [6], and habitat and environmental monitoring ([12, 60]) applications are examples of multi-agent systems. For example, in environmental applications agents may be motes, with some processing capabilities, whose goal is monitoring some quantity of the environment, such as temperature. Other examples of multi-agent systems can be found in robotics, where agents are team of robots; examples of these applications are agents playing some game, such as soccer [70] or "capture the flag" [23, 71]. Examples from biology are flocks of birds or shoals of fishes ([20, 10]), where agents are birds or fishes whose goal is to form and maintain a specific spatial pattern.

In this Thesis, we focus on collaborative multi-agent systems where agents interact using very simple nearest-neighbor rules. We consider hybrid systems, i.e., systems where agents may have both discrete and continuous components. For example, we consider a multi-agent system consisting of vehicles whose goal is to reach a specific desired spatial configuration. In this multi-agent system, the variables describing the state of a vehicles can be discrete or continuous. For example, a discrete variable may represent the mode of the vehicle, e.g. straight or maneuvering; this variable can drive the dynamics of the vehicles, e.g. position and acceleration, that are represented using continuous variables.

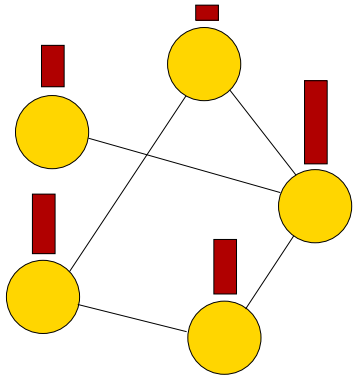
1.3 Communication Models for Multi-Agent Systems

In this Section, we present two communication models for multi-agent systems. In the first model, agents communicate via shared variables, while in the second model agents communicate by exchanging messages over an unreliable communication medium.

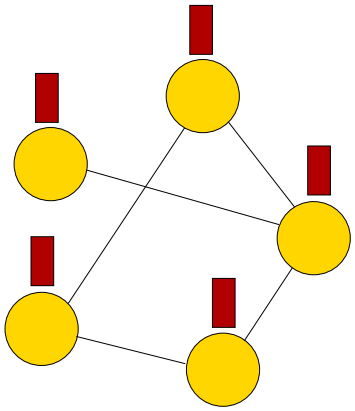
In a shared-state system, agents share memory and use this memory to communicate with each other. Using this common memory, each agent exposes some variables of its own local state to other agents. Each agent is responsible for its exposed variables, as it can read and write them; an agent can only read the exposed variables of other agents. Examples of these systems are multi-threaded systems, where threads store local variables in the stack memory, while storing exposed variables in the heap memory.

In a message-passing system, agents communicate with each other by sending messages. An agent can send messages containing the current values of its local variables to other agents. Agents can read and write their local variables and access the values in the received messages. Messages may be lost, delayed or received out of order. Examples of these systems are processes on different machines communicating over the Internet. We consider message-passing systems with bounded transmission delay.

The same algorithm executed over these two communication models can lead to different final system configurations. When communicating via shared variables, agents update their states using the current state of other agents. When communicating via message-passing, each agent updates its state using the latest received message. This message may contain information from an old copy of the state of some other agent, since messages may be delayed or received out-of-order. As a consequence, properties of configurations of a shared-state multi-agent system may not hold for the corresponding message-passing systems.



(a) A feasible configuration of the load balancing multi-agent system.



(b) Corresponding desired configuration of the load balancing multi-agent system.

Figure 1.1: Representation of a feasible and corresponding desired configuration of the load balancing multi-agent system. Nodes represent processors, edges represent dedicated communication between nodes and bars represent workload at each processor. The total amount of workload in Figure 1.1(a) and Figure 1.1(b) is the same.

1.4 Multi-Agent Systems in the Presence of Exogenous Inputs

In this Section, we introduce the notion of exogenous inputs for multi-agent systems. These inputs are arbitrary quantities injected into the system. When injected, they can change the state of the system.

For example, consider a load balancing multi-agent system [21]. This system consists of a network of processors, and the goal of the system is to distribute the workload evenly across the network in a decentralized manner. We refer to Figure 1.1 for a pictorial representation of this multi-agent system. In this Figure, processors are depicted as nodes, communication links between processors are represented as edges and the amount of workload at each node is represented as a rectangle. In Figure 1.1(a), we present a configuration of the system, where each processor stores a given amount

of workload; in Figure 1.1(b), we present the corresponding desired configuration where all processors store the same amount of workload. The total workloads in Figure 1.1(a) and Figure 1.1(b) are the same. We first study systems in which we are given an initial workload at each agent and no work is added or completed; the question of interest is: "Can agents interact locally so that the system state converges to the desired state?". Exogenous inputs change the work at each agent while the load balancing algorithm is executing. New work is added and work may be completed. In this case the system may never converge to the desired state in which all agents have the same workload. The problem of interest is determining a bound (if one exists) of the error given by the distance between the actual state and the desired state.

As another example, we consider a system consisting of a network of sensors that exchange messages in an inaccessible environment. The goal of the system is to track some object traveling across some area. A system without exogenous inputs estimates the location of a stationary object. The movement of the object is modeled by exogenous inputs.

Other examples, include distributed coordination and flocking [34], vehicle formation [25, 61, 19] and sensor fusion [52, 49] systems. In the presence of exogenous inputs, the goal state of the system can change with time. We present conditions that ensure that the system is able to track these time-varying quantities.

1.5 Motivating Example

In this Section, we present theory for proving that a general class of multi-agent systems satisfy their specifications. Agents communicate via message-passing over an unreliable communication medium. We allow for lost, delayed, duplicated, and received out-of-order messages. The goal of agents in these systems is to form and maintain a time-varying regular spatial configuration starting from some arbitrary positions.

For example, in Figure 1.3 we present a sequence of snapshots of a multi-agent system whose goal is to form and maintain a regular grid. Examples of regular grids are presented in Figure 1.2 and Figure 1.5. This specific system consists of four leader agents and 76 follower agents. The desired configuration is a regular grid with corners given by the four leader agents. In this specific execution, leaders move according to the dynamics presented in Figure 1.4; in Figure 1.6, we present another sequence of snapshots of the same multi-agent systems. In this case, the dynamics of the leaders are presented in Figure 1.7.

We consider very simple protocols for agents where the new position of an agent is computed as the weighted average of the positions of the agents in its neighborhood. Agents move from their current locations to the newly computed one according to some dynamics. The specific dynamics depends on the current state of the agent. An agent may not reach its newly computed target

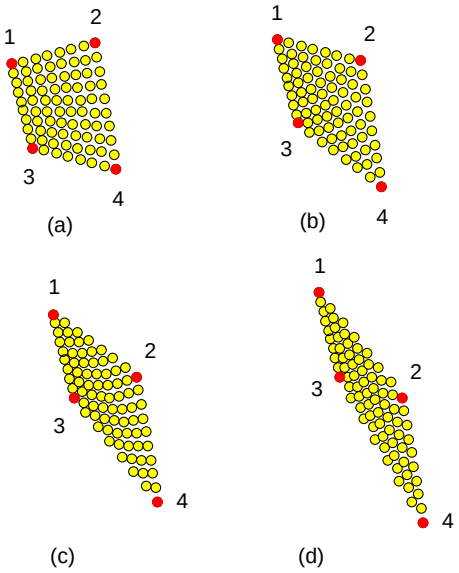


Figure 1.2: Examples of regular grids. The corner agents are depicted as dark filled circles and have identifiers ranging from 1 through 4.

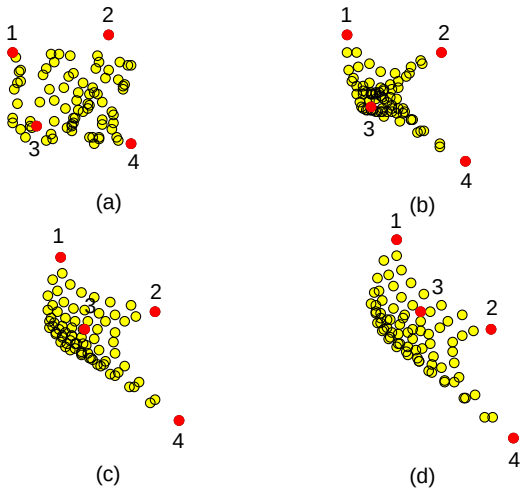


Figure 1.3: Snapshots of the execution of a multi-agent system whose goal is to form and maintain a time-varying spatial configuration. Leader agents having have identifiers ranging from 1 through 4 and are depicted as dark filled circles. The goal configuration, depicted in Figure 1.2 in the case of snapshots (a)-(d), is a regular grid with extremes given by the leader agents. The dynamics of the four leader agents is presented in Figure 1.4.

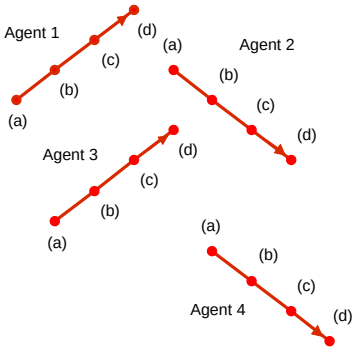


Figure 1.4: The dynamics of the leader agents in the execution presented in Figure 1.3. For each leader agent we emphasize the position of the leader agent at the snapshots (a)-(d) of Figure 1.3.

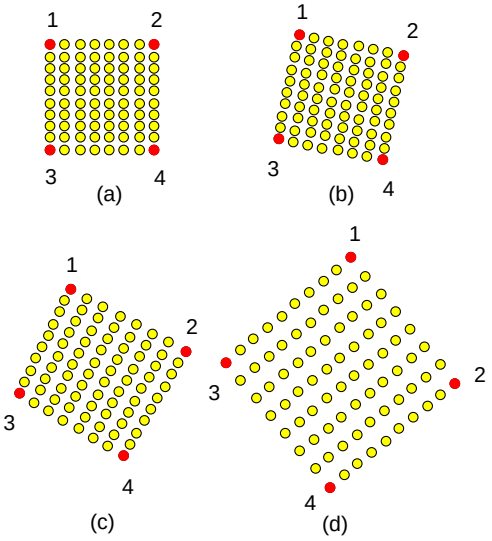


Figure 1.5: Examples of regular grids. The corner agents are depicted as dark filled circles and have identifiers ranging from 1 through 4.

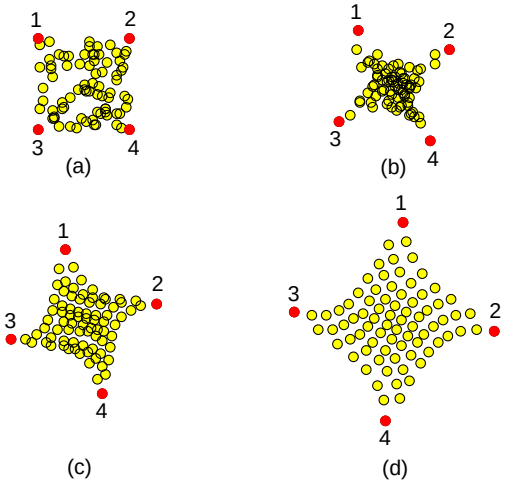


Figure 1.6: Snapshots of an execution of the multi-agent in Figure 1.3. The goal configuration, depicted in Figure 1.5 in the case of snapshots (a)-(d), is a regular grid with extremes given by the leader agents. The dynamics of the leader agents are presented in Figure 1.7.

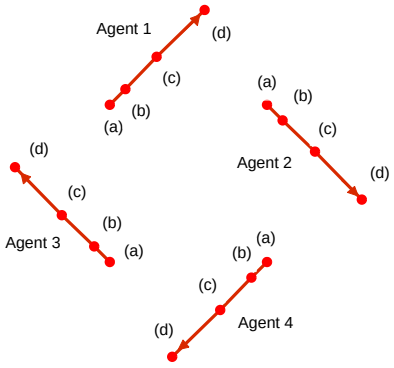


Figure 1.7: The dynamics of the leader agents in the execution presented in Figure 1.6. For each leader agent we emphasize the position of the leader agent at the snapshots (a)-(d) of Figure 1.6.

position, because it may receive a message from a neighbor while it is traveling, and this may change its target position. An agent updates its target state using the positions stored in the received messages that are potentially old, corrupted and computed at different times.

As a specific example, we consider the case where the positions of the agents are points on the real line. The system consists of two leader agents and $N-1$ follower agents. The goal of the follower agents is to form an equi-spaced line with extremes given by the two leader agents. The specific equi-spaced line changes with time, since leader agents can move with arbitrary dynamics. This is the 1-dimensional version of the multi-agent system presented in Figure 1.3 and Figure 1.6. In this specific system, we assume that agents have unique identifiers: the identifiers of the leader agents are 0 and N, while the identifiers of the follower agents are in the set $\{1, \ldots, N-1\}$. Agents exchange messages containing agents locations. Each follower agent keeps the last message that it received from any agent with lower identifier and it also keeps the last message from any agent with higher identifier. The updating rule of the follower agent is very simple. It computes its target position as the weighted average of the values stored in these two last received messages. The weights depend on the identifiers of the senders of these two messages. It then moves towards it according to some dynamics.

Proving the correctness of systems in this class is very challenging. We must prove that the distance between the current configuration of the system and the corresponding time-varying desired configuration is bounded. In this Thesis, we present a strategy for proving correctness of these systems. We next describe the main ideas discussed in this Thesis:

1. We start by proving the correctness of a shared-state multi-agent system where agents instantaneously update their current position with the newly computed one. In these systems, agents update their positions using the current positions of other agents. We assume that the desired final configuration does not change with time. This defines a shared-state multi-agent system with discrete, instantaneous actions. For this system, we prove that the system eventually reaches the desired configuration.

   For example, in the case of the multi-agent system whose goal is forming a equi-spaced line, we have that the leader agents are stationary and, in a nondeterministic fashion, an agent chooses two other agents and updates its location with the weighted average of these other two agents. We refer to this shared-state multi-agent system as the Line-Up multi-agent system.

2. We then relax the assumption on the agent dynamics. We allow agents to evolve their state from their current to the newly computed one according to some explicit dynamics. The proof of correctness of the shared-state multi-agent system with discrete actions remains valid in this system assuming specific dynamics for agents. For example, agents may move from their current locations towards their newly computed with constant velocity and straight trajectory.

3. We then relax the assumption on the communication. Agents do not operate over a perfect communication medium. The medium is unreliable. We allow for messages that can be lost, delayed, duplicated or received out-of-order. Values stored in the messages in transit correspond to potentially old locations of the agents. The proof of correctness of the shared-state system remains valid in the case of message-passing systems. This is not true in general; in this Thesis, we derive conditions on the shared-state systems that ensure it. As shown in this Thesis, this class of multi-agent systems satisfies these conditions.

4. We finally model noise in the transmission and time-varying formations using exogenous inputs. This general system is able to track the time-varying final configuration. This is not true in general; in this Thesis, we derive conditions on the agents protocol in the absence of exogenous inputs and on the exogenous inputs that ensure this property.

1.6 Structure of the Thesis

In this Section, we discuss the structure of the Thesis.

Chapter 2

In Chapter 2, we introduce the automaton with timed actions model. We use this model for representing multi-agent systems and exogenous inputs.

An automaton with timed actions consists of a set of states, a set of initial states, a set of timed actions, an enabling predicate and a transition function. In the case of the multi-agent system, the state space of the automaton models the state of its agents and, in case of message-passing communication, the state of the communication medium. The set of actions along with the enabling condition and transition function models the behaviour of the agents of the system. In the case of exogenous inputs, the state space of the automaton models the state space of the multi-agent system and its set of actions models the behaviour of these exogenous injections. We assume that exogenous inputs can modify any agent in the system.

We prove the correctness of a multi-agent system by showing properties of its executions. These properties are described using the linear logic operators always $\Box$ and eventually $\Diamond$ [59]. For example, the proof of correctness of the Line-Up multi-agent system requires to prove that the error of the system never increases, that is a safety property, and that the error of the system eventually decreases, that is a progress property. We also define fairness for multi-agent systems.

Chapter 3

In Chapter 3 we discuss properties of automata with timed actions. We use these properties for proving correctness of the corresponding multi-agent systems.

In this Chapter, we introduce the notion of equilibrium state. An equilibrium state is a fixed point of the executions of the automaton. Equilibrium states model goal configurations of multi-agent systems. For example, the goal configuration of the Line-Up multi-agent system is an equilibrium state of the corresponding Line-Up automaton.

In this Chapter, we introduce stability and convergence properties. Informally, an equilibrium state is stable if every execution of the automaton that starts close to the equilibrium state remains close to it. The system converges to an equilibrium state if any fair execution of the automaton that starts close to the equilibrium state converges to it. In this Chapter, we provide sufficient conditions on the automaton that ensure stability and convergence. We also discuss properties of automata in the presence of exogenous inputs. When injecting exogenous inputs, we are interested in the robustness of the system. We model this robustness property with the notion of bounded automaton. An automaton in the presence of exogenous inputs is bounded if eventually-always the distance between the system and the set of equilibrium states of the automaton in the absence of exogenous inputs is bounded.

The material covered in this Chapter has been published in [14]. Our work extends the work of [66] to systems with timed actions, in the special case of metric state space.

Chapter 4

In Chapter 4 we discuss stability and convergence properties of multi-agent systems. We consider both shared-state and message-passing multi-agent systems.

We first discuss properties of shared-state multi-agent systems. We model these systems as automata with discrete or timed actions. We prove correctness of these systems using the results of Chapter 3.

We, then, introduce a generalization of the shared-state multi-agent system model, where agents can update their state using the states of other agents computed at some times in the past. This is different from the shared-state automaton model where agents use the current state of the other agents. In this new model, called shared-state multi-agent system with sliding window, agents cannot read arbitrary old values; if t is the current time of the execution, agents can read the state of other agents up to $t - \mathcal{B}$. We present conditions on the shared-state system that preserve correctness when transforming the shared-state automaton into a shared-state with sliding window. We show that message-passing multi-agent systems with bounded delay can be modeled using shared-state automata with sliding window.

We finally discuss multi-agent systems with concurrent actions. In these systems, agents can execute actions concurrently. For example, in the Line-Up multi-agent system with concurrent actions, multiple agents can move at the same time. We model both shared-state and message-passing systems with concurrent actions and derive conditions that ensure stability and convergence of these systems.

The material of this Chapter has been partially presented in [16] and extends the work of [66, 8] where they prove stability and correctness of multi-agent systems with concurrent discrete actions.

Chapter 5

In Chapter 5 we discuss the correctness of a general class of iterative schemes. The goal of systems in this class is solving a system of linear equations of the form $A \cdot x = b$. These systems iteratively compute vector x starting from an initial guess vector $x0$. These are decentralized schemes where each agent is responsible for solving a specific variable using a specific equation of the system of linear equations. For example, agent i would be responsible for computing $x(i)$ using as updating rule the i-th equation of the system. We consider both shared-state and message-passing multi-agent systems.

We, first, consider shared-state multi-agent systems. We prove the correctness of schemes in this class using the results in Chapter 3. We require the matrix A to satisfy specific assumptions. We, then, consider message-passing multi-agent systems. In these systems, agent i repeatedly broadcasts a message containing the current value of $x(i)$. Agent i uses the i-th equation as its updating rule. In this equation, variable $x(i)$ is the only unknown, since agent i uses the last message received from agent j to represent $x(j)$ for all $j \neq i$. We prove correctness of message-passing systems using results of Chapter 4.

In this Chapter, we discuss a linear robot pattern formation protocol that can be modeled as a multi-agent system whose goal is solving a system of equations. This protocol is a special case of the Line-Up multi-agent system. We derive its proof of correctness from the convergence property of systems in this class.

The material covered in this Chapter has been published in [14]. It extend the work of [28] in the linear case, and the work of [18] relaxing some assumptions on the equations in the system.

Chapter 6

In Chapter 6 we present a library of PVS [53] meta-theories that can be used to verify message-passing multi-agent systems discussed in Chapter 5. Our framework can be downloaded from [58] and consists of over 200 lemmas and approximately 8700 proof steps. It consists of a library of PVS meta-theories built on the top of I/O automata [42, 3, 2, 4] with the extension for timed and hybrid

systems [36, 46, 45, 35, 38].

In this Chapter, we detail the structure of the framework. It consists of three main libraries. The first library describes the state space, initial states, actions, enabling condition and transition function of the message-passing multi-agent system. The state of the system consists of the state of the agents and the state of the communication medium. The second library encodes the proof of correctness of these systems in PVS. The third library presents auxiliary lemmas on predefined data structures such as vectors and matrices.

When using this framework, the end-user is required to discharge some assumptions on the matrix A. As an example, we apply our verification framework for proving correctness of the robot pattern formation protocol described in Chapter 5.

The material covered in this Chapter has been published in [56, 57]. Our work follows a very large body of literature where theorem provers have been used for modeling [29, 30, 11, 37] and verification [33, 27, 44]. A theorem prover is an appropriate tool when modeling nondeterministic systems with dense state spaces. Other tools that rely on exhaustive state space exploration, such as model checkers, may present difficulties when dealing with this issue. However, there are some exceptions. For example, in [31], the author checks the time to reach agreement of a consensus protocol using the UPPAAL model checker [7]. The author is able to reduce the state space of the system using a key compositional property of the protocol.

Chapter 7

In Chapter 7 we discuss properties of the multi-agent systems in the presence of exogenous inputs.

We prove that the system in the presence of exogenous inputs is bounded, if the multi-agent system in the absence of exogenous inputs and the exogenous inputs injected in the system satisfy specific properties. These conditions require the system in the absence of exogenous inputs to execute additive protocols and require the exogenous inputs to be uniformly bounded quantities added to the agents.

We apply these results to the class of systems presented in Chapter 5. The goal of systems in this class is solving systems of linear equations. We obtain that in the presence of exogenous inputs systems in this class are bounded. For example, we consider the robot pattern formation protocol discussed in Chapter 5. For this system, we discuss two exogenous inputs models. In the first one, we consider exogenous inputs that modify the locations of the leader agents. This corresponds to modeling a system where the goal configuration changes with time. We prove that the system eventually-always is closed to an equi-spaced line. In the second model, we consider exogenous inputs that modify the locations of follower agents. This corresponds to modeling a system where the communication between agents is noisy. In this case, we also prove that the system eventually-always gets close to the goal configuration, assuming that the transmitting noise is bounded.

The material covered in this Chapter has been published in [55]. Our work extends previous work on multi-agent systems in the presence of adversarial conditions such as [62, 63, 65, 26, 68, 69]. In these papers, authors focus on shared-state multi-agent systems solving consensus-type problems [17, 22, 64, 9], i.e. the goal of these systems is tracking time-varying quantities.

Chapter 2

Formal Models for Multi-Agent Systems

In this Chapter, we present formal models for multi-agent systems. We use these models to formally represent multi-agent systems and describe their behaviors.

In Section 2.1 we review the automaton model. In Section 2.2 we present a new extension of the automaton model, that allows modeling systems with continuous-time actions. In Section 2.3 we introduce a novel notion of fairness for automata. This notion of fairness will be used throughout the Thesis. In Section 2.4 we discuss an automaton that models multi-agent systems in the presence of exogenous inputs. Finally, in Section 2.5 we relate the main results of this Chapter to the literature.

2.1 Automata

In this Section, we review the automaton model and present an example.

2.1.1 Automaton Model

We refer to [5] for a discussion of the concept of automata. Formally, an *automaton* has the following structure.

Definition 1. *An automaton is a quintuple* $(S, S0, A, E, T)$ *consisting of*

- *nonempty set of states* S
- *nonempty set of start (initial) states* $S0$
- *a set of actions* A
- *an enabling predicate* $E : S \times A \to \mathcal{B}$,
- *a transition function* $T : S \times A \to S$

The set $\mathcal{B}$ denotes the set of boolean values, $\mathcal{B} = \{true, false\}$. For $s \in S$ and $a \in A$, $E(s, a)$ holds if and only if action a can be executed in the state s. If this is the case, we say that a is *enabled* in s. For convenience, we write $s \xrightarrow{a} s'$ to denote $(s, a, s') \in T$.

The semantic of an automaton is defined in terms of its executions; these describe the behavior of the system.

Definition 2. *An execution fragment is a possibly infinite alternating sequence of states and actions* $s_0, a_0, s_1, a_1, \ldots$ *such that* $s_{i+1} = T(s_i, a_i)$ *and* $E(s_i, a_i)$ *holds.*
An execution fragment is a system execution if $s_0 \in S0$.

Given an execution π of the system, we denote by $\pi.fstate$ the first state of the execution. If π is finite, we denote by $\pi.lstate$ the last state of the execution.

We next introduce the concept of *reachability*. Given $s, s' \in S$, we say that s' is *reachable from* s, if there exists a finite (possibly empty) execution fragment starting from s that reaches s'. We denote by $RF(s)$ the set of states reachable from s, defined formally as:

Definition 3. *Given* $s \in S$, *the set* $RF(s)$ *is defined as*

$$RF(s) = \{s' \in S \mid \exists \pi : \pi.fstate = s \wedge \pi.lstate = s'\}$$

We can extend this definition to a set of states $\hat{S}$. The set of reachable states from $\hat{S}$ is the union of the set of reachable states from its elements. Formally,

Definition 4. *Given* $\hat{S} \in S$,

$$RF(\hat{S}) = \{s' \in S \mid \exists \hat{s} \in \hat{S},\ \exists \pi : \pi.fstate = \hat{s} \wedge \pi.lstate = s'\}$$

We introduce the reachable predicate as follows:

Definition 5. *Given* $s \in S$,

$$r(s) \equiv (s \in RF(S0))$$

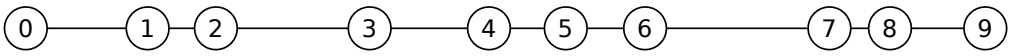
The predicate holds if s is reachable from some initial state. If $r(s)$ holds, we say that s is *reachable*.

Automata are used for formalizing discrete-time systems. Actions of the automaton have constant discrete execution times. When an action is executed by the automaton, the clock of the system is advanced by some fixed discrete time. For this reason, these automata are usually called discrete automata.

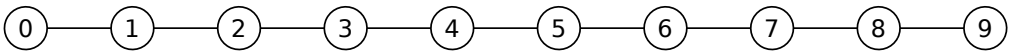
2.1.2 Line-Up Automaton

In this Section, we present an example of discrete automaton. We consider a system consisting of $N+1$ agents, each having a unique identifier $i \in \{0, 1, \ldots, N\}$. These agents start at arbitrary positions

(stored in the vector $x0$), and, through interactions, their goal is to converge to a configuration where agents are located, in order, at equidistant points on a straight line with extremes $x0(0)$ and $x0(N)$. Without loss of generality, we present the one-dimensional version of the protocol, where agent positions are real values. Figure 2.1 shows two configurations of the Line-Up multi-agent system when the system consists of ten agents; Figure 2.1(a) represents a generic configuration while Figure 2.1(b) represents the desired final configuration, where all agents are in order and equispaced on the line.



(a) Pictorial representation of an initial state of the Line-Up multi-agent system.



(b) Pictorial representation of the corresponding goal state.

Figure 2.1: A Line-Up multi-agent system consisting of 10 agents.

Agents update their state in a nondeterministic fashion. Agent i chooses nondeterministically two agents l, r with $l < i < r$ and sets its new position x' as the weighted average of the positions of l, r:

$$x' = \frac{r-i}{r-l}x(l) + \frac{i-l}{r-l}x(r) \tag{2.1}$$

where x is the vector of agent positions. We denote this updating rule by $avg_{l,i,r}$. Figure 2.2 pictorially represents this updating rule for a given example; Figure 2.2(a) represents agents l, i, r before i executes the updating rule and Figure 2.2(b) represents the same agents after the execution of the rule. As discussed before, only agent i modifies its position; its new position is a linear combination of the positions of l, r. Notice that, in the case when $l = i - 1$ and $r = i + 1$, we have that x' is the average of the positions of $x(i-1)$ and $x(i+1)$, since $\frac{r-i}{r-l} = \frac{i-l}{r-l} = \frac{1}{2}$.

The goal configuration of the Line-Up system is the vector $\hat{x}$ where $\forall i \leq N$

$$\hat{x}(i) = \frac{N-i}{N}x0(0) + \frac{i}{N}x0(N) \tag{2.2}$$

(a) Configuration of agents l, i and r.

(b) Configuration of agents l, i and r after agent i executes the updating rule.

Figure 2.2: Pre- and post configurations of the updating rule in the case when the identifiers of the agents are $i = 2$, $l = 1$ and $r = 4$.

We refer to Figure 2.1(b) for a pictorial representation of the desired goal configuration for a system of 10 agents.

The automaton modeling the Line-Up multi-agent system has the following structure:

- $S = \mathbb{R}^{N+1}$, since the state space of each agent is $\mathbb{R}$.
- $S0 = \{s0\}$, with $s0 = x0$,
- $A = \{A_i\}_{i \in I}$ with $A_i = \{avg_{l,i,r}\}_{l<i<r}$,
- $E : S \times A \to true$
- $T : S \times A \to S$, defined as $\forall a = avg_{l,i,r} \in A$, $\forall s \in S$,

$$T(s,a) = \left(s(0), s(1), s(2), \ldots, \frac{r-i}{r-l}s(l) + \frac{i-l}{r-l}s(r), \ldots, s(N)\right)$$

For this model, agents 0 and N are stationary.

As presented in Equation 2.1, the goal configuration of the Line-Up system is the state $\hat{s}$ where $\forall i \leq N$

$$\hat{s}(i) = \frac{N-i}{N}s0(0) + \frac{i}{N}s0(N) \tag{2.3}$$

with $s0 \in S0$.

In Figure 2.3, we present a formalization of the system in the PVS theorem prover. The system consists of $N+1$ agents, with $N > 0$. Each agent has a unique identifier. This is a natural number in the interval $[0, N]$. In PVS, the type of agent identifiers is `I`. The state space of each agent is the set of real numbers. The state space of the system is defined by the function `S`, that maps each agent identifier to a real value. This theory has an input parameter given by the state `s0`. This parameter stores the initial configuration of the system. The set of initial states is encoded in PVS using the initial state predicate `start?`. The set of actions `A` of the system consists of the action `avg`. This action has three parameters; these are identifiers of three agents: `i,left,right`. As encoded in the enabling condition predicate `E`, this action can be executed only if agent `i` is properly contained in the interval `[left, right]`. When executing this action, the post-state of the action is equal to the pre-state, with the exception of the i-th coordinate. This entry stores the weighted average of `s(left)` and `s(right)`. The execution of this action is encoded in PVS by the transition function `T`.

2.2 Automata with Timed Actions

In this Section, we present a novel model, called automaton with timed actions. It extends the model presented in Section 2.1. It allows for actions with non-constant time duration. It can be

```
% Number of agents of the system.
N: posnat

% Agent Identifier.
% It is a natural number in the interval [0,N].
I: TYPE = upto(N)

% State definition.
% A state maps each agent identifier into a real value.
S: TYPE = [I -> real]

% PVS meta-theory.
% s0 is an input of the theory.
% s0 represents the initial configuration of the system.
LineUP[s0: S]: THEORY
BEGIN
        i,left,right: VAR I

        % Action set of the system.
        A: DATATYPE
        BEGIN
                % avg action
                % its parameters are left, i, right
                avg(left,i,right): avg?
        END A

        s: VAR S
        a: VAR A

        % Initial State Predicate.
        start?(s): bool = (s = s0)

        % Enabling Predicate.
        E(s,a): bool =
        CASES a OF
                % avg is enabled if left<i<right
                avg(left,i,right): (left<i) AND (i<right)
        ENDCASES

        % Transition Function.
        T(s,a): S =
        CASES a OF
                % Agent i sets its value to the weighted average of
                % the values of left and right
                avg(left,i,right): LET
                        w_left: posreal = (right-i)/(right-left),
                        w_right: posreal = (i-left)/(right-left)
                        IN
                        s WITH [(i):= w_left*s(left) + w_right*s(right)]
        ENDCASES

END LineUP
```

Figure 2.3: PVS representation of the Line-Up multi-agent system.

used for modeling systems with continuous-time dynamics, such as mobile multi-robot systems.

We first present the model and discuss it. We then define the concepts of executions and reachability for this automaton. We then extend the meaning of the temporal operators always $\Box$ and eventually $\Diamond$ to this class. We finally present an example and relate this model to other timed system models.

2.2.1 Automaton Model

Informally, an automaton with timed actions is a generalization of the discrete automaton model presented in Section 2.1. An automaton with timed actions has a state space S, a set of initial states $S0$ and a set of actions A. Unlike the discrete automaton model, actions have time duration. Formally,

Definition 6. *A automaton with time actions $\mathcal{A}$ is a tuple $(S, S0, A, E, T)$ consisting of*

- *nonempty set of states S,*
- *nonempty set of start (initial) states $S0$,*
- *a set of actions A,*
- *an enabling predicate $E : S \times A \to \mathcal{B}$,*
- *a transition function $T : S \times A \to (\mathbb{T} \to S)$ with*

$$\mathbb{T} = \{[0, \tau] \mid \epsilon \leq \tau < \infty\}$$

such that $\forall s \in S, a \in A$,

$$T(s, a)(0) = s$$

The duration and behaviour of the actions of $\mathcal{A}$ are encoded in the transition function T. Parameters of T are a state in S and an action in A. By construction of the transition function, we have that the same action can have different time duration and behaviour when executed at different states. The transition function T specifies the behaviour of the action throughout its duration. Given a state s and an action a, $T(s, a)$ defines a mapping from a closed finite time interval to the state space of $\mathcal{A}$. The state $T(s, a)(0)$ is equal to state s, since $T(s, a)(0)$ specifies the behavior of the action at the beginning of the interval. The closed finite interval is lower bounded by some constant $\epsilon > 0$, i.e. we do not allow for Zeno executions. As an example, Figure 2.4 presents a pictorial representation of the behaviour of a timed action a. The system consists of a single real-valued variable. When a is executed in state $s = 0$, the action has time duration equal to 1 time unit and evolves the state

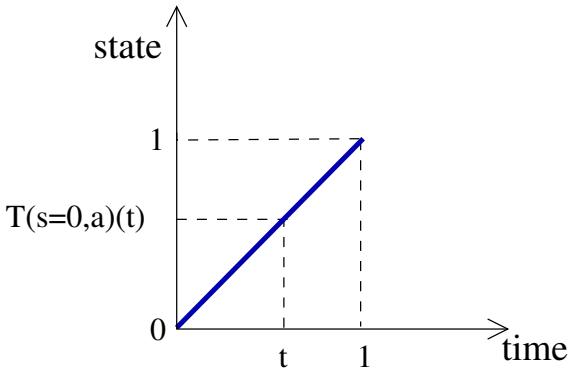


Figure 2.4: Pictorial representation of behavior of a timed action. The system consists of variable x, that evolves with constant velocity from value 0 to value 1 for 1 time unit.

with constant velocity $v = 1$. Mathematically, the behaviour of the timed action a can be expressed using the function T as follows: $\forall t \in [0, 1]$, $T(s, a)(t) = s + v \cdot t$, with $s = 0$ and $v = 1$.

In our model, the set of actions has a special action, called no-op action. This action advances the time of the automaton, in the case when the execution of the system reaches a state where no action is enabled. We require this action since the execution cannot move forward in time if no action is enabled. This is analogous to the *skip* action in UNITY [15]. This action is enabled in state $s \in S$ if no other action is enabled in s, i.e.

$$E(s, \text{no-op}) = (\forall a \in A, a \neq \text{no-op}, \neg(E(s, a)))$$

The time duration of this action in s is ϵ

$$T(s, \text{no-op}) = [0, \epsilon] \rightarrow S$$

When executed in s, it does not modify s, i.e. $\forall t \in [0, \epsilon]$ we have that

$$T(s, \text{no-op})(t) = s$$

This action advances the execution of the automaton by ϵ time units.

This model generalizes the discrete automaton model presented in Section 2.1. An automaton $\mathcal{A}$ can be encoded as an automaton with timed actions $\mathcal{A}_{time}$. Automaton $\mathcal{A}_{time}$ has the same state space, initial states and actions of $\mathcal{A}$. Actions of $\mathcal{A}$ can be modeled as actions in $\mathcal{A}_{time}$ as follows. The time duration of each action of $\mathcal{A}$ in $\mathcal{A}_{time}$ is 1 unit of time. When executing action a from state s, the system remains in state s for all $t < 1$ and moves to $T(s, a)$ at time 1. This action models the discrete behaviour of the execution of a in $\mathcal{A}$. Formally, $\mathcal{A} = (S, S0, A, E, T)$ can be modeled as the timed automaton $\mathcal{A}_{time} = (S_{time}, S0_{time}, A_{time}, E_{time}, T_{time})$ as follows:

- $S_{time} = S$
- $S0_{time} = S0$
- $A_{time} = A$
- $\forall s \in S_{time}, a \in A_{time}$, $E_{time}(s,a) = E(s,a)$
- $\forall s \in S_{time}, a \in A_{time}$, $T(s,a) : [0,1] \to S$ with

$$\begin{aligned} \forall t < 1 \quad : \quad & T_{time}(s,a)(t) = s \\ & T_{time}(s,a)(1) = T(s,a) \end{aligned}$$

2.2.2 Executions and Reachability

In this Section, we extend the concepts of executions and reachability to automata with timed actions. Throughout the Thesis, given an automaton with timed actions $\mathcal{A} = (S, S0, A, E, T)$, given $s \in S, a \in A$, we denote by $f_{s,a}$ the function $T(s,a)$, and by $\tau_{s,a}$ the right extreme of the domain of $f_{s,a}$, i.e. $f_{s,a} : [0, \tau_{s,a}] \to S$; by construction, $\tau_{s,a}$ is finite. We denote by $\text{end}_{s,a}$ the state $f_{s,a}(\tau_{s,a})$, i.e. it is the state resulting into the evaluation of the function $f_{s,a}$ at the right extreme of the interval. For convenience, we denote $(s, a, f_{s,a}) \in T$ as $s \xrightarrow{a} \text{end}_{s,a}$.

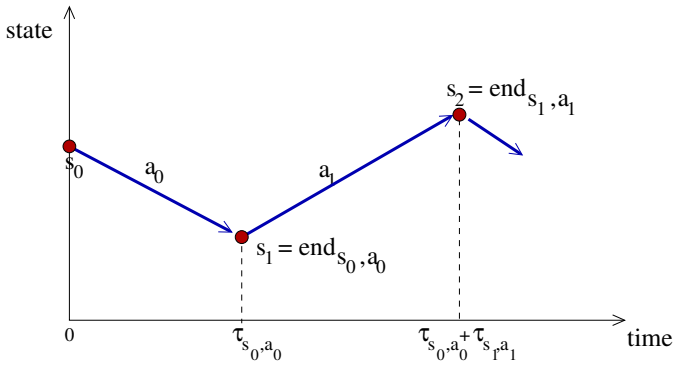


Figure 2.5: Pictorial representation of an end-state execution fragment. Dark filled circles represent end-states of the execution and arrows represent the execution of the actions. This end-state execution fragment starts at state s_0 and executes actions $a_0, a_1, \ldots$. When executed from state s_0, action a_0 has time duration τ_{s_0,a_0}. Action a_1 has time duration τ_{s_1,a_1} when executed from state $s_1 = \text{end}_{s_0,a_0}$.

We next define the concept of execution fragment. We first consider a special set of executions, called end-state execution fragments. These executions are the natural generalization of the notion of execution fragments of discrete-action automata. Formally,

Definition 7. *An end-state execution fragment π is a possibly infinite alternating sequence of states and actions $\pi = s_0, a_0, s_1, a_1 \ldots$ such that $s_{i+1} = end_{s_i,a_i}$, and $E(s_i, a_i)$ holds.*
If the execution is finite, i.e. $\pi = s_0, a_0, s_1, a_1 \ldots a_{N-1}, s_N$, then its time duration is $\tau = \sum_{i=0}^{N-1} \tau_{s_i,a_i}$.

An end-state execution fragment starts from a state s_0 and executes action a_0 from state s_0. The execution of a_0 has a time duration τ_{s_0,a_0} and end-state equal to s_1. In state s_1, it executes action a_1 that has time duration τ_{s_1,a_1} and end-state equal to s_2 and so on. If 0 is the time of the execution π at state s_0, we have that the time of π at state s_1 is τ_{s_0,a_0}, the time of the execution at state s_2 is $\tau_{s_0,a_0} + \tau_{s_1,a_1}$ and so on. Given π we denote by $t_0, t_1, \ldots$ the possibly infinite sequence of times with t_i being the time of π at state s_i. This sequence of times can be defined recursively: $t_0 = 0$ and $t_i = t_{i-1} + \tau_{s_i,a_i}$. Figure 2.5 presents an example of end-state execution fragment; this fragment starts at s_0 and executes actions $a_0, a_1, \ldots$.

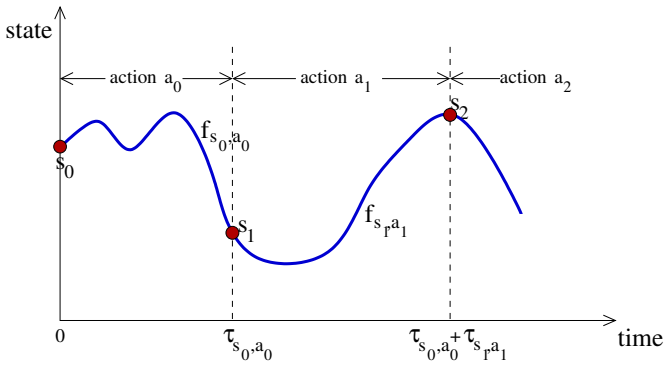


Figure 2.6: Pictorial representation of the function describing the end-state execution fragment depicted in Figure 2.5. In this specific example, $\hat{t} = 0$. Dark filled circles represent end-states of the execution. The curve in the time interval $[0, \tau_{s_0,a_0}]$ represents the behavior of action a_0 when executed in state s_0. This behaviour is described by function f_{s_0,a_0}. The curve in the time interval $[\tau_{s_0,a_0}, \tau_{s_0,a_0} + \tau_{s_1,a_1}]$ represents the behavior of action a_1 when executed in state s_1. This behavior is described by function f_{s_1,a_1}.

In general, given an end-state infinite execution fragment $\pi = s_0, a_0, s_1, a_1 \ldots$, and a time $\hat{t}$ we have that π can be represented as a function $\hat{\pi} : [\hat{t}, \infty) \to S$ such that $\forall i \in \mathbb{N}$:

$$(\hat{\pi}(t_i) = s_i) \wedge (E(\hat{\pi}(t_i), a_i)) \wedge (\forall t \in [t_i, t_{i+1}] \ : \ \hat{\pi}(t) = f_{s_i,a_i}(t))$$

where the infinite sequence of times $t_0, t_1, \ldots$ is recursively defined as $t_0 = \hat{t}$ and $t_i = t_{i-1} + \tau_{s_i,a_i}$. We next briefly discuss these three conditions: (1) the first condition ensures that s_i is the state of the execution at time t_i, (2) the second condition ensures that action a_i is enabled in state s_i, (3) the third condition ensures that action a_i is executed in state s_i and, together with the first condition, that the post-state of the execution is s_{i+1}. For example, Figure 2.6 represents the function corresponding to the end-state execution fragment in Figure 2.5.

Similarly, if the execution fragment π is finite, i.e. $\pi = s_0, a_0, s_1, \ldots, s_{N-1}, a_{N-1}, s_N$ we have

that the function corresponding to π is defined as $\hat{\pi} : [\hat{t}, \hat{t} + \tau] \to S$, with τ being the time duration of π. In this specific case, we have that the sequence of times is finite, and given by $t_0, t_1, \ldots, t_N$ with $t_0 = \hat{t}$ and $t_i = t_{i-1} + \tau_{s_i,a_i}$. The conditions on the function are the same as the conditions for the case of the infinite execution fragment, and, thus, they are not reported.

We denote by $\mathcal{E}_{end-state}$ the set of all functions corresponding to end-state execution fragments of the system. Throughout this Thesis, we do not distinguish between execution fragments and their functional representations. We use them interchangeably.

In our model, actions have time duration. For this reason, feasible execution fragments may start during the execution of some action of the fragment. Also, in case of finite fragments, they may end during the execution of some action or they may start and end during the execution of some action. We refer to Figure 2.7, Figure 2.8 and Figure 2.9 for a pictorial representation of these three cases.

The finite fragment in Figure 2.8 starts at state s_0 and executing action a_0 in state s_0 reaches state s_1. It then initiates action a_1 starting from s_1 for only $t - \tau_{s_0,a_0}$ time units. It stops before the action is completed at time $\tau_{s_0,a_0} + \tau_{s_1,a_1}$. In our model, this fragment is a valid execution fragment.

The fragment in Figure 2.9 starts from a state s'_0 with $s'_0 = T(s_0, a_0)(t)$ and $t > 0$. The fragment starts while action a_0 is executing. Then, it reaches s_1 and executes a_1, and so on. In our model, also this fragment is a valid execution fragment.

We next introduce the sub-fragment operator.

Definition 8. *Given an end-state execution fragment $\pi : \Delta \to S$ and a time interval $\hat{\Delta}$, the sub-fragment of π with respect to $\hat{\Delta}$, denoted by $sub(\pi, \hat{\Delta})$, is a function $\hat{\pi} : \hat{\Delta} \to S$ such that*

- *$\hat{\Delta}$ is a sub-interval of Δ and*
- *$\hat{\pi}$ is the restriction of π to $\hat{\Delta}$.*

Figure 2.7, Figure 2.8 and Figure 2.9 are examples of sub-fragments of the end-state execution fragment presented in Figure 2.6. In the sub-fragment depicted in Figure 2.7 the time interval $\hat{\Delta}$ is $[t_1, t_2]$; in the sub-fragment depicted in Figure 2.8, $\hat{\Delta} = [0, t]$; in the sub-fragment depicted in Figure 2.9, $\hat{\Delta} = [t, \infty)$.

If $\hat{\Delta}$ is a prefix of Δ, we say that $\hat{\pi}$ is a prefix of π. For example, the execution fragment in Figure 2.8 is a prefix of the end-state execution fragment presented in Figure 2.6 since $\hat{\Delta} = [0, t]$ is a prefix of $[0, \infty)$. Instead, if $\hat{\Delta}$ is a suffix of Δ, we say that $\hat{\pi}$ is a suffix of π. For example, the execution fragment in Figure 2.9 is a suffix of the end-state execution fragment presented in Figure 2.6 since $\hat{\Delta} = [t, \infty)$ is a suffix of $[0, \infty)$.

Given this operator, we next define the notion of execution fragment.

Definition 9. *A function $\hat{\pi} : \hat{\Delta} \to S$ is an execution fragment if there exists an end-state execution fragment π such that $\hat{\pi} = sub(\pi, \hat{\Delta})$.*

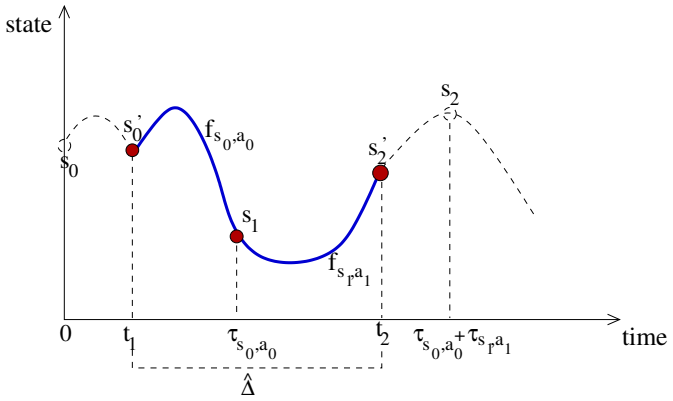


Figure 2.7: Pictorial representation of a finite sub-fragment of the end-state execution fragment depicted in Figure 2.6. This sub-fragment has domain $\hat{\Delta} = [t_1, t_2]$. It starts in state s'_0 with $s'_0 = T(s_0, a_s)(t_1)$ and it ends in state s'_2 with $s'_2 = T(s_1, a_1)(t_2 - \tau_{s_0,a_0})$. It executes action a_0 starting from time t_1, then it executes action a_2 for $t_2 - \tau_{s_0,a_0}$ time units. The dark solid line represents the behavior of the actions in this sub-fragment, while the dashed lines represent the behavior of the actions in the end-state execution fragment presented in Figure 2.6.

Given an execution fragment π, we denote by $\pi.fstate$ the first state of π and $\pi.lstate$ the last state of π, if the execution π is finite. If π is a finite execution fragment, then the time interval $\hat{\Delta}$ is closed and finite. Instead, if π is an infinite end-state execution fragment, then $\hat{\Delta}$ can be either closed and finite or left-closed and infinite. The function in Figure 2.8 is an execution fragment since it is the restriction of the end-state execution fragment presented in Figure 2.6 to the interval $[0, t]$. Similarly, the function in Figure 2.9 is an execution fragment since it is restriction of the end-state execution fragment presented in Figure 2.6 to the interval $[t, \infty)$.

Given an end-state execution fragment π we denote by $\mathit{Suffix}(\pi)$ the set of suffix execution fragments of π. If π is an infinite execution fragment, we have that the set $\mathit{Suffix}(\pi)$ is infinite. Instead, if π is a finite execution fragment, we have that the set $\mathit{Suffix}(\pi)$ contains a finite number of execution fragments.

We denote by $\mathcal{E}$ the set of execution fragments and by $\mathcal{E}_\infty$ the set of infinite execution fragments (i.e. $\mathcal{E}_\infty \subset \mathcal{E}$). By construction, $\mathcal{E}$ is the closure of $\mathcal{E}_{end-state}$ under the sub-fragment operator. An execution fragment $\pi : \Delta \to S$ is an execution, if 0 is the left extreme of Δ and $\pi.fstate \in S0$. We denote by $\mathcal{E}_{S0}$ the set of executions and by $\mathcal{E}_{S0,\infty}$ the set of infinite executions.

Given a state $s \in S$, we next define the set of reachable states from s.

Definition 10. *Given $s \in S$, the set of reachable states from s is*

$$RF(s) = \{s' \mid \exists \pi \in \mathcal{E} \ : \ \pi.fstate = s \wedge \pi.lstate = s'\}$$

A state is reachable if there exists a finite execution fragment that reach it. Given $\hat{S} \subseteq S$, we can generalize the notion of reachability to set of states.

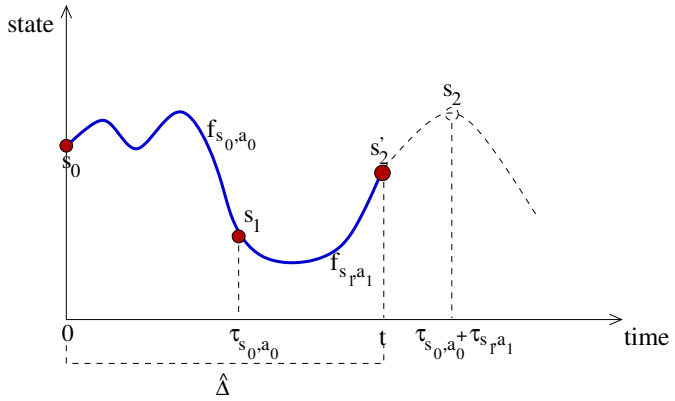


Figure 2.8: Pictorial representation of a prefix of the end-state execution fragment depicted in Figure 2.6. This prefix has domain $\hat{\Delta} = [0, t]$. It starts in state s_0 and it ends in state s'_2 with $s'_2 = T(s_1, a_1)(t - \tau_{s_0,a_0})$. It executes action a_0 starting from state s_0, it then executes action a_2 for $t - \tau_{s_0,a_0}$ time units. The dark solid line represents the behavior of the actions in this sub-fragment, while the dashed line represents the behavior of the actions in the end-state execution fragment presented in Figure 2.6.

Definition 11. *Given $\hat{S} \in S$, the set of reachable states from the set $\hat{S}$ is*

$$RF(\hat{S}) = \{s' \mid \exists \hat{s} \in \hat{S}, \exists \pi \in \mathcal{E} \ : \ \pi.fstate = \hat{s} \wedge \pi.lstate = s'\}$$

2.2.3 Temporal Operators

In this Section, we extend the definitions of the temporal operators always ($\Box$) and eventually ($\Diamond$).

Before proceeding with these definitions, we review the concept of predicate. A predicate P on the state space of the automaton $\mathcal{A}$ is a function that returns a boolean value, i.e. $P : S \rightarrow \mathcal{B}$. We say that predicate P holds in state $s \in S$, if $P(s) = true$. We next extend this definition to execution fragments and automata. Predicate P holds for an execution fragment π, denoted by P_π, if P holds at $\pi.fstate$, i.e. if predicate P holds at the initial state of π. A predicate P holds for an automaton $\mathcal{A}$ if P holds for all system executions.

The $\Box$ operator is defined as follows.

Definition 12. *Given a predicate P,*

- *$\Box$ P holds for an execution fragment π, denoted by $\Box_\pi$ P, if $\forall \hat{\pi} \in Suffix(\pi)$: $P_{\hat{\pi}}$*
- *$\Box$ P holds for an automaton $\mathcal{A} = (S, S0, A, E, T)$, if $\forall \pi \in \mathcal{E}_{S0,\infty}$: $\Box_\pi$ P*

$\Box$ P holds for an execution fragment π if the predicate P holds for all suffix execution fragments of π. Therefore, $\Box$ P holds for π if P holds for every state in π. The execution π can be finite or infinite. Figure 2.10 presents an execution fragment where $\Box$ P holds, while Figure 2.11 presents an execution fragment where $\Box$ P does not hold. $\Box$ P holds for an automaton $\mathcal{A}$ if it holds for all its infinite executions.

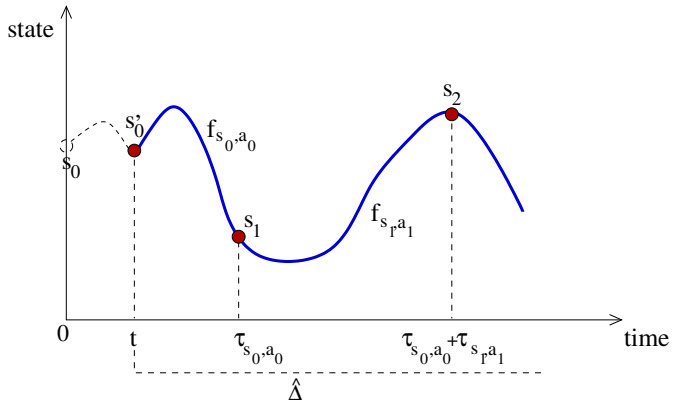


Figure 2.9: Pictorial representation of a suffix of the end-state execution fragment depicted in Figure 2.6. This suffix has interval $\hat{\Delta} = [t, \infty)$. This is an infinite sub-fragment that starts in state s'_0 with $s'_0 = T(s_0, a_0)(t)$. It executes action a_0 starting from time t, then it executes action a_2 and so on. The dark solid line represents the behavior of the actions in this sub-fragment, while the dashed line represents the behavior of the actions in the end-state execution fragment presented in Figure 2.6.

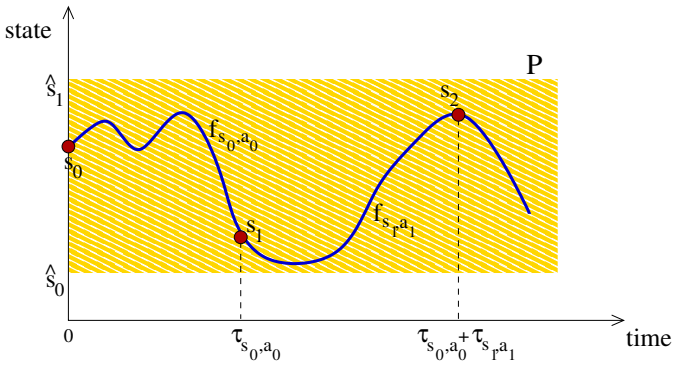


Figure 2.10: Execution fragment where $\Box\ P$ holds; predicate P holds in s if $s \in [\hat{s}_0, \hat{s}_1]$. The shaded area represents the region where predicate P holds.

Similarly, the eventually operator is defined as follows.

Definition 13. *Given a predicate P,*

- $\Diamond\ P$ *holds for an execution fragment π, denoted by $\Diamond_\pi\ P$, if $\exists \hat{\pi} \in Suffix(\pi)\ :\ P_{\hat{\pi}}$*
- $\Diamond\ P$ *holds for an automaton $\mathcal{A} = (S, S0, A, E, T)$, if $\forall \pi \in \mathcal{E}_{S0,\infty}\ :\ \Diamond_\pi\ P$*

$\Diamond\ P$ holds for a possibly infinite execution fragment π if the predicate P holds for some suffix execution fragments of π. Therefore, $\Diamond\ P$ holds for π if P holds in some state of π. In Figure 2.11, we presents an execution fragment satisfying $\Diamond\ P$ and in Figure 2.12, we represent an execution fragment where $\Diamond\ P$ does not hold. $\Diamond\ P$ holds for an automaton $\mathcal{A}$ if it holds for all its infinite executions.

From the definition of reachability, it follows that

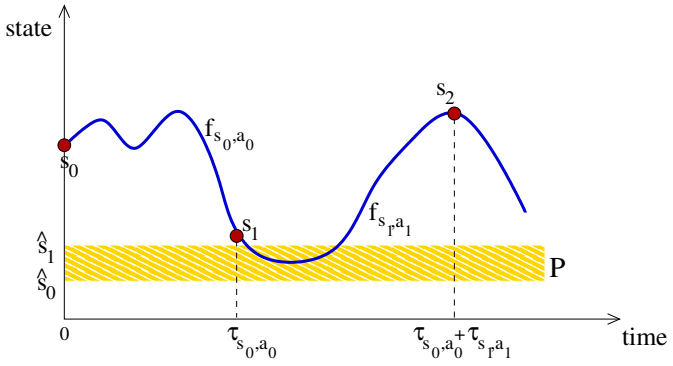


Figure 2.11: Execution fragment where $\diamond\ P$ hold; predicate P holds in s if $s \in [\hat{s}_0, \hat{s}_1]$. The shaded area represents the region where predicate P holds.

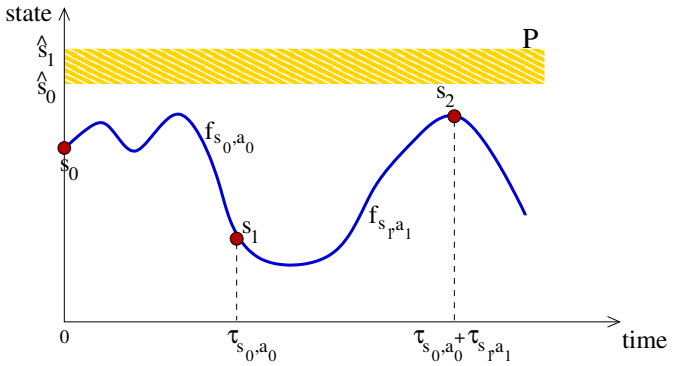


Figure 2.12: Execution fragment where $\diamond\ P$ does not hold; predicate P holds in s if $s \in [\hat{s}_0, \hat{s}_1]$. The shaded area represents the region where predicate P holds.

Lemma 1.

$$\Box\ P \equiv (RF(S0) \subseteq P)$$

Proof. It follows directly from the definition of $\Box\ P$ and reachability. $\Box$

We next discuss the meaning of the main temporal logic formulas used in this Thesis. These are $\diamond\Box\ P$, $\Box\,(P \Rightarrow \diamond\ Q)$ and $\Box\,(P \Rightarrow \Box\ P)$.

We start discussing $\diamond\Box\ P$. Informally, $\diamond\Box\ P$ holds for an execution if there exists a time t in the execution such that the predicate P holds after time t. Formally, given an execution π, $\diamond\Box\ P$ holds for π if $\exists t' \geq 0$ such that $\forall t \geq t'$, $P(\pi(t))$. In Figure 2.13, we present an execution where $\diamond\Box\ P$ holds. Similarly, the temporal logic formula $\diamond\Box\ P$ holds for an automaton $\mathcal{A}$ if it holds for

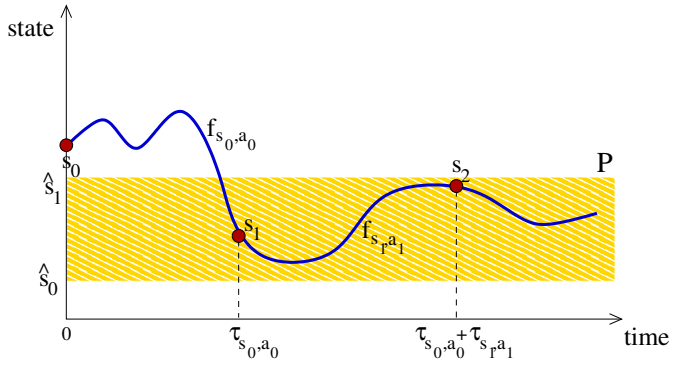


Figure 2.13: Execution fragment where $\Diamond\Box\ P$ holds; predicate P holds in s if $s \in [\hat{s}_0, \hat{s}_1]$. The shaded area represents the region where predicate P holds.

all its executions. We next formally derive the meaning of this formula:

$$\begin{aligned}
\Diamond\Box\ P &\equiv (\forall \pi \in \mathcal{E}_{S0,\infty}\ :\ \Diamond_\pi(\Box\ P)) \\
&\equiv (\forall \pi \in \mathcal{E}_{S0,\infty}\ :\ \exists \hat{\pi} \in \mathit{Suffix}(\pi)\ :\ \Box_\pi\ P) \\
&\equiv (\forall \pi \in \mathcal{E}_{S0,\infty}\ :\ \exists \hat{\pi} \in \mathit{Suffix}(\pi)\ :\ \forall \bar{\pi} \in \mathit{Suffix}(\hat{\pi})\ : P(\bar{\pi}.fstate)) \\
&\equiv (\forall \pi : \mathbb{R}_{\geq 0} \rightarrow S, \pi(0) \in S0\ :\ \exists t' \geq 0\ :\ \forall t \geq t'\ : P(\pi(t)))
\end{aligned}$$

We next discuss the meaning of the temporal logic formula $\Box\,(P \Rightarrow \Diamond\ Q)$. Informally, the formula $\Box\,(P \Rightarrow \Diamond\ Q)$ holds for an execution fragment, if for all states in the execution satisfying P there exists a state later in the execution satisfying Q. Formally, given an execution π, $\Box\,(P \Rightarrow \Diamond\ Q)$ holds for π if for all $t \geq 0$, such that $P(\pi(t))$, there exists $t' \geq t$, such that $Q(\pi(t'))$. In Figure 2.14, we present an execution where $\Box\,(P \Rightarrow \Diamond\ Q)$ holds. Similarly, the temporal logic formula $\Box\,(P \Rightarrow \Diamond\ Q)$ holds for an automaton $\mathcal{A}$ if it holds for all its executions. We next formally derive the meaning of this formula:

$$\begin{aligned}
\Box\,(P \Rightarrow \Diamond\ Q) &\equiv (\forall \pi \in \mathcal{E}_{S0,\infty}\ :\ \Box_\pi\ (P \Rightarrow \Diamond\ Q)) \\
&\equiv (\forall \pi \in \mathcal{E}_{S0,\infty}\ :\ \forall \hat{\pi} \in \mathit{Suffix}(\pi)\ :\ (P \Rightarrow \Diamond\ Q)_{\hat{\pi}}) \\
&\equiv (\forall \pi \in \mathcal{E}_{S0,\infty}\ :\ \forall \hat{\pi} \in \mathit{Suffix}(\pi)\ :\ (P_{\hat{\pi}} \Rightarrow \Diamond_{\hat{\pi}}\ Q)) \\
&\equiv (\forall \pi \in \mathcal{E}_{S0,\infty}\ :\ \forall \hat{\pi} \in \mathit{Suffix}(\pi)\ :\ (P(\hat{\pi}.fstate) \Rightarrow \exists \bar{\pi} \in \mathit{Suffix}(\hat{\pi})\ :\ Q(\bar{\pi}.fstate))) \\
&\equiv (\forall \pi : \mathbb{R}_{\geq 0} \rightarrow S, \pi(0) \in S0\ :\ \forall t \geq 0\ :\ (P(\pi(t)) \Rightarrow \exists t' \geq t\ :\ Q(\pi(t'))))
\end{aligned}$$

We finally discuss the meaning of temporal formula $\Box\,(P \Rightarrow \Box\ P)$. Informally, the formula $\Box\,(P \Rightarrow \Box\ P)$ holds for an execution fragment, if for all states in the execution fragment if P holds in the state then P continues to hold forever. Formally, given an execution π, $\Box\,(P \Rightarrow \Box\ P)$ holds π if for all $t \geq 0$, if $P(\pi(t))$ holds, then $\forall t' \geq t$, $P(\pi(t'))$ holds. In Figure 2.13 and Figure 2.12, we

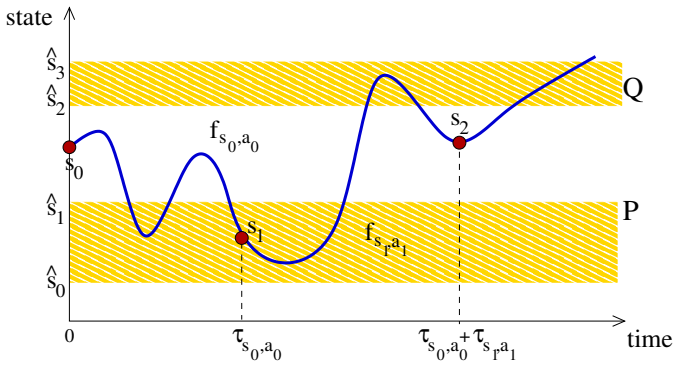


Figure 2.14: Execution fragment where $\Box\,(P \Rightarrow \Diamond\, Q)$ holds; predicate P holds in s if $s \in [\hat{s}_0, \hat{s}_1]$ and predicate Q holds in s if $s \in [\hat{s}_2, \hat{s}_3]$. The shaded areas represent the regions where predicate P and predicate Q hold.

present two execution where $\Box\,(P \Rightarrow \Box\, P)$ holds. Similarly, the temporal logic formula $\Box\,(P \Rightarrow \Box\, P)$ holds for an automaton $\mathcal{A}$ if it holds for all its executions. We next formally derive the meaning of this formula:

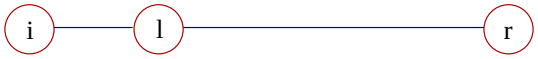
$$
\begin{aligned}
\Box\,(P \Rightarrow \Box\, P) \quad &\equiv \quad (\forall \pi \in \mathcal{E}_{S0,\infty} \;:\; \Box_\pi\; (P \Rightarrow \Box\, P)) \\
&\equiv \quad (\forall \pi \in \mathcal{E}_{S0,\infty} \;:\; \forall \hat{\pi} \in \mathit{Suffix}(\pi) \;:\; (P \Rightarrow \Box\, P)_{\hat{\pi}}) \\
&\equiv \quad (\forall \pi \in \mathcal{E}_{S0,\infty} \;:\; \forall \hat{\pi} \in \mathit{Suffix}(\pi) \;:\; (P_{\hat{\pi}} \Rightarrow \Box_{\hat{\pi}}\, P)) \\
&\equiv \quad (\forall \pi \in \mathcal{E}_{S0,\infty} \;:\; \forall \hat{\pi} \in \mathit{Suffix}(\pi) \;:\; (P(\hat{\pi}.fstate) \Rightarrow \forall \bar{\pi} \in \mathit{Suffix}(\hat{\pi}) \;:\; P(\bar{\pi}.fstate))) \\
&\equiv \quad (\forall \pi : \mathbb{R}_{\geq 0} \rightarrow S, \pi(0) \in S0 \;:\; \forall t \geq 0 \;:\; (P(\pi(t)) \Rightarrow \forall t' \geq t \;:\; P(\pi(t'))))
\end{aligned}
$$

2.2.4 Line-Up automaton with dynamics

In this Section, we present the Line-Up multi-agent system with dynamics and model it using the automaton with timed actions framework. This multi-agent system generalizes the Line-Up multi-agent system presented in Section 2.1.2. The system consists of $N+1$ agents whose goal is to converge to a configuration where agents are located, in order, at equidistant points on a straight line with extremes given by the initial positions of agent 0 and agent N.

In the Line-Up multi-agent system with dynamics, the updating rule of agent i, presented in Figure 2.15, is defined as follows. Agent i chooses two other agents l, r with $l < i < r$, it computes its new position x' using the formula defined in Equation 2.1, and continuously moves from its current location to its destination position x'. We suppose that agent i reaches x' in finite time. For example, agent i can move towards x' with constant velocity or it can instantaneously jump from its current position to its newly computed one. The jump dynamics models the protocol of the Line-Up multi-agent system presented in Section 2.1.2.

The automaton with timed actions modeling the Line-Up multi-agent system with dynamics has



(a) Positions of agents l, i and r before executing the updating rule.

(b) Positions of agents l, i and r while agent i executes the updating rule. Agent i moves towards its new computed position.

(c) Positions of agents l, i and r while agent i executes the updating rule. Agent i moves towards its newly computed position.

(d) Positions of agents l, i and r after executing the updating rule.

Figure 2.15: Agent i executes the updating rule and moves from its current position to the newly computed one.

the following structure:

- $S = \mathbb{R}^{N+1}$, since the state space of each agent is $\mathbb{R}$.
- $S0 = \{s0\}$, with $s0 = x0$,
- $A = \{A_i\}_{i \in I}$ with $A_i = \{\widehat{avg}_{l,i,r}\}_{l<i<r}$,
- $E : S \times A \to true$
- $T : S \times A \to (\mathbb{T} \to S)$, defined as $\forall s \in S$, $\forall a \equiv \widehat{avg}_{l,i,r} \in A$,

$$\begin{aligned} \forall j \neq i, \forall t \leq \tau_{s,a} \quad &: \quad (T(s,a)(t))(j) = s(j) \\ \forall t \leq \tau_{s,a} \quad &: \quad (T(s,a)(t))(i) = f_{s,a}(t) \end{aligned}$$

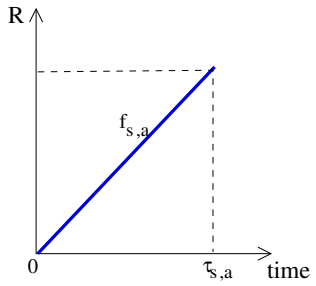
with $f_{s,a} : [0, \tau_{s,a}] \to S$ having

$$\begin{aligned} f_{s,a}(0) &= s(i) \\ f_{s,a}(\tau_{s,a}) &= \frac{r-i}{r-l} s(l) + \frac{i-l}{r-l} s(r) \end{aligned}$$

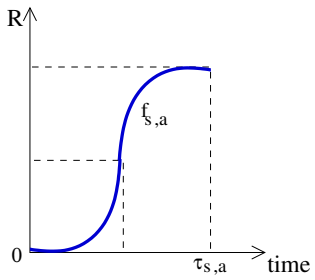
In this automaton, agents 0 and N are stationary. The remaining agents move toward their destination positions computed using the formula in Equation 2.1. The function $f_{s,a}$ models the dynamics of agent i. While agent i moves, the other agents are stationary. In this automaton, we do not model concurrent agent movement. Function $f_{s,a}$ of agent i depends on the pre-state of the action;

this means that agent i can have different dynamics when starting from different states. In general, in our model, different agents can have different dynamics and the duration of their actions can be different as well. Examples of feasible dynamics for agent i are presented in Figure 2.16(a) and Figure 2.16(b).

The goal state of the automaton is the same goal state of the automaton presented in Section 2.1.2. We refer to Equation 2.3 for its definition.



(a) Agent i moves with constant velocity for $\tau_{s,a}$ time units.



(b) Agent i moves with constant acceleration for $\frac{\tau_{s,a}}{2}$ time units, then it moves with constant deceleration for $\frac{\tau_{s,a}}{2}$ time units.

Figure 2.16: Feasible dynamics for agent i when executing action $a = \widehat{avg}_{l,i,r}$ in state s with $s(j) = 0$, $\forall j \in \{0, \ldots, N\}$.

2.3 Fairness

In this Section, we present the fairness criterion used in this Thesis. This is a new fairness criterion, that we have formally introduced in [14]. Before describing this criterion, we introduce an example that motivates our definition.

Consider the multi-agent system presented in Figure 2.17. This system consists of three agents, u, v, w and three bidirectional communication channels, one for each pair of agents. Suppose that

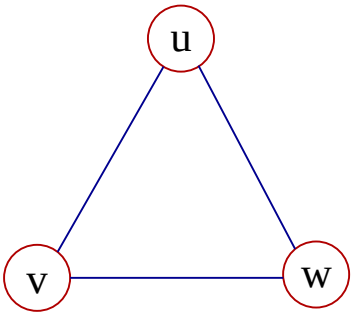


Figure 2.17: A multi-agent system consisting of three agents. Circles represent agents, lines represent communication channels.

the goal of the system is to compute some function of its initial state. For example, agents may want to compute the average or the minimum of their initial values. If any agent is permanently partitioned from the other two, the system will be not able to compute the desired quantity. Hence, feasible infinite executions of the system must satisfy the requirement that there are no permanent partitions in the system. This requirement can be encoded as follows. Denote by

(a) $a_{u,v}$ the action resulting in the interaction between u and v,

(b) $a_{u,w}$ the action resulting in the interaction between u and w, and

(c) $a_{v,w}$ the action resulting in the interaction between v and w,

and by

(d) F_u the set of actions $\{a_{u,v}, a_{u,w}\}$,

(e) F_v the set of actions $\{a_{u,v}, a_{v,w}\}$,

(f) F_w the set of actions $\{a_{u,w}, a_{v,w}\}$.

The no-permanently partitioned requirement on u holds if in any infinite execution, actions from the set F_u occur infinitely often; for example, $a_{u,v}$ executes infinitely often or $a_{u,w}$ executes infinitely often. Similar conditions hold for agents v, w. Hence, the no-permanent partition requirement holds if in any infinite execution, actions from the sets F_u, F_v, F_w occur infinitely often. An example of feasible execution is an execution where $a_{u,v}, a_{u,w}$ are executed infinitely often, but $a_{v,w}$ is never executed.

The no-permanent partition criterion cannot be specified using *weak fairness*. Under weak fairness, we can only model executions where each action is executed infinitely often. Weak fairness criterion is too strong. It rules out feasible executions, such as the one mentioned before. We need a fairness criterion weaker than weak fairness. In this new fairness criterion, we require fairness to be defined with respect to set of actions, instead of single actions.

We next formally define this concept of fairness in the context of automaton with timed actions $\mathcal{A}$.

Definition 14. *Given an automaton $\mathcal{A}$ and a family of actions $\mathcal{F} = \{F\}$, with $F \subseteq A$, an infinite execution $s_0, a_1, s_1, a_2, \ldots$ is $\mathcal{F}$-fair if for all $F \in \mathcal{F}$, actions in F occur infinitely often in the execution, i.e.,*

$$\forall F \in \mathcal{F},\ \forall m,\ \exists k\ :\ k > m\ :\ a_k \in F \tag{2.4}$$

The family $\mathcal{F}$ is called a fairness condition for $\mathcal{A}$.

This notion of fairness is weaker than weak fairness. It reduces to weak fairness when $\forall F \in \mathcal{F}$, F has cardinality 1. Recalling our previous example, we have that the no-permanent partition

requirement can be encoded as

$$\mathcal{F} = \{\{a_{u,v}, a_{u,w}\}, \{a_{u,v}, a_{v,w}\}, \{a_{u,w}, a_{v,w}\}\}$$

while weak fairness condition can be encoded as

$$\mathcal{F}_{weak} = \{\{a_{u,v}\}, \{a_{u,w}\}, \{a_{v,w}\}\}$$

In the Line-Up automaton, presented in Section 2.1.2, and its generalization with dynamics, presented in Section 2.2.4, fair executions consist of infinite sequences of actions where each agent is not permanently partitioned. For this two automata, the set $F_i = A_i$ for all i and $\mathbb{F} = \{F_i\}_{0<i<N}$.

2.4 Automata in the Presence of Exogenous Inputs

In this Section, we present a model for formally describing multi-agent systems in the presence of exogenous inputs. Exogenous inputs have been introduced in Chapter 1. Our goal is to investigate properties of the exogenous inputs that ensure robustness of the overall system.

These inputs help modeling quantities of the system that vary with time due to external or internal conditions. For example, consider a multi-agent system where agents store temperature readings at several locations. Sensors produce new readings at regular time intervals. As a consequence, the temperatures stored in the agents need to be updated at regular intervals of time. We model these updates as external inputs.

We next discuss another example. Consider a generalization of the Line-Up multi-agent system where agent 0 and N can move arbitrarily. The goal of the agents is to form and maintain an equispaced spatial configuration that changes with time. In this example, the positions of agents 0 and N, stored in the system, need to be updated periodically. We model their updates as external inputs.

These exogenous inputs can be modeled as discrete or timed actions. For example, the temperature update can be modeled as a discrete action while the evolution of the positions of agent 0 and N in the Line-Up multi-agent system can be modeled using a timed action. The state space of the system with and without exogenous inputs are the same. Each action of the exogenous inputs has an enabling condition and a transition function, describing the behaviour of the input. For example, an action modeling the movements of agents 0 and N in the Line-Up system can be the action *move* defined as follows. It operates over the set of $\mathbb{R}^{N+1}$; it is always enabled, i.e.

$$\forall s \in \mathbb{R}^{N+1} \ : \ E(s, move) = true \tag{2.5}$$

When executed, agents 0 and N may, for example, evolve their position linearly for an interval of time Δ:

$$\forall s \in \mathbb{R}^{N+1}, \forall t \in \Delta \ : \ T(s, move)(t) = (s + v \cdot t) \tag{2.6}$$

where v is a vector of real numbers, modeling velocities, with $v(i) \neq 0$ for $i \in \{0, N\}$ and 0 otherwise. Action *move* models an exogenous input where agents 0 and N move at constant independent velocities.

In this Thesis, we model the set of exogenous input and the system in the presence of exogenous inputs as automata with timed actions. We denote by $\mathcal{A} = (S, S0, A, E, T)$ the automaton model for the system in the absence of exogenous inputs. We, first, model the set of exogenous inputs as an automaton. We denote this automaton by $\bar{\mathcal{A}}$. This automaton has the same state space and initial conditions of $\mathcal{A}$. It defines a set of actions $\bar{A}$ that is disjoint from the set of actions of $\mathcal{A}$. This set of actions has an enabling predicate $\bar{E}$ and transition function $\bar{T}$. Formally, the automaton of the exogenous inputs $\bar{\mathcal{A}}$ is as follows:

Definition 15. *Given $\mathcal{A} = (S, S0, A, E, T)$, the automaton of the exogenous inputs $\bar{\mathcal{A}}$ is the tuple $(S, S0, \bar{A}, \bar{E}, \bar{T})$, with $\bar{A} \cap A = \emptyset$.*

For example, the automaton with timed actions modeling the exogenous input of the Line-Up system is $\bar{\mathcal{A}}_{Line-Up} = (\mathbb{R}^{N+1}, \{x0\}, \{move\}, E, T)$ where $x0$ is the vector of initial positions, E and T have been defined in Equation 2.5 and Equation 2.6.

Giving this automaton, we can define the automaton modeling a system in presence of exogenous inputs. This automaton, called exogenous automaton, is a composition of $\mathcal{A}$ and $\bar{\mathcal{A}}$ as follows.

Definition 16. *Given $\mathcal{A} = (S, S0, A, E, T)$ and $\bar{\mathcal{A}} = (S, S0, \bar{A}, \bar{E}, \bar{T})$ with $A \cap \bar{A} = \emptyset$, the exogenous automaton $\mathcal{A}_{exog} = (S, S0, A_{exog}, E_{exog}, T_{exog})$ where*

- *$A_{exog} = A \cup \bar{A}$,*
- *the enabling predicate E_{exog} defined as, $\forall s \in S, a \in A_{exog}$,*

$$E_{exog}(s, a) = \begin{cases} E(s, a) & a \in A \\ \bar{E}(s, a) & a \in \bar{A} \end{cases}$$

- *the transition function T_{exog} defined as, $\forall s \in S, a \in A_{exog}, \forall t \in [0, \tau_{s,a}]$,*

$$T_{exog}(s, a)(t) = \begin{cases} T(s, a)(t) & a \in A \\ \bar{T}(s, a)(t) & a \in \bar{A} \end{cases}$$

This automaton operates on the same state space of the automaton in the absence of exogenous inputs and executes actions from both $\mathcal{A}$ and $\bar{\mathcal{A}}$.

Executions of $\mathcal{A}_{exog}$ can be projected on $\mathcal{A}$ and $\bar{\mathcal{A}}$. In the case of $\mathcal{A}$, the projection is obtained by removing all actions of the automaton of the exogenous input $\bar{\mathcal{A}}$ and executing the remaining actions starting from the same initial state. Similarly, the projection on $\bar{\mathcal{A}}$ is obtained by considering only the actions of $\bar{\mathcal{A}}$.

2.5 Discussion

In this Section, we discuss the automaton model with timed actions and relate it to other models for timed systems.

In Section 2.2, we have introduced the automaton model with timed actions. This model extends the automaton model; it allows for actions with different time durations. We have defined the notion of executions and reachability for this model, and we have defined the meaning of the temporal operator $\Box$ and $\Diamond$ for automata with timed actions. We have also modeled the Line-Up multi-agent systems with explicit dynamics using this framework.

The automaton with timed actions models action-deterministic systems, i.e. systems where the behaviour of the action is determined by the action and the pre-state of the action. We can easily extend the definition and model action nondeterministic systems. In this case, given a state and an action, the execution of the action can have different behaviors and/or different time durations.

This model extends the ideas discussed in [13], where the author informally presents a model for continuous systems and discusses the structure of a logic for reasoning about them. It can be used for modeling system where some portions of the system have discrete behavior and other portions of the systems have continuous behaviour, i.e. hybrid systems.

As an example of hybrid system, we may consider the single agent timed system presented in Example 1.1 of [32]. This system models the controller of the temperature of a room. The system consists of a thermostat controlling the temperature of a room. When the heater is off, the temperature of the room falls according to the differential equation d_1. When the heater is on the temperature rises according to the differential equation d_2. The heater may go on as soon as the temperature falls below t_{on} degree and may go off as soon as the temperature rise above t_{off}. We assume $t_{on} < t_{off}$. We assume that the temperature of the room is always in the interval $[t_{\min}, t_{\max}]$.

A possible nondeterministic automaton with timed actions modeling this system is the following. Its state space S consists of two variables: x storing the temperature of the room with domain $[t_{\min}, t_{\max}]$, and *heater* storing the state of the heater. The variable *heater* is a binary (discrete) variable; its value is 1, if the heater is on and 0 if the heater is off. The set of actions A of the system consists of three actions *heater_on*, *heater_off* and *update*. The *heater_on* action toggles the *heater* variable from 0 (off) to 1 (on). It is enabled in state $s \in S$ if $s.x < t_{on}$. The *heater_off* action toggles the *heater* variable from 1 (on) to 0 (off). It is enabled in state $s \in S$ if $s.x > t_{off}$.

The *update* action evolves the temperature and it is always enabled. When executed in state s, if the heater is on, i.e. $s.heater = 1$, it evolves x according to the differential equation d_2 for some arbitrary time; instead, if the heater is off, i.e. $s.heater = 0$, it evolves x according to the differential equation d_1 for some arbitrary time.

This model is different from other models for hybrid systems, such as hybrid automata [32] and hybrid I/O [41] and timed I/O automata [36] models. Our model is similar to the [41, 36]. In [41] and its extension [36], the authors distinguish between discrete actions and actions with time duration, called trajectories. Executions are alternating sequences of actions and trajectories. They assume that the set of feasible trajectories is closed under prefix, suffix and concatenation. This implies that given a state, the system can execute the complete action or any prefix of it. They need these assumptions to model composed systems where any trajectory of any component automaton may be interrupted at any time by a discrete transition of another component. In our model, actions cannot be interrupted and we do not require the action set to be closed under prefix, suffix and concatenation. For example, these automata models would not be able to model the behaviour of the multi-agent system described in Section 2.2.4, The main limitation is that agents can stop before reaching their locations, since trajectories are closed under prefix. Hence, these systems can model executions where portions of the evolutions of the states of different agents may interleave. This implies that when an agent moves, its new location may be computed using locations of other agents that have not been computed by the algorithm.

This model cannot be used for time-critical systems [1]. In a future continuation of this work, we would like to extend the automaton and the meaning of the temporal operators to time-critical systems.

Chapter 3

Stability and Convergence Properties of Automata

In this Chapter, we discuss properties of states of automata with timed actions. We use these properties to prove correctness of multi-agent systems.

In Section 3.1 we present the concept of equilibrium state for automata with timed actions. In Section 3.2 we present the concept of Lyapunov function and level sets; we use these concepts for proving properties of equilibrium states. In Section 3.3 we define the notion of stability for equilibrium states and present conditions for proving it while in Section 3.4, we define the notion of asymptotically stability and present conditions for proving it. In Section 3.5 we discuss properties of states in the presence of exogenous inputs. Finally, in Section 3.6 we relate the definitions and theorems of this Chapter to other works on stability and convergence.

Throughout this Chapter, $\mathcal{A}$ denotes the automaton with timed actions $(S, S0, A, E, T)$, and $\hat{s}$ a state of $\mathcal{A}$, i.e., $\hat{s} \in S$.

3.1 Equilibria in Automata

In this Section, we define the concept of equilibrium state of automata. Informally, an equilibrium state is a stationary configuration of the system with respect to its action set. Formally,

Definition 17. *$\hat{s}$ is an equilibrium state of $\mathcal{A}$ if $RF(\hat{s}) = \{\hat{s}\}$.*

An equilibrium state is a *trapping state* of the system. The definition of equilibrium state can be generalized to set of states.

Definition 18. *$\hat{S} \subseteq S$ is a set of equilibrium states of $\mathcal{A}$ if $RF(\hat{S}) = \hat{S}$.*

The set $RF(\hat{S})$ denotes the set of all reachable states from $\hat{S}$. We refer to Section 2.2.2 for the definition of $RF(\hat{S})$.

We next discuss two specific examples. We first present the set of equilibrium states of the Line-Up automaton defined in Section 2.1.2. This is an example of discrete automaton. We then discuss the equilibrium states of the Line-Up automaton with dynamics presented in Section 2.2.4. This is an example of automaton with timed actions.

We consider the Line-Up automaton defined in Section 2.1.2. We show that the state defined in Equation 2.3 is an equilibrium state of this automaton. We briefly recall the multi-agent system. This system consists of $N+1$ agents and the goal of the agents is to form a configuration where agents are located, in order, at equidistant points on a straight line. The end points of this line are the initial positions of agent 0 and N. We refer to Figure 2.1(b) for a pictorial representation of the final configuration state.

The goal state of the system is:

$$\hat{s}(i) = \frac{N-i}{N} s0(0) + \frac{i}{N} s0(N) \;\; \forall i \in \{0, \ldots, N\}$$

where $s0$ is the initial state of the automaton. We next show that $\hat{s}$ is an equilibrium state.

Lemma 2. *$\hat{s}$ is an equilibrium state of the Line-Up automaton.*

Proof. Our goal is to show that $\forall a \in A$, with $a = avg_{l,i,r}$ and $l < i < r$, we have that

$$\hat{s} = T(\hat{s}, avg_{l,i,r})$$

Consider an arbitrary action $a = avg_{l,i,r}$, with $l < i < r$. By construction, we have that $\forall j \neq i$,

$$\hat{s}(j) = T(\hat{s}, avg_{l,i,r})(j)$$

This is because action $avg_{l,i,r}$ modifies only component i of the state.

We next prove that $\hat{s}(i) = T(\hat{s}, avg_{l,i,r})(i)$. By algebra manipulation and definition of $\hat{s}$, the following chain of inequalities holds

$$\begin{aligned}
T(\hat{s}, avg_{l,i,r})(i) &= \frac{r-i}{r-l}\hat{s}(l) + \frac{i-l}{r-l}\hat{s}(r) \\
&= \frac{r-i}{r-l}\left(\frac{N-l}{N} s0(0) + \frac{l}{N} s0(N)\right) + \\
&\quad \frac{i-l}{r-l}\left(\frac{N-r}{N} s0(0) + \frac{r}{N} s0(N)\right) \\
&= \frac{(r-l)(N-i)}{(r-l)N} s0(0) + \frac{i(r-l)}{(r-l)N} s0(N) \\
&= \frac{N-i}{N} s0(0) + \frac{i}{N} s0(N) \\
&= \hat{s}(i)
\end{aligned}$$

□

Furthermore, any equilibrium state of the Line-Up automaton is a straight equidistance line.

Lemma 3. *$\bar{s} \in S$ is an equilibrium state of the Line-Up automaton if and only if $\forall i \leq N$*

$$\bar{s}(i) = \frac{N-i}{N}\ \bar{s}(0) + \frac{i}{N}\ \bar{s}(N)$$

Proof. Assume that $\bar{s} \in S$ is not straight equidistance line, i.e. $\exists j$, with $0 < j < N$, such that

$$\bar{s}(j) \neq \frac{N-j}{N}\ \bar{s}(0) + \frac{j}{N}\ \bar{s}(N)$$

We prove that $\bar{s}$ is not an equilibrium state. We next show that $\exists l, r \in \{0, 1, \ldots, N\}$, such that $T(\bar{s}, avg_{l,j,r}) \neq \bar{s}$.

Consider the action $avg_{0,j,N}$. When executing this action, the state of agent j is modified as follows:

$$T(\bar{s}, avg_{0,j,N})(j) = \frac{N-j}{N}\ \bar{s}(0) + \frac{j}{N}\ \bar{s}(N)$$

Putting all together, we have that

$$\begin{aligned} \bar{s}(j) \quad &\neq \quad \frac{N-j}{N}\ \bar{s}(0) + \frac{j}{N}\ \bar{s}(N) \\ &= \quad T(\bar{s}, avg_{0,j,N})(j) \end{aligned}$$

Hence, $\bar{s}$ is not an equilibrium state.

Assume, instead, that $\bar{s} \in S$ is a straight equidistance line, i.e. $\forall i \leq N$,

$$\bar{s}(i) = \frac{N-i}{N}\ \bar{s}(0) + \frac{i}{N}\ \bar{s}(N)$$

The proof of this case is similar to the proof of Lemma 2 and not reported. □

We next consider the Line-Up automaton with dynamics introduced in Section 2.2.4. We prove that state $\hat{s}$ defined in Equation 2.3 is an equilibrium state of the automaton under specific assumptions on the dynamics of the agents. Similarly to Lemma 2, we can show that $\forall a \in A$, state $T(\hat{s}, a)(\tau_{\hat{s},a})$ is equal to $\hat{s}$. In order to prove that $\hat{s}$ is an equilibrium state, we require that $T(\hat{s}, a)(t)$ is equal to $\hat{s}$, for all $t \in (0, \tau_{\hat{s},a})$. This means that we require agents to be stationary when executing actions in state $\hat{s}$. For example, $\hat{s}$ is an equilibrium state if the dynamics of the agents satisfy the following condition: $\forall s \in S$, $\forall a \in A$, $\forall t \in (0, \tau_{s,a})$, $T(\hat{s}, a)(t)$ is a convex combination of s and $T(\hat{s}, a)(\tau_{s,a})$. This is because any convex combination of $\hat{s}$ and $T(\hat{s}, a)(\tau_{\hat{s},a})$ remains in $\hat{s}$, since $T(\hat{s}, a)(\tau_{\hat{s},a}) = \hat{s}$.

Lemma 4. *$\hat{s}$ is an equilibrium state of the Line-Up automaton with dynamics if and only if*

$$\forall a \in A, \forall t \in (0, \tau_{\hat{s},a}) \ : \ f_{\hat{s},a}(t) = f_{\hat{s},a}(\tau_{\hat{s},a}) \tag{3.1}$$

Proof. By construction of the action set, we have that

$$\forall a = \widehat{avg}_{l,i,r} \in A \ : \ f_{\hat{s},a}(\tau_{\hat{s},a}) = \hat{s}(i) \tag{3.2}$$

This is because,

$$\begin{aligned}
f_{\hat{s},a}(\tau_{\hat{s},a}) &= \frac{r-i}{r-l}\hat{s}(l) + \frac{i-l}{r-l}\hat{s}(r) \\
&= \frac{r-i}{r-l}\left(\frac{N-l}{N}s0(0) + \frac{l}{N}s0(N)\right) + \\
&\quad \frac{i-l}{r-l}\left(\frac{N-r}{N}s0(0) + \frac{r}{N}s0(N)\right) \\
&= \frac{(r-l)(N-i)}{(r-l)N}s0(0) + \frac{i(r-l)}{(r-l)N}s0(N) \\
&= \frac{N-i}{N}s0(0) + \frac{i}{N}s0(N) \\
&= \hat{s}(i)
\end{aligned}$$

Hence, by construction of the action set, we have that

$$\forall a \in A \ : \ T(\hat{s}, a)(\tau_{\hat{s},a}) = \hat{s}$$

We first assume that the conditions defined in Equation 3.1 hold. Our goal is to show that

$$\forall a = \widehat{avg}_{l,i,r} \in A, \ \forall t \in (0, \tau_{\hat{s},a}] \ : \ f_{\hat{s},a}(t) = \hat{s}(i)$$

since by construction we require that $\hat{s}$ is the only reachable state from $\hat{s}$ and agent i is the only agent modified by the execution of action $a = \widehat{avg}_{l,i,r}$.

Consider an arbitrary $a = \widehat{avg}_{l,i,r} \in A$. From Equation 3.2, we have that $f_{\hat{s},a}(\tau_{\hat{s},a})$ is equal to $\hat{s}(i)$. Using conditions defined in Equation 3.1, we derive that $f_{\hat{s},a}(t) = \hat{s}(i)$ for all $t \in (0, \tau_{\hat{s},a})$.

We then assume that $\hat{s}$ is an equilibrium state. Our goal is to show that the conditions in Equation 3.1 hold.

Consider an arbitrary $a = \widehat{avg}_{l,i,r} \in A$. By definition of equilibrium state, we have that

$$\forall t \in (0, \tau_{\hat{s},a}] \ : \ T(\hat{s}, a)(t) = \hat{s}$$

By construction of action a, we have that

$$\forall t \in (0, \tau_{\hat{s},a}] \ : \ (T(\hat{s}, a)(t))(i) = f_{\hat{s},a}(t)$$

Combining previous equations with Equation 3.2, we have that Equation 3.1 holds. □

We next prove that any equilibrium state of the Line-Up automaton with dynamics can be represented as a equi-spaced straight line.

Lemma 5. *$\bar{s} \in S$ is an equilibrium state of the Line-Up automaton with dynamics if and only if*

$$\forall i \leq N \ : \ \bar{s}(i) = \frac{N-i}{N} \ \bar{s}(0) + \frac{i}{N} \ \bar{s}(N)$$

and

$$\forall a \in A, \forall t \in (0, \tau_{\hat{s},a}) \ : \ f_{\hat{s},a}(t) = f_{\hat{s},a}(\tau_{\hat{s},a})$$

Proof. The proof is similar to the proof of Lemma 3 and Lemma 4. Therefore it is not reported. □

3.2 Lyapunov Function and Level Sets

In this Section, we discuss Lyapunov functions [43, 39] for automata. This tool is used for proving stability and convergence of equilibrium states. A Lyapunov function for $\mathcal{A}$ is a mapping $V : S \to \mathbb{P}$, where $\mathbb{P}$ is a totally ordered set. The set $\mathbb{P}$ may be the set of non-negative reals ($\mathbb{R}_{\geq 0}$), the set of natural numbers ($\mathbb{N}$) or a set of tuples, where elements of the tuples are compared using lexicographic order. For example, a Lyapunov function for the Line-Up automaton with dynamics, discussed in Section 2.2.4, can be the function $\hat{V} : S \to \mathbb{R}_{\geq 0}$ defined as follows: $\forall s \in S$

$$\hat{V}(s) = \sum_{i=0}^{N} (\hat{s}(i) - s(i))^2$$

This function maps a state s to a non-negative real number, that represents the square of the distance of the state from the equilibrium state $\hat{s}$. This metric is computed as the sum of squares of the distances of the single agents from their corresponding goal positions.

We next introduce the concept of level sets of the Lyapunov function V. We will use this concept throughout the thesis. We define the level sets of V, as follows.

Definition 19. *For each $p \in \mathbb{P}$, the level set L_p is defined as*

$$L_p = \{s \in S : V(s) \leq p\}$$

This set includes all states mapped by V to a value upper bounded by p.

For example, in the case of the Line-Up automaton with dynamics, we have that this family of level sets denoted by $\{\hat{L}_p\}_{p\in\mathbb{R}_{\geq 0}}$ is defined as follows. The level set $\hat{L}_0$ of $\hat{V}$ is equal to the set consisting of the equilibrium state $\hat{s}$, i.e. $\hat{L}_0 = \{\hat{s}\}$. For all $p > 0$, $\hat{L}_p$ includes all states of S such that $\hat{V}(s) \leq p$.

The family of level sets $\{L_p\}_{p\in\mathbb{P}}$ of the Lyapunov function V is *monotonic*, by construction, i.e.,

$$\forall p, q \in \mathbb{P},\ q < p,\ L_q \subseteq L_p$$

We say that $\{L_p\}_{p\in\mathbb{P}}$ is *strictly monotonic*, if each level set L_q of the family is strictly contained in all level sets L_p with $q < p$, i.e.,

$$\forall p, q \in \mathbb{P},\ q < p,\ L_q \subsetneq L_p$$

For example, in the case of the Line-Up automaton with dynamics with the given Lyapunov function, it is easy to show that the family of level sets of $\hat{V}$ is strictly monotonic. This is because, for all $p \in \mathbb{R}_{\geq 0}$, we can construct a state having distance from $\hat{s}$ equal to p.

In the special case of multi-agent systems, we introduce the notion of set of states in *conjunctive form*. In the case of multi-agent systems, the state space S of a system is defined as the Cartesian product of the state spaces of its agents, i.e. $S = S_1 \times S_2 \times \ldots S_N$ with S_i being the state space of agent i and N being the total number of agents in the system. We refer to Section 4.1 for a detailed discussion of multi-agent systems. Informally, a set of states Q is in conjunctive form if it can be expressed as the Cartesian product of $Q_1, Q_2, \ldots Q_N$ with $Q_i \subseteq S_i$. Formally,

Definition 20. *A set of states $Q \subseteq S$ is in conjunctive form if $\exists Q_1, Q_2, \ldots Q_N$ with $Q_i \subseteq S_i$ such that*

$$Q = \bigcap_{i=1}^{N} \{s \in S \mid s(i) \in Q_i\}$$

The set of state Q can be expressed as the intersection of sets, where the i-th set restricts the state space of agent i to Q_i, i.e. the i-th set is equal to

$$S_1 \times \ldots \times Q_i \times S_N$$

By construction, the state space S of the system is in conjunctive form: the set Q_i is equal to S_i for all $i = \{1, \ldots, N\}$. Figure 3.1 presents examples of sets in conjunctive form and sets that are not in conjunctive form.

For example, in the case of the Line-Up automaton with dynamics, we can define the following Lyapunov function and family of level sets in conjunctive form. The Lyapunov function $\bar{V} : S \rightarrow$

$\mathbb{R}_{\geq 0}^{N+1}$ maps each state of the automaton into a tuple of non-negative real numbers such that $\forall s \in S$

$$\bar{V}(s) = \big(\bar{V}_0(s(0)), \ldots, \bar{V}_i(s(i)), \ldots, \bar{V}_N(s(N))\big)$$

where $\bar{V}_i(s(i)) = (s(i) - \hat{s}(i))^2$. This function maps state s into a tuple of $N+1$ elements, where the i-th entry stores the distance between the current position of agent i and its goal position. Given $p = (p_0, \ldots, p_i, \ldots, p_N)$, we have that the level set $\bar{L}_p$ of $\bar{V}$ is defined as

$$\bar{L}_p = \{s' \in S \mid \forall i \in \{0, \ldots, N\} \ : \ \bar{V}_i(s'(i)) \leq p_i\}$$

The state $\bar{L}_p$ is in conjunctive form, since it can be written as the intersection of the following sets

$$\bar{L}_p = \bigcap_{i=1}^{N} \{s' \in S \mid \bar{V}_i(s'(i)) \leq p_i\}$$

For all $p \in \mathbb{P}$, we say that L_p is *stable* if any execution that starts in L_p remains in L_p, formally,

Definition 21. *L_p is stable if*

$$RF(L_p) \subseteq L_p$$

The set $RF(L_p)$ denotes the set of all reachable states from L_p. We refer to Section 2.2.2 for its definition. The concept of stability for a level set is pictorially represented in Figure 3.2. For example, in the case of the Line-Up automaton with dynamics, we have that $\hat{L}_0$ is stable, since $\hat{L}_0$ is a set of cardinality one, consisting of an equilibrium state.

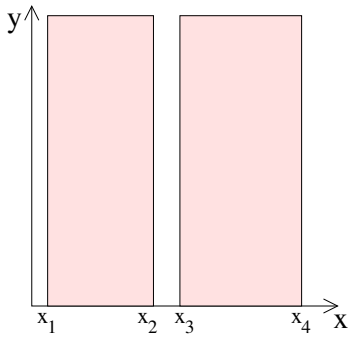
We say that the family $\{L_p\}_{p \in \mathbb{P}}$ is stable if all its level sets are stable. We next derive sufficient and necessary conditions for proving stability of a family of level sets. We require that the execution of any enabled action does not increase the value of the Lyapunov function.

Lemma 6. *$\{L_p\}_{p \in \mathbb{P}}$ is stable if and only if*

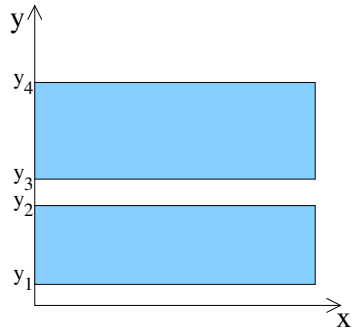
$$\forall s \in S, a \in A \ : \ E(s,a) \implies (\forall t \in \tau_{s,a} \ : \ V(f_{s,a}(t)) \leq V(s))$$

Proof. Suppose that the family of level sets $\{L_p\}_{p \in \mathbb{P}}$ is stable. Consider an arbitrary state $s \in S$ and an arbitrary action $a \in A$, such that $E(s,a)$. By assumption, the level set L_p with $p = V(s)$ is stable. By definition of reachability, we have that

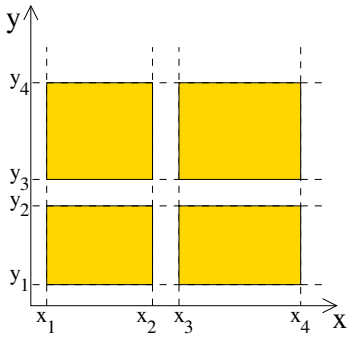
$$\forall t \in \tau_{s,a} \ : \ f_{s,a}(t) \in RF(L_p)$$



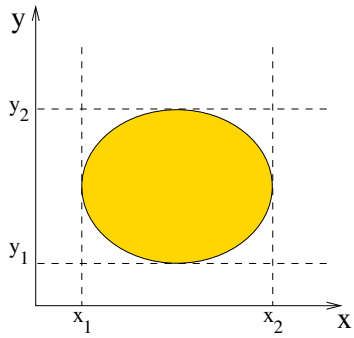
(a) An example of set in conjunctive form. The set is the union of the shaded areas. This set can be represented as $Q_x \times S_y$, where Q_x is the set containing all $x \in S_x$ with $x_1 \leq x \leq x_2$ or $x_3 \leq x \leq x_4$.



(b) An example of set in conjunctive form. The set is the union of the shaded areas. This set can be represented as $S_x \times Q_y$, where Q_y is the set containing all $y \in S_y$ with $y_1 \leq y \leq y_2$ or $y_3 \leq y \leq y_4$.



(c) An example of set in conjunctive form. The set is the union of the shaded areas. This set can be expressed as the intersection of $Q_x \times S_y$ (defined in Figure 3.1(a)) and $S_x \times Q_y$ (defined in Figure 3.1(b)).



(d) An example of set that is not conjunctive. The set is the shaded area.

Figure 3.1: Pictorial representation of examples of sets. In these examples, the system consists of two agents x and y. The state space of agent x is denoted by S_x and the state space of agent y is denoted by S_y. Figure 3.1(a), Figure 3.1(b) and Figure 3.1(c) are examples of conjunctive sets while Figure 3.1(d) is an example of set that is not conjunctive.

and by definition of stability, we have that $RF(L_p) \subseteq L_p$. Hence,

$$\forall t \in \tau_{s,a} \ : \ f_{s,a}(t) \in L_p$$

or equivalently,

$$\forall t \in \tau_{s,a} \ : \ V(f_{s,a}(t)) \leq V(s)$$

and this part of the Lemma follows.

Suppose, instead, that

$$\forall s \in S, a \in A : \ E(s,a) \implies (\forall t \in \tau_{s,a} \ : \ V(f_{s,a}(t)) \leq V(s))$$

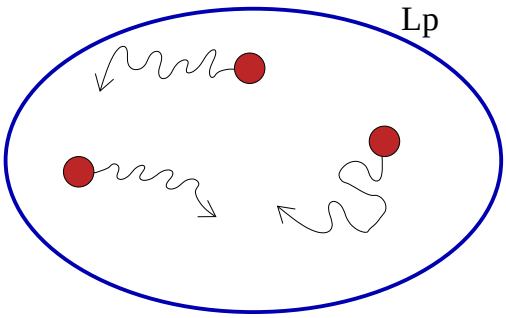


Figure 3.2: Pictorial representation of a stable level set L_p. Dark filled circles contained in the level set L_p represent states of the systems, while arrow lines represent execution fragments. In this Figure, any execution fragment starting from states in L_p remains in L_p.

Consider an arbitrary $p \in \mathbb{P}$. Our goal is to show that L_p is stable, i.e. $RF(L_p) \subseteq L_p$. We next show that

$$\forall s \in L_p,\ \forall s' \in RF(s)\ :\ V(s') \leq V(s) \leq p$$

Consider an arbitrary $s \in L_p$ and $s' \in RF(s)$. By definition of reachability, there exists a execution fragment π such that $\pi.fstate = s$ and $\pi.lstate = s'$. Iterating the assumption along the actions of π, we get that $V(s') \leq V(s)$. Furthermore, since $s \in L_p$ we have that $V(s') \leq V(s) \leq p$ and the Lemma follows. □

We next present a property of stable level sets.

Lemma 7. *If L_p with $p \in \mathbb{P}$ is stable then*

$$\forall U \subseteq L_p\ :\ RF(U) \subseteq L_p$$

Proof. It follows from the definitions of reachability and stability. □

Suppose that the state space S is a metric space, i.e. we can define a function $d : S \times S \to \mathbb{R}_{\geq 0}$ satisfying $\forall s, s', \bar{s} \in S$,

- $d(s, s') = 0$ if and only if $s = s'$,
- $d(s, s') = d(s', s)$ and
- $d(s, s') \leq d(s, \bar{s}) + d(\bar{s}, s')$.

In the case of the Line-Up automaton, a feasible distance function is the two-norm of the distance between pair of states, i.e. the function $d : S \times S \to \mathbb{R}_{\geq 0}$ defines as, $\forall s, s' \in S$

$$d(s, s') = \sum_{i=0}^{N} (s(i) - s'(i))^2$$

Assuming the state space to be a metric space, we can define the concept of ϵ-ball around a state with $\epsilon \geq 0$ as follows

Definition 22. *The ϵ-ball around $s \in S$, denoted by $B_\epsilon(s)$, is*

$$B_\epsilon(s) = \{s' \in S \mid d(s, s') \leq \epsilon\}$$

The ϵ-ball around s consists of the set of states having at most distance ϵ from s. For example, in the case when $\epsilon = 0$, the 0-ball consists only of the state s itself; this follows from the definition of distance function. The concept of ϵ-ball is unrelated to the concept of reachability; states in $B_\epsilon(s)$ do not have to be reachable from s. The concept of ϵ-ball around $s \in S$ can be extended to set of states as follows, given $\hat{S} \subseteq S$:

$$B_\epsilon(\hat{S}) = \{s \in S \mid \exists \hat{s} \in \hat{S} \ : \ d(\hat{s}, s) \leq \epsilon\}$$

3.3 Stable Equilibria

In this Section, we present a formalization of the concept of stable equilibrium points in the context of automata. Informally, $\hat{s} \in S$ is *stable* if every execution fragment that starts close to $\hat{s}$ remains close to $\hat{s}$. Formally,

Definition 23. *$\hat{s}$ is a stable state of $\mathcal{A}$ if $\forall \epsilon > 0$ $\exists \delta > 0$ such that*

$$RF(B_\delta(\hat{s})) \subseteq B_\epsilon(\hat{s})$$

From automaton reachability definition, it follows that $0 < \delta \leq \epsilon$. This is because, by construction, a state is always reachable from itself. From this definition, it follows that any execution fragment starting from a state at distance at most δ from $\hat{s}$ remains within distance ϵ from $\hat{s}$. We refer to Figure 3.3 for a pictorial representation of the definition of stability.

The definition of stability is extended to set of states. Informally, $\hat{S} \subseteq S$ is *stable* if every execution fragment that starts close to $\hat{S}$ remains close to $\hat{S}$. Formally,

Definition 24. *$\hat{S}$ is a* stable *set of $\mathcal{A}$ if $\forall \epsilon > 0$ $\exists \delta > 0$ such that*

$$RF(B_\delta(\hat{S})) \subseteq B_\epsilon(\hat{S})$$

We next discuss sufficient conditions that ensure stability of equilibrium states of the automaton. These conditions are defined in terms of a Lyapunov function $V : S \to \mathbb{P}$. Specifically, if the family of level sets $\{L_p\}_{p \in \mathbb{P}}$ is stable and $\forall \epsilon \geq 0$, $\exists p \in \mathbb{P}, \delta \geq 0$ such that $B_\delta(\hat{s}) \subseteq L_p \subseteq B_\epsilon(\hat{s})$, then $\hat{s}$ is

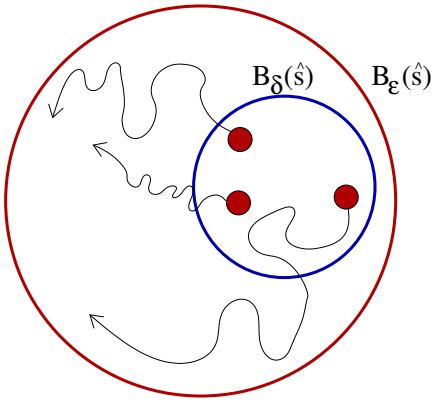


Figure 3.3: Pictorial representation of stable equilibrium state in terms of ϵ and δ balls around $\hat{s}$. Dark filled circles represent states contained in the δ-ball around $\hat{s}$, while arrow lines represent execution fragments. In this Figure, we have that all execution fragments starting from states in the δ-ball around $\hat{s}$ remain inside the ϵ-ball around $\hat{s}$.

stable. This is because all states reachable from $B_\delta(\hat{s})$ belong to the stable set L_p. We state these conditions formally in the following theorem:

Theorem 8. *If there exists $V : S \to \mathbb{P}$ that satisfies the following conditions:*

B1. $\forall \epsilon > 0,\ \exists p \in \mathbb{P}\ :\ \ L_p \subseteq B_\epsilon(\hat{s}),$

B2. $\forall p \in \mathbb{P},\ \exists \epsilon > 0\ :\ B_\epsilon(\hat{s}) \subseteq L_p,$

B3. $\{L_p\}_{p \in \mathbb{P}}$ *is stable*

then $\hat{s}$ is a stable state of $\mathcal{A}$.

Before proceeding with the proof, we briefly discuss these assumptions. B1 requires that every ϵ-ball around $\hat{s}$ contains a level set L_p. B2 is a symmetric assumption that requires that every level set of V contains an ϵ ball. These two conditions ensure the existence of L_p, with $B_\delta(\hat{s}) \subseteq L_p \subseteq B_\epsilon(\hat{s})$. B3 states that for any state $s \in S$ the set of reachable states from s is inside $L_{V(s)}$. This is a sufficient condition for ensuring stability of the level sets.

We next prove the theorem and refer to Figure 3.4 for a graphical representation of the proof.

Proof. Let us fix an $\epsilon > 0$. We have to show that there exists a $\delta > 0$, such that any execution fragment that starts in $B_\delta(\hat{s})$ remains within $B_\epsilon(\hat{s})$. From Assumption B1,

$$\exists p \in \mathbb{P} :\ L_p \subseteq B_\epsilon(\hat{s})$$

From Assumption B2,

$$\exists \nu \geq 0 :\ B_\nu(\hat{s}) \subseteq L_p$$

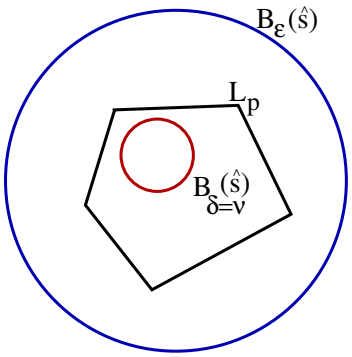


Figure 3.4: A graphical representation of the proof of Theorem 8.

Set $\delta = \nu$. From Assumption B3, L_p is stable, hence, since $B_\delta(\hat{s}) \subseteq L_p$, we get that

$$RF(B_\delta(\hat{s})) \subseteq L_p \subseteq B_\epsilon(\hat{s})$$

where the first inclusion follows from Lemma 7. □

3.4 Asymptotically Stable Equilibria

In this Section, we model asymptotically stable equilibrium states. Informally, $\hat{s} \in S$ is an *asymptotically stable* equilibrium state or, equivalently, $\mathcal{A}$ converges to $\hat{s}$, if every fair infinite execution of the automaton eventually gets and remains arbitrarily close to $\hat{s}$. Formally,

Definition 25. *$\mathcal{A}$ converges to $\hat{s}$ if* $\forall \epsilon > 0$*:* $\Diamond\Box \ \ (s \in B_\epsilon(\hat{s}))$

We refer to Chapter 2 for the meaning of the two temporal operators, $\Box$ and $\Diamond$. These two operators together ensure that $\forall \epsilon > 0$, $\exists t_\epsilon$ such that for all fair infinite execution π, and $t \geq t_\epsilon$, $d(\pi(t), \hat{s}) < \epsilon$. Figure 3.5 presents a graphical representation of this definition.

We next provide sufficient conditions for proving convergence in terms of a Lyapunov function $V : S \to \mathbb{P}$. Specifically, if the family of levels set $\{L_p\}_{p \in \mathbb{P}}$ of V is stable and $\forall \epsilon > 0$, $\exists p \in \mathbb{P}$ such that $L_p \subseteq B_\epsilon(\hat{s})$, then it is sufficient to show that the system eventually enters L_p. This is because, by stability of L_p, the system remains in L_p and, hence, it converges to $\hat{s}$. Next we give sufficient conditions that allow us to prove that the systems eventually reaches L_p. Suppose that (1) $\mathbb{P}$ is a well-ordered set, meaning that every nonempty subset of $\mathbb{P}$ has a least element, (2) the family $\{L_p\}_{p \in \mathbb{P}}$ is strictly monotonic and (3) for each level set there is an action, such that, when executed, the system moves from the level set to a strictly contained one. Under these assumptions, we can prove that the systems eventually reaches L_p. We state these conditions in the following theorem.

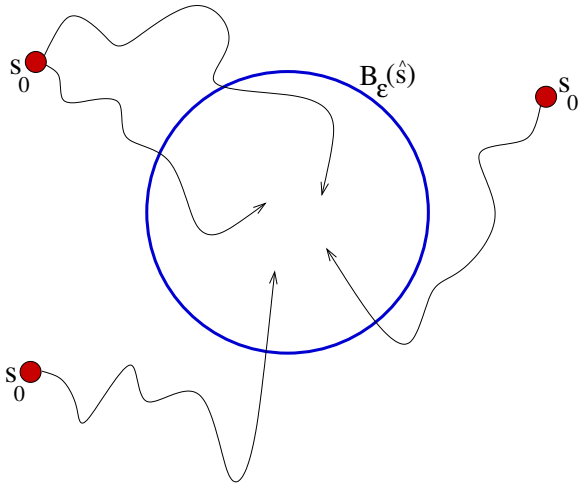


Figure 3.5: Pictorial representation of asymptotical stability of $\hat{s}$. Dark filled circles represent initial states of the system and arrow lines represent system executions. In this Figure, any execution eventually enters and remains in the ϵ-ball around $\hat{s}$.

Theorem 9. *If there exists $V : S \to \mathbb{P}$, with $\mathbb{P}$ a totally ordered set that satisfies the following conditions:*

C1. $\{L_p\}_{p\in\mathbb{P}}$ *is strictly monotonic,*

C2. $\forall \epsilon > 0, \exists p \in \mathbb{P}$ *such that* $L_p \subseteq B_\epsilon(\hat{s})$,

C3. $\{L_p\}_{p\in\mathbb{P}}$ *is stable,*

C4. $\forall p \in \mathbb{P}$, *with* $L_p \neq \{\hat{s}\}$, $\exists q < p$ *such that* $\Box\ (L_p \implies \Diamond\ L_q)$

C5. $\mathbb{P}$ *is a well-ordered set*

then $\mathcal{A}$ converges to $\hat{s}$.

Proof. Consider an arbitrary $\epsilon > 0$. Using Assumption C2, we know that $\exists p \in \mathbb{P}$ such that

$$L_p \subseteq B_\epsilon(\hat{s})$$

Our goal is to show that $\Diamond\ (s \in L_p)$. This is enough for proving convergence to $\hat{s}$, since, by Assumption C3, L_p is stable. We refer to Figure 3.6 for a pictorial representation of the goal of the proof.

Consider an arbitrary fair execution π of the automaton and denote by $p_0, p_1, \ldots \hat{p}$ the sequence of level sets visited when executing actions in π. Specifically, execution π starts in L_{p_0}, i.e. $s0 \in L_{p_0}$, then, executing actions from π, eventually enters the set L_{p_1} and so on. This sequence is a decreasing sequence of values with minimum element $\hat{p}$. The decreasing property follows from Assumptions C1, C3 and C4 while the minimum element property follows from Assumption C5.

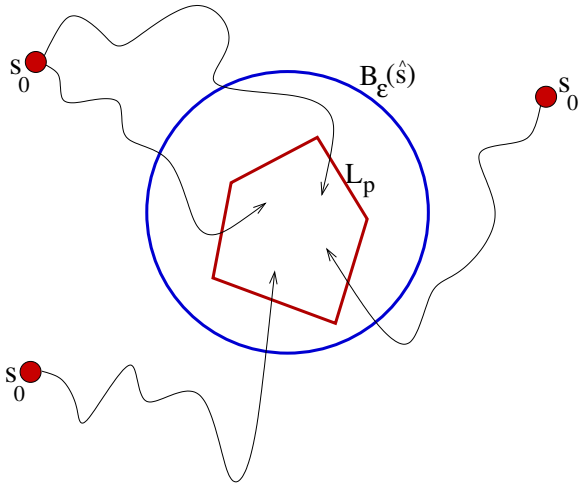


Figure 3.6: The system eventually enters L_p in Theorem 9. Dark filled circles represent initial states of the system and arrow lines represent executions of the system.

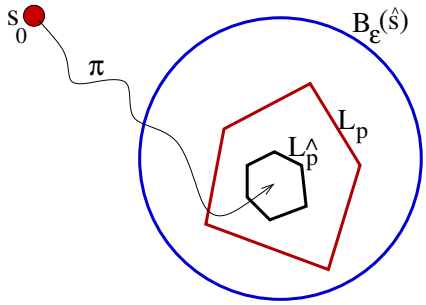


Figure 3.7: $L_{\hat{p}} \subseteq L_p$ in Theorem 9.

We want to show that

$$L_{\hat{p}} \subseteq L_p$$

or equivalently that $\hat{p} \leq p$. This condition ensures that execution π enters L_p. We pictorially represent this condition in Figure 3.7.

If $L_{\hat{p}} = \{\hat{s}\}$, then π converges to $\hat{s}$, and this concludes the proof.

Assume, instead, that

$$L_{\hat{p}} \neq \{\hat{s}\}$$

The proof proceeds by contradiction. Assume that $\hat{p} > p$. Using Assumption C4, we have that $\forall s \in L_{\hat{p}}$ there exists a fair action such that when executed the execution enters in a level set L_q with $q < \hat{p}$. This is a contradiction since $\hat{p}$ is assumed to be the minimum of the sequence. Hence, $\hat{p} \leq p$ and the Lemma follows. $\square$

In certain applications, the distance function d of the metric space can itself be used as a Lyapunov function for proving convergence. Given d, we can define the Lyapunov function to be $V(s) = d(s, \hat{s})$ with $\hat{s}$.

Corollary 1. *Given a function $V : S \to \mathbb{R}_{\geq 0}$ defined as $V(s) = d(s, \hat{s})$, $\forall s \in S$, if there exists a strictly decreasing infinite sequence $p_0, p_1, \ldots \in \mathbb{R}_{>0}$ of valuations of V that converges to 0 satisfying*

D1. $\{L_{p_i}\}_{p_i \in \mathbb{R}_{>0}}$ *is strictly monotonic*

D2. $\{L_{p_i}\}_{p_i \in \mathbb{P}}$ *is stable*

D3. $\forall i$, *with* $p_i \neq 0$, $\Box \left(L_{p_i} \Rightarrow \Diamond L_{p_{i+1}}\right)$

Then $\mathcal{A}$ converges to $\hat{s}$.

Proof. Condition C1-5 follow from Assumptions D1-3 of the decreasing sequence. □

In the special case when the decreasing sequence is of the form $C, C\alpha, C\alpha^2, C\alpha^3, \ldots$ with C being some positive constant and $0 \leq \alpha < 1$, we have that the automaton converges linearly to $\hat{s}$:

Corollary 2. *If there exists α, with $0 \leq \alpha < 1$, such that conditions D1-3 hold for the sequence $C, C\alpha, C\alpha^2, \ldots$ with $C \in \mathbb{R}_{>0}$, then $\mathcal{A}$ converges to $\hat{s}$.*

Proof. Follows from the previous Corollary where the sequence $C, C\alpha, C\alpha^2, C\alpha^3, \ldots$ is strictly decreasing. □

We next introduce the notion of convergence to a function.

Definition 26. *Given $g : S \to \hat{S}$, $\mathcal{A}$ converges to g if $\forall s \in S$ the automaton $\mathcal{A}_s = (S, \{s\}, A, T, E)$ converges to $g(s)$.*
$\mathcal{A}$ converges linearly to g with rate α, $0 \leq \alpha < 1$, if $\forall s \in S$ the automaton $\mathcal{A}_s = (S, \{s\}, A, T, E)$ converges linearly to $g(s)$ with rate α.

In this definition, for all $s \in S$ the automaton $\mathcal{A}_s$ has the same state space, same set of actions, same enabled condition and transition function of automaton $\mathcal{A}$. However, the two automata differ for the set of initial states: the set of initial states of $\mathcal{A}_s$ consists only of state s.

3.5 Properties of Automata in the Presence of Exogenous Inputs

In this Section, we discuss properties of automata with timed actions in the presence of exogenous inputs. Given $\mathcal{A}$, we denote by $\mathcal{A}_{exog}$ the corresponding exogenous automaton. We denote by $\hat{S}$ the set of equilibrium states of $\mathcal{A}$. We refer to Section 2.4 for the definition of $\mathcal{A}_{exog}$.

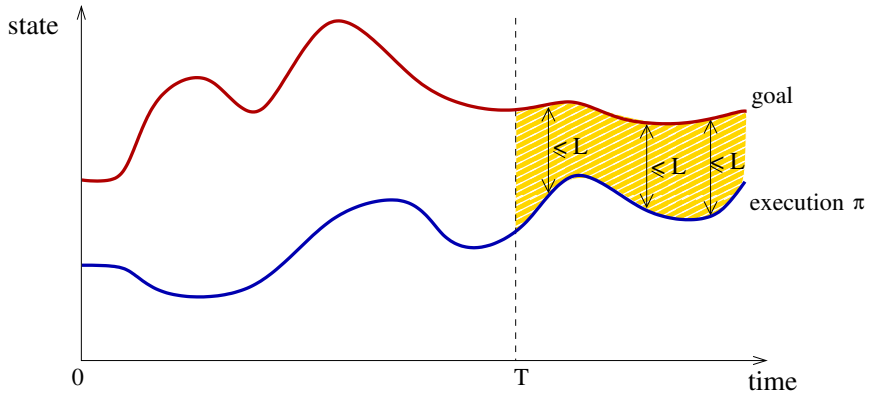


Figure 3.8: Pictorial representation of a L-bounded execution π of $\mathcal{A}_{exog}$ with respect to function g. In this Figure, for all $t \in \mathbb{R}_{\geq 0}$, the goal function at time t is defined as $\text{goal}(t) = g(\pi(t))$. We assume that the automaton in the absence of exogenous input $\mathcal{A}$ converges to g. By definition of bounded exogenous automaton, there exists a time T, such that $\forall t \geq T$, the distance between the state $\pi(t)$ and $\text{goal}(t)$ is bounded by L. The shaded area represent the time interval $[T, \infty)$.

In the presence of exogenous inputs, states in $\hat{S}$ may not be equilibrium states for $\mathcal{A}_{exog}$. Also, $\mathcal{A}_{exog}$ may not have equilibrium states at all. For example, we consider the generalization of the Line-Up multi-agent system presented in Section 2.4. In this system, agent 0 and N can move arbitrarily and the goal of the system is to form and maintain a equi-spaced straight line with extreme points given by the time-varying positions of agents 0 and N. The corresponding exogenous automaton $\mathcal{A}_{exog}$ combines the Line-Up automaton in absence of exogenous inputs described in Section 4.1 and the automaton of the exogenous inputs for the Line-Up multi-agent system described in Section 2.1.2. The exogenous automaton modeling the Line-Up multi-agent system in presence of exogenous inputs has no equilibrium states. This is because the execution of the action of the exogenous automaton modifies the positions of agents 0 and N.

In the presence of exogenous inputs, we are not interested in equilibrium states. Instead, we are interested in tracking the distance between the current state of the exogenous automaton and the set of equilibrium states of $\mathcal{A}$. This time-varying quantity measures how close the system in the presence of exogenous inputs is to $\hat{S}$. In particular, we are interested in systems where eventually-always this time-varying quantity is bounded by some finite constant. Formally,

Definition 27. *Given a function $g : S \to \hat{S}$ on $\mathcal{A}$ such that $\mathcal{A}$ converges to g, and given $L \in \mathbb{R}_{\geq 0}$, the exogenous automaton $\mathcal{A}_{exog}$ is L-bounded with respect to g if*

$$\Diamond\Box\left(d(s, g(s)) \leq L\right)$$

The exogenous automaton $\mathcal{A}_{exog}$ is bounded with respect to g if $\exists L \in \mathbb{R}_{\geq 0}$ such that $\mathcal{A}_{exog}$ is L-bounded with respect to g.

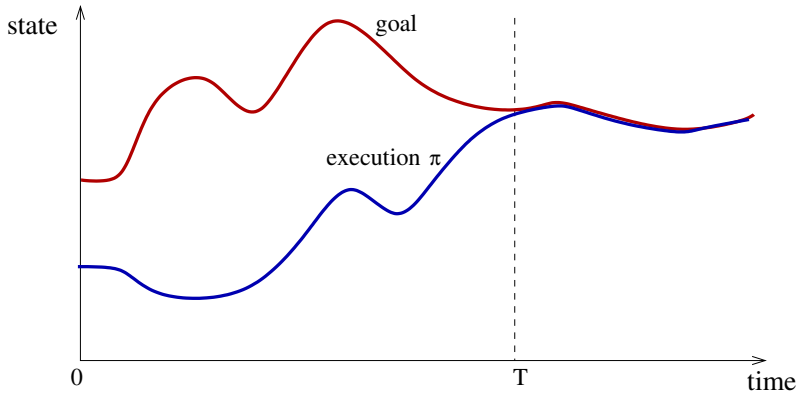


Figure 3.9: Pictorial representation of a 0-bounded execution π of $\mathcal{A}_{exog}$ with respect to function g. In this Figure, for all $t \in \mathbb{R}_{\geq 0}$, the goal function at time t is defined as $\text{goal}(t) = g(\pi(t))$. We assume that the automaton in the absence of exogenous input $\mathcal{A}$ converges to g. By definition of 0-bounded exogenous automaton, there exists a time T, such that $\forall t \geq T$, $\pi(t) = \text{goal}(t)$.

We recall that $d : S \times S \to \mathbb{R}_{\geq 0}$ is a distance function on the state space S. We briefly discuss this definition. It requires that the states of the automaton $\mathcal{A}_{exog}$ eventually get close and remain close to $g(s)$. Figure 3.8 presents a pictorial representation of this definition. If $L = 0$, we say that the automaton $\mathcal{A}_{exog}$ is 0-bounded with respect to g. We refer to Figure 3.9 for a pictorial representation of a 0-bounded automaton.

In Chapter 7, we provide conditions on g and on the set of actions of the system in the absence of exogenous inputs and on the actions of the exogenous inputs that ensure the system in the presence of exogenous inputs to be bounded or 0-bounded with respect to g. For example, in Chapter 7, we show that if the exogenous input are bounded, the Line-Up system in the presence of exogenous inputs is bounded with respect to the function $g : S \to \hat{S}$ defined as: $\forall s \in S, \forall i \leq N$, $g(s)(i) = \frac{N-i}{N} \cdot s(0) + \frac{i}{N} \cdot s(N)$.

3.6 Discussion

In this Section, we discuss the concepts of stability and convergence of automata with timed actions and relate them to the literature.

Throughout this thesis, stability and convergence are key concepts. We use them for proving correctness of multi-agent systems. We model multi-agents system as automata, and prove their correctness by showing that their goal configurations are asymptotically stable equilibrium states, or equivalently, that these systems converge to their goal states.

The correctness of distributed systems is usually defined in terms of termination, rather than convergence. A distributed system terminates, if it eventually reaches its goal configuration. We refer to [40] for a discussion of terminating distributed systems; for example, in the Byzantine

consensus problem the components of the system reach consensus in a finite number of rounds. Many of the terminating distributed systems have discrete state spaces. In distributed systems with dense state spaces, termination does not always hold, while the weaker convergence property may hold. The Line-Up multi-agent system, presented in Section 2.1.2, is an example of non terminating distributed systems with dense state space. For this example, we can show that the executions of the system get closer and closer to the goal configuration, but never reach it, i.e. that the system converges to the equilibrium state.

The definitions and theorems of this Chapter extend the work of [66] to systems with timed actions where the state space is a metric space. In [66], the author considers a discrete-time system and defines stability and convergence for this system in terms of a topological structure around $\hat{s}$, called a neighborhood system around $\hat{s}$. The author assumes weak fairness of the action set. We generalize the definitions and the theorems in [66] to systems with the weaker notion of fairness presented in Section 2.3. Our generalization assumes a specific neighborhood system, defined by the ϵ-balls around $\hat{s}$. This topological structure assumes the state space to be a metric space. The definitions and results presented in this Chapter are similar to [14, 47]; in this papers, authors generalize in a similar way the results of [66] to hybrid and timed I/O automata and formalize these properties in the PVS theorem prover.

Chapter 4

Stability and Convergence Properties of Multi-Agent Systems

In this Chapter, we present a general result for proving stability and convergence properties of equilibrium states of multi-agent systems.

In Section 4.1 we discuss stability and convergence properties of shared-state multi-agent systems where agents are not allowed to execute concurrent actions. In Section 4.2 we present a generalization of the shared-state multi-agent system where agents can read the state of other agents at some time in the past. In this Section, we derive conditions for proving stability and convergence properties of these systems. In Section 4.3 we model message-passing multi-agent systems with bounded delay and derive conditions for proving stability and convergence properties of theses systems. In Section 4.4 we discuss stability and convergence properties of multi-agent systems with concurrent actions. Finally, in Section 4.5 we relate the main results of this Chapter to the literature.

Throughout this Thesis, given a function $f : [t_1, t_2] \rightarrow S$, we denote by f^{Δ} the function f shifted by Δ. The function f^{Δ} has domain $[t_1 + \Delta, t_2 + \Delta]$ and it is defined as $\forall t \in [t_1 + \Delta, t_2 + \Delta]$, $f^{\Delta}(t) = f(t - \Delta)$.

4.1 Shared-State Multi-Agent Systems

In this Section, we model shared-state multi-agent systems. We have informally presented these systems in Chapter 1. A shared-state system consists of a collection of agents that communicate via shared variables. In this Section, we model systems where at any given time only one agent can perform an action: concurrent actions by multiple agents are not allowed. A more general model where agents can execute actions concurrently is discussed in Section 4.4.

In this Thesis, we consider shared-state systems where actions of an agent can read the state of all agents in the system, but only modify its own state. This is a restriction on the general shared-state multi-agent system class, where the action of an agent can modify a group of agents.

4.1.1 Shared-State Automaton

We model a multi-agent system using the automaton with the timed action model. In shared-state systems, the state of the system S is defined as the Cartesian product of the states of the agents composing the system. Similarly, the set of initial state $S0$ is the cartesian product of the sets of initial states of the agents in the system. Each agent has its own set of actions. Actions of an agent can read the states of other agents, but only modify its own state. Formally,

Definition 28. *A shared-state MAS with N agents can be modeled as an automaton with timed actions $\mathcal{A} = (S, S0, A, E, T)$ with:*

- $S = S_1 \times S_2 \times \ldots S_N$, *where S_i is the state of agent i,*
- $S0 = S0_1 \times S0_2 \times \ldots S0_N$, *where $S0_i$ is the initial state of agent i,*
- $A = \bigcup_{i=1}^{N} A_i$, *where A_i is the set of actions of agent i*
- $\forall s \in S, a_i \in A_i$, $\forall t \in [0, \tau_{s,a_i}]$, *only agent i's state changes:*

$$T(s, a_i)(t) = (s(1), s(2), \ldots, s(i-1), f_{s,a_i}(t), s(i+1), \ldots, s(N))$$

In Figure 4.1, we represent an execution of an automaton consisting of 3 agents.

Example. We next discuss an example of multi-agent system and corresponding automaton. The system and automaton are depicted in Figure 4.2. As shown in Figure 4.2(a), the system consists of two agents: u and v. The state of agent u consists of a single real-valued variable x, with $x \in [0, 2]$. The set of initial states of u is the interval $[0, 1]$. We refer to Figure 4.2(b) for a pictorial representation of the state of u. Similarly, the state of agent v consists of the real-valued variable y, with $y \in [0, 2]$, and its set of initial states is the interval $[0, 1]$. The state of the overall system is defined as the cartesian product of the agents states, i.e. $S = [0, 2] \times [0, 2]$. This set corresponds to the shaded (green) area in Figure 4.2(c). Similarly, the set of initial states is the cartesian product of the agents initial states, i.e. $S0 = [0, 1] \times [0, 1]$. This set corresponds to the light shaded rectangle in Figure 4.2(c). The set of actions consists of the set of actions of u and the set of actions of v. Actions of u (respectively v) read the states of u and v, but change only the state of u (respectively u). In Figure 4.2(d), we represent a feasible execution of the system.

The Line-Up multi-agent system presented in Section 2.1.2 and its generalization with dynamics presented in Section 2.2.4 are examples of shared-state multi-agent systems. In these systems, the state space of the system is the cartesian product of the state spaces of the agents and each action of the system reads the state of the system, but only modifies the state of a single agent.

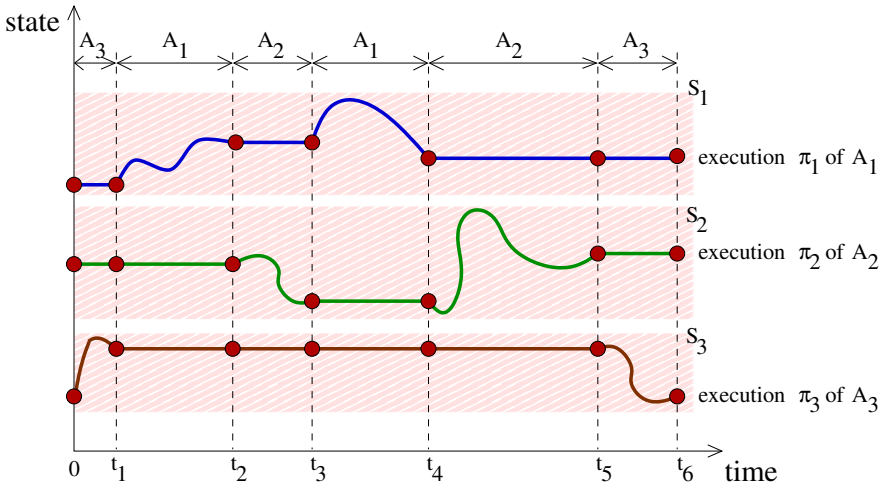
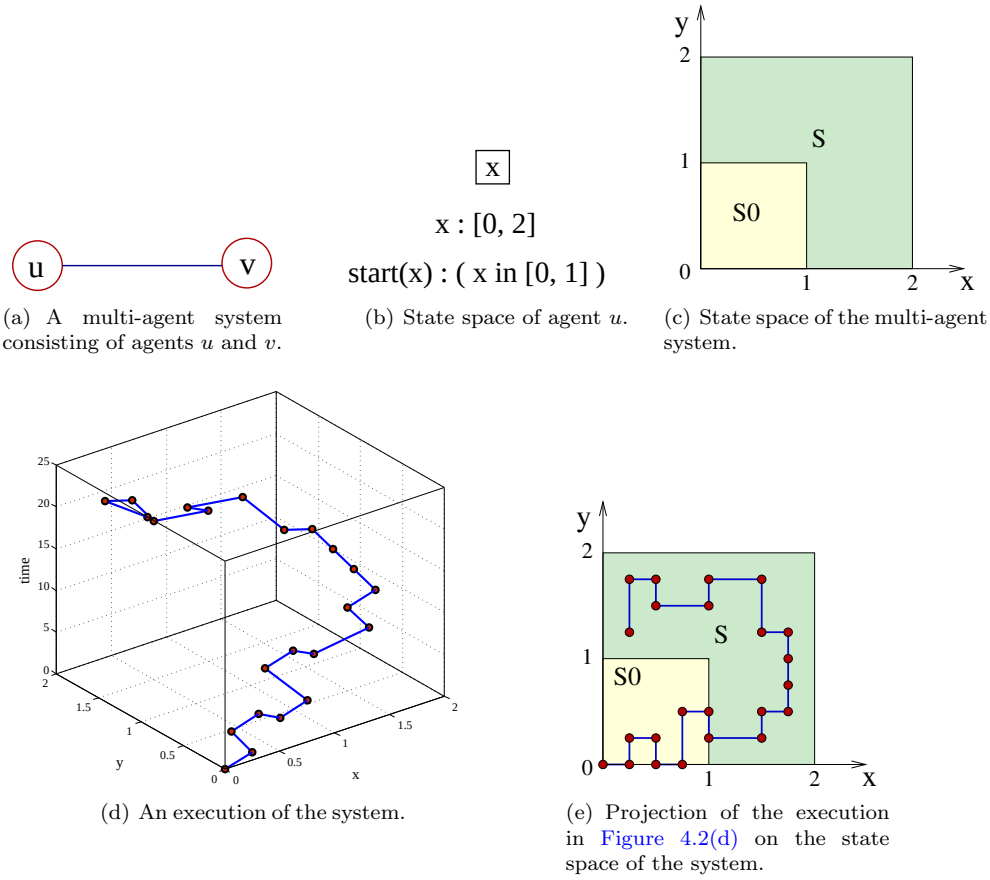


Figure 4.1: An execution of a multi-agent system consisting of three agents A_1, A_2 and A_3. For each agent i, function π_i is the projection of the system execution on agent A_i. The state of the system at time t is given by the triple $(\pi_1(t), \pi_2(t), \pi_3(t))$. The system start at state $(\pi_1(0), \pi_2(0), \pi_3(0))$. In this state, agent A_3 executes an action. This action updates the state of agent A_1 and leaves the state of the other agents unchanged. The end-state of this action is state $(\pi_1(t_1), \pi_2(t_1), \pi_3(t_1))$ with $\pi_1(t_1) = \pi_1(0)$, $\pi_2(t_1) = \pi_2(0)$. Then, starting from this state, agent A_1 executes an action and so on. The complete sequence of actions executed is $a_1, a_2, a_3, a_4, a_5, a_6, \ldots$ with $a_1 \in A_3$, $a_2 \in A_1$, $a_3 \in A_2$, $a_4 \in A_1$, $a_5 \in A_2$, $a_6 \in A_3$ and so on. The three shaded areas represent the state spaces of the three agents denoted by S_1, S_2 and S_3.

Theorem Proving Model. In Figure 4.3, we encode a generic multi-agent shared-state system in PVS. In this model, we assume that the system is represented by a discrete automaton. This system consists of $N + 1$ agents, with $N > 0$. In this generic system, each agent has an identifier. These identifiers are natural numbers in the interval $[0, N]$. In PVS, the type of agent identifiers is `I`. This generic multi-agent system models homogenous systems, i.e. systems where all agents have the same state space. The type of the state space of the agents is represented by the PVS type `L`. This is an uninterpreted type, i.e. it is a generic type, that can be instantiated to any concrete type. For example, the state space of an agent can be the set of real numbers. The state space of the system is the cartesian product of the states of its agents; this is encoded using the PVS array function `S`. This function maps each agent identifier to its state space. The set of initial states is encoded in PVS using the predicate `start?`. This predicate is defined as a conjunction of boolean conditions. Each condition checks properties of a single agent. This predicate is in conjunctive form, since the set of starting states of the system is the cartesian product of the set of starting states of the single agents. In the PVS meta-theory, the type `A` defines the set of actions of the system. Each action has an enabling condition, encoded by the predicate `E`, and a body, encoded in PVS by the function `T`. For each state `s` in the state space and action `a` of the system, predicate `E` holds if action `a` can be executed in state `s`. The resulting state of this execution is given by `T(s,a)`.



(a) A multi-agent system consisting of agents u and v.

(b) State space of agent u.

(c) State space of the multi-agent system.

(d) An execution of the system.

(e) Projection of the execution in Figure 4.2(d) on the state space of the system.

Figure 4.2: An example of shared-state multi-agent system and the corresponding automaton.

4.1.2 Shared-State Automaton with Explicit Arbitrary Dynamics

In this Section, we construct a generic shared-state multi-agent system given a discrete multi-agent system. We denote by $\mathcal{A}_{\mathcal{D}} = (S_{\mathcal{D}}, S0_{\mathcal{D}}, A_{\mathcal{D}}, E_{\mathcal{D}}, T_{\mathcal{D}})$ a discrete automaton modeling a shared-state multi-agent system, by $\hat{s}_{\mathcal{D}}$ an equilibrium of $\mathcal{A}_{\mathcal{D}}$ and by $V_{\mathcal{D}} : S_{\mathcal{D}} \to \mathbb{P}$ a Lyapunov function on $\mathcal{A}_{\mathcal{D}}$. Actions of $\mathcal{A}_{\mathcal{D}}$ are instantaneous. Given $\mathcal{A}_{\mathcal{D}}$, we next construct a generic automaton with timed actions $\mathcal{A} = (S, S0, A, E, T)$ and present conditions that ensure stability and convergence properties of $\mathcal{A}$.

Automaton $\mathcal{A}$ relaxes the assumption of $\mathcal{A}_{\mathcal{D}}$ and explicitly models the dynamics of the agents. When $\mathcal{A}$ executes an action, it evolves the current state of some agent i towards its newly computed one according to some dynamics. In this more general automaton, an agent, when evolving its state, can stop before reaching the newly computed value. At each time of the execution, the state of the system stores both the current state of the multi-agent system and the newly computed one. In this model, we refer to the newly computed state as the destination state. Initially, the destination state is equal to the initial state of the automaton.

We next describe the structure of the automaton. A state s of $\mathcal{A}$ consists of a pair of states

```
% Number of agents of the system.
N: posnat

% Agent Identifier.
% It is a natural number in the interval [0,N].
I: TYPE = upto(N)

% Agent Type.
% It is an uninterpreted type and can store any type.
L: TYPE

% PVS meta-theory.
SharedState: THEORY
BEGIN

        % State definition.
        % A state maps each agent identifier into a value.
        S: TYPE = [I -> L]

        % Action set of the system.
        A: DATATYPE
        BEGIN
                % Action definition.
                % This definition includes the input paramenters of the action.
        END A

        s: VAR S
        a: VAR A

        % Initial State Predicate.
        % This predicate defines the set of initial states.
        start?(s): bool = % conditions in conjunctive form

        % Enabling Predicate.
        E(s,a): bool =
        CASES a OF
                % For each action of the system, this predicate defines
                % a set of enabling conditions.
        ENDCASES

        % Transition Function.
        T(s,a): S =
        CASES a OF
                % For each action of the system,
                % this function encodes the body of the action.
        ENDCASES

END SharedState
```

Figure 4.3: Generic PVS model of a shared-state multi-agent system.

of $\mathcal{A}_{\mathcal{D}}$, where the first component represents the current state of the automaton and the second component represents its destination state. Given s we refer to the first component of s as $s.x$ and to the second component as $s.z$. A state $s \in S$ is an initial state of $\mathcal{A}$, if both $s.x$ and $s.z$ are equal to the same initial state of $\mathcal{A}_{\mathcal{D}}$. The set of actions of $\mathcal{A}$ is equal to the set of actions of $\mathcal{A}_{\mathcal{D}}$. Given a state $s \in S$, and an action $a \in A$, the action is enabled in s, if the corresponding discrete action is enabled in state $s.x$. This is because, by construction, $s.x$ stores the current state of the multi-agent system. In the automaton $\mathcal{A}$, for all $s \in S$, $a \in A$, the execution of action a in state s has duration $\tau_{s,a}$. We denote by i the agent executing action a. When a is executed, agent i updates its current and destination states, then evolves its current state towards its newly computed destination state. The other agents do not change their state.

Formally, the automaton representation follows.

Definition 29. *Given $\mathcal{A}_{\mathcal{D}} = (S_{\mathcal{D}}, S0_{\mathcal{D}}, A_{\mathcal{D}}, E_{\mathcal{D}}, T_{\mathcal{D}})$, the automaton with time actions $\mathcal{A} = (S, S0, A, E, T)$ modeling a shared-state MAS with N agents and explicit arbitrary dynamics has:*

- $S = (S_{\mathcal{D}}, S_{\mathcal{D}})$,
- $S0 = \{s \in S \mid \exists s0 \in S0_{\mathcal{D}} \ : \ s.x = s.z = s0\}$,
- $A = A_{\mathcal{D}}$
- $\forall s \in S$, $\forall a \in A$, $E(s,a) = E_{\mathcal{D}}(s.x, a)$,
- $\forall s \in S$, $\forall a \in A$, *action a is executed by agent i, has duration $\tau_{s,a}$ and its behaviour is as follows,* $\forall t \in (0, \tau_{s,a}]$,

$$(T(s,a)(t)).x(i) = f_{s,a}(t)$$
$$(T(s,a)(t)).z(i) = T_{\mathcal{D}}(s.x, a)(i)$$

and $\forall j \neq i$,

$$\begin{aligned} (T(s,a)(t)).x(j) &= s.x(j) \\ (T(s,a)(t)).z(j) &= s.z(j) \end{aligned}$$

The fairness criterion of $\mathcal{A}$ is equal to the fairness criterion of $\mathcal{A}_{\mathcal{D}}$. Given an equilibrium state $\hat{s}_{\mathcal{D}}$ of $\mathcal{A}_{\mathcal{D}}$, the corresponding equilibrium state in S, denoted by $\hat{s}$, satisfies the property that $\hat{s}.x = \hat{s}.z = \hat{s}_{\mathcal{D}}$. We next present conditions on the structure of the level sets of $V_{\mathcal{D}}$ and condition on the function f that ensure stability and convergence of $\mathcal{A}$. Given $V_{\mathcal{D}}$, we construct a Lyapunov function $V : S \to \mathbb{P}$ for $\mathcal{A}$. For each state $s \in S$, the value $V(s) = \max\{V(s.x), V(s.z)\}$.

Line-Up example. We can construct a generic Line-Up multi-agent system with explicit dynamics. We refer to Section 2.1.2 for a detailed description of the Line-Up multi-agent system and corresponding discrete automaton.

Given the discrete Line-Up automaton $\mathcal{A}_\mathcal{D} = (S_\mathcal{D}, S0_\mathcal{D}, A_\mathcal{D}, E_\mathcal{D}, T_\mathcal{D})$ defined in Section 2.1.2, the automaton with time actions $\mathcal{A} = (S, S0, A, E, T)$ modeling the Line-Up shared-state MAS with N agents and explicit arbitrary dynamics has:

- $S = (\mathbb{R}^{N+1}, \mathbb{R}^{N+1})$,
- $S0 = \{(s0, s0)\}$ with $s0$ the initial state of $\mathcal{A}_\mathcal{D}$,
- $A = \{A_i\}_{i \in I}$ with $A_i = \{avg_{l,i,r}\}_{l<i<r}$,
- $E : S \times A \to true$
- $\forall s \in S$, $\forall a = avg_{l,i,r} \in A$, action a is executed by agent i, has duration $\tau_{s,a}$ and its behaviour is as follows, $\forall t \in (0, \tau_{s,a}]$,

$$\begin{aligned} (T(s,a)(t)).x(i) &= f_{s,a}(t) \\ (T(s,a)(t)).z(i) &= \frac{r-i}{r-l}s(l) + \frac{i-l}{r-l}s(r) \end{aligned}$$

and $\forall j \neq i$,

$$\begin{aligned} (T(s,a)(t)).x(j) &= s.x(j) \\ (T(s,a)(t)).z(j) &= s.z(j) \end{aligned}$$

In the case of stability we show that the following Theorem holds.

Theorem 10. *If*

E1. $V_\mathcal{D}$ satisfies Assumptions B1-3 of Theorem 8,

E2. $\forall k \in \mathbb{P}$, level set $L_{k_\mathcal{D}}$ of $V_\mathcal{D}$ is a convex set,

E3. $\forall s \in S, \forall a \in A$, $\forall t \in [0, \tau_{s,a}]$, $(T(s,a)(t)).x$ is a convex combination of $s.x$ and $T_\mathcal{D}(s.x, a)$,

then $\hat{s}$ is a stable equilibrium state of $\mathcal{A}$.

We briefly discuss these Assumptions. E1 requires that $V_\mathcal{D}$ is a certificate of stability for $\hat{s}_\mathcal{D}$ in $\mathcal{A}_\mathcal{D}$. E2 requires that the level sets of $V_\mathcal{D}$ are convex sets. E3 requires that $\forall s \in S, \forall a \in A$, the states of the execution fragment π, having $\pi.fstate = s.x$, $\pi.lstate = T_\mathcal{D}(s.x, a)$ and obtained by executing a on $s.x$, are linear combinations of $s.x$ and $T_\mathcal{D}(s.x, a)$. This condition is pictorially represented

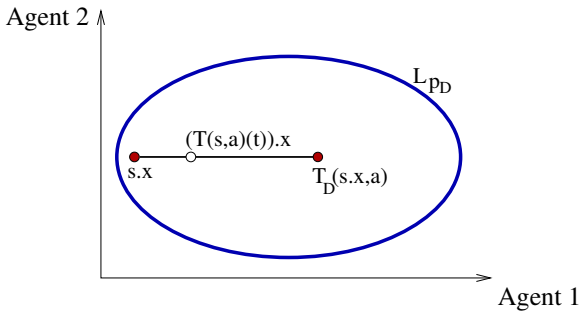


Figure 4.4: Pictorial representation of the conditions of Theorem 10. We consider a system consisting of two agents. E2 requires that the level sets of $\mathcal{A}_{\mathcal{D}}$ are convex. E3 requires that $(T(s,a)(t)).x$ is a convex combination of $s.x$ and $T_{\mathcal{D}}(s.x,a)$. In this Figure, $s.x$ and $T_{\mathcal{D}}(s.x,a)$ belong to $L_{p_{\mathcal{D}}}$ and are represented as dark filled circles; $(T(s,a)(t)).x$ is represented as a white filled circle.

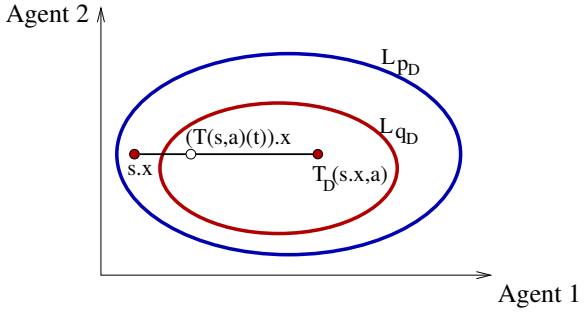


Figure 4.5: Pictorial representation of the conditions of Theorem 11. We consider a system consisting of two agents. In this Figure, state $s.x \in L_{p_{\mathcal{D}}} - L_{q_{\mathcal{D}}}$ and $T_{\mathcal{D}}(s.x,a) \in L_{q_{\mathcal{D}}}$. F2 requires that the level sets of $\mathcal{A}_{\mathcal{D}}$ are convex. F3 requires that $(T(s,a)(t)).x$ is a convex combination of $s.x$ and $T_{\mathcal{D}}(s.x,a)$. F4 requires that $\exists t \in [0,\tau_{s,a}]$, such that $(T(s,a)(t)).x \in L_{q_{\mathcal{D}}}$. In this Figure, $s.x$ and $T_{\mathcal{D}}(s.x,a)$ are represented as dark filled circles, and $(T(s,a)(t)).x$ is represented as a white filled circle.

in Figure 4.4. In Section 3.1, we have used condition E3. Specifically, in Lemma 4, we have shown that Equation 2.3 is an equilibrium state equilibrium state of the Line-Up automaton with dynamics if and only if the Line-Up automaton with dynamics satisfies E3.

Proof. We want to show that V on $\mathcal{A}$ satisfies Assumptions B1-3 of Theorem 8. Conditions B1 and B2 hold by hypothesis. We next show that the family of level sets $\{L_k\}_{k\in\mathbb{P}}$ of V is stable.

Fix an arbitrary $k \in \mathbb{P}$. Consider an arbitrary state $s \in L_k$ and action $a \in A$ with $E(s,a)$. We next show that $\forall t \in [0,\tau_{s,a}]$, $T(s,a)(t) \in L_k$.

By Assumption B3 of function $V_{\mathcal{D}}$, $T_{\mathcal{D}}(s.x,a) \in L_{k_{\mathcal{D}}}$, since by construction of $V_{\mathcal{D}}$, $s.x \in L_{k_{\mathcal{D}}}$. Hence, by construction, $\forall t \in [0,\tau_{s,a}]$, $(T(s,a)(t)).z \in L_{k_{\mathcal{D}}}$. By Assumption E2, $L_{k_{\mathcal{D}}}$ is a convex set; hence, any convex combination of $s.x$ and $T_{\mathcal{D}}(s.x,a)$ is in $L_{k_{\mathcal{D}}}$. By Assumption E3, $\forall t \in [0,\tau_{s,a}]$, $(T(s,a)(t)).x$ is a convex combination of $s.x$ and $T_{\mathcal{D}}(s.x,a)$. Hence, $\forall t \in [0,\tau_{s,a}]$, $(T(s,a)(t)).x \in L_{k_{\mathcal{D}}}$. Putting all together, by construction of V, we have that $\forall t \in [0,\tau_{s,a}]$, $T(s,a)(t) \in L_k$. □

In the case of convergence the following Theorem holds.

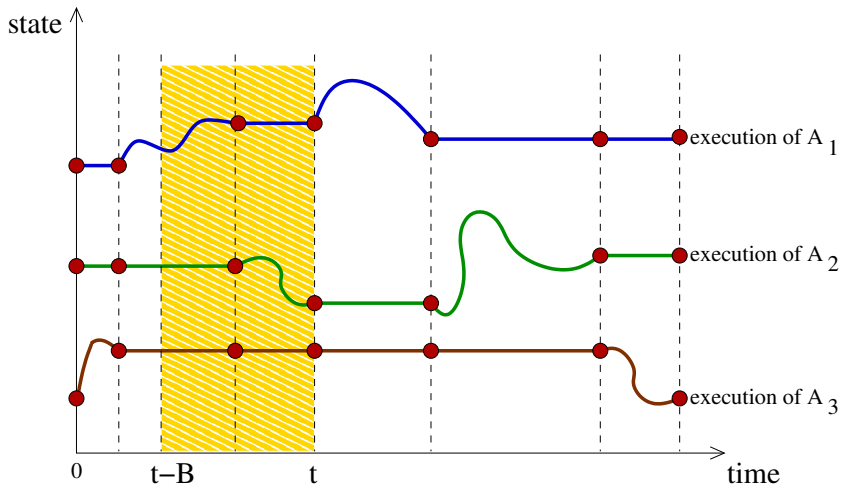


Figure 4.6: An execution of a multi-agent system consisting of three agents where at time t, each agent can access the state of other agents in the interval $[t-\mathcal{B}, t]$. We refer to Figure 4.1 for a description of this execution. The shaded area represents the time interval $[t-\mathcal{B}, t]$.

Theorem 11. *If*

F1. $V_{\mathcal{D}}$ *satisfies Assumptions C1-5 of Theorem 9,*

F2. $\forall k \in \mathbb{P}$*, level set* $L_{k_{\mathcal{D}}}$ *of* $V_{\mathcal{D}}$ *is a convex set,*

F3. $\forall s \in S, \forall a \in A, \forall t \in [0, \tau_{s,a}]$*,* $(T(s,a)(t)).x$ *is a convex combination of* $s.x$ *and* $T_{\mathcal{D}}(s.x, a)$*,*

F4. $\forall s \in S$*,* $\forall a \in A$*, if* $\exists p, q \in \mathbb{P}$*, with* $q < p$*, such that* $s.x \in L_{p_{\mathcal{D}}} - L_{q_{\mathcal{D}}}$ *and* $T_{\mathcal{D}}(s.x, a) \in L_{p_{\mathcal{D}}}$ *then* $\exists t \in [0, \tau_{s,a}]$ *such that* $T(s,a)(t).x \in L_{q_{\mathcal{D}}}$

then $\mathcal{A}$ *converges to* $\hat{s}$*.*

Before proceeding with the proof we briefly discuss Assumptions F1 and F4. Assumption F1 requires that $V_{\mathcal{D}}$ is a certificate of convergence of $\hat{s}_{\mathcal{D}}$ in $\mathcal{A}_{\mathcal{D}}$. Assumption F3 is the same as Assumption E3 of Theorem 10. F4 requires that $\forall s \in S, \forall a \in A$, if, when executing action a in state $s.x$, $\mathcal{A}_{\mathcal{D}}$ moves from level set $L_{p_{\mathcal{D}}}$ to $L_{q_{\mathcal{D}}}$, with $L_{q_{\mathcal{D}}} \subsetneq L_{p_{\mathcal{D}}}$, then, when executing action a in state s, $\mathcal{A}$ enters a strictly contained level set. This condition is pictorially represented in Figure 4.5.

Proof. We want to show that V on $\mathcal{A}$ satisfies Assumptions C1-5 of Theorem 9. Conditions C1, C2 and C5 hold by hypothesis. The proof of Assumption C3 is similar to the proof of condition B3 in Theorem 10 and not reported. We next show that condition C4 holds.

Fix an arbitrary $k \in \mathbb{P}$ and consider an arbitrary state $s \in L_k$. Using Assumption C4 on $\mathcal{A}_{\mathcal{D}}$, we have that $\mathcal{A}_{\mathcal{D}}$ eventually reaches a state $\bar{s}$ with $\bar{s} \in L_q$ and $L_q \subset L_k$. By construction and Assumption F4, $\mathcal{A}$ eventually reaches $\bar{s}$ as well. Hence, condition C4 holds. □

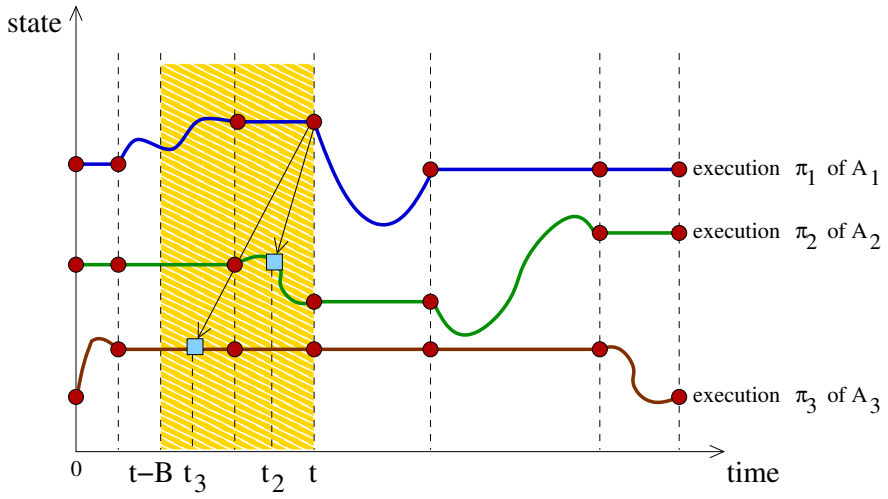


Figure 4.7: In the execution of Figure 4.6, agent A_1 at time t reads past states of agent A_2 and A_3. It reads the state of agent A_2 at time t_2 and of agent A_3 at time t_3. Agent A_1 executes an action at time t on the state $(\pi_1(t), \pi_2(t_2), \pi_3(t_3))$. The behaviour of this action is different from the behavior of the same action in the execution depicted in Figure 4.1. This is because the two actions are executed on different states. In Figure 4.1 the action is executed on the current state $(\pi_1(t), \pi_2(t), \pi_3(t))$ of the system.

4.2 Shared-State Multi-Agent System with Sliding Window

In this Section, we introduce a generalization of the shared-state multi-agent systems. By construction, in a shared-state system agents update their state using the current state of the other agents. We next introduce a more general shared-state model where agents can update their state using the past states of other agents. In this new model, called shared-state system with sliding window, if t is the current time of the execution, agents can read the state of other agents in the time interval $[t - \mathcal{B}, t]$ where $\mathcal{B}$ is a constant and $\mathcal{B} \geq 0$. For example, Figure 4.6 presents the execution of a multi-agent system where agent A_1 at time t can access the state of agents A_2 and A_3 in the interval $[t-\mathcal{B}, t]$. As shown in Figure 4.7, agent A_1 reads the state of agent A_2 at time t_2 and of agent A_3 at time t_3, with $t_2, t_3 \in [t - \mathcal{B}, t]$. In these figures, the shaded area emphasizes the execution fragment from time $t - \mathcal{B}$ to t. This area is a window of size $\mathcal{B}$.

4.2.1 Automaton with sliding window

We denote by $\mathcal{A} = (S, S0, A, E, T)$ the shared state automaton and by $\mathcal{A}_\mathcal{B} = (S_\mathcal{B}, S0_\mathcal{B}, A_\mathcal{B}, E_\mathcal{B}, T_\mathcal{B})$ the corresponding shared-state automaton with sliding window. We next informally present the structure of $\mathcal{A}_\mathcal{B}$.

A state in $\mathcal{A}_\mathcal{B}$ is a finite execution fragment of $\mathcal{A}$. The duration of this fragment is $\mathcal{B}$ and the final state of the fragment is the state of the shared-state multi-agent system at some time t. Hence,

a state $s_{\mathcal{B}} \in S_{\mathcal{B}}$ can be represented as a function that maps the interval $[-\mathcal{B}, 0]$ to S, where $s_{\mathcal{B}}(0)$ represents the state of the multi-agent system at some time t and $s_{\mathcal{B}}(\hat{t})$, with $-\mathcal{B} \leq \hat{t} < 0$, represents the state of the multi-agent system at time $t - \hat{t}$. Given $s_{\mathcal{B}} \in S_{\mathcal{B}}$ we denote by $s_{i\mathcal{B}}$ the projection of the state $s_{\mathcal{B}}$ on agent i. Formally, $s_{i\mathcal{B}}$ is a function from the interval $[-\mathcal{B}, 0]$ to S_i, defined as $\forall t \in [-\mathcal{B}, 0]$, $s_{i\mathcal{B}}(t) = s_{\mathcal{B}}(t)(i)$. We next discuss the set of initial states of $A_{\mathcal{B}}$. For initial state $s0 \in S0$, the past is not defined, since the execution starts at $s0$ at time 0. For these states, the corresponding initial state $s0_{\mathcal{B}} \in S0_{\mathcal{B}}$ repeats $s0$ throughout the interval $[-\mathcal{B}, 0]$, i.e. $\forall t \in [-\mathcal{B}, 0]$, $s0_{\mathcal{B}}(t) = s0$.

Agents read past states. For example, in the execution π of the multi-agent system presented in Figure 4.7, agent A_1 at time t chooses a time t_2 for agent A_2 and a time t_3 for agent A_3, with $t_2, t_3 \in [t - \mathcal{B}, t]$. Given this sequence of times, agent A_1 executes an action on the state $(\pi(t)(1), \pi(t_2)(2), \pi(t_3)(3))$, where $\pi(t)(1)$ is the state of agent A_1 at time t, $\pi(t_2)(2)$ is the state of agent A_2 at time t_2 and $\pi(t_3)(3)$ is the state of agent A_3 at time t_3.

In general, given a state $s_{\mathcal{B}} \in S_{\mathcal{B}}$, and given a N-tuple of times $t \in [-\mathcal{B}, 0]^N$, we call the tuple $(s_{1\mathcal{B}}(t(1)), \ldots, s_{N\mathcal{B}}(t(N)))$ an asynchronous view of $s_{\mathcal{B}}$. In an asynchronous view, the i-th value of the tuple is the state of agent i in $s_{\mathcal{B}}$ at time $t(i)$. Intuitively, this view is a tuple where the state of each agent can either be its current state or some state in the past. In an asynchronous view, the times in t are independent from each other. If $t = (\hat{t}, \hat{t}, \ldots, \hat{t})$ for all i, then the asynchronous view corresponds to the state of the system at time $\hat{t}$. An asynchronous view of $s_{\mathcal{B}}$ is a valid state of S. This is because, by construction, S is defined as the cartesian product of the states of the agents of the system. Given a state $s \in S$ and a state $s_{\mathcal{B}} \in S_{\mathcal{B}}$, we introduce the asynchronous view relation $H \subseteq S \times S_{\mathcal{B}}$ as follows:

$$(s, s_{\mathcal{B}}) \in H \equiv \big(\exists t \in [-\mathcal{B}, 0]^N \ : \ \forall i \leq N \ : \ s_{i\mathcal{B}}(t(i)) = s(i)\big)$$

We denote $(s, s_{\mathcal{B}}) \in H$ as $H(s, s_{\mathcal{B}})$. These two states are in the asynchronous view relation if state s is an asynchronous view of state $s_{\mathcal{B}}$. Given this relation, we define the set of all asynchronous view of $s_{\mathcal{B}} \in S_{\mathcal{B}}$. We denote this set by $\mathcal{H}(s_{\mathcal{B}})$. Formally, this set is

$$\mathcal{H}(s_{\mathcal{B}}) = \{s \in S \mid H(s, s_{\mathcal{B}})\}$$

The notion of asynchronous view is used for defining the behaviour of the actions of $\mathcal{A}_{\mathcal{B}}$. The set of actions of $\mathcal{A}_{\mathcal{B}}$ is constructed from the set of actions of A. For each action $ai \in A_i$, we construct action $ai_{\mathcal{B}} \in \mathcal{A}_{\mathcal{B}}$. Given a state $s_{\mathcal{B}}$, the execution of action $ai_{\mathcal{B}}$ consists of the execution of action ai on an asynchronous view of $s_{\mathcal{B}}$. The specific asynchronous view s of $s_{\mathcal{B}}$ is nondeterministically chosen by $ai_{\mathcal{B}}$. The only constraint on s is that the state of agent i in s is equal to the current state of i in $\bar{s}$, i.e. $s(i) = s_{i\mathcal{B}}(0)$. In the shared-state model with sliding window, the automaton is non-

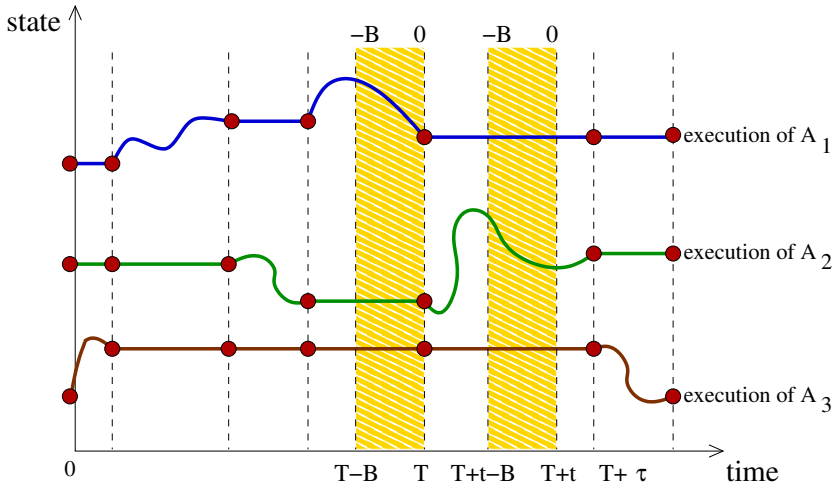


Figure 4.8: Pictorial representation of two non-overlapping windows defined on the execution in Figure 4.6. The time interval of the first window is $[T - \mathcal{B}, T]$, while the time interval of the second window is $[T - \mathcal{B} + t, T + t]$ with $t > \mathcal{B}$. These two time intervals do not overlap. The quantity τ denotes the time duration of the action executed by agent A_2 at time T. The shaded areas represent the two time intervals, i.e. the two windows.

deterministic, because given a state $s_{\mathcal{B}}$ and an action $ai_{\mathcal{B}}$, the execution of the action does not have a unique behaviour. It depends on the asynchronous view chosen by the action. The duration of $ai_{\mathcal{B}}$ is equal to $\tau_{s,ai}$, which denotes the duration of ai when executed from state s; and the behaviour of agent i is given by $f_{s,ai}$. Given $s_{\mathcal{B}}$, $ai_{\mathcal{B}}$ and the asynchronous view s, the transition function $T_{\mathcal{B}}$ at time t, for all $t \in [0, \tau_{s,ai}]$, is defined as follows. As represented in Figure 4.8, if $t > \mathcal{B}$, the window of size $\mathcal{B}$ ending at time t and the window of size $\mathcal{B}$ ending at time 0 do not overlap. In this case, the state $T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t)$ corresponds to the sequence of states of $T(s, ai)$ from time $t - \mathcal{B}$ to t. If $t \leq \mathcal{B}$, the window at time t and the window ending at time 0 overlap. As represented in Figure 4.9, in this case the portion of the state $T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t)$ from time $-\mathcal{B}$ to $-t$ is given by the portion of state $s_{\mathcal{B}}$ from time $-\mathcal{B} + t$ to 0; the portion of the state $T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t)$ from time $-t$ to 0 is given by the portion of state $T(s, ai)$ from time 0 to time t.

The definition of shared-state automaton with sliding window follows.

Definition 30. *Given an shared-state automaton with time action $\mathcal{A} = (S, S0, A, E, T)$, and given a non-negative real constant $\mathcal{B}$, we define the shared-state automaton with sliding window $\mathcal{A}_{\mathcal{B}} = (S_{\mathcal{B}}, S0_{\mathcal{B}}, A_{\mathcal{B}}, E_{\mathcal{B}}, T_{\mathcal{B}})$ with:*

- $S_{\mathcal{B}} = [-\mathcal{B}, 0] \rightarrow S$,
- $S0_{\mathcal{B}} = \{\ s0_{\mathcal{B}} \in S_{\mathcal{B}} \mid \exists s0 \in S0, \forall t \in [-\mathcal{B}, 0]\ :\ s0_{\mathcal{B}}(t) = s0\ \}$
- $\forall ai \in A_i$ *we construct action* $ai_{\mathcal{B}} \in A_{\mathcal{B}}$,

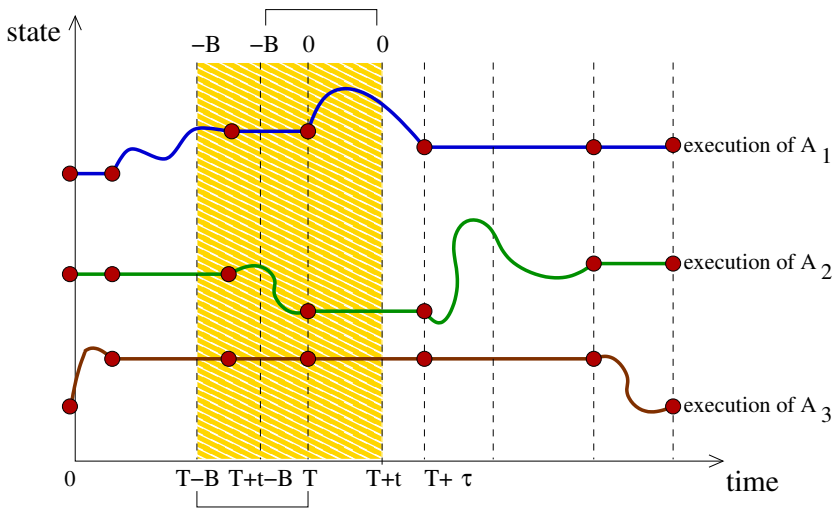


Figure 4.9: Pictorial representation of two overlapping windows defined on the execution in Figure 4.6. The time interval of the first window is $[T-\mathcal{B}, T]$, while the time interval of the second window is $[T-\mathcal{B}+t, T+t]$ with $t < \mathcal{B}$. These two time intervals overlap. The quantity τ denotes the time duration of the action executed by agent A_1 at time T. The shaded area represents the two overlapping time intervals.

- $\forall s_{\mathcal{B}} \in S_{\mathcal{B}}$, $\forall ai_{\mathcal{B}} \in A_{\mathcal{B}}$,

$$E_{\mathcal{B}}(s_{\mathcal{B}}, ai_{\mathcal{B}}) = (\forall s \in \mathcal{H}(s_{\mathcal{B}}) \ : \ E(s, ai))$$

- $\forall s_{\mathcal{B}} \in S_{\mathcal{B}}$, $\forall ai_{\mathcal{B}} \in A_{\mathcal{B}}$, *agent* i *chooses a state* $s \in S$, *with* $H(s, s_{\mathcal{B}})$ *and* $s(i) = s_{i\mathcal{B}}(0)$

  $\forall t \in [0, \tau_{s,ai}], t \leq \mathcal{B}$

$$\begin{aligned}(T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t))[-\mathcal{B}, -t] &= (s_{\mathcal{B}}[t-\mathcal{B}, 0])^{-t} \\ (T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t))[-t, 0] &= (T(s, ai)[0, t])^{-t}\end{aligned}$$

  $\forall t \in [0, \tau_{s,ai}], t > \mathcal{B}$

$$(T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t))[-\mathcal{B}, 0] = (T(s, ai)[t-\mathcal{B}, t])^{-t}$$

We notice that if $\mathcal{B} = 0$, then the system reduces to a shared-state system.

We next discuss equilibrium states of the shared-state automaton with sliding window. Given an equilibrium state $\hat{s} \in S$, the state $\hat{s}_{\mathcal{B}} \in S_{\mathcal{B}}$ consisting of the repetition of $\hat{s}$, i.e. $\hat{s}_{\mathcal{B}}(t) = \hat{s}$, $\forall t \in [-\mathcal{B}, 0]$ is an equilibrium state of $S_{\mathcal{B}}$. This follows by construction of the state space and action set. In general, an equilibrium state of $\mathcal{A}_{\mathcal{B}}$ requires that all its asynchronous views are equilibrium states of $\mathcal{A}$.

4.2.2 Line-Up with Sliding Window

For example, consider the Line-Up multi-agent system presented in Section 2.2.4. We next present the generalization of this automaton.

In the Line-Up automaton with sliding window, $S_{\mathcal{B}}$ maps the interval $[-\mathcal{B}, 0]$ to $\mathbb{R}^{N+1}$; this is because, by construction, $S = \mathbb{R}^{N+1}$. The initial set of states $S0_{\mathcal{B}}$ consists of the state $s0_{\mathcal{B}}$, defined as $\forall t \in [-\mathcal{B}, 0]$, $s0_{\mathcal{B}}(t) = s0$. For each action $a = \widehat{avg}_{l,i,r} \in A$, we construct an action $\widehat{avg}_{l,i,r_{\mathcal{B}}}$. By construction, this action is always enabled. When executed in state $s_{\mathcal{B}}$, agent i chooses an asynchronous view s of $s_{\mathcal{B}}$ with $s(i) = s_{i\mathcal{B}}(0)$ and executes action $\widehat{avg}_{l,i,r}$ in state s. The time duration of $a_{\mathcal{B}}$ is $\tau_{s,\widehat{avg}_{l,i,r}}$. For all $t \in [0, \tau_{s,\widehat{avg}_{l,i,r}}]$, if $t > \mathcal{B}$, then the state of $T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})$ at time t is given by $T(s, \widehat{avg}_{l,i,r})$ restricted to the interval $[t-\mathcal{B}, t]$. Instead, if $t \leq \mathcal{B}$, we have that $T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t)$ restricted to the interval $[-\mathcal{B}, -t]$ is equal to $s_{\mathcal{B}}$ restricted to the interval $[-\mathcal{B}+t, 0]$; $T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t)$ restricted to the interval $(-t, 0]$ is equal to $T(s, \widehat{avg}_{l,i,r})$ restricted to the interval $(0, t]$

The automaton modeling the Line-Up multi-agent system with dynamics and sliding window follows:

- $S_{\mathcal{B}} = [-\mathcal{B}, 0] \to \mathbb{R}^{N+1}$
- $S0_{\mathcal{B}} = \{s0_{\mathcal{B}}\}$, with $\forall t \in [-\mathcal{B}, 0]$: $s0_{\mathcal{B}}(t) = s0$,
- $A_{\mathcal{B}} = \{A_{i\mathcal{B}}\}_{i \in \{0,\ldots,N\}}$ with $A_{i\mathcal{B}} = \{\widehat{avg}_{l,i,r_{\mathcal{B}}}\}_{l<i<r}$,
- $E_{\mathcal{B}} : S_{\mathcal{B}} \times A_{\mathcal{B}} \to true$
- $T_{\mathcal{B}} : S_{\mathcal{B}} \times A_{\mathcal{B}} \to S_{\mathcal{B}}$, defined as $\forall a_{\mathcal{B}} = \widehat{avg}_{l,i,r_{\mathcal{B}}} \in A_{\mathcal{B}}$, $\forall s_{\mathcal{B}} \in S_{\mathcal{B}}$, agent i chooses a state $s \in S$, with $H(s, s_{\mathcal{B}})$ and $s(i) = s_{i\mathcal{B}}(0)$
  $\forall t \in [0, \tau_{s,\widehat{avg}_{l,i,r}}], t \leq \mathcal{B}$

$$\begin{aligned}(T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t))[-\mathcal{B}, -t] &= (s_{\mathcal{B}}[t-\mathcal{B}, 0])^{-t} \\ (T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t))[-t, 0] &= (T(s, \widehat{avg}_{l,i,r})[0, t])^{-t}\end{aligned}$$

$\forall t \in [0, \tau_{s,ai}], t > \mathcal{B}$

$$(T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t))[-\mathcal{B}, 0] = (T(s, \widehat{avg}_{l,i,r})[t-\mathcal{B}, t])^{-t}$$

4.2.3 Line-Up with Explicit Arbitrary Dynamics and Sliding Window

In this Section, we generalize the automaton for the Line-Up multi-agent system presented in Section 4.1.2. In the automaton presented in Section 4.1.2 agents have arbitrary dynamics. When the system executes action $avg_{l,i,r}$, agent i computes its new destination state using Equation 2.1 and evolves its current state towards its destination state.

In the Line-Up automaton with explicit arbitrary dynamics and sliding window, $S_{\mathcal{B}}$ maps the interval $[-\mathcal{B}, 0]$ to the pair $(\mathbb{R}^{N+1}, \mathbb{R}^{N+1})$. For each action $a = avg_{l,i,r} \in A$, we construct an action $avg_{l,i,r_{\mathcal{B}}}$. By construction, this action is always enabled. When executed in state $s_{\mathcal{B}}$, agent i chooses an asynchronous view s of $s_{\mathcal{B}}$ with $s(i) = s_{i\mathcal{B}}(0)$ and executes action $avg_{l,i,r}$ in state s.

The automaton modeling the Line-Up multi-agent system with explicit arbitrary dynamics and sliding window follows:

Definition 31. *The automaton $A_{\mathcal{B}} = (S_{\mathcal{B}}, S0_{\mathcal{B}}, A_{\mathcal{B}}, E_{\mathcal{B}}, T_{\mathcal{B}})$ modeling the Line-Up MAS with explicit arbitrary dynamics and sliding window has:*

- $S_{\mathcal{B}} = [-\mathcal{B}, 0] \to (\mathbb{R}^{N+1}, \mathbb{R}^{N+1})$
- $S0_{\mathcal{B}} = \{s0_{\mathcal{B}}\}$, *with* $\forall t \in [-\mathcal{B}, 0] \ : \ s0_{\mathcal{B}}(t) = s0$,
- $A_{\mathcal{B}} = \{avg_{l,i,r}\}_{l<i<r}$,
- $E_{\mathcal{B}} : S_{\mathcal{B}} \times A_{\mathcal{B}} \to true$
- $T_{\mathcal{B}} : S_{\mathcal{B}} \times A_{\mathcal{B}} \to S_{\mathcal{B}}$, *defined as* $\forall a_{\mathcal{B}} = avg_{l,i,r} \in A_{\mathcal{B}}$, $\forall s_{\mathcal{B}} \in S_{\mathcal{B}}$, *agent* i *chooses a state* $s \in S$, *with* $H(s, s_{\mathcal{B}})$ *and* $s(i) = s_{i\mathcal{B}}(0)$
  $\forall t \in [0, \tau_{s,avg_{l,i,r}}], t \leq \mathcal{B}$

$$\begin{aligned} (T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t))[-\mathcal{B}, -t] &= (s_{\mathcal{B}}[t - \mathcal{B}, 0])^{-t} \\ (T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t))[-t, 0] &= (T(s, avg_{l,i,r})[0, t])^{-t} \end{aligned}$$

$\forall t \in [0, \tau_{s,ai}], t > \mathcal{B}$

$$(T_{\mathcal{B}}(s_{\mathcal{B}}, a_{\mathcal{B}})(t))[-\mathcal{B}, 0] = (T(s, avg_{l,i,r})[t - \mathcal{B}, t])^{-t}$$

4.2.4 Lyapunov Function and Level Sets

In this Section, we present a feasible Lyapunov function for $\mathcal{A}_{\mathcal{B}}$ and discuss the structure of its level sets. This Lyapunov function is derived from a Lyapunov function defined on the automaton $\mathcal{A}$. Given a Lyapunov function $V : S \to \mathbb{P}$ on $\mathcal{A}$, we next construct a Lyapunov function $V_{\mathcal{B}} : S_{\mathcal{B}} \to \mathbb{P}$ on $\mathcal{A}_{\mathcal{B}}$ using the notion of asynchronous view.

The evaluation of $V_{\mathcal{B}}$ at state $s_{\mathcal{B}} \in S_{\mathcal{B}}$ is the maximum of the evaluations of V at the asynchronous views of $s_{\mathcal{B}}$. Formally,

Definition 32. *Given a Lyapunov function $V : S \to \mathbb{P}$ for $\mathcal{A}$, the Lyapunov function $V_{\mathcal{B}} : S_{\mathcal{B}} \to \mathbb{P}$ for $\mathcal{A}_{\mathcal{B}}$ is as follows, $\forall s_{\mathcal{B}} \in S_{\mathcal{B}}$,*

$$V_{\mathcal{B}}(s_{\mathcal{B}}) = \max_{s \in \mathcal{H}(s_{\mathcal{B}})} V(s)$$

Functions $V_{\mathcal{B}}$ and V have the same range $\mathbb{P}$.

We next discuss the structure of the level sets of $V_{\mathcal{B}}$. We next show that the level sets of $V_{\mathcal{B}}$ can be rewritten in terms of the level sets of V. We denote by $L_{k\mathcal{B}}$ the k-level set of $V_{\mathcal{B}}$, and by L_k the k-th level set of V. We have that $L_{k\mathcal{B}}$ can be expressed as

$$L_{k\mathcal{B}} = \{s_{\mathcal{B}} \in S_{\mathcal{B}} \mid \forall s \in \mathcal{H}(s_{\mathcal{B}}) \ : \ L_k(s)\}$$

i.e. $s_{\mathcal{B}} \in L_{k\mathcal{B}}$ if all its asynchronous views are in L_k. This holds because:

$$\begin{aligned} L_{k\mathcal{B}} &= \{s_{\mathcal{B}} \in S_{\mathcal{B}} \mid V_{\mathcal{B}}(s_{\mathcal{B}}) \leq k\} \\ &= \{s_{\mathcal{B}} \in S_{\mathcal{B}} \mid \forall s \in \mathcal{H}(s_{\mathcal{B}}) \ : \ V(s) \leq k\} \\ &= \{s_{\mathcal{B}} \in S_{\mathcal{B}} \mid \forall s \in \mathcal{H}(s_{\mathcal{B}}) \ : \ L_k(s)\} \end{aligned}$$

where the first inequality follows by construction; the second inequality follows by definition of $V_{\mathcal{B}}$ and the last inequality follows by definition of L_k. Furthermore, if P_k denotes the predicate defining L_k, we have that

$$L_{k\mathcal{B}} = \{s_{\mathcal{B}} \in S_{\mathcal{B}} \mid \forall s \in \mathcal{H}(s_{\mathcal{B}}) \ : \ P_k(s)\}$$

i.e. $s_{\mathcal{B}} \in L_{k\mathcal{B}}$ if P_k holds in all asynchronous views of $s_{\mathcal{B}}$. This holds because:

$$\begin{aligned} L_{k\mathcal{B}} &= \{s_{\mathcal{B}} \in S_{\mathcal{B}} \mid \forall s \in \mathcal{H}(s_{\mathcal{B}}) \ : \ L_k(s)\} \\ &= \{s_{\mathcal{B}} \in S_{\mathcal{B}} \mid \forall s \in \mathcal{H}(s_{\mathcal{B}}) \ : \ P_k(s)\} \end{aligned}$$

We denote by $P_{k\mathcal{B}}$ the predicate corresponding to the k-th level set of $V_{\mathcal{B}}$. This predicate is defined as $\forall s \in S_{\mathcal{B}}$

$$P_{k\mathcal{B}}(s_{\mathcal{B}}) \equiv (\forall s \in \mathcal{H}(s_{\mathcal{B}}) \ : \ P_k(s))$$

We next discuss some properties of the family of level sets of $V_{\mathcal{B}}$. We prove that is the family of level sets $L_{k k\in\mathbb{P}}$ is in conjunctive form, then family of level sets of $V_{\mathcal{B}}$ is in conjunctive form as well. The Lemma follows:

Lemma 12. *If $\{L_k\}_{k\in\mathbb{P}}$ is in conjunctive form then $\{L_{k\mathcal{B}}\}_{k\in\mathbb{P}}$ is in conjunctive form.*

Proof. Consider an arbitrary $k \in \mathbb{P}$. By construction,

$$L_{k\mathcal{B}} = \{s_{\mathcal{B}} \in S_{\mathcal{B}} \mid \forall s \in \mathcal{H}(s_{\mathcal{B}}) \ : \ P_k(s)\}$$

where P_k is the predicate of L_k. By assumption, P_k is in conjunctive form, i.e. $\forall s \in S$,

$$P_k(s) \equiv \left(\bigwedge_{i \in \{1,\ldots,N\}} P_{(k,i)}(s(i)) \right)$$

Hence, $L_{k\mathcal{B}}$ can be rewritten as

$$L_{k\mathcal{B}} = \left\{ s_{\mathcal{B}} \in S_{\mathcal{B}} \mid \bigwedge_{i \in \{1,\ldots,N\}} P_{(k,i)_{\mathcal{B}}}(s_{i\mathcal{B}}) \right\} \tag{4.1}$$

where $P_{(k,i)_{\mathcal{B}}}$ is defined as

$$P_{(k,i)_{\mathcal{B}}}(s_{i\mathcal{B}}) \equiv \left(\forall t \in [-\mathcal{B}, 0] \ : \ P_{(k,i)}(s_{i\mathcal{B}}(t)) \right)$$

The predicate $P_{(k,i)_{\mathcal{B}}}$ depends only on the state of the i-th agent. Hence, $L_{k\mathcal{B}}$ is in conjunctive form. Equation 4.1 holds, since,

$$\begin{aligned} L_{k\mathcal{B}} &= \left\{ s_{\mathcal{B}} \in S_{\mathcal{B}} \mid \left(\forall s \in \mathcal{H}(s_{\mathcal{B}}) \ : \ \bigwedge_{i \in \{1,\ldots,N\}} (P_{(k,i)}(s(i))) \right) \right\} \\ &= \left\{ s_{\mathcal{B}} \in S_{\mathcal{B}} \mid \bigwedge_{i \in \{1,\ldots,N\}} (\forall t \in [-\mathcal{B}, 0] \ : \ P_{k,i}(s_{i\mathcal{B}}(t))) \right\} \\ &= \left\{ s_{\mathcal{B}} \in S_{\mathcal{B}} \mid \bigwedge_{i \in \{1,\ldots,N\}} P_{(k,i)_{\mathcal{B}}}(s_{i\mathcal{B}}) \right\} \end{aligned}$$

where the first inequality follows by construction; the second inequality follows by definition of the set of asynchronous views and the third inequality follows by definition of $P_{(k,i)_{\mathcal{B}}}$. □

4.2.5 Stability

In this Section, we present a stability result for shared-state automata with sliding window. The stability property of an equilibrium state of $\mathcal{A}_{\mathcal{B}}$ is derived from the stability of the corresponding equilibrium state in $\mathcal{A}$. Specifically, given an equilibrium state $\hat{s} \in S$ and a Lyapunov function V for $\mathcal{A}$, we derive conditions on the structure of V that ensure that $V_{\mathcal{B}}$ is a certificate of the stability of $\hat{s}_{\mathcal{B}} \in S_{\mathcal{B}}$. These conditions require the level sets of V to be in conjunctive form. The main result follows.

Theorem 13. *If*

G1. V satisfies Assumptions B1-3 of Theorem 8,

G2. family $\{L_k\}_{k \in \mathbb{P}}$ of V is in conjunctive form

then $\hat{s}_{\mathcal{B}}$ is a stable equilibrium state of $A_{\mathcal{B}}$.

Proof. Our goal is to show that $V_{\mathcal{B}}$ satisfies Assumptions B1-3 of Theorem 8. By construction and assumptions on V, B1-2 hold. We next prove that Assumption B3 holds, i.e. the family of level sets of $V_{\mathcal{B}}$ is stable.

We denote the family of level set of $V_{\mathcal{B}}$ by $\{L_{k\mathcal{B}}\}_{k\in\mathbb{P}}$. Using Assumption G2 and Lemma 6, the family $\{L_{k\mathcal{B}}\}_{k\in\mathbb{P}}$ is stable if and only if $\forall s_{\mathcal{B}} \in S_{\mathcal{B}}, \forall ai_{\mathcal{B}} \in A_{\mathcal{B}}$, with $E_{\mathcal{B}}(s_{\mathcal{B}}, ai_{\mathcal{B}})$, $\forall s \in \mathcal{H}(s_{\mathcal{B}})$, with $s(i) = s_{i\mathcal{B}}(0)$

$$\forall t \in [0, \tau_{s,ai}] \ : \ V_{\mathcal{B}}(T_{\mathcal{B}}(s_{\mathcal{B}}, ai_{\mathcal{B}})(t)) \leq V_{\mathcal{B}}(s_{\mathcal{B}})$$

We next show that this condition holds.

Consider an arbitrary state $s_{\mathcal{B}} \in S_{\mathcal{B}}$, an arbitrary action $ai_{\mathcal{B}} \in A_{\mathcal{B}}$, with $E_{\mathcal{B}}(s_{\mathcal{B}}, ai_{\mathcal{B}})$, and an arbitrary asynchronous view s, with $s(i) = s_{i\mathcal{B}}(0)$. Denote by $s'_{\mathcal{B}}(t)$ the state $T_{\mathcal{B}}(s_{\mathcal{B}}, ai_{\mathcal{B}})(t)$ for all $t \in [0, \tau_{s,ai}]$. Our goal is to show that $s'_{\mathcal{B}}(t) \in L_{k\mathcal{B}}$, where $k = V_{\mathcal{B}}(s_{\mathcal{B}})$. This implies that $V_{\mathcal{B}}(s'_{\mathcal{B}}(t)) \leq V_{\mathcal{B}}(s_{\mathcal{B}})$

Using Lemma 12, since $\{L_k\}_{k\in\mathbb{P}}$ is in conjunctive form, we have that $L_{k\mathcal{B}}$ is in conjunctive form and has the following structure

$$L_{k\mathcal{B}} = \left\{ \bar{s} \in S_{\mathcal{B}} \mid \bigwedge_{j\in\{1,\ldots,N\}} P_{(k,j)_{\mathcal{B}}}(\bar{s}_{i_{\mathcal{B}}}) \right\}$$

with

$$P_{(k,j)_{\mathcal{B}}}(\bar{s}i_{\mathcal{B}}) \equiv \big(\forall t \in [-\mathcal{B}, 0] \ : \ P_{(k,j)}(\bar{s}_{i_{\mathcal{B}}}(t))\big)$$

where $P_{(k,j)}$ is the j-th predicate in P_k. Hence,

$$s'_{\mathcal{B}} \in L_{k\mathcal{B}} \Leftrightarrow \ \big(\forall j \in \{1, \ldots, N\} \ : \ P_{(k,j)_{\mathcal{B}}}(s'_{j_{\mathcal{B}}})\big)$$

By construction, predicate $P_{(k,j)_{\mathcal{B}}}$ depends only on agent j. Consider an arbitrary agent $j \in \{1, \ldots, N\}$, with $j \neq i$. When action $ai_{\mathcal{B}}$ is executed, no new values for agent j are added to the system. Hence, $P_{(k,j)_{\mathcal{B}}}(s'_{j_{\mathcal{B}}})$ holds, since $P_{(k,j)_{\mathcal{B}}}(s_{j_{\mathcal{B}}})$ holds. Consider agent i. By construction of $V_{\mathcal{B}}$, $s \in L_k$. By assumption, function V satisfies B3; hence, all new values of i added to the system satisfy predicate $P_{(k,i)}$, i.e. $P_{(k,i)_{\mathcal{B}}}(s'_{i_{\mathcal{B}}})$ holds. □

4.2.6 Convergence

In this Section, we discuss convergence property of shared-state automata with sliding window. Similarly to the previous subsection, we derive convergence of the equilibrium state $\hat{s}_{\mathcal{B}} \in S_{\mathcal{B}}$ from the convergence property of the equilibrium state $s \in S$. In this case, we require the level sets of V

to be in conjunctive form as well.

Theorem 14. *If*

H1. V satisfies Assumptions C1-5 of Theorem 9,

H2. family $\{L_k\}_{k\in\mathbb{P}}$ of V is in conjunctive form

then $\mathcal{A}_\mathcal{B}$ converges to $\hat{s}_\mathcal{B}$.

Proof. Our goal is to show that $V_\mathcal{B}$ satisfies the assumptions of Theorem 9. Assumptions C1-2 and C5 hold, by hypothesis and by construction of $V_\mathcal{B}$. Proof of C3 is similar to the proof of B3 in Theorem 13 and, therefore, is not reported. We next prove Condition C4.

Fix an arbitrary $k \in \mathbb{P}$. We denote by L_k the k-th level set of V and by $L_{k\mathcal{B}}$ the k-th level set of $V_\mathcal{B}$.

Using Assumption H2 and Lemma 12, we have that $L_{k\mathcal{B}}$ is in conjunctive form.

Using Assumptions H2 and C4, we have that if $\mathcal{A}$ starts in L_k, then it eventually enters and remains in $L_q \subset L_p$. We denote by a_i the action executed when the system enters L_q, and i denotes the agent executing the action. By Assumption H2, $P_{(q,i)} \Rightarrow P_{(k,i)}$ and $P_{(q,j)} = P_{(k,j)}$ for all $j \neq i$.

Using Assumption C4 on V, we have that if $A_\mathcal{B}$ starts in $L_{k\mathcal{B}}$, then eventually enters in $s_\mathcal{B} \in L_{k\mathcal{B}}$ where it executes a_i on an asynchronous view $s \in \mathcal{H}(s_\mathcal{B})$ with $s(i) = s_{i\mathcal{B}}(0)$. Denote by $\bar{s}_\mathcal{B}$ the state $\bar{s}_\mathcal{B} = T_\mathcal{B}(s_\mathcal{B}, a_\mathcal{B})(\tau_{s,a_i})$. Predicate $P_{(q,j)}(\bar{s}_{j_\mathcal{B}}(t))$ holds $\forall j \neq i, \forall t \in [-\mathcal{B}, 0]$; this is because $P(k,j)(\bar{s}_{j_\mathcal{B}}(t))$ holds. Furthermore, in any state of any execution fragment starting from $\bar{s}_\mathcal{B}$, predicate $P_{(q,j)}$ holds. Instead, in the case of agent i, there exists $\hat{t} \in [-\tau_{s,a_i}, 0]$ such that $\forall t \geq \hat{t}$, predicate $P_{(q,i)}(\bar{s}_{i_\mathcal{B}}(t))$ holds. Hence, we have that predicate $P_{(q,i)}$ holds in any new state of any execution fragment starting from $\bar{s}_\mathcal{B}$; this is because it holds in the initial state of the execution and it continues to hold since P_q is stable. Hence, the system eventually enters a state $s'_\mathcal{B}$ such that $\forall t \in [-\mathcal{B}, 0]$, $P_{(q,i)}(s'_{i\mathcal{B}}(t))$. This implies that predicate $L_{q\mathcal{B}}$ holds eventually; this is because predicate $L_{q\mathcal{B}}$ is in conjunctive form. □

4.3 Message-Passing Multi-Agent Systems

In this Section, we present message-passing multi-agent systems with bounded delay. We model them using the automaton with sliding window model and derive conditions on the stability and convergence properties of their equilibrium states.

4.3.1 Message-Passing Communication Model

Message-passing multi-agent systems, informally presented in Chapter 1, consist of a collection of agents communicating by means of an unreliable communication medium. In these systems, agents

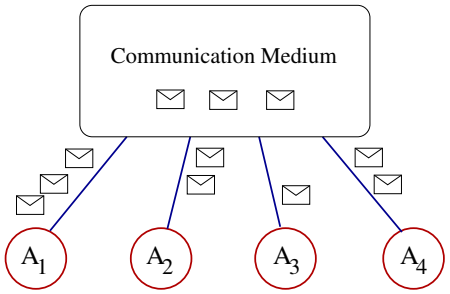


Figure 4.10: Pictorial representation of the message-passing communication model. Circles represent agents and links represent communication channels. Agents send messages to each other using the communication medium.

communicate via message-passing, i.e. they exchange messages, that may be lost, delayed, duplicated or received out-of-order.

In our formalization, agents do not interact directly with other agents; instead, as shown in Figure 4.10, agents interact through a communication medium. They send messages to the medium and receive messages from the medium. The task of the medium is to implement the communication protocol. The medium decides on the set of recipients of the message. For example, in the case of a broadcast communication protocol, the medium sends the message to all agents. The medium can delete, delay, duplicate or change the order of messages in transit. For example, it can decide to delete a message in transit, deliver a more recent message before an older one, deliver a message more than one time, or deliver a message to a proper subset of its recipients. However, it cannot modify the content of the messages in transit.

This formalization is very general and can model unicast, multicast or broadcast communication protocols. The content of the messages in transit is some function defined on the state of the agent sending the message. In our model, each agent stores some set of variables describing its local state. It also stores for each other agent the last received message.

We assume fairness in the transmission; we do not allow permanent partitions between communicating agents. If the communication allows agent j to receive messages from agent i, it is not possible that all messages sent by i are deleted. Fairness ensures that j will receives infinitely many messages sent from i. We also assume that each agent sends infinitely many messages; however, the number of messages sent within a finite time interval is finite.

In our model, we assume a communication medium with bounded, but unknown, transmitting delay. This unknown constant is denoted by b. This class of message-passing systems is called message-passing systems with bounded delay. A message received by the medium at time t is either sends to its recipients by time $t + b$ or deleted.

4.3.2 Message-Passing Automaton

In this Section, we model a message-passing system using the shared-state automaton with sliding window. Throughout this Section, we denote by $\mathcal{A}_{mp}$ the automaton of the message-passing system.

In the message-passing system model, the action set of an agent consists of the *send*, *receive* actions and of a set of internal actions. When an agent executes a *send* action, it broadcasts some value, that can be its current state or can be some function of its current value. In the case of a *send* action, the agent does not need to access the state of the other agents. When an agent executes a *receive* action, it receives a message and executes an action based on its current state and the last received messages from its neighbors. We assume that each agent stores in its local state a vector of length N, where entry j contains the last message received from agent j. As explained later, the message-passing communication mechanism is modeled using the notion of asynchronous view. Intuitively, the local vector storing the last received messages corresponds to the agent accessing the state of the other agents at some time in the past.

Automaton $\mathcal{A}_{mp}$ is defined as the shared-state automaton with sliding window $\mathcal{A}_{\mathcal{B}}$ having $\mathcal{B}$ equals to the transmitting delay b. In this automaton, a state of $\mathcal{A}_{mp}$ describes an execution fragment of the message-passing system, where the agents execute *send*, *receive* or internal actions. The set of initial states of $\mathcal{A}_{mp}$ models a system where all communication channels are initially empty. When executing a *send* action in state s_{mp}, the system does not change. When executing a *receive* action in state s_{mp}, agent i constructs the vector of the last received messages using the notion of asynchronous view. Differently from the general model where the asynchronous view is chosen nondeterministically, in the message-passing model, the N-tuple of times $t \in [-\mathcal{B}, 0]^N$, is such that for all $j \neq i$, $s_{j\,\mathcal{B}}(t(j))$ is the state of agent j when executing action *send*. The nonnegative value $-t(j)$ represents the delay of the message sent by j and received by i.

In this model, agents do not physically send or receive messages. When an agent executes a *receive* action, it constructs the vector of the last received messages using the notion of asynchronous view. For example, consider the execution in Figure 4.11. Agent A_1 executes a *receive* action and consider as the last received message the state of agent A_2 at time t_2 and the state of agent A_3 at time t_3. This model is very general and allows modeling delayed, lost, duplicated or received out-of-order messages. Given an execution fragment, a message sent at time t from agent i is duplicated if some agent j, with $j \neq i$, executes two *receive* actions using as last received message from i the state of agent i at time t. The message sent at time t is lost if no agent in the interval $[t, t + \mathcal{B}]$ accesses the state of agent i at time t. Two messages sent by i at time t_1 and t_2 are received out-of-order if some agent j, with $j \neq i$ executes two *receive* actions: the first one with the state at time t_2 of agent i and the second one with the state at time t_1 of agent i.

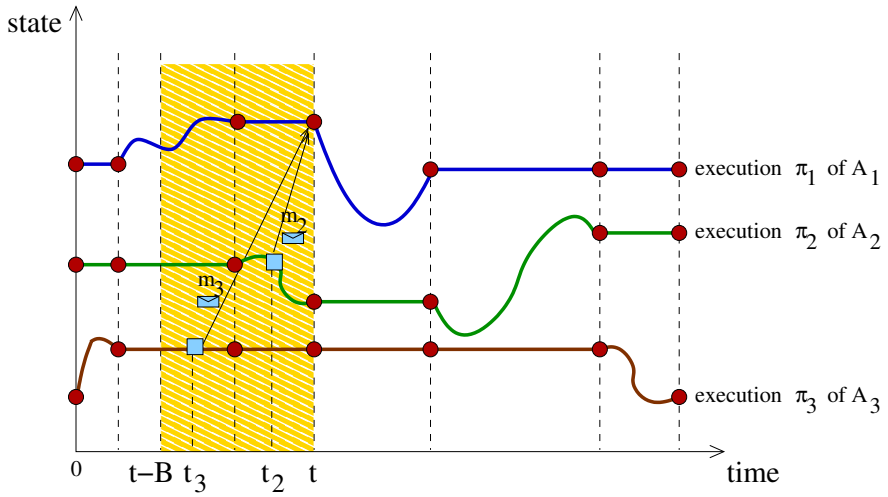


Figure 4.11: An execution of a multi-agent system consisting of three agents where at time t, agent A_1 receives the two messages m_2 and m_3. Message m_2 has been sent by agent A_2 at time t_2 and message m_3 by agent A_3 at time t_3. Message m_2 stores $\pi_2(t_2)$ and message m_3 stores $\pi_3(t_3)$. Agent A_1 executes an action at time t on the state $(\pi_1(t), m_2, m_3)$. The communication delay is bounded by $\mathcal{B}$. Agent A_1 at time t can receive any message sent in the time interval $[t-\mathcal{B}, t]$. In this Figure, this interval is represented by the shaded area.

4.3.3 Stability and Convergence

In this Section, we discuss stability and convergence properties of message-passing multi-agent systems. We show that stability and convergence of a message-passing system can be derived from the stability and convergence of the corresponding shared-state multi-agent system. This is because we have shown that a message-passing system with bounded delay is a special case of shared-state system with sliding window.

We denote by $\mathcal{A}$ a shared-state automaton, by $\hat{s}$ an equilibrium state of $\mathcal{A}$ and V a Lyapunov function on $\mathcal{A}$. We denote by $\hat{s}_{mp}$ the state in $\mathcal{A}_{mp}$ corresponding to $\hat{s}$. We have that the following two Theorems hold:

Theorem 15. *If G1-2 hold, then $\hat{s}_{mp}$ is a stable equilibrium state of $\mathcal{A}_{mp}$.*

Proof. It follows directly from Theorem 13. □

Theorem 16. *If H1-2 hold, then $\mathcal{A}_{mp}$ converges to $\hat{s}_{mp}$.*

Proof. It follows directly from Theorem 14. □

4.4 Multi-Agent Systems with Concurrent Actions

In this Section, we model multi-agent systems where agents can execute actions concurrently. We use the automaton with timed action model to model both shared-state and message-passing systems.

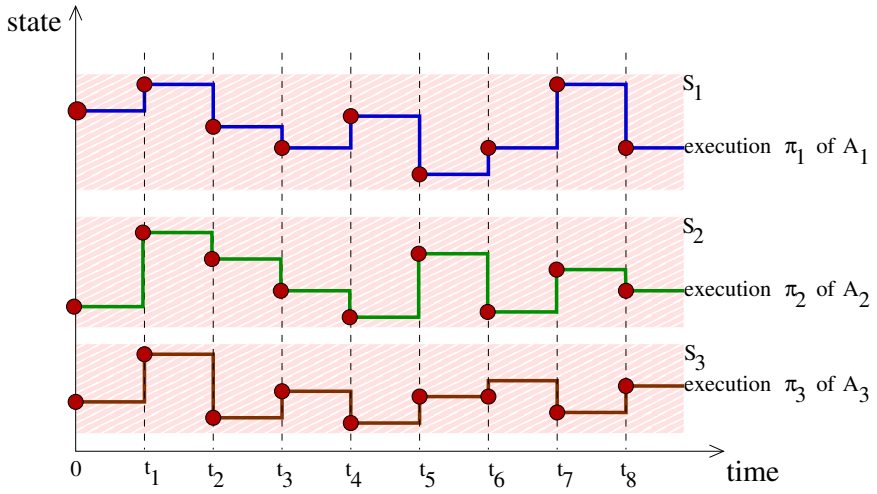


Figure 4.12: An execution of a multi-agent system consisting of three agents where agents execute discrete actions concurrently. The three shaded areas represent the state spaces of the three agents denoted by S_1, S_2 and S_3.

4.4.1 Shared-State Multi-Agent Systems with Concurrent Actions

We model shared-state multi-agent systems where actions of an agent can read the state of all agents in the system, but only modify its own state.

Before proceeding with the definitions, we denote by S_i the state of agent i in the multi-agent system, and by S the cartesian product of the states of the agents, i.e. $S = S_1 \times \ldots \times S_N$. We denote by $S0_i$ the set of initial states of agent i. We assume that associated with each agent there is a set of actions A_i. For each action $a_i \in A_i$, action a_i is enabled in state $s \in S$ if $E_i(s, a_i)$ holds; when executed, it has time duration τ_{s,a_i} and it evolves the state of agent i according to the function $f_{s,a_i} : [0, \tau_{s,a_i}] \rightarrow S_i$.

We next discuss two shared-state models: a shared-state automaton with discrete concurrent actions, and a shared-state automaton with time-varying concurrent actions.

4.4.1.1 Shared-State Multi-Agent Systems with Discrete Actions

In this Section, we discuss shared-states multi-agent systems with concurrent discrete actions. At each point of the execution, all agents pick an action and execute it. For example, in Figure 4.12, we present an execution of a multi-agent system consisting of three agents.

The automaton $\mathcal{A}_{\mathcal{DC}} = (S_{\mathcal{DC}}, S0_{\mathcal{DC}}, A_{\mathcal{DC}}, E_{\mathcal{DC}}, T_{\mathcal{DC}})$ modeling a shared-state multi-agent system with concurrent discrete actions is defined as follows. Its state space, set of initial states and set of action are defined as cartesian products. Specifically, its state space $S_{\mathcal{DC}}$ is the cartesian product of the state spaces of the agents, i.e. $S_{\mathcal{DC}} = S_1 \times \ldots \times S_N$, with S_i being the state space of agent i. Similarly, its set of initial states $S0_{\mathcal{DC}}$ is the cartesian product of the initial sets of states of its agents,

i.e. $S0_{\mathcal{DC}} = S0_1 \times \ldots \times S0_N$, with $S0_i$ being the set of initial states of agent i. Its set of actions $A_{\mathcal{DC}}$ is the cartesian product of the sets of actions of the agents, i.e. $A_{\mathcal{DC}} = A_1 \times \ldots \times A_N$, with A_i being the set of actions of agent i. An action $a \in A_{\mathcal{DC}}$ is of the form $a = (a1, \ldots, aN)$, where $a(i)$ is the action executed by agent i. An action $a \in A_{\mathcal{DC}}$ is enabled in state $s \in S_{\mathcal{DC}}$ if for each agent i action $a(i)$ is enabled in state s. When executing action a in state s, the i-th component of the post-state is given by the execution of $a(i)$ in state s i.e. $T_{\mathcal{DC}}(s, a) = (T_1(s, a(1)), \ldots, T_i(s, a(i)), \ldots, T_N(s, a(N)))$ with T_i being the transition function of agent i.

We next formally present the automaton.

Definition 33. *The automaton $\mathcal{A}_{\mathcal{DC}} = (S_{\mathcal{DC}}, S0_{\mathcal{DC}}, A_{\mathcal{DC}}, E_{\mathcal{DC}}, T_{\mathcal{DC}})$ modeling a shared-state multi-agent system with N agents and concurrent discrete actions has:*

- $S_{\mathcal{DC}} = S_1 \times \ldots \times S_N$,
- $S0_{\mathcal{DC}} = S0_1 \times \ldots \times S0_N$,
- $A_{\mathcal{DC}} = A_1 \times \ldots \times A_N$,
- $\forall s \in S_{\mathcal{DC}}$, $\forall a \in A_{\mathcal{DC}}$,
$$E_{\mathcal{DC}}(s, a) = \bigwedge_{i \in \{1,\ldots,N\}} E_i(s, a(i))$$
- $\forall s \in S_{\mathcal{DC}}$, $\forall a \in A_{\mathcal{DC}}$,
$$T_{\mathcal{DC}}(s, a) = (T_1(s, a(1)), \ldots, T_i(s, a(i)), \ldots, T_N(s, a(N)))$$

As an example, we consider a generalization of the Line-Up multi-agent system presented in Section 2.1.2. In this generalization, we allow agents to execute actions concurrently. This generalization defines a nondeterministic system where at each time of the execution, all agents choose an action. Agents i, with $0 < i < N$, chooses an action in A_i, with $A_i = \{avg_{l,i,r}\}_{l<i<r}$ where $avg_{l,i,r}$ implements the updating rule in Equation 2.1. Agent 0 and N are stationary. The automaton follows.

Definition 34. *The automaton modeling the Line-Up multi-agent discrete system with concurrent actions has the following structure:*

- $S_{\mathcal{DC}} = \mathbb{R}^{N+1}$, *since the state space of each agent is* $\mathbb{R}$.
- $S0_{\mathcal{DC}} = \{s0\}$, *with* $s0$ *being the initial configuration*
- $\mathcal{A}_{\mathcal{DC}} = A_1 \times \ldots \times A_{N-1}$ *where* $A_i = \{avg_{l,i,r}\}_{l<i<r}$
- $E_{\mathcal{DC}} : S_{\mathcal{DC}} \times \mathcal{A}_{\mathcal{DC}} \rightarrow true$

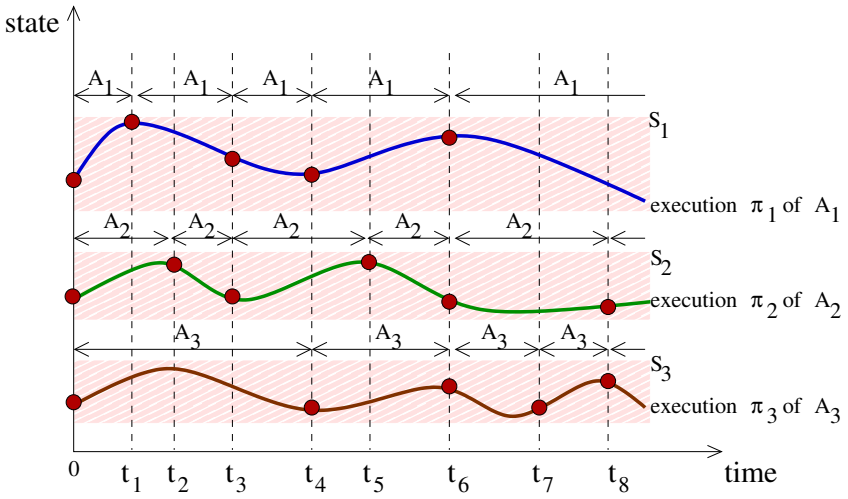


Figure 4.13: An execution of a multi-agent system consisting of three agents where agents execute actions concurrently. At time 0, all three agents execute an action. The time duration of the action executed by agent A_1 is t_1; the time duration of the action executed by agent A_2 is t_2 and the time duration of the action executed by agent A_3 is t_4. At time t_1 agent A_1 executes an action for $t_3 - t_1$ time units; at time t_3, it executes an action for $t_4 - t_3$ time units; at time t_4, it executes an action for $t_6 - t_4$ time units. Similarly, at time t_2 agent A_2 executes an action for $t_3 - t_2$ time units; at time t_3, it executes an action for $t_5 - t_3$ time units; at time t_5, it executes an action for $t_6 - t_5$ time units; at time t_6, it executes an action for $t_8 - t_6$ time units. At time t_4 agent A_3 executes an action for $t_6 - t_4$ time units; at time t_6, it executes an action for $t_7 - t_6$ time units; at time t_7, it executes an action for $t_8 - t_7$ time units. The three shaded areas represent the state spaces of the three agents denoted by S_1, S_2 and S_3. Dark filled circles represent end-states of executions of actions.

- $T_{\mathcal{DC}} : S_{\mathcal{DC}} \times \mathcal{A}_{\mathcal{DC}} \to S_{\mathcal{DC}}$, *defined as* $\forall a = (a_1, \ldots, a_i = avg_{l,i,r}, \ldots, a_{N-1}) \in \mathcal{A}_{\mathcal{DC}}$, $\forall s \in S_{\mathcal{DC}}$,

$$T_{\mathcal{DC}}(s,a)(i) = \begin{cases} s(i) & i \in \{0, N\} \\ \left(\frac{r-i}{r-l}s(l) + \frac{i-l}{r-l}s(r)\right) & 0 < i < N \end{cases}$$

4.4.1.2 Shared-State Multi-Agent Systems with Timed Actions

In this Section, we discuss a more general automaton model where actions are executed concurrently and can have different time durations. We denote by $\mathcal{A}_{\mathcal{C}} = (S_{\mathcal{C}}, S0_{\mathcal{C}}, A_{\mathcal{C}}, E_{\mathcal{C}}, T_{\mathcal{C}})$ the automaton with timed actions modeling a shared-state system with concurrent timed actions. When concurrent timed actions are executed, at each time of the execution an agent can either pick an action and execute it or it can been executing an action started at some time in the past. For example, in Figure 4.13, we present the execution of a multi-agent system consisting of three agents. In this execution, agent A_1 and agent A_2 at time t_3 pick an action and execute it; instead, agent A_3 at time t_3 is executing an action started at time 0.

This is more general than the model with concurrent discrete actions where at each time of the

execution all agents pick an action. For this reason, differently from the automaton with discrete concurrent actions, a state in $\mathcal{A}_\mathcal{C}$ requires to store the state of each agent and, in case the agent is executing an action, the function describing the behaviour of the action for the remaining time interval. This is equivalent to store for each agent a function from a finite closed time interval to S_i with S_i being the state space of agent i. The length of the time interval is equal to the remaining duration of the action, and the function describes the behaviour of the action in this interval. We assume that the finite time interval has left extreme equals to 0. If the agent is not currently executing an action, the function maps the interval $[0, 0]$ to its current state. We denote by $\mathcal{I}$ the set of finite closed intervals with left extreme 0, i.e.

$$\mathcal{I} = \{[0, t] \mid t \geq 0\}$$

A state $s_\mathcal{C}$ in $S_\mathcal{C}$ is a tuple of length N, where the i-th entry is a function from $\mathcal{I}$ to S_i, i.e. $s_\mathcal{C}$ is of the form $(g_1, g_2, \ldots, g_N)$ where g_i is a function with domain $[0, t_i]$ and defined as $g_i : [0, t_i] \to S_i$, for all $i \in \{1, 2, \ldots, N\}$. Given $s_\mathcal{C} \in S_\mathcal{C}$, we denote by $s_\mathcal{C}.g_i$ the i-th entry of $s_\mathcal{C}$ and by $s_\mathcal{C}.t_i$ the right extreme of the domain of function $s_\mathcal{C}.g_i$. Given a state $s_\mathcal{C}$, we denote by $s^{(0)}$ the tuple $(s1, \ldots, sN)$ where $si = s_\mathcal{C}.g_i(0)$ for all $i \in \{1, \ldots N\}$; $s^{(0)}$ is the tuple where each entry stores the evaluation of function $s_\mathcal{C}.g_i$ at time 0. This tuple represents the current state of the multi-agent system. By construction, this tuple is well-defined. A state $s_\mathcal{C} \in S_\mathcal{C}$ is an initial state of $\mathcal{A}_\mathcal{C}$ if for all $i \in \{1, \ldots, N\}$ the state of agent i at time 0 is an initial state of agent i and if $s_\mathcal{C}.t_i > 0$, there exists an action $a_i \in A_i$ and time $t' \in [0, \tau_{s^{(0)}, a_i}]$ such that function $s_\mathcal{C}.g_i$ corresponds to the behaviour of action a_i in state $s^{(0)}$ in the interval $[t', \tau_{s^{(0)}, a_i}]$. The set of actions $A_\mathcal{C}$ is the cartesian product of the set of actions of the agents, i.e. $A_\mathcal{C} = A_1 \times \ldots \times A_N$. By construction, the set of agents that can execute an action in state $s_\mathcal{C}$ are the agents with $s_\mathcal{C}.t_i = 0$; this is because they are the only agents not executing actions at the current time. Given $s_\mathcal{C}$, we define this set of agents as follows:

$$\mathit{agents}(s_\mathcal{C}) = \{i \in \{1, \ldots, N\} \mid s_\mathcal{C}.t_i = 0\}$$

An action $a_\mathcal{C} \in A_\mathcal{C}$ is enabled in the state $s_\mathcal{C} \in S_\mathcal{C}$ if for all agents not already executing an action, the corresponding action in $a_\mathcal{C}$ is enabled in $s^{(0)}$. Associated with each agent i, there is a time $\hat{t}_i$. This time is the time duration of action $a(i)$ in state $s^{(0)}$ if the agent is not currently executing an action (i.e. $i \in \mathit{agents}(s_\mathcal{C})$), and it is the remaining time duration $s_\mathcal{C}.t_i$ otherwise ($i \notin \mathit{agents}(s_\mathcal{C})$). The time duration of $a_\mathcal{C}$, denoted by τ_m, is the minimum of the $\{\hat{t}_i\}_{i \in \{1, \ldots, N\}}$. This time defines the next time instant when there are agents that have completed the execution of some action and are ready to execute a new one. The behavior of $a_\mathcal{C}$ depends on the specific agent. If agent i is already executing an action, i.e $i \notin \mathit{agents}(s_\mathcal{C})$, then for all $t \in [0, \tau_{s^{(0)}, a_\mathcal{C}(i)}]$, the state of the agent i at time t is the restriction of function $s_\mathcal{C}.g_i$ in the interval $[t, s_\mathcal{C}.t_i]$. If agent i is not already executing an

action, i.e $i \in agents(s_{\mathcal{C}})$, then for all $t \in (0, \tau_{s^{(0)},a_{\mathcal{C}}(i)}]$, agent i executes action $a_{\mathcal{C}}(i)$ in state $s^{(0)}$ and the state of the agent i at time t is the restriction of function $f_{s^{(0)},a_{\mathcal{C}}(i)}$ in the interval $[t, \tau_{s^{(0)},a_{\mathcal{C}}(i)}]$.

The definition of the automaton follows.

Definition 35. *A shared-state MAS with N agents and concurrent actions can be modeled as an automaton with timed action $\mathcal{A}_{\mathcal{C}} = (S_{\mathcal{C}}, S0_{\mathcal{C}}, A_{\mathcal{C}}, E_{\mathcal{C}}, T_{\mathcal{C}})$ where*

- $S_{\mathcal{C}} = ((\mathcal{I} \to S_1), \ldots, (\mathcal{I} \to S_N))$, *where S_i the state of agent i,*
- $s_{\mathcal{C}} \in S0_{\mathcal{C}}$ *if*

$$\begin{aligned} \forall i \in \{1, \ldots, N\} \quad &: \quad s_{\mathcal{C}}.g_i(0) \in S0_i \\ \forall s_{\mathcal{C}}.t_i > 0 \quad &: \quad \exists a_i \in A_i,\ \exists t \in [0, \tau_{s^{(0)},a_i}] \ : \ s_{\mathcal{C}}.g_i = (f_{s^{(0)},a_i}[t, \tau_{s^{(0)},a_i}])^{-t} \end{aligned}$$

- $\mathcal{A}_{\mathcal{C}} = (A_1 \times A_2 \times \ldots \times A_N)$, *where A_i is the set of actions of agent i;*
- $\forall s_{\mathcal{C}} \in S_{\mathcal{C}}$, $\forall a_{\mathcal{C}} \in \mathcal{A}_{\mathcal{C}}$,

$$E_{\mathcal{C}}(s_{\mathcal{C}}, a_{\mathcal{C}}) = \Big(\forall i \in agents(s_{\mathcal{C}}) \ : \ E_i(s^{(0)}, a_{\mathcal{C}}(i))\Big)$$

- $\forall s_{\mathcal{C}} \in S_{\mathcal{C}}$, $\forall a_{\mathcal{C}} \in \mathcal{A}_{\mathcal{C}}$,
  let

$$\tau_m = \min\big(\{\tau_i \mid \tau_i = \tau_{s^{(0)},a_{\mathcal{C}}(i)} \wedge i \in agents(s_{\mathcal{C}})\} \cup \{\tau_i \mid \tau_i = s_{\mathcal{C}}.t_i \wedge i \notin agents(s_{\mathcal{C}})\}\big)$$

  then
  $\forall i \notin agents(s_{\mathcal{C}})$, $\forall t \in [0, \tau_m]$

$$((T_{\mathcal{C}}(s_{\mathcal{C}}, a_{\mathcal{C}}))(t)).g_i = (s_{\mathcal{C}}.g_i[t, \tau_i])^{-t}$$

  $\forall i \in agents(s_{\mathcal{C}})$, $\forall t \in (0, \tau_m]$

$$((T_{\mathcal{C}}(s_{\mathcal{C}}, a_{\mathcal{C}}))(t)).g_i = (f_{s^{(0)},a_{\mathcal{C}}(i)}[t, \tau_i])^{-t}$$

We next present an example of automaton with concurrent timed actions. We consider the generalization of the Line-Up multi-agent system presented in Section 2.2.4, where we model the evolution of the state of the agents from their current state to their newly computed one. In this system, we have that the evolution of action a in state s is defined by the function $f_{s,a}$ with time duration $\tau_{s,a}$. The automaton when we allow for concurrent actions is defined as follows.

Definition 36. *A shared-state MAS with N agents and concurrent actions for the Line-Up multi-agent system can be modeled as an automaton with timed action $\mathcal{A}_{\mathcal{C}} = (S_{\mathcal{C}}, S0_{\mathcal{C}}, A_{\mathcal{C}}, E_{\mathcal{C}}, T_{\mathcal{C}})$ where*

- $S_{\mathcal{C}} = ((\mathcal{I} \to \mathbb{R}), \ldots, (\mathcal{I} \to \mathbb{R}))$,
- $s_{\mathcal{C}} \in S0_{\mathcal{C}}$ *if* $s^{(0)} = s0$ *and*

$$\forall s_{\mathcal{C}}.t_i > 0 \ : \ \exists \widehat{avg}_{l,i,r} \in A_i, \ \exists t \in [0, \tau_{s^{(0)}, \widehat{avg}_{l,i,r}}] \ : \ s_{\mathcal{C}}.g_i = (f_{s^{(0)}, \widehat{avg}_{l,i,r}}[t, t + s_{\mathcal{C}}.t_i])^{-t}$$

- $\mathcal{A}_{\mathcal{C}} = A_1 \times A_2 \times \ldots \times A_N$,
- $E_{\mathcal{C}} = S_{\mathcal{C}} \times \mathcal{A}_{\mathcal{C}} \to true$,
- $\forall s_{\mathcal{C}} \in S_{\mathcal{C}}$, $\forall a_{\mathcal{C}} \in \mathcal{A}_{\mathcal{C}}$,
  let

$$\tau_m = \min\left(\{\tau_i \mid \tau_i = \tau_{s^{(0)}, a_{\mathcal{C}}(i)} \wedge i \in agents(s_{\mathcal{C}})\} \cup \{\tau_i \mid \tau_i = s_{\mathcal{C}}.t_i \wedge i \notin agents(s_{\mathcal{C}})\}\right)$$

  then
  $\forall i \notin agents(s_{\mathcal{C}})$, $\forall t \in [0, \tau_m]$

$$((T_{\mathcal{C}}(s_{\mathcal{C}}, a_{\mathcal{C}}))(t)).g_i = (s_{\mathcal{C}}.g_i[t, \tau_i])^{-t}$$

  $\forall i \in agents(s_{\mathcal{C}})$, $\forall t \in (0, \tau_m]$

$$((T_{\mathcal{C}}(s_{\mathcal{C}}, a_{\mathcal{C}}))(t)).g_i = (f_{s^{(0)}, a_{\mathcal{C}}(i)}[t, \tau_i])^{-t}$$

4.4.2 Shared-State Multi-Agent Systems with Sliding Window and Concurrent Actions

In this Section, we extend shared-state multi-agent systems with concurrent actions. We allow agents to update their state using the state of other agents computed at some times in the past. Given the shared-state automaton with concurrent actions, we construct a shared-state automaton with sliding window and concurrent actions using a procedure similar to the one presented in Section 4.2. In Section 4.2, we have constructed a sliding window automaton for a shared-state automaton where concurrent actions are not allowed.

We assume that agents cannot read arbitrary old values. If t is the current time of the execution, every agent can read the state of other agents in the interval $[t - \mathcal{B}, t]$, with $\mathcal{B} \geq 0$. For example, in Figure 4.14, agents A_1 at time t can read the state of agents A_1, A_2 and A_3 in the interval $[t-\mathcal{B}, t]$.

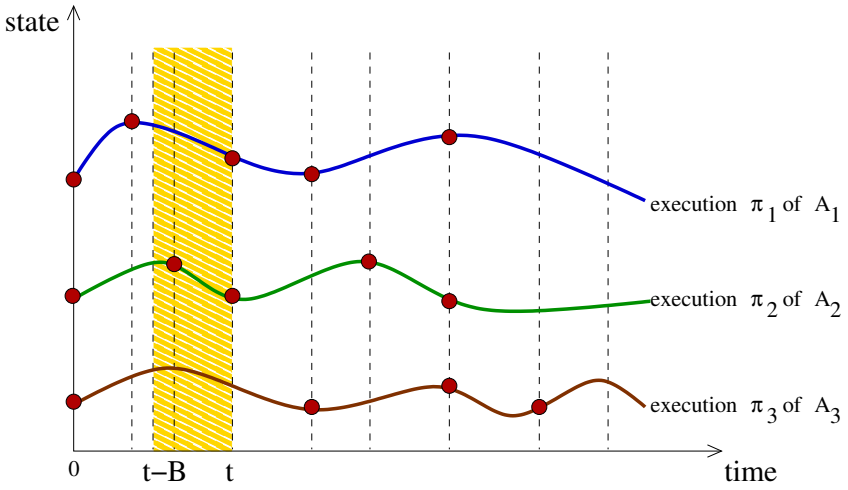


Figure 4.14: An execution of a multi-agent system consisting of three agents where agents execute actions concurrently and agent can read past states. For example, at time t, agent A_1 can read the state of agent A_2 and A_3 in the time interval $[t-\mathcal{B}, t]$. Similarly, agent A_2 can read the state of agent A_1 and A_3 in the time interval $[t-\mathcal{B}, t]$. At time t agent A_3 is executing an action started at time 0. The shaded area represents the time interval $[t-\mathcal{B}, t]$. Dark filled circles represent end-states of executions of actions.

Given $\mathcal{A}_{\mathcal{C}} = (S_{\mathcal{C}}, S0_{\mathcal{C}}, A_{\mathcal{C}}, E_{\mathcal{C}}, T_{\mathcal{C}})$, we next construct the automaton $\mathcal{A}_{\mathcal{BC}} = (S_{\mathcal{BC}}, S0_{\mathcal{BC}}, A_{\mathcal{BC}}, E_{\mathcal{BC}}, T_{\mathcal{BC}})$. This automaton generalizes $\mathcal{A}_{\mathcal{C}}$; in particular, when $\mathcal{B} = 0$, then the two automata $\mathcal{A}_{\mathcal{BC}}$ and $\mathcal{A}_{\mathcal{C}}$ are equal. For each agent i, a state of $\mathcal{A}_{\mathcal{BC}}$ requires to store: its current state, its past states (for a bounded time interval of length $\mathcal{B}$), and, in case the agent is executing an action, the function describing its behaviour. This state structure generalizes the state structure of agent $\mathcal{A}_{\mathcal{C}}$, where the agent state is not required to store past states. We model the state of an agent as a function from a finite closed time interval to S_i. This interval is of the form $[-\mathcal{B}, t]$. The function evaluated at time 0 corresponds to the current state of the agent, the function evaluated in the interval $[-\mathcal{B}, 0)$ to the past states and the function evaluated in the interval $(0, t_i]$ to the behavior of the agent when it executes an action at time 0. We denote by $\mathcal{I}_{\mathcal{B}}$ the family of finite closed intervals with left extreme $-\mathcal{B}$, i.e.

$$\mathcal{I}_{\mathcal{B}} = \{[-\mathcal{B}, t] \mid t \geq 0\}$$

Hence, a state $s_{\mathcal{BC}}$ in $S_{\mathcal{BC}}$ is the N-tuple $s_{\mathcal{BC}} = (g_1, \ldots, g_N)$ where $g_i : [-\mathcal{B}, t_i] \rightarrow S_i$, for all $i \in \{1, 2, \ldots, N\}$. Given $s_{\mathcal{BC}} \in S_{\mathcal{BC}}$, we denote by $s_{\mathcal{BC}}.g_i$ the i-th component of $s_{\mathcal{BC}}$ and by $s_{\mathcal{BC}}.t_i$ the right extreme of the domain of $s_{\mathcal{BC}}.g_i$. Given a state $s_{\mathcal{BC}} \in S_{\mathcal{BC}}$ and a state $s_{\mathcal{C}} \in S_{\mathcal{C}}$, we next introduce the asynchronous view relation for automata with concurrent actions $H_{\mathcal{C}} \subseteq S_{\mathcal{C}} \times S_{\mathcal{BC}}$ as follows:

$$(s_{\mathcal{C}}, s_{\mathcal{BC}}) \in H_{\mathcal{C}} \equiv \left(\exists t \in [-\mathcal{B}, 0]^N \ : \ \forall i \in \{1, \ldots, N\} \ : \ s_{\mathcal{C}}.g_i = (s_{\mathcal{BC}}.g_i[t(i), s_{\mathcal{BC}}.t_i])^{-t(i)}\right)$$

This notion of asynchronous generalizes the notion of asynchronous view defined in Section 4.2. In this more general notion, the asynchronous view describes the state of the system for a time interval. Given this relation, we define the set of all asynchronous view of $s_{\mathcal{BC}} \in S_{\mathcal{BC}}$. This set denoted as $\mathcal{H}_{\mathcal{C}}(s_{\mathcal{BC}})$ is defined as follows:

$$\mathcal{H}_{\mathcal{C}}(s_{\mathcal{BC}}) = \{s_{\mathcal{C}} \in S_{\mathcal{C}} \mid H_{\mathcal{C}}(s_{\mathcal{C}}, s_{\mathcal{BC}})\}$$

The set of actions $A_{\mathcal{BC}}$ is constructed using the set $A_{\mathcal{C}}$. For each action $a_{\mathcal{C}} \in A_{\mathcal{C}}$, we construct an action $a_{\mathcal{BC}}$. Given a state $s_{\mathcal{BC}} \in S_{\mathcal{BC}}$, the execution of action $a_{\mathcal{BC}}$ consists of the execution of action $a_{\mathcal{BC}}(i)$ on an asynchronous view of $s_{\mathcal{BC}}$. By construction, $a_{\mathcal{BC}}(i)$ and $a_{\mathcal{BC}}(j)$ can be executed on different asynchronous views. Action $a_{\mathcal{BC}} \in A_{\mathcal{BC}}$ is enabled in state $s_{\mathcal{BC}} \in S_{\mathcal{BC}}$, if the corresponding action $a_{\mathcal{C}}$ is enabled in all asynchronous view of $s_{\mathcal{BC}}$. The definition of the automaton with sliding window follows.

Definition 37. *Given $\mathcal{A}_{\mathcal{C}} = (S_{\mathcal{C}}, S0_{\mathcal{C}}, A_{\mathcal{C}}, E_{\mathcal{C}}, T_{\mathcal{C}})$, a shared-state MAS with N agents, concurrent actions and sliding window of size $\mathcal{B}$ can be modeled as an automaton with timed action $\mathcal{A}_{\mathcal{BC}} = (S_{\mathcal{BC}}, S0_{\mathcal{BC}}, A_{\mathcal{BC}}, E_{\mathcal{BC}}, T_{\mathcal{BC}})$ having*

- $S_{\mathcal{BC}} = ((\mathcal{I}_{\mathcal{B}} \rightarrow S_1), \ldots, (\mathcal{I} \rightarrow S_N))$,
- $s_{\mathcal{BC}} \in S0_{\mathcal{BC}}$ *if*

$$\begin{aligned} \exists s_{\mathcal{C}} \in S0_{\mathcal{C}} \quad &: \quad \forall i \in \{1, \ldots, N\} \ : \ s_{\mathcal{BC}}.g_i[0, s_{\mathcal{BC}}.t_i] = s_{\mathcal{C}}.g_i \\ \forall t \in [-\mathcal{B}, 0) \quad &: \quad s_{\mathcal{BC}}.g_i(t) = s_{\mathcal{BC}}.g_i(0) \end{aligned}$$

- $\mathcal{A}_{\mathcal{BC}} = \mathcal{A}_{\mathcal{C}}$,
- $\forall s_{\mathcal{BC}} \in S_{\mathcal{BC}}$, $\forall a_{\mathcal{BC}} \in \mathcal{A}_{\mathcal{BC}}$,

$$E_{\mathcal{BC}}(s_{\mathcal{BC}}, a_{\mathcal{BC}}) = (\forall s_{\mathcal{C}} \in \mathcal{H}_{\mathcal{C}}(s_{\mathcal{BC}}) \ : \ E(s_{\mathcal{C}}, a_{\mathcal{BC}}))$$

- $\forall s_{\mathcal{BC}} \in S_{\mathcal{BC}}$, $\forall a_{\mathcal{BC}} \in \mathcal{A}_{\mathcal{BC}}$, *for all* $i \in \{1, \ldots, N\}$, *agent* i *chooses a state* $s_{\mathcal{C}_i} \in S_{\mathcal{C}}$ *with* $H(s_{\mathcal{C}_i}, s_{\mathcal{BC}})$

  Let

$$\tau_m = \min\Big(\{\tau_i \mid \tau_i = \tau_{s_i^{(0)}, a_{\mathcal{BC}}(i)} \wedge \ i \in agents(s_{\mathcal{BC}})\} \cup \{\tau_i \mid \tau_i = s_{\mathcal{BC}}.t_i \wedge i \notin agents(s_{\mathcal{BC}})\}\Big)$$

  then

$\forall i \notin agents(s_{\mathcal{BC}})$, $\forall t \in [0, \tau_m]$

$$(T_{\mathcal{BC}}(s_{\mathcal{BC}}, a_{\mathcal{BC}})(t)).g_i = (s_{\mathcal{BC}}.g_i[t - \mathcal{B}, \tau_i])^{-t}$$

$\forall i \in agents(s_{\mathcal{BC}})$, $\forall t \in (0, \tau_m]$ *with* $t \leq \mathcal{B}$,

$$\begin{aligned}(T_{\mathcal{BC}}(s_{\mathcal{BC}}, a_{\mathcal{BC}})(t)).g_i[-\mathcal{B}, -t) &= (s_{\mathcal{BC}}.g_i[t - \mathcal{B}, 0))^{-t} \\ (T_{\mathcal{BC}}(s_{\mathcal{BC}}, a_{\mathcal{BC}})(t)).g_i[-t, \tau_i - t] &= (f_{s_i^{(0)}, a_{\mathcal{BC}}(i)}[0, \tau_i])^{-t}\end{aligned}$$

and $t > \mathcal{B}$,

$$(T_{\mathcal{BC}}(s_{\mathcal{BC}}, a_{\mathcal{BC}})(t)).g_i = (f_{s_i^{(0)}, a_{\mathcal{BC}}(i)}[t - \mathcal{B}, \tau_i])^{-t}$$

Theorem 13 and Theorem 14 can be extended to this more general class of systems.

4.5 Discussion

In this Section, we discuss the main results of this Chapter and relate them to the literature.

In this Chapter, we have presented formal models for shared-state and message-passing systems. We have modeled systems with sequential actions and with concurrent actions. In the case of shared-state systems, we have restricted our attention on stared-state systems where the execution of an action can change the state of a single agent. In the case of message-passing systems, we have focused on systems with bounded transmitting delay where messages may be lost, delayed or received out-of-order.

We have modeled these systems using the automaton with timed actions model. We model the state of the multi-agent system at each point of the execution as the cartesian product of the states of the single agents. We next motivate this specific structure for the state space. In shared-state systems, we require that the execution of any action can change the state of only one agent. By construction, when an action is executed, the post-state of the action is equal to the pre-state of the action except for the agent executing the action. If we assume a cartesian product structure, we can ensure that the post-state of the action is a valid state. In message-passing systems, we have that the tuple consisting of the state of an agent and the last received messages is a valid state. This constraint requires a cartesian product structure for the state space.

We first present the shared-state model with discrete actions. We then generalize this model and, using the shared-state model with discrete actions, we construct a model for shared-state systems with explicit arbitrary dynamics. In this model, the state of an agent consists of a pair: its current state and the newly computed one. We present conditions on the stability and convergence of these systems. We require a specific structure for the Lyapunov function and for the dynamics for the

agents. Specifically, in the case of stability, we require the level sets of the Lyapunov function to be convex sets (see Assumption E2) and trajectories of the agents to be convex combinations from their current state and newly computed one (see Assumption E3). This specific structure ensures that, when executing a timed action, the trajectory of the system remains in the same level set. In the case of convergence, we also require that an agents eventually moves towards the newly computed states (see Assumption F4). This requirement ensures that the system eventually enters a strictly contained level set.

We present message-passing systems as a special case of shared-state automata with sliding window. In shared-state automata with sliding window, a state describes the behavior of a shared-state system for a time interval. When executing an action, an agent picks the states of the other agents nondeterministically in this time interval and executes the action assuming that this asynchronous view is the current state of the system. We derive conditions on the Lyapunov function that ensure stability and convergence. We require the level sets of the Lyapunov function to be in conjunctive form (see Assumptions G2 and H2). Under this assumption, a level set can be represented as a conjunction of independent predicates, each predicate defined on the state of a single agent. This structure of the predicate ensures that if the states of a time interval satisfy the predicate, then any asynchronous view obtained combining these states satisfies the predicate as well. Using the bounded delay assumption, we have that the system eventually enters in a strictly contained level set.

We present shared-state and message-passing system with concurrent timed actions. In these systems, at each point of the computation, an agent is either starting executing a new action or it is executing some action started at some time in the past. We derive conditions on the Lyapunov function that ensure stability and convergence. These assumptions are similar to the assumptions in the case of sequential actions.

The material presented in this Chapter has been partially presented in [16]. Previous work includes [24, 66, 8]. In [24], the authors investigate a consensus problem in a distributed system with bounded communication delay. Our work extends [66, 8]. In [66, 8], the authors investigate stability and convergence of shared-state systems with concurrent discrete actions and sliding window. Our contribution is to model these systems as automata, extend them to timed actions and extend their results to concurrent systems with timed actions.

Chapter 5

An Application to Distributed Control

In this Chapter, we discuss correctness of a general class of iterative schemes. The goal of protocols in this class is solving a system of linear equations in a decentralized way. We are interested in this class of distributed systems, because it has many practical applications in areas such as in distributed robot pattern formation protocols [50, 51, 16]. We prove correctness of these schemes using the results presented in Chapter 3 and Chapter 4.

In Section 5.1 we discuss the class of iterative schemes assuming shared-state communication. In this Section, we present the class of protocols, model them as automata, and prove their correctness using results from Chapter 3. In Section 5.2 we generalize this class and allows for unreliable message-passing communication. In this Section, we model protocols in this class as automata and prove their correctness. We derive the proof of correctness of these protocols from the proof of correctness of the corresponding shared-state protocols using results presented in Chapter 4. In Section 5.3 we discuss correctness of a distributed robot pattern formation protocol. Finally, in Section 5.4 we relate the main results of this Chapter to the literature.

5.1 Systems of Linear Equations via Shared Variables

In this Section, we discuss a class of multi-agent systems whose goal is to solve a system of linear equations.

5.1.1 MAS solving Systems of Linear Equations

In this Section, we discuss the class of multi-agent systems consisting of shared-state iterative schemes for solving systems of linear equations of the form $A \cdot x = b$ where A is a real-valued invertible matrix of size $N \times N$, while x, b are real valued vectors of length N. The matrix is weakly diagonally dominant (see Assumption L3) and strictly diagonally dominant in at least one row (see Assump-

tion L4). The invertibility assumption of A ensures the existence of the solution of the system of linear equations. The solution of the system of linear equations is given by $A^{-1}b$. The goal of these systems is to iteratively compute the vector x starting from an initial guess vector $x0$. The Gauss, Jacobi and Gauss-Seidel algorithms are examples of iterative schemes belonging to this class.

We model this class as shared-state multi-agent systems with N agents. The goal of the agents is to compute the solution of the system of linear equations in a decentralized way. To do so, each agent is responsible for solving a specific component of the vector x; for example, agent i would be responsible for solving variable $x(i)$. Agent i computes $x(i)$ by applying the following updating rule:

$$x(i) := b(i) - \sum_{j \neq i} A(i,j) \cdot x(j) \tag{5.1}$$

where $x(j)$ is the current value of agent j. Agent i can access the value of agent j (for $j \neq i$), but not modify it. When executing this updating rule, agent i sets the value of $x(i)$ to the solution of the i-th equation of the system of linear equations. Within this updating rule and more generally throughout this Chapter, we assume that the diagonal entries of matrix A are all equal to 1.

We consider shared-state multi-agent systems where concurrent actions are not allowed. Agents execute the updating scheme in Equation 5.1 in a nondeterministic fashion. We assume weak fairness, i.e., each agent solves its equation infinitely often. Nondeterministic versions of Jacobi and Gauss-Seidel algorithms belong to this class.

5.1.2 System of Linear Equations Shared-State Automaton

In this Section, we model this class of system using automata. The discrete automaton is defined as follows:

Definition 38. *The generic discrete shared-state automaton* $\mathcal{A}_\mathcal{D} = (S_\mathcal{D}, S0_\mathcal{D}, A_\mathcal{D}, E_\mathcal{D}, T_\mathcal{D})$ *with* N *agents is as follows:*

- A, b *and* $x0$ *are parameters with* A *of size* $N \times N$, *and* b *and* $x0$ *vectors of length* N,
- $S_\mathcal{D} = \mathbb{R}^N$ *with* $S_i = \mathbb{R}$,
- $S0_\mathcal{D} = \{x0\}$,
- $A_\mathcal{D} = \cup_i A_i$ *with* $A_i = \{le_i\}$,
- $E_\mathcal{D} : S_\mathcal{D} \times A_\mathcal{D} \rightarrow true$,
- $\forall s_\mathcal{D} \in S_\mathcal{D}$, $\forall a_\mathcal{D} = le_i \in A_\mathcal{D}$

$$T_\mathcal{D}(s_\mathcal{D}, le_i) = \left(s_\mathcal{D}(1), \ldots, s_\mathcal{D}(i-1), \left(b(i) - \sum_{j \neq i} A(i,j) \cdot s_\mathcal{D}(j) \right), s_\mathcal{D}(i+1), \ldots, s_\mathcal{D}(N) \right)$$

We assume weak fairness, i.e. $\mathcal{F}_{\mathcal{D}} = \{le_i\}_{i\in\{1,\ldots,N\}}$.

We next discuss equilibrium states of $\mathcal{A}_{\mathcal{D}}$. Given matrix A and vector b, the automaton has a unique equilibrium state; this state is the solution of the system of linear equations. The state $\hat{s}_{\mathcal{D}}$ denotes this equilibrium state, i.e. $\hat{s}_{\mathcal{D}} = A^{-1}b$. By definition of equilibrium state, it follows that $\forall i \in \{1,\ldots,N\}$,

$$\hat{s}_{\mathcal{D}}(i) = b(i) - \sum_{j\neq i} A(i,j)\cdot \hat{s}_{\mathcal{D}}(j) \tag{5.2}$$

We refer to this property as the fixed point property.

5.1.3 Proof of Correctness

In this Section, we discuss the proof of correctness of systems in this class. A system in this class is correct, if the corresponding automaton converges to $\hat{s}_{\mathcal{D}}$. We next construct a Lyapunov function around $\hat{s}_{\mathcal{D}}$ and prove that, under specific restrictions on the matrix, this function satisfies the assumptions of Theorem 9.

5.1.3.1 Matrix A

In this Section, we present the Assumptions on matrix A. Later in this Section, we prove convergence of $\mathcal{A}_{\mathcal{D}}$ to $\hat{s}_{\mathcal{D}}$ under these assumptions on A. We make the following assumptions on A:

Assumptions.

L1. A is invertible,

L2. A has all entries along the main diagonal equal to 1*,*

L3. A is weakly diagonally dominant, i.e. $\forall i \in \{1,\ldots,N\}$

$$\sum_{j\neq i} \mid A(i,j) \mid \leq \mid A(i,i) \mid$$

L4. A is strictly diagonally dominant in at least one row, i.e.

$$\exists k \in \{1,\ldots,N\} \ : \ \sum_{j\neq k} \mid A(k,j) \mid < \mid A(k,k) \mid$$

Assumption L2 is made for convenience without loss of generality.

5.1.3.2 Communication Graph G

In this Section, we introduce the notion of communication graph. In this class of systems, agent i reads the state of agent j for all $j \neq i$ with $A(i,j) \neq 0$. We formalize this communication between

agents using a directed graph. There is a directed edge with source i and destination j if and only if agent i reads local variables of agent j. The structure of matrix A does not require symmetric communication. The graph is defined as follows:

Definition 39. *Given matrix A, the communication graph $G = (V, E)$ has $V = \{1, \ldots, N\}$ and $(j, k) \in E$ if $A(j, k) \neq 0$.*

We next define the notion of strictly diagonally dominant vertex. A vertex i of $G = (V, E)$ is strictly diagonally dominant if the corresponding row of matrix A is strictly diagonally dominant. Formally,

Definition 40. *Given matrix A and communication graph $G = (V, E)$, $i \in V$ is strictly diagonally dominant if $\sum_{j \neq i} \mid A(i, j) \mid < \mid A(i, i) \mid$.*

We make the following assumption on $G = (V, E)$:

Assumptions.

M1. $\forall i \in V$ there exists a directed path from i to a strictly diagonally dominant vertex of G.

5.1.3.3 Strictly Diagonally Dominant Rooted Forest $\mathbb{F}$

In this Section, we introduce the notion of strictly diagonally dominant rooted forest. Given the communication graph $G = (V, E)$, a rooted forest $\mathbb{F}$ of G is strictly diagonally dominant if it is rooted at strictly diagonally dominant vertices of G. Given $\mathbb{F}$ and $i \in V$, we denote by $p(i)$ the parent of i in $\mathbb{F}$ and $ancestors(i)$ the set of vertices in the path from i to a root vertex in $\mathbb{F}$. By construction, $i \in ancestors(i)$. If i is a root of the forest, then $p(i) = \perp$ and $ancestors(i) = \{i\}$. We introduce the predicate $root$ on the vertices of $\mathbb{F}$; $root(i)$ holds if i is a root of the forest, i.e. row i of A is strictly diagonally dominant.

5.1.3.4 Error Function e

In this Section, we introduce the notion of error for the agents in the system. Informally, the error of agent i is the distance between its value and the value of the i-th component of the solution vector. We next formally define the error function of the agents:

Definition 41. *The error function $e : S_{\mathcal{D}} \times \{1, \ldots, N\} \to \mathbb{R}_{\geq 0}$ is defined as follows, $\forall s_{\mathcal{D}} \in S_{\mathcal{D}}$, $\forall i \in \{1, \ldots, N\}$,*

$$e\left(s_{\mathcal{D}}, i\right) = \mid s_{\mathcal{D}}(i) - \hat{s}_{\mathcal{D}}(i) \mid$$

The state $\hat{s}_{\mathcal{D}}$ denotes the equilibrium state of the system of equations; it has been defined in Equation 5.2. The error of the system is defined as the maximum of the errors of the agents.

Definition 42. *The error function* $e : S_{\mathcal{D}} \to \mathbb{R}_{\geq 0}$ *is defined as,* $\forall s_{\mathcal{D}} \in S_{\mathcal{D}}$,

$$e(s_{\mathcal{D}}) = \max_{i \in \{1,\ldots,N\}} e\left(s_{\mathcal{D}}, i\right)$$

In the next Lemma, we relate the errors of the agents when executing an action.

Lemma 17. $\forall s_{\mathcal{D}} \in S_{\mathcal{D}}$, $\forall a_{\mathcal{D}} = le_j \in A_{\mathcal{D}}$,

$$e\left(T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}}), j\right) \leq \sum_{i \neq j} \mid A(j,i) \mid \cdot\, e\left(s_{\mathcal{D}}, i\right)$$

Proof. We denote by $s'_{\mathcal{D}}$ the state $T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}})$. By construction of the transition function, we have that,

$$s'_{\mathcal{D}}(j) = b(j) - \sum_{i \neq j} A(j,i) \cdot s_{\mathcal{D}}(i)$$

Using the fixed point property of $\hat{s}_{\mathcal{D}}$, we have that

$$\hat{s}_{\mathcal{D}}(j) = b(j) - \sum_{i \neq j} A(j,i) \cdot \hat{s}_{\mathcal{D}}(i)$$

Hence, by definition of $e\left(s'_{\mathcal{D}}, j\right)$

$$\begin{aligned} e\left(s'_{\mathcal{D}}, j\right) &= \left| \sum_{i \neq j} A(j,i) \cdot \left(s_{\mathcal{D}}(i) - \hat{s}_{\mathcal{D}}(i)\right) \right| \\ &\leq \sum_{i \neq j} \mid A(j,i) \mid \cdot \mid s_{\mathcal{D}}(i) - \hat{s}_{\mathcal{D}}(i) \mid \\ &\leq \sum_{i \neq j} \mid A(j,i) \mid \cdot\, e\left(s_{\mathcal{D}}, i\right) \end{aligned}$$

where the first inequality holds by triangle inequality and the last one by definition of the error function. □

This Lemma ensures that when agent j executes an action, the error of j in the post-state $T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}})$ of the action is bounded by the weighted average of the errors of the remaining agents in the pre-state $s_{\mathcal{D}}$ of the action. These weights are the absolute values of the j-th row of matrix A.

5.1.3.5 Agents Weights

In this Section, we define a weight for each agent of the system. These weights are recursively constructed along a strictly diagonally dominant rooted forest.

Definition 43. *Given a strictly diagonally dominant rooted forest* $\mathbb{F}$*, the weight of agent* j *in* $\mathbb{F}$*,*

denoted by $w(j)$*, is:*

$$w(j) = \begin{cases} \sum_{k \neq j} \mid A(j,k) \mid & root(j) \\ \mid A(j,p(j)) \mid \cdot w(p(j)) + \sum_{k \notin \{j,p(j)\}} \mid A(j,k) \mid & otherwise \end{cases}$$

We next show that these weights are nonnegative and strictly smaller than 1.

Lemma 18. *Given a strictly diagonally dominant rooted forest* $\mathbb{F}$*, we have that*

$$\forall j \in \{1, \ldots, N\} \ : \ 0 \leq w(j) < 1$$

Proof. We denote by j an arbitrary agent in the system. The proof follows by induction along the forest.

Base Case. Assume that $root(j)$ holds. By construction, j is a strictly diagonally dominant vertex. By construction, its weight $w(j)$ is nonnegative, since it is the sum of nonnegative values, and strictly smaller than 1, since row j of matrix A satisfies Assumption L4.

Induction Case. Assume that $\neg root(j)$ holds. By induction hypothesis, $0 \leq w(p(j)) < 1$. By construction of $w(j)$, we have that the weight $w(j)$ is nonnegative, since it is the sum of nonnegative terms. It is strictly smaller than 1, because

$$\begin{aligned} w(j) &< \mid A(j,p(j)) \mid + \sum_{k \notin \{j,p(j)\}} \mid A(j,k) \mid \\ &= \sum_{k \neq j} \mid A(j,k) \mid \\ &\leq 1 \end{aligned}$$

where the first inequality holds, since $w(p(j)) < 1$ by induction hypothesis, and last inequality holds since A satisfies Assumptions L2-3. □

We conclude this section with the definition of α; this is a nonnegative real constant defined as the maximum of the weights of the agents.

Definition 44. *Given a strictly diagonally dominant rooted forest* $\mathbb{F}$*,*

$$\alpha = \max_{j \in I} w(j)$$

By construction, $0 \leq \alpha < 1$, i.e. α is a contraction factor.

5.1.3.6 Totally Ordered Set $\mathbb{P}$

In this Section, we present a totally ordered set $\mathbb{P}$. This set defines the range of the Lyapunov function used to prove convergence of the automaton. Without loss of generality, we fix a strictly

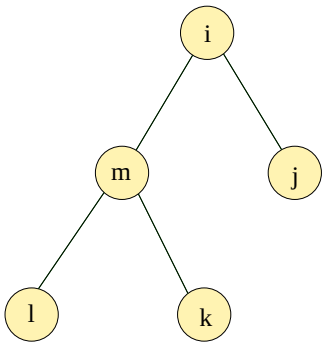
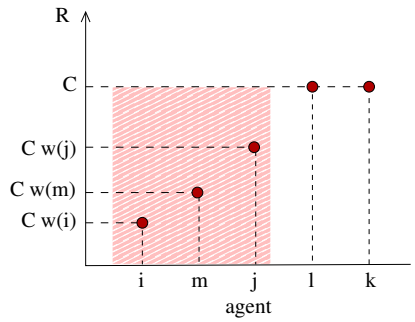


Figure 5.1: Representation of a tree. The Breadth-first traversal ordering is $i <_{BF} m <_{BF} j <_{BF} l <_{BF} k$.

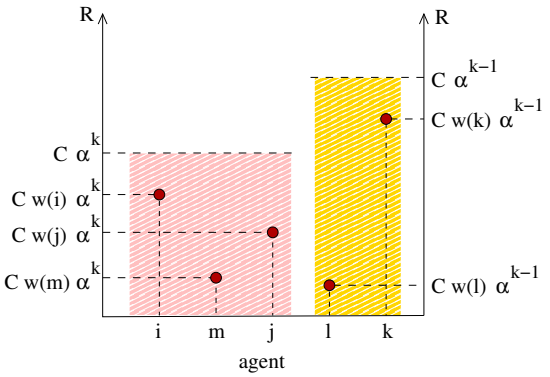
diagonally dominant rooted forest $\mathbb{F}$ of the communication graph G, and assume that this forest consists of only one tree. We denote this tree by $\mathbb{T}$, the set of vertices of $\mathbb{T}$ is $\{1, \ldots, N\}$. We also fix a nonnegative real constant C.

We consider the Breadth-first traversal of $\mathbb{T}$; this a level-order traversal that ensures that every node on a level is visited before going to a lower level. The Breadth-first travels defines a total ordering of the vertices. Given two vertices i and j of $\mathbb{T}$, with $i \neq j$, we have that vertex i is less than j with respect to the Breadth-first traversal, denoted by $i <_{BF} j$, if vertex i precedes vertex j in the traversal; while i is larger than j with respect to the Breadth-first traversal, denoted by $i >_{BF} j$, if vertex i follows j in the traversal, i.e. either i is on the same level on the right hand side of j or i belongs to a lower level. For example, in the tree presented in Figure 5.1 we have that $i <_{BF} j$, $k <_{BF} l$ and $k <_{BF} j$. We denote by $\min_{BF}(\mathbb{T})$ the smallest vertex in $\mathbb{T}$ with respect to the Breadth-first traversal, i.e. $\min_{BF}(\mathbb{T}) \leq_{BF} j$, for all $j \in \{1, \ldots, N\}$. By construction of $\mathbb{T}$, $\min_{BF}(\mathbb{T})$ is a root vertex. Similarly, we denote by $\max_{BF}(\mathbb{T})$ the largest vertex in $\mathbb{T}$ with respect to the Breadth-first traversal.

We next define the elements of $\mathbb{P}$. An element in $q_{(k,j)} \in \mathbb{P}$ is a vector of length N and depends on two parameters: a natural number k and an agent identifier j. Vector $q_{(k,j)}$ contains nonnegative real values. If $k = 0$, then $\forall i \leq_{BF} j$, the value $q_{(k,j)}(i)$ stores the number $C \cdot w(i)$, while $\forall i >_{BF} j$, the value of $q_{(k,j)}(i)$ is C. In this case, we have that all entries smaller or equals to j with respect to the Breadth-first traversal are strictly smaller of all entries larger than j with respect to the Breadth-first traversal. An example of the case when $k = 0$ is presented in Figure 5.2(a). If $k > 0$, then $\forall i \leq_{BF} j$, the value $q_{(k,j)}(i)$ stores the number $C \cdot \alpha^k \cdot w(i)$, while $\forall i >_{BF} j$, the value of $q_{(k,j)}(i)$ is $C \cdot \alpha^{k-1} \cdot w(i)$. In this case, we have that all entries strictly larger than j (with respect to the Breadth-first traversal) store numbers upper-bounded by $C \cdot \alpha^{k-1}$, while all entries smaller or equals to j (with respect to the Breadth-first traversal) are upper-bounded by $C \cdot \alpha^k$. By construction, $C \cdot \alpha^k < C \cdot \alpha^{k-1}$. An example of the case when $k > 0$ is presented in Figure 5.2(b).



(a) Example of $q_{(k1,j1)}$ for the case when $k1 = 0$ and $j1 = j$; as represented in the shaded area, agents i, m and j are strictly smaller than C.



(b) Example of $q_{(k1,j1)}$ for the case when $k1 > 0$ and $j1 = j$; as represented in the shaded area on the left, agents i, m and j are strictly smaller than $C \cdot \alpha^k$ while, as represented in the shaded area on the right, agents l and k are strictly smaller than $C \cdot \alpha^{k-1}$.

Figure 5.2: Examples of $q_{(k1,j1)}$ for the tree in Figure 5.1.

Formally,

Definition 45. *Given* $k \in \mathbb{N}$ *and* $j \in \{1, \ldots, N\}$, $q_{(k,j)} = (q_{(k,j)}(1), \ldots, q_{(k,j)}(N)) \in \mathbb{P}$ *if* $\forall i \in \{1, \ldots, N\}$,

$$q_{(k,j)}(i) = \begin{cases} C \cdot \alpha^k \cdot w(i) & i \leq_{BF} j \\ C \cdot \alpha^{k-1} \cdot w(i) & (i >_{BF} j) \wedge (k > 0) \\ C & (i >_{BF} j) \wedge (k = 0) \end{cases}$$

We next define the ordering relation $\leq_{\mathbb{P}}$ on $\mathbb{P}$. Given $q_{(k1,j1)}, q_{(k2,j2)}$, this ordering is a lexicographic order on the pairs $(k1, j1), (k2, j2)$, where $k1$ and $k2$ are compared using the total ordering on the naturals, and $j1$ and $j2$ are compared using the $\leq_{BF}$ ordering. Formally,

Definition 46. *Given* $q_{(k1,j1)}, q_{(k2,j2)} \in \mathbb{P}$,

$$q_{(k1,j1)} \leq_{\mathbb{P}} q_{(k2,j2)} \equiv ((k2 < k1) \vee ((k1 = k2) \wedge (j2 \leq_{BF} j1)))$$

The relation $\leq_{\mathbb{P}}$ is a total order. Hence, $\mathbb{P}$ is a totally ordered set. Set $\mathbb{P}$ has the property that every non-empty subset of $\mathbb{P}$ has a least element with respect to $\leq_{\mathbb{P}}$, i.e., $\mathbb{P}$ is a well-ordered set.

5.1.3.7 Lyapunov Function and Level Sets

In this Section, we present a Lyapunov function V for $\mathcal{A}_{\mathcal{D}}$ and its level sets. The function $V : S_{\mathcal{D}} \to \mathbb{P}$ maps a state $s_{\mathcal{D}} \in S_{\mathcal{D}}$ into the smallest element of $\mathbb{P}$ that upper-bounds the error of all the agents in state $s_{\mathcal{D}}$. Formally,

Definition 47. *The Lyapunov function $V : S_{\mathcal{D}} \to \mathbb{P}$ is as follows, $\forall s_{\mathcal{D}} \in S_{\mathcal{D}}$,*

$$V(s_{\mathcal{D}}) = \min_{k \in \mathbb{N}, j \in \{1,\dots,N\}} \{q_{(k,j)} \mid \forall i \in \{1,\dots,N\} \ : \ e\left(s_{\mathcal{D}}, i\right) \leq q_{(k,j)}(i)\}$$

We denote by $L_{(k,j)}$ the level set of V corresponding to the value $q_{(k,j)} \in \mathbb{P}$. This level set includes all states of $\mathcal{A}_{\mathcal{D}}$ where $\forall i$, with $i \leq_{BF} j$, the error of agent i is bounded by $C \cdot \alpha^k \cdot w\left(i\right)$. $\forall i$, with $i >_{BF} j$, the error of agent i is bounded by $C \cdot \alpha^{k-1} \cdot w\left(i\right)$, if $k > 0$, and by C otherwise. This level set is of the form:

$$L_{(k,j)} = \left\{s_{\mathcal{D}} \in S_{\mathcal{D}} \mid \forall i \in \{1,\dots,N\} \ : \ e\left(s_{\mathcal{D}}, i\right) \leq q_{(k,j)}(i)\right\}$$

We denote by $\mathcal{Q}_{(k,j)}$ the predicate associated with $L_{(k,j)}$, defined as $\forall s_{\mathcal{D}} \in S_{\mathcal{D}}$

$$\mathcal{Q}_{(k,j)}(s_{\mathcal{D}}) \equiv \left(\bigwedge_{i \in \{1,\dots,N\}} \left(e\left(s_{\mathcal{D}}, i\right) \leq q_{(k,j)}(i)\right) \right)$$

This predicate is in conjunctive form, since it can be expressed as the conjunction of simpler predicates defined on the state of single agents. Predicate $\mathcal{Q}_{(k,j)_i}$ is defined as

$$\mathcal{Q}_{(k,j)_i}(s_{\mathcal{D}}) \equiv \left(e\left(s_{\mathcal{D}}, i\right) \leq q_{(k,j)}(i)\right)$$

5.1.3.8 Properties of Level Sets of V

In this Section, we discuss some properties of the level sets of V. We first show that each level set $L_{(k,j)}$ is stable.

Lemma 19. *For all $q_{(k,j)} \in \mathbb{P}$, $L_{(k,j)}$ is stable.*

Proof. Our goal is to prove that $\forall s_{\mathcal{D}} \in S_{\mathcal{D}}$, $\forall a_{\mathcal{D}} \in \mathcal{A}_{\mathcal{D}}$,

$$s_{\mathcal{D}} \in L_{(k,j)} \Rightarrow T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}}) \in L_{(k,j)}$$

Consider an arbitrary state $s_{\mathcal{D}} \in S_{\mathcal{D}}$ and an arbitrary action $a_{\mathcal{D}} = le_i \in A_{\mathcal{D}}$, with $a_{\mathcal{D}} \in A_i$. Denote by $s'_{\mathcal{D}}$ the post-state of the execution of $a_{\mathcal{D}}$, i.e. $s'_{\mathcal{D}} = T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}})$.

When executing action $a_{\mathcal{D}}$, by construction, the states $s'_{\mathcal{D}}$ and $s_{\mathcal{D}}$ differ only in the i-th component. Hence, $\forall l \in \{1,\dots,N\}$, with $l \neq i$, the predicate $\mathcal{Q}_{(k,j)_l}(s'_{\mathcal{D}})$ holds.

We next prove that $\mathcal{Q}_{(k,j)_i}(s'_{\mathcal{D}})$ holds. Using Lemma 17, we have that the error of i in $s'_{\mathcal{D}}$ is bounded by the weighted average of the errors of its neighbors:

$$e\left(s'_{\mathcal{D}}, i\right) \leq \sum_{l \neq i} \mid A(i,l) \mid \cdot \, e\left(s_{\mathcal{D}}, l\right) \tag{5.3}$$

We distinguish two cases, $i \leq_{BF} j$ and $i >_{BF} j$.

Assume that $i \leq_{BF} j$. If i is a root of the tree and $k > 0$, then, using Equation 5.3, we have that

$$\begin{aligned} e\left(s'_{\mathcal{D}}, i\right) &\leq C \cdot \alpha^{k-1} \cdot \sum_{l \neq i} \left(\mid A(i,l) \mid \cdot w\left(l\right)\right) \\ &\leq C \cdot \alpha^{k} \cdot \sum_{l \neq i} \mid A(i,l) \mid \\ &\leq C \cdot \alpha^{k} \cdot w\left(i\right) \\ &= q_{(k,j)}(i) \end{aligned}$$

where the first inequality follows since, by assumption, $s_{\mathcal{D}} \in L_{(k,j)}$; the second inequality follows, by definition of α; the third inequality holds by definition of $w\left(i\right)$. Hence, $\mathcal{Q}_{(k,j)_i}(s'_{\mathcal{D}})$, if $root(i) \wedge k > 0$. The predicate holds also in the case when $k = 0$. This case is similar to the case when $root(i) \wedge k > 0$ and, therefore, is not reported.

Consider the case when i is not a root of $\mathbb{T}$. In this case, assuming $k > 0$ and using Equation 5.3, the error of i in $s'_{\mathcal{D}}$ can be bounded as follows:

$$\begin{aligned} e\left(s'_{\mathcal{D}}, i\right) &\leq \mid A(i,p(i)) \mid \cdot e\left(s_{\mathcal{D}}, p(i)\right) + \sum_{l \notin \{i, p(i)\}} \mid A(i,l) \mid \cdot e\left(s_{\mathcal{D}}, l\right) \\ &\leq C \cdot \alpha^{k} \cdot \mid A(i,p(i)) \mid \cdot w\left(p(i)\right) + \sum_{l \notin \{i, p(i)\}} \left(\mid A(i,l) \mid \cdot C \cdot \alpha^{k-1} \cdot w\left(l\right)\right) \\ &\leq C \cdot \alpha^{k} \cdot \left(\mid A(i,p(i)) \mid \cdot w\left(p(i)\right) + \sum_{l \notin \{i, p(i)\}} \mid A(i,l) \mid \right) \\ &\leq C \cdot \alpha^{k} \cdot w\left(i\right) \\ &= q_{(k,j)}(i) \end{aligned}$$

The first inequality is equivalent to Equation 5.3 and obtained by rewriting it appropriately; the second inequality follows since, by assumption, $s_{\mathcal{D}} \in L_{(k,j)}$; the last inequality follows from definition of $w\left(i\right)$. Hence, $\mathcal{Q}_{(k,j)_i}(s'_{\mathcal{D}})$, if $\neg root(i) \wedge k > 0$. The case when $k = 0$ is similar and not reported. This completes the proof of the first case. We have shown that predicate $\mathcal{Q}_{(k,j)_i}(s'_{\mathcal{D}})$ holds if $i \leq_{BF} j$.

We next consider the case when $i >_{BF} j$. Using Equation 5.3 and assuming $k > 0$, we have that

$$\begin{aligned} e\left(s'_{\mathcal{D}}, i\right) &\leq \sum_{l \neq i} \mid A(i,l) \mid \cdot C \cdot \alpha^{k-1} \cdot w\left(l\right) \\ &\leq C \cdot \alpha^{k} \\ &= q_{(k,j)}(i) \end{aligned}$$

The first equality follows from the assumption that $s_{\mathcal{D}} \in L_{(k,j)}$; second inequality follows from the weakly diagonally dominant assumption of A (see Assumption L3) and definition of α. Hence, $\mathcal{Q}_{(k,j)_i}(s'_{\mathcal{D}})$, if $k > 0$. The case when $k = 0$ is similar and not reported.

Putting all together, we have that $\forall t \in \{1, \ldots, N\}$, $\mathcal{Q}_{(k,j)_t}(s'_{\mathcal{D}})$ holds, i.e. $s'_{\mathcal{D}} \in L_{(k,j)}$. □

We next prove that each level set is a convex set.

Lemma 20. *For all $q_{(k,j)} \in \mathbb{P}$, $L_{(k,j)}$ is a convex set, i.e. $\forall s_{\mathcal{D}}, s'_{\mathcal{D}} \in S_{\mathcal{D}}, \forall \beta \in [0,1]$,*

$$(\beta \cdot s_{\mathcal{D}} + (1-\beta) \cdot s'_{\mathcal{D}}) \in S_{\mathcal{D}}$$

Proof. Consider an arbitrary level set $L_{(k,j)}$ with $q_{(k,j)} \in \mathbb{P}$, two arbitrary states $s_{\mathcal{D}}, s'_{\mathcal{D}} \in S_{\mathcal{D}}$ and an arbitrary value $\beta \in [0,1]$. Denote by $\bar{s}_{\mathcal{D}}$ the state given by $\beta \cdot s_{\mathcal{D}} + (1-\beta) \cdot s'_{\mathcal{D}}$. By definition of $\mathcal{A}_{\mathcal{D}}$, the state $\bar{s}_{\mathcal{D}} \in S_{\mathcal{D}}$.

We next show that $\bar{s}_{\mathcal{D}} \in L_{(k,j)}$. The following chain of inequalities holds $\forall i \in \{1, \ldots, N\}$,

$$\begin{aligned} e(\bar{s}_{\mathcal{D}}, i) &= |\, \bar{s}_{\mathcal{D}}(i) - \hat{s}_{\mathcal{D}}(i) \,| \\ &= |\, \beta \cdot (s_{\mathcal{D}}(i) - \hat{s}_{\mathcal{D}}(i)) + (1-\beta) \cdot (s'_{\mathcal{D}}(i) - \hat{s}_{\mathcal{D}}(i)) \,| \\ &\leq \beta \cdot e(s_{\mathcal{D}}, i) + (1-\beta) \cdot e(s'_{\mathcal{D}}, i) \\ &\leq q_{(k,j)}(i) \end{aligned}$$

where the first inequality follows by definition of the error function; the second inequality follows by construction of $\bar{s}_{\mathcal{D}}$; the third inequality follow by triangle inequality and definition of error function; the last inequality follow since $s_{\mathcal{D}}, s'_{\mathcal{D}} \in L_{(k,j)}$. Hence, $\bar{s}_{\mathcal{D}} \in L_{(k,j)}$. □

We next prove that the family of level sets $\{L_{(k,j)}\}_{q_{(k,j)} \in \mathbb{P}}$ is strictly monotonic; we refer to Section 3.1 for the definition of strictly monotonic families.

Lemma 21. *The family $\{L_{(k,j)}\}_{q_{(k,j)} \in \mathbb{P}}$ is strictly monotonic.*

Proof. We denote by $q_{(k1,j1)}, q_{(k2,j2)}$ two elements of $\mathbb{P}$ with $q_{(k1,j1)} <_{\mathbb{P}} q_{(k2,j2)}$. By construction, $L_{(k1,j1)} \subseteq L_{(k2,j2)}$ and our goal is to show that $L_{(k1,j1)} \subsetneq L_{(k2,j2)}$.

We distinguish two cases, $k2 = 0$ and $k2 > 0$.

Consider the case when $k2 > 0$. We define the state $s_{\mathcal{D}} \in S_{\mathcal{D}}$ as follows, $\forall i \in \{1, \ldots, N\}$

$$e(s_{\mathcal{D}}, i) = \begin{cases} C \cdot \alpha^{k2} \cdot w(i) & i \leq_{BF} j2 \\ C \cdot \alpha^{k2-1} \cdot w(i) & i >_{BF} j2 \end{cases}$$

By construction, $s_{\mathcal{D}} \in L_{(k2,j2)}$. We next show that $s_{\mathcal{D}} \notin L_{(k1,j1)}$. We distinguish two cases. In the

first case we assume that $k1 > k2$. By definition, we have that

$$\forall {s'}_{\mathcal{D}} \in L_{(k1,j1)} \ : \ e\left({s'}_{\mathcal{D}}, \min_{BF}(\mathbb{T})\right) \leq C \cdot \alpha^{k1} \cdot w\left(\min_{BF}(\mathbb{T})\right)$$

By construction,

$$e\left(s_{\mathcal{D}}, \min_{BF}(\mathbb{T})\right) = C \cdot \alpha^{k2} \cdot w\left(\min_{BF}(\mathbb{T})\right)$$

Since $k2 < k1$ we have that $\alpha^{k2} > \alpha^{k1}$. Hence, $s_{\mathcal{D}} \notin L_{(k1,j1)}$.
We consider the case when $k1 = k2$ and $j1 >_{BF} j2$. By definition, we have that

$$\forall {s'}_{\mathcal{D}} \in L_{(k1,j1)} \ : \ e\left({s'}_{\mathcal{D}}, j1\right) \leq C \cdot \alpha^{k1} \cdot w\left(j1\right)$$

while

$$e\left(s_{\mathcal{D}}, j1\right) = C \cdot \alpha^{k1-1} \cdot w\left(j1\right)$$

Since $\alpha^{k1-1} > \alpha^{k1}$, we have that $s \notin L_{(k1,j1)}$. This concludes the case when $k2 > 0$.

Consider the case when $k2 = 0$. In this case, we have that $k1 = 0$ and $j2 <_{BF} j1$. We define the state $s_{\mathcal{D}} \in S_{\mathcal{D}}$ as follows, $\forall i \in \{1, \ldots, N\}$:

$$e\left(s_{\mathcal{D}}, i\right) = \begin{cases} C \cdot w\left(i\right) & i \leq_{BF} j2 \\ C & i >_{BF} j2 \end{cases}$$

By construction, $s_{\mathcal{D}} \in L_{(k2,j2)}$. Instead, the state $s_{\mathcal{D}}$ does not belong to $L_{(k1,j1)}$, since $e\left(s_{\mathcal{D}}, j1\right) = C$ while $\forall {s'}_{\mathcal{D}} \in L_{(k1,j1)}$, $e\left({s'}_{\mathcal{D}}, j1\right) \leq C \cdot w\left(j1\right) < C$. $\square$

We finally prove that for each level set of V, except for the smallest one, there exists an action that leads the execution in a strictly contained level set.

Lemma 22. *$\forall q_{(k,j)} \in \mathbb{P}$, with $q_{(k,j)} \neq q_{(\infty, \max_{BF}(\mathbb{T}))}$, $\exists a_{\mathcal{D}} \in \mathcal{A}_{\mathcal{D}}$, such that $\forall s_{\mathcal{D}} \in L_{(k,j)}$,*

$$s_{\mathcal{D}} \in L_{(k,j)} \Rightarrow T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}}) \in L_{(k1,j1)}$$

with $q_{(k1,j1)} \in \mathbb{P}$ and $q_{(k1,j1)} <_{\mathbb{P}} q_{(k,j)}$.

Proof. Consider an arbitrary element $q_{(k,j)} \in \mathbb{P}$, and an arbitrary state $s_{\mathcal{D}} \in L_{(k,j)}$. We distinguish two cases: $j = \max_{BF}(\mathbb{T})$ and and $j <_{BF} \max_{BF}(\mathbb{T})$.

Assume that $j = \max_{BF}(\mathbb{T})$. We define $a_{\mathcal{D}} = le_{\min_{BF}(\mathbb{T})}$ and denote by ${s'}_{\mathcal{D}}$ the state $T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}})$. Our goal is to show that ${s'}_{\mathcal{D}} \in L_{(k1,j1)}$, with $k1 = k + 1$ and $j1 = \min_{BF}(\mathbb{T})$.

Using Lemma 19, we have that $\mathcal{Q}_{(k,j)}({s'}_{\mathcal{D}})$ holds. By construction, predicate $\mathcal{Q}_{(k1,j1)}$ can be

rewritten as

$$\mathcal{Q}_{(k1,j1)} \equiv \mathcal{Q}_{(k1,j1)_{\min_{BF}(\mathbb{T})}} \wedge \left(\bigwedge_{i \neq \min_{BF}(\mathbb{T})} \mathcal{Q}_{(k,j)_i} \right)$$

Hence, we only need to prove that $\mathcal{Q}_{(k1,j1)_{\min_{BF}(\mathbb{T})}}(s'_{\mathcal{D}})$ holds, i.e.

$$e\left(s'_{\mathcal{D}}, \min_{BF}(\mathbb{T})\right) \leq C \cdot \alpha^{k1} \cdot w\left(\min_{BF}(\mathbb{T})\right)$$

Using Lemma 17, we have that the error of agent $\min_{BF}(\mathbb{T})$ in $s'_{\mathcal{D}}$ can be bounded as follows:

$$\begin{aligned} e\left(s'_{\mathcal{D}}, \min_{BF}(\mathbb{T})\right) &\leq C \cdot \alpha^{k} \cdot \sum_{i \neq \min_{BF}(\mathbb{T})} \left(\left| A\left(\min_{BF}(\mathbb{T}), i\right) \right| \cdot w\left(i\right) \right) \\ &\leq C \cdot \alpha^{k} \cdot \alpha \cdot \sum_{i \neq \min_{BF}(\mathbb{T})} \left| A\left(\min_{BF}(\mathbb{T}), i\right) \right| \\ &\leq C \cdot \alpha^{k1} \cdot w\left(\min_{BF}(\mathbb{T})\right) \end{aligned}$$

where the first inequality follows since $s'_{\mathcal{D}} \in L_{(k,j)}$; the second inequality follows from definition of α; the last inequality follows from definition of $w\left(\min_{BF}(\mathbb{T})\right)$. Hence, $\mathcal{Q}_{(k1,j1)_{\min_{BF}(\mathbb{T})}}(s'_{\mathcal{D}})$ holds.

We next consider the case when $j <_{BF} \max_{BF}(\mathbb{T})$. We define $a_{\mathcal{D}} = le_{j1}$ where $j1$ is the smallest of the agent identifiers that succeed j with respect to the Breadth-first traversal. We denote by $s'_{\mathcal{D}}$ the state $T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}})$. Our goal is to prove that $s'_{\mathcal{D}} \in L_{(k,j1)}$.

Predicate $\mathcal{Q}_{(k,j)}(s'_{\mathcal{D}})$ holds, since, by Lemma 19, $L_{(k,j)}$ is stable. By construction, predicate $\mathcal{Q}_{(k,j1)}$ can be rewritten as

$$\mathcal{Q}_{(k,j1)} \equiv \mathcal{Q}_{(k,j1)_{j1}} \wedge \left(\bigwedge_{i \neq j1} \mathcal{Q}_{(k,j)_i} \right)$$

Hence, we only need to prove that $\mathcal{Q}_{(k,j1)_{j1}}(s'_{\mathcal{D}})$ holds, i.e.

$$e\left(s'_{\mathcal{D}}, j1\right) \leq C \cdot \alpha^{k} \cdot w\left(j1\right)$$

Using Lemma 17, we have that when $k1 > 0$, the error of agent $j1$ in $s'_{\mathcal{D}}$ can be bounded as follows:

$$\begin{aligned} e\left(s'_{\mathcal{D}}, j1\right) &\leq \mid A(j1, p(j1)) \mid \cdot C \cdot \alpha^{k} \cdot w\left(p(j1)\right) + \sum_{i \neq \{j1, p(j1)\}} \mid A(j1, i) \mid \cdot C \cdot \alpha^{k-1} \cdot w\left(i\right) \\ &\leq C \cdot \alpha^{k} \cdot \left(\mid A(j1, p(j1)) \mid \cdot w\left(p(j1)\right) + \sum_{i \neq \{j1, p(j1)} \mid A(j1, i) \mid \right) \\ &\leq C \cdot \alpha^{k} \cdot w\left(j1\right) \end{aligned}$$

The first inequality holds, since, by construction $s'_{\mathcal{D}} \in L_{(k,j)}$ and $p(j1) \leq_{BF} j1$; the second inequality follows from definition of α; the last inequality follows from definition of $w\left(j1\right)$. Hence,

$\mathcal{Q}_{(k,j1)_{j1}}(s'_{\mathcal{D}})$. The case when $k1 = 0$ is similar and not reported. Hence, $\mathcal{Q}_{(k,j1)}(s'_{\mathcal{D}})$ holds. □

5.1.3.9 Convergence Property

In this section, we show that $\hat{s}_{\mathcal{D}} = A^{-1}b$ is an asymptotically stable equilibrium state of $\mathcal{A}_{\mathcal{D}}$. This result follows from Theorem 9.

Theorem 23. *The automaton $\mathcal{A}_{\mathcal{D}}$ converges to $\hat{s}_{\mathcal{D}}$.*

Proof. The Lyapunov function V satisfies the assumptions of Theorem 9. Assumption C1 holds, from Lemma 21; Assumption C2, C5 follow from the structure of $\mathbb{P}$; Assumption C3 follows from Lemma 19 while Assumption C4 from Lemma 22, since the system assumes weak fairness. □

We notice that automaton $\mathcal{A}_{\mathcal{D}}$ converges linearly to $\hat{s}_{\mathcal{D}}$ with rate α.

5.1.4 Solving Systems of Linear Equations with Dynamics

In this Section, we further generalize this class of systems. In the schemes presented so far, agents update their state instantaneously. In this Section, we relax this assumption and explicitly model the dynamics of agents. We are interested in this generalization, because in real-worlds applications, it is unrealistic to assume instantaneous updates of the solution vector x. For example, in robotic applications systems of equations are used in pattern formation protocols, where x can be thought of as agent positions. When a robot executes an action, the move from its current location to its newly computed one is not instantaneous. Instead, the robot moves according to some time-related dynamics.

In this class, when an agent executes the updating scheme in Equation 5.1, it evolves its current value towards the newly computed one according to some dynamics. Systems in this class allow an agent to stop before reaching the newly computed value. We model these systems as automata with time actions. Given $\mathcal{A}_{\mathcal{D}}$, we can construct the generic automaton with timed actions $\mathcal{A}_{dyn} = (S_{dyn}, S0_{dyn}, \mathcal{A}_{dyn}, E_{dyn}, T_{dyn})$ modeling the shared-state multi-agent system with explicit arbitrary dynamics using the procedure presented in Section 4.1.2.

We next present the automaton and refer to Section 4.1.2 for the details of its construction.

Definition 48. *Given $\mathcal{A}_{\mathcal{D}} = (S_{\mathcal{D}}, S0_{\mathcal{D}}, A_{\mathcal{D}}, E_{\mathcal{D}}, T_{\mathcal{D}})$, the automaton with time actions $\mathcal{A}_{dyn} = (S_{dyn}, S0_{dyn}, \mathcal{A}_{dyn}, E_{dyn}, T_{dyn})$ modeling a shared-state MAS with N agents with explicit arbitrary dynamics solving a system of linear equations has:*

- $S_{dyn} = (\mathbb{R}^N, \mathbb{R}^N)$,
- $S0_{dyn} = \{(s0, s0)\}$, *with* $s0 \in S0_{\mathcal{D}}$,

- $\mathcal{A}_{dyn} = A_{\mathcal{D}} = \cup_i A_i$ *with* $A_i = \{le_i\}$,
- $E_{dyn} : S_{dyn} \times \mathcal{A}_{dyn} \to true$,
- $\forall s \in S_{dyn}$, $\forall a = le_i \in A_{dyn}$, *action* a *has time duration* $\tau_{s,a}$ *and its behaviour is* $\forall t \in (0, \tau_{s,a}]$,

$$(T_{dyn}(s,a)(t)).x(i) = f_{s,a}(t)$$
$$(T_{dyn}(s,a)(t)).z(i) = \left(b(i) - \sum_{j \neq i} A(i,j) \cdot s.x(j) \right)$$

and $\forall j \neq i$,

$$\begin{aligned} (T_{dyn}(s,a)(t)).x(j) &= s.x(j) \\ (T_{dyn}(s,a)(t)).z(j) &= s.z(j) \end{aligned}$$

The equilibrium state of $\mathcal{A}_{dyn}$ is $\hat{s}_{dyn}$ defined as $(\hat{s}_{\mathcal{D}}, \hat{s}_{\mathcal{D}})$. A Lyapunov function for $\mathcal{A}_{dyn}$ is the function V_{dyn} defined as follows. Given a state $s_{dyn} \in S_{dyn}$, $V_{dyn}(s_{dyn}) = \max\{V(s_{dyn}.x), V(s_{dyn}.z)\}$. By construction, the level sets of V_{dyn} are in conjunctive form, they are stable and satisfies condition C4 of Theorem 9.

Differently from the discrete automaton $\mathcal{A}_{\mathcal{D}}$, in this more general automaton, agents may update their state using values that have not computed using the updating rule. These values derive from the evolution of the agents state according to some dynamics. Under specific assumptions on the dynamics of the agents, we next show that systems in this class are correct, i.e. they converge to the solution of the system of linear equations. Using function V_{dyn}, we next prove convergence of $\mathcal{A}_{dyn}$.

Theorem 24. *If* $\mathcal{A}_{dyn}$ *satisfies Assumptions F3-4, the* $\mathcal{A}_{dyn}$ *converges to* $\hat{s}_{dyn}$.

Proof. It follows from Theorem 11. Assumption F1 holds from Theorem 23. Assumption F2 holds from Lemma 20 and Assumptions F3-4 hold by hypothesis. □

For example, agents moving with constant velocity from their current state to the desired one satisfy Assumptions F3-4.

5.2 Solving Systems of Linear Equations via Message-Passing

In this Section, we present a class of message-passing schemes for solving systems of linear equations. Systems in this class are the message-passing version of the schemes presented in Section 5.1. In these schemes agents interact by sending messages via an unreliable communication medium. We allow for lost, delayed, duplicated or delivered out-of-order messages. The message-passing version

of the Gauss, Jacobi, and Gauss-Seidel algorithms are examples of iterative schemes belonging to this class. We assume a message-passing system with bounded delay d. Under this assumption, messages sent at time t either they are received by time $t + d$ or they are lost.

5.2.1 MAS solving Systems of Linear Equations

This class consists of message-passing decentralized iterative schemes for solving systems of linear equations of the form $A \cdot x = b$. We model this class as message-passing multi-agent systems with N agents. Each agent is responsible for computing the value of a specific variable using a specific equation of the system of linear equations. We assume that agent i is responsible for solving variable $x(i)$. In our model, agent i stores $x(i)$ and repeatedly broadcasts a message containing the current value of this variable. Agent i also stores a vector y_i of length N. The j-th component of y_i contains the content of the last message that i has received from j, for all $j \neq i$. We assume agents broadcast their value infinitely often, where the number of messages sent within a finite time interval is assumed to be finite. Upon receiving a message, agent i updates the corresponding entry in the vector y_i, computes a new value for $x(i)$ as follows

$$x(i) := b(i) - \sum_{j \neq i} A(i,j) \cdot y_i(j) \tag{5.4}$$

and moves towards it according to some dynamics. This updating rule corresponds to the i-th equation of the system where $x(i)$ is the only unknown: agent i uses the values stored in y_i to represent the values of $x(j)$ for all $j \neq i$.

In our model, agent i broadcast the value of $x(i)$ infinitely often. That value is subsequently store in y_j, for some $j \neq i$, as the content of the last message received from i. In our model, for each agent i, there are several values associated with i in the system: one stored in $x(i)$, another in $y_j(i)$ for all $j \neq i$ and a potentially infinite number of messages in transit. The values stored in $y_j(i)$, for all $j \neq i$, and messages in transit can be old copies of $x(i)$. Hence, differently from the shared-state multi-agent system, in this case, an agent can update its value using data that are old and potentially computed at different times.

5.2.2 System of Linear Equations Message-Passing Automaton

As discussed in Section 4.3, we model this message-passing multi-agent system with bounded delay using the automaton with sliding window presented in Section 4.2. The construction of the sliding window automaton requires a shared-state automaton. We consider as shared-state automaton the automaton $\mathcal{A}_{dyn}$. This allows modeling the most general message-passing scheme, where agents compute the new value using the updating rule defined in Equation 5.4 and move towards it.

The definition of the message-passing automaton $\mathcal{A}_{mp}$ follows.

Definition 49. *Given $\mathcal{A}_{dyn} = (S_{dyn}, S0_{dyn}, \mathcal{A}_{dyn}, E_{dyn}, T_{dyn})$, we have that the message-passing automaton $\mathcal{A}_{mp} = (S_{mp}, S0_{mp}, A_{mp}, E_{mp}, T_{mp})$ modeling a message-passing multi-agent system with bounded delay $\mathcal{B}$ and explicit arbitrary dynamics solving a system of linear equations has:*

- $S_{mp} = [-\mathcal{B}, 0] \rightarrow S_{dyn}$,
- $S0_{mp} = \{s0_{mp}\}$, *with* $\forall t \in [-\mathcal{B}, 0] \ : \ s0_{mp}(t) \in S0_{dyn}$,
- $\mathcal{A}_{mp} = A_{dyn}$
- $E_{mp} : S_{mp} \times A_{mp} \rightarrow true$,
- $\forall s_{mp} \in S_{mp}$, $\forall a_{mp} = le_i \in \mathcal{A}_{mp}$, *agent i chooses a state $s \in S_{dyn}$, with $H(s, s_{mp})$ and $s(i) = s_{mp}(0)$*
  $\forall t \in [0, \tau_{s,\widehat{le_i}}], t \leq \mathcal{B}$

$$\begin{aligned} T_{mp}(s_{mp}, a_{mp})(t)[-\mathcal{B}, -t] &= (s_{mp}[t - \mathcal{B}, 0])^{-t} \\ T_{mp}(s_{mp}, a_{mp})(t)[-t, 0] &= (T_{dyn}(s, le_i)[0, t])^{-t} \end{aligned}$$

$\forall t \in [0, \tau_{s,\widehat{le_i}}], t > \mathcal{B}$

$$T_{mp}(s_{mp}, a_{mp})(t)[-\mathcal{B}, 0] = (T_{dyn}(s, le_i)[t - \mathcal{B}, t])^{-t}$$

We denote by $\hat{s}_{mp}$ the equilibrium state of $\mathcal{A}_{mp}$. This state satisfies the property that all its asynchronous views are equal to $\hat{s}_{dyn}$.

5.2.3 Proof of Correctness

In this Section, we show that $\mathcal{A}_{mp}$ converges to $\hat{s}_{mp}$. We derive this property from Theorem 16.

Theorem 25. *If $\mathcal{A}_{dyn}$ satisfies Assumptions F3-4, then $\mathcal{A}_{mp}$ converges to $\hat{s}_{mp}$.*

Proof. If follows from Theorem 16. Using Assumptions F3-4, Theorem 24 holds. Hence, Assumption H1 of Theorem 16 holds for V_{dyn}. Assumption H2 of Theorem 16 holds, since by construction the level sets of the Lyapunov function V_{dyn} are in conjunctive form. □

For example, agents moving with constant velocity from their current state to the desired one satisfy Assumptions F3-4.

5.3 Linear Robot Pattern Formation Protocol

In this Section, we discuss a specific example. We consider a multi robot system operating over an unreliable communication medium. The goal of the system is to form an equispaced straight line. We prove the correctness of this system using the results of the previous Section.

5.3.1 Linear Robot Patter Formation Multi-Agent System

In this Section, we describe the system. The system consists of $N+1$ robots, each with a unique identifier. Robots of the system start at some arbitrary locations and their goal is to form a specific spatial configuration using simple local rules. We denote by $x0$ the vector of the initial robot positions. Without loss of generality, we assume that robot positions are real values. The system consists of two leaders with identifiers 0 and N and $N-1$ followers. In the final configuration, robots form a straight equispaced line with leaders at the extremes of the line. Figure 2.1(a) represents a generic configuration of the system consists of ten agents, while Figure 2.1(b) represents the corresponding goal configuration.

Agents in the system use a leader-follower protocol where leaders are stationary while followers update their positions using their immediate neighbors. The follower agent i with $i \in \{1, \ldots, N-1\}$ computes its new position as the average of the positions of agents $i-1$ and $i+1$ and moves towards it. Agents communicate via message-passing where messages may lost, delayed or received out-of-order.

This system is a special case of the Line-Up multi-agent system. If we restrict the set of actions of the Line-Up multi-agent system and allow agents to communicate only with their immediate neighbors, i.e. the action set is $\{avg_{i-1,i,i+1}\}_{0<i<N}$, we obtain an automaton modeling the linear robot pattern formation multi-agent system.

5.3.2 Solving a System of Linear Equations

In this Section, we model this system as a system for solving linear equations. We instantiate matrix A and vector b; then, we show that the updating rules executed by the robots can be expressed using the equations of the system of linear equations and we show that final goal of the multi-agent system is equal to the solution $A^{-1}b$ of the system of equations.

Matrix $A \in \mathbb{R}^{N+1} \times \mathbb{R}^{N+1}$ is defined as follows. Matrix A is a tri-diagonal matrix with all elements along the main diagonal are equal to 1, i.e.

$$\forall i \in \{0, \ldots, N\} \ : \ A(i,i) = 1$$

The entries of the second left diagonal are equal to -0.5, except for entry N that is equal to 0, i.e.

$$(A(N, N-1) = 0) \wedge (\forall i \in \{1, \dots N-1\} \ : \ A(i, i-1) = -0.5)$$

and all entries of the second right diagonal are equal to -0.5, except for entry 1 that is equal to 0, i.e.

$$(A(0, 1) = 0) \wedge (\forall i \in \{1, \dots N-1\} \ : \ A(i, i+1) = -0.5)$$

For example, in the case when $N = 5$, the matrix A is

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -0.5 & 1 & -0.5 & 0 & 0 \\ 0 & -0.5 & 1 & -0.5 & 0 \\ 0 & 0 & -0.5 & 1 & -0.5 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Vector $b \in \mathbb{R}^{N+1}$ has the following structure. Entries of the leader agents are their initial positions, i.e. $b(0) = x0(0)$ and $b(N) = x0(N)$; entries of the followers are equal to 0, i.e. $\forall i \neq \{0, N\} : b(i) = 0$. For example, in the case when $N = 5$, the vector b is $(x0(0), 0, 0, 0, x0(N))$.

Plugging A, b in the updating rule in Equation 5.1 of agent i, we get that

$$\begin{array}{rcll} x(0) & := & x0(0) & \\ x(i) & := & \dfrac{x(i-1) + x(i+1)}{2} & \forall i, 0 < i < N \\ x(N) & := & x0(N) & \end{array}$$

This protocol ensures that agents 0 and N are stationary; while agent i, with $0 < i < N$, computes its destination location as the average of the positions of agent $i-1$ and agent $i+1$.

The solution of the system $A^{-1}b$ is equal to the vector in Equation 2.3. We refer to [67] for computing the inverse of a tri-diagonal matrix.

5.3.3 Proof of Correctness

In this Section, we present the automaton model for the multi-agent system and prove its correctness. The automaton $\mathcal{A}^l_{mp}$ modeling the robot pattern formation MAS specialized the automaton presented in Section 5.2.2. It can be also obtained by restricting the set of actions of of the automaton $A_{\mathcal{B}}$ presented in Section 4.2.3. The automaton $\mathcal{A}^l_{mp}$ has the same state and set of initial states of $A_{\mathcal{B}}$. The set of actions of $\mathcal{A}^l_{mp}$ is a subset of the set of actions of $A_{\mathcal{B}}$; the set of actions of $\mathcal{A}^l_{mp}$ consists of $\{avg_{i-1,i,i+1}\}_{0<i<N}$.

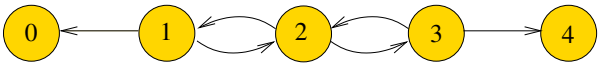


Figure 5.3: The communication graph $G = (V, E)$ corresponding to matrix A when $N = 5$.

The correctness of this system can be derived from Theorem 25. This Theorem holds if matrix A satisfies Assumptions L1-4 and the communication graph of matrix A satisfies Assumption M1.

Lemma 26. *Matrix A satisfies Assumptions L1-4*

Proof. Matrix A satisfies Assumption L1, since its determinant is non-zero (see [67]). All entries of its main diagonal are equal to 1, thus satisfying Assumption L2. Matrix A satisfies Assumption L3, i.e. it is weakly diagonally dominant. This is because the sum of the absolute values of the non-diagonal entries along row 0 and N is 0 and it is 1 along the remaining rows. These values are bounded by 1, that is the value along the main diagonal. Matrix A satisfies Assumption L4; it is strictly diagonally dominant in row 0 and N where the sum of the absolute values of the non-diagonal entries is 0 and the value along the diagonal is 1. □

Graph $G = (V, E)$ is a linear graph with the following structure. The set of vertices V is equal to $\{1, \dots N\}$. Vertices 0 and N are strictly diagonally dominant. The out-degree of these two vertices is 0 and their in-degree is 1; vertex 0 has an incoming edge from vertex 1, and vertex N has an incoming edge from vertex $N-1$. The generic vertex i, with $0 < i < N$, has two outgoing edges, towards $i-1$ and $i+1$. Vertices 1 and $N-1$ have one incoming edges from 2 and $N-2$ respectively and vertex i, with $1 < i < N-1$, has two incoming edges, from $i-1$ and $i+1$. For example, Figure 5.3 represents the graph $G = (V, E)$ in the case when $N = 5$.

Lemma 27. *Communication graph $G = (V, E)$ satisfies Assumption M1,*

Proof. G satisfies Assumption M1, because from vertex i, with $0 < i < N$, there exists a path leading towards 0 and one leading towards N. These two paths are $i, i-1, i-2, \dots, 0$ and $i, i+1, i+2, \dots, N$. □

Hence, we can now prove the main theorem.

Theorem 28. *If the dynamics of the agents satisfy Assumptions F3-4, then $\mathcal{A}^l_{mp}$ converges to $\hat{s}^l_{mp}$.*

Proof. It follows from Theorem 25, since matrix A satisfies Assumptions L1-4 (see Lemma 26) and graph G satisfies Assumption M1 (see Lemma 27). □

We briefly discuss the proof of Theorem 25. This proof requires to fix a strictly diagonally dominant rooted forest and construct a Lyapunov function using this forest. A feasible forest $\mathbb{F}$ consists of two trees, denoted by T_0 and T_N. Tree T_0, rooted at vertex 0, is of the form $(V(T_0), E(T_0))$ with $V(T_0) = \{0, 1, \dots, N-1\}$ and $E(T_0) = \{(i-1, i) \mid 0 < i < N\}$. In this tree, the parent of

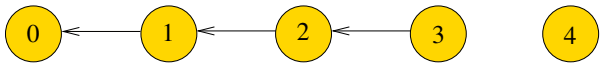


Figure 5.4: A strictly diagonally dominant rooted forest of the communication graph in Figure 5.3.

vertex i, with $i \in \{1, \ldots, N-1\}$ is $i-1$. Tree T_N consists of only vertex N. For example, Figure 5.4 represents $\mathbb{F}$ in the case when $N = 5$.

Given this $\mathbb{F}$, the weights of the agents are as follows. The weight of agents 0 and N are 0. For agent i, with $0 < i < N$, the weight of agent i is

$$w(i) = \left(1 - \frac{1}{2^i}\right)$$

This sequence of weights defines an increasing sequence of values with respect to the Breadth-first traversal of T_0, i.e. given $i, j \in V(T_0)$ if $i <_{BF} j$ then $w(i) < w(j)$. The value of α is the maximum of this sequence, i.e. $\alpha = w(N-1)$. The value of constant C depends on the initial positions of the robots, and is defined as the maximum error of the initial positions, i.e.

$$C = \max_{i \in \{0,\ldots,N\}} e(x0, i)$$

Assuming forest $\mathbb{F}$, the execution starts at level set $L_{0,0}$ where the errors of agent 0 and N are bounded by 0 and the error of the remaining agents is bounded by C. Then, it eventually enters level set $L_{0,1}$ where the errors of agent 0 and N are bounded by 0, the error of agent 1 is bounded by $0.5 \cdot C$ and the error of the other agents is bounded by C. Then it eventually enters level set $L_{0,2}$ and so on. Hence, any execution is contained in this sequence of level sets given by:

$$L_{0,0}, L_{0,1}, \ldots, L_{0,N-1}, L_{1,0}, L_{1,0}, \ldots L_{1,N-1}, L_{2,0}, \ldots L_{2,N-1}, L_{3,0}, \ldots,$$

For example, in Figure 5.5, we represent the level set $L_{k,3}$.

5.4 Discussion

In this Section, we discuss the main results of this Chapter and relate them to the literature.

In this Chapter, we have presented a general class of multi-agent systems. Systems in this class are iterative decentralized schemes whose goal is solving a system of linear equations.

We have shown their correctness using the results presented in the previous Chapters. In the case of shared-state multi-agent systems, we have applied the results of Chapter 3. In the case of message-passing systems, we have derive their correctness from the correctness proof of the corresponding shared-state systems using the results of Chapter 4. The proofs of shared-state and message-passing

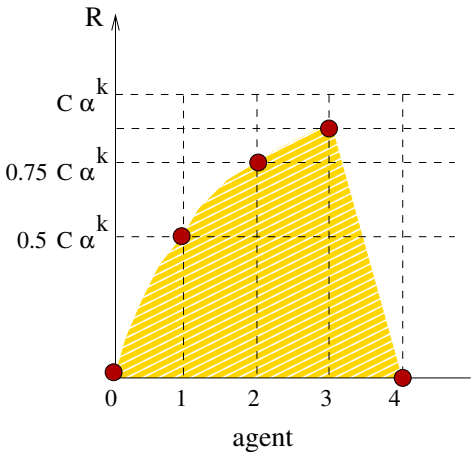


Figure 5.5: Pictorial representation of $L_{k,3}$ in the case when the system consists of 5 agents, i.e. $N = 5$. A vector v belongs to $L_{k,3}$ if $v(0) = v(4) = 0$ and $v(1) \in [0, 0.5 \cdot C \cdot \alpha^k]$, $v(2) \in [0, 0.75 \cdot C \cdot \alpha^k]$ and $v(3) \in [0, 0.875 \cdot C \cdot \alpha^k]$. The set $L_{k,3}$ corresponds to the shaded area.

systems require a specific structure of matrix A. The matrix has to be weakly diagonally dominant (see Assumption L3) and strongly diagonally dominant in at least one row (see Assumption L4). We also require the matrix to be invertible (see Assumption L1) and the main diagonal entries to be all 0 (see Assumption L2). The invertibility property ensures the existence and uniqueness of the solution of the system.

In these proofs, we show that the error of the system at each point of the computation eventually decreases by a factor of α. This property is proven using an induction scheme over the set of agents. As a base case, we show that eventually the maximum error of agents corresponding to strictly diagonally dominant rows is reduced by α. Then, assuming that this is the case for agents at distance k from some agent satisfying Assumption L4, we show that the property holds for all agents at distance $k+1$. This induction scheme is represented as a Breadth-first visit on a forest of trees rooted at agents corresponding to strictly diagonally dominant rows. Iterating this property, we have that, denoting by C the initial error of the system, the error eventually decreases to $\alpha \cdot C$, then to $\alpha^2 \cdot C$, $\alpha^3 \cdot C$, and so on, converging to 0 as time tends to infinity.

Assumptions L3-4 are crucial in the proof of correctness. Assumption L3 ensures that the level sets of the Lyapunov function are stable, and Assumption L4 ensures that the system eventually enters a strictly contained level set. Informally, Assumption L3 ensures that the error of the system at each point of the computation does not increase while Assumption L4 ensures that it eventually decreases by a factor of α.

Schemes in this class can solve the system of linear equations even if they operate over an unreliable communication medium. When executing these schemes via message-passing, agents compute a new estimate of the solution vector using values that are potentially old and computed

at different times. Hence, the system converges even if the execution is not time-stepped. This is a desirable property from an implementation point of view. This is because, we do not need to implement a synchronization mechanism, such as a barrier, to simulate rounds. Agents can proceed at different speeds and agents with faster computational capabilities do not need to wait on agents with slower computational capabilities. We assume that an agent cannot be infinitely faster than the others.

The material covered in this Chapter has been published in [14]. Our work extends previous work on systems of equations such as [18, 28]. In [18], the authors consider message-passing systems whose goal is solving systems of linear equations. They prove correctness of these schemes in the case when matrix A is strictly diagonally dominant, i.e. it is strictly diagonally dominant in all its rows. Our results relax this assumption; we only require that the matrix is strictly diagonally dominant in at least one row. In [28], the authors consider shared-state systems whose goal is solving linear and non-linear systems of equations. They provide conditions that ensure convergence of this general class of systems. In the case of linear schemes, these conditions require the matrix A to be strictly diagonally dominant.

Chapter 6

PVS Verification Framework

In this Chapter, we present a framework for verifying the class of distributed message-passing systems discussed in Chapter 5. This tool has been implemented within the PVS theorem prover and can be downloaded from [58].

Section 6.1 describes the architecture of this framework. Section 6.2, Section 6.3 and Section 6.4 present implementation details of the tool. Section 6.5 discusses the challenges in the implementation of the tool. Section 6.6 presents an application of our framework to the linear robot pattern formation multi-agent system.

6.1 Systems of Linear Equations PVS Verification Framework

In this Section, we describe the structure of the tool. It consists of a set of PVS meta-theories, built on top of I/O automata meta-theories [72, 42, 3, 2, 4] with extensions for timed and hybrid systems [36, 46, 45]. This tool partially uses the PVS NASA libraries [37].

It consists of three main libraries:

- Mathematical PVS library
- Message-Passing System PVS library
- Verification PVS library

The Mathematical PVS library includes definitions and properties of vectors and square matrices. The Message-Passing System PVS library models the message-passing distributed system as a I/O automaton. The Verification PVS library encodes the proof of correctness of the generic protocol for solving systems of linear equations.

We refer to Figure 6.1 for a pictorial representation of the interactions among these libraries. Specifically, the Message-Passing System PVS library uses the data structures defined in the Math-

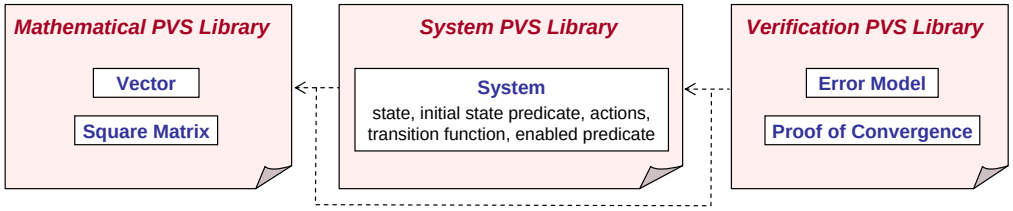


Figure 6.1: Architecture of the PVS Verification Framework.

ematical PVS Library. The Verification PVS libraries proves convergence of the I/O automaton modeled in the Message-Passing System library using theorems from the Mathematical PVS library.

In Section 6.2 we present the Mathematical PVS library, in Section 6.3 we discuss the Message-Passing System library and in Section 6.4 we discuss the Verification PVS library.

6.2 Mathematical Library

In this Section, we present the mathematical library. It consists of two main meta-theories, `Vector` meta-theory and `Matrix` meta-theory. We first discuss the `Vector` meta-theory and then the `Matrix` meta-theory.

6.2.1 `Vector` PVS meta-theory

The `Vector` meta-theory extends the PVS NASA Vector meta-theory. A vector is encoded in PVS as a function from `Index` to real numbers:

```
Index: TYPE = upto(N)
Vector: TYPE = [Index->real]
```

where `Index`, defined in the PVS NASA library [37], is the set $\{0, 1, \ldots, N\}$ with `N` being the number of agents of the system.

The `Vector` library extends the PVS NASA library for vectors. It includes definitions of vector operators, such as absolute value operator and cross product operator, that are not defined in the PVS NASA library for Vectors. In this theory, we also prove some properties of these operators.

The main operators defined in this library are presented in Figure 6.2(a). We briefly discuss their meaning and PVS implementations:

- The `abs` operator computes the absolute value of a vector, defined as the absolute value of its

components. It is encoded in PVS as a function from `Vector` to `Vector`; it uses the lambda PVS operator for accessing the input vector components. This operator is used for defining diagonally dominant vectors.

- The `prod` operator computes the dot product of two vectors, i.e., it returns a new vector where each of its entries consists of the product of the corresponding entries in the input vectors. In PVS, it is a function from `Vector` $\times$ `Vector` to `Vector` and uses the lambda PVS operator for accessing the components of the input vectors. This operator is used for defining the cross product of two vectors.

- The `sum` operator computes the sum of the elements of a vector. In PVS, it is encoded as a recursive function from `Vector` to `real`. Together with `prod` operator, it defines the cross product of two vectors.

- The `cross_product` operator computes the cross product of two vectors. We encode it in PVS as a function from `Vector` to `real`. By definition, it combines `prod` and `sum` operators. This operator implements the protocol executed by agents in the system, see Equation 5.1.

The `Vector` library contains predicates on vectors, as well. In Figure 6.2(b), we present the PVS definition and implementation of the main ones. We briefly discuss their meaning:

- Predicate `eq?` holds if the two input vectors `v1` and `v2` are equal, i.e., all their components are equal. For example, we can use this predicate for encoding the commutativity property of the `prod` operator in PVS.

- Predicate `pos?` holds if the input vector is nonnegative, i.e., all its components are nonnegative. For example, this predicate can be used for encoding the positivity property of the `abs` operator.

- Predicate `dd?` holds if the input vector `v` is weakly diagonally dominant with respect to its i-th component. Specifically this predicate requires that the sum of all components of the vector except i is bounded by the value of the i-th component. This predicate is extensively used for encoding the weakly diagonally dominant assumption of the rows of matrix A presented in Assumption L3.

- Predicate `sdd?` holds if the input vector `v` is strictly diagonally dominant with respect to its i-th component. This predicate is used for encoding the strictly diagonally dominant assumption of matrix A (see Assumption L4).

```
% absolute value operator
abs(v: Vector): Vector = LAMBDA(i: Index): abs(v(i))

% dot product operator
prod(v1,v2: Vector): Vector = LAMBDA(i: Index): v1(i)*v2(i)

% sum operator
sum(v: Vector, i: Index):  RECURSIVE real =
IF (i=0) THEN v(0) ELSE v(i) + sum(v,i-1) ENDIF
MEASURE (i)
sum(v: Vector) : real = sum(v,N-1)

% cross product operator
cross_product(v1,v2: Vector): real = sum(prod(v1,v2))
```

(a) Operators in the `Vector` meta-theory.

```
% equality predicate
eq?(v1,v2: Vector): bool = FORALL(i: Index): v1(i) = v2(i)

% positivity predicate
pos?(v: Vector):    bool = FORALL(i: Index): v(i) >= 0

% diagonally dominance predicate
dd?(v: Vector,i: Index): bool = sum(v)-v(i) <= v(i)

% strictly diagonally dominance predicate
sdd?(v: Vector,i: Index): bool = sum(v)-v(i) < v(i)
```

(b) Predicates in the `Vector` meta-theory.

Figure 6.2: Predicates and Operators defined in the PVS `Vector` meta-theory.

6.2.2 `Matrix` PVS meta-theory

The `Matrix` meta-theory introduces the type `Matrix` and defines operators and predicates on square matrices. We define a square matrix in PVS as a function from pair of `Index` to real numbers, in PVS

```
Matrix: TYPE = [Index, Index -> real]
```

The type `Matrix` is our extension to the PVS NASA library [37].

This meta-theory encodes operators and predicates on square matrices. Figure 6.3(a) presents the main and operators, while Figure 6.3(b) presents the main predicates. We discuss their main features:

- The PVS `row` operator extracts row `r` of the input matrix `m`. The type of the extracted row is `Vector`. This operator uses the lambda PVS operator for constructing this vector. Similarly, the PVS `col` operator extracts column `c` of the input matrix `m`. We use these operators for encoding properties of rows and columns of the matrix.

- The PVS `abs` operator computes the absolute value of the input matrix `m`. It is a function from `Matrix` to `Matrix`. Each entry of the resulting matrix stores the absolute value of the corresponding entry of matrix `m`. We use this function for encoding the diagonally dominance properties.

- The PVS `prod` operator computes the product of the two input matrices `m1` and `m2`. It is encoded as a function from `Matrix` $\times$ `Matrix` to `Matrix`. Entry (i, j) of the resulting matrix stores the cross product of row i and column j. We use this function for encoding the invertibility property of matrix A (see Assumption L1).

- The PVS `eq?` predicate holds if the two input matrices `n` and `m` are equal, i.e. for each entry they store the same value. We use this predicate for encoding the invertibility property of matrix A.

- The PVS `diag?` predicate holds if the input matrix is an identity matrix, i.e. its main diagonal consists of all 1 and the remaining entries are equal to 0. Together with `prod` and `eq?`, we use this predicate for encoding the invertibility property of matrix A.

- The PVS `inv?` predicate holds if the input matrix is invertible. We check the invertibility property by showing the existence of a left and right inverse matrix. This predicate encodes Assumption L1 of matrix A.

- The PVS `dd?` predicate holds if the matrix is weakly diagonally dominant, i.e., each row of the matrix is weakly diagonally dominant. This predicate encodes Assumption L3 of matrix A.

```
% extract row operator
row(m: Matrix, r: Index): Vector = LAMBDA(c: Index): m(r,c)

% extract column operator
col(m: Matrix, c: Index): Vector = LAMBDA(r: Index): m(r,c)

% absolute value operator
abs(m: Matrix): Matrix = LAMBDA(r,c: Index): abs(m(r,c))

% product operator
prod(m1,m2: Matrix): Matrix = LAMBDA (r,c: Index): cross_product(row(m1,r),col(m2,c))
```

(a) Operators of the `Matrix` meta-theory.

```
% identity matrix predicate
diag?(m): bool = FORALL (r,c) : IF r=c THEN m(r,c)=1 ELSE m(r,c)=0 ENDIF

% equality predicate
eq?(m,n): bool = FORALL (r,c) : m(r,c)=n(r,c)

% invertibility predicate
inv?(m): bool = EXISTS(n:Matrix): eq?(prod(m,n),prod(n,m)) AND diag?(prod(n,m))

% diagonally dominant predicate
dd?(m):  bool = FORALL(r:Index): dd?(row(abs(m),r),r)

% strictly diagonally dominant predicate
sdd?(m): bool = EXISTS(r:Index): sdd?(row(abs(m),r),r)
```

(b) Predicates of the `Matrix` meta-theory.

Figure 6.3: Predicates and Operators defined in the PVS `Matrix` meta-theory.

- The PVS `sdd?` predicate holds if the matrix is strongly diagonally dominant, i.e., there exists a strongly diagonally dominant row. This predicate encodes Assumption L4 of matrix A.

The library includes lemmas on matrices needed in the proof of correctness.

6.3 Message-Passing System PVS Library

In this Section, we model the multi-agent system in PVS. We specialize the automaton meta-theory [46] for the case of message-passing systems. The automaton models the state of the system, its set of actions, initial predicate, enabling conditions and transition function.

6.3.1 System state

The state of the system is made up of the state of the agents, along with the state of the communication channel. An agent is responsible for its current and target values, along with the set of messages in the communication channel for whom it is the recipient. We define the system state in PVS with the type `S`, outlined in Figure 6.4. The fields `target`, and `lastmsg` describe the state of the agents, while `buffer` describes the state of the channel. The field `now` corresponds to the

```
S: TYPE = [# target:  Vector,                    % vector of agents
             lastmsg: Matrix,                    % matrix of values
             buffer: [Index, Index -> Pset],     % state of the channel
             now:     nonneg_real,               % system clock
             next:   [Index -> nonneg_real] #]  % agent send deadlines
```

Figure 6.4: System state. Refer to Figure 6.5 for the definition of `Pset`.

```
Msg :TYPE = [# loc: real,
                id: Index #]
Pkt :TYPE = [# msg: Msg,
               ddl: posreal #]
Pset:TYPE = set[Pkt]
b   :posreal
d   :posreal
```

Figure 6.5: Channel Types Components of the system automaton.

clock of the system, storing the current time. The field `next` is a vector containing, for each agent, the future time that agent is allowed to execute a *send* action. The `target` field stores the target value of each agent. Finally, `lastmsg` is a matrix in which its diagonal entries hold the current value of each agent; the non-diagonal entries store the last message that agent i has received from agent j. The `target` and the diagonal entries of `lastmsg` fields correspond to the variables x and z, respectively, from the mathematic model outlined in Chapter 5.

The initial condition of the system is described using the predicate *starts*?. It holds in the state `s` if the global clock of this state is set to 0, the `target` to the initial guess `x0`, and ensures that `next` does not violate the parameter `d` (defined in Section 6.3.2):

```
start?(s: S): bool =
  now(s) = 0 AND (FORALL(i: Index): next(s)(i) <= d) AND
  target(s) = x0 AND (FORALL(i, j: Index): lastmsg(s)(i,j) = x0(i))
```

Note that the communication channels are not necessarily empty initially.

6.3.2 Communication Medium

The communication layer is a broadcast channel allowing for lost, delayed, or out-of-order messages. The architecture of the communication medium has been presented in Section 4.3.1. We briefly discuss it.

We model the faulty communication medium by defining packets, channels and timing variables. Figure 6.5 outlines the PVS datatypes used for this purpose. Messages sent between agents (`Msg`) is represented as a record, consisting of the agents location and identifier. Messages, along with their

```
ACS: DATATYPE BEGIN
 send(p:Pkt, i:Index, d1:posreal): send?
 receive(p:Pkt, i:Index): receive?
 move(i:Index, delta_t:posreal): move?
 msgloss(p:Pkt, i:Index): msgloss?
 nu_traj(delta_t:posreal): nu_traj?
END ACS
```

Figure 6.6: Actions of the system.

delivery deadline, are contained within packets (`Pkt`). Sets of packets (`Pset`) make up a dedicated, directed, channel between two agents. Because a set lacks ordering, it makes it an appropriate type for a communication channel that allows for out-of-order messages. Timing within the channel are handled by the constants `b` and `d`. The former is an upper bound on packet deadlines—each packet deadline is at most `b` units of time—and is used to model message delay. The constant `d` is an upper bound on the interval between consecutive *send* actions. Using ⟨`d`,`next`⟩, we ensure that the *send* action is executed infinitely often.

6.3.3 System actions

Actions within our system consist of agent movement and message transmission, channel manipulation, and system clock maintenance. Our PVS definitions are outlined in Figure 6.6. The `send`, `receive`, and `move` actions are executed by agents, while the `msgloss` action is used by the communication channel to simulate packet loss. Finally, the `nu_traj` action updates the system time. This section describes the behavior of each, as well as when they are enabled.

New trajectory. The `nu_traj` action advances the time variable of the system, `now`, by `delta_t` units, where `delta_t` is the input parameter of the action;

```
nu_traj(delta_t: posreal): s WITH [now := now(s) + delta_t]
```

It is enabled when the new value of the global clock does not violate a packet deadline:

```
nu_traj(delta_t): FORALL(p: Pkt): ddl(p) >= now(s) + delta_t
```

Agent move. The `move` action models the movement of an agent from its current value to a new value based on its locally computed solution to the equation of the system. The parameters of the action are the agent that moves and the time interval. In our implementation, agent i sets $z[i]$ (stored in `lastmsg(i,i)`) to $x[i]$ (stored in `target(i)`) and advances the global clock of `delta_t` units; in PVS

```
move(i: Index, delta_t: posreal): s WITH
```

```
  [ lastmsg := lastmsg(s) WITH [ (i, i) := target(s)(i) ],
        now := now(s) + delta_t]
```

This action can be executed only if packet deadlines are violated by the new time of the system:

```
move(i, delta_t): FORALL(p: Pkt): ddl(p) >= now(s) + delta_t
```

Agent send. When executing a `send` action, agent `i` broadcasts its packet `p` to all agents in the system and schedules its next send:

```
send(p: Pkt, i: Index, d1: posreal): s WITH [
  buffer := LAMBDA (k, j: Index):
   IF ((k = i) AND (j /= i)) THEN union(p, buffer(s)(k, j))
   ELSE buffer(s)(k, j)
   ENDIF,
  next := next(s) WITH [(i):= next(s)(i) + d1 ]]
```

In updating the buffer, the agent is adding its packet, `p`, to all of its outgoing channels. Notice that an agent does not send a message to itself. The `send` action is executed only if the time when the agent is allowed to send equals to the global time of the system. Furthermore, the sent packet must contain the identifier of the agent, its current target location, and correct packet deadline. The detailed PVS code follows

```
send(p, i, d1): next(s)(i) = now(s) AND d1 <= d AND
                id(msg(p)) = i AND loc(msg(p)) = target(s)(i) AND
                ddl(p) = now(s) + b
```

Agent receive. When agent `i` receives packet `p`, it updates the `lastmsg` variable, computes a new value for its `target`, and removes the packet from the channel:

```
receive(p: Pkt, i: Index):
 LET m: Msg = msg(p), j: Index = id(m), l: real = loc(m),
     Ci: vector = update(row(lastmsg(s), i), j, l) IN s WITH
  [ buffer := buffer(s) WITH
    [ (j,i)   := remove(p, buffer(s)(j, i)) ],
      lastmsg := lastmsg(s) WITH [ (i, j) := l ],
      target  := target(s)  WITH [ (i) := gauss(Ci, i) ]]
```

The `gauss` function implements Equation 5.1. The action can be executed if the `p` is in the channel from `msg(p)` to `i`, and its deadline does not violate the global time of the system,

```
receive(p, i): buffer(s)(id(msg(p)), i)(p) AND ddl(p) >= now(s)
```

Message loss. Message loss is modeled by removing a given packet from a directed channel:

```
msgloss(p: Pkt, i: Index): LET m: Msg = msg(p), j: Index = id(m) IN s
  WITH [ buffer := buffer(s)
    WITH [ (j, i) := remove(p, buffer(s)(j, i)) ]]
```

It is enabled only if the packet belongs to this channel:

```
msgloss(p, i): buffer(s)(id(msg(p)), i)(p)
```

6.4 Verification PVS Library

In this Section, we describe the proof of correctness in PVS. The library carries out the proof for the message-passing system without reducing it to a shared-state system as described in Chapter 5. We refer to [14] for details on this proof. We cannot encode the proof of Theorem 16, because it requires a PVS framework for proving the correctness of generic message-passing multi-agent systems from the corresponding shared-state ones. The implementation of this general framework is currently under investigation.

In the following subsections, we discuss error function representation (encoded in `Error Model` meta-theory), assumptions of the problem, stability and convergence proofs (encoded in `Proof of Correctness` meta-theory).

6.4.1 Error Model

The error model is defined in the `Error Model` theory. The error of an agent is defined as the distance between its current value and its desired value. The desired value `xstar` is an input of the theory and it satisfies the fixed point Assumption:

```
xstar_def_ax: ASSUMPTION xstar(i) = gauss(xstar,i)
```

During system execution, the value of an agent is represented in three places: within its state, within the set of packets in transit on its outgoing channels, and within the received message field of other agents. Although these are all values of the same agent, agent dynamics, message delay and reordering do not guarantee their equality. In the proof of correctness, we define an error function for each of these errors. Their definition is presented in Figure 6.7.

We define the error of an agent `mes` as the maximum of them,

```
mes(s,i): nonnegative_real = max(mae(s,i), mbe(s,i))
```

```
% error of target position of agent i
de(s,i):  nonnegative_real = abs(xstar(i)-target(s)(i))

% error of current position of agent i
le(s,i):  nonnegative_real = abs(xstar(i)-lastmsg(s)(i,i))

% error of the position of agent i stored by agent j
re(s,i,j):nonnegative_real = abs(xstar(i)-lastmsg(s)(j,i))

% error of position of agent i stored in packet p
be(p:Packet): nonnegative_real = abs(xstar(id(msg(p)))-loc(msg(p)))
```

Figure 6.7: Error values of agent i.

```
% maximum error function within outgoing channels
% defined axiomatically
mbe : [S,I -> nonnegative_real]

% maximum value is an upper bound on the errors within outgoing channels
mbe_all_error:
  AXIOM FORALL(p,j):
  buffer(s)(id(msg(p)),j)(p) IMPLIES be(p)<=mbe(s,id(msg(p)))

% maximum value is stored in a packet
mbe_ex_error :
  AXIOM FORALL(i):
  EXISTS(p,j) :  id(msg(p))=i AND buffer(s)(i,j)(p) AND be(p)=mbe(s,i)


% maximum error within agent states
mre(s,i): nonnegative_real = max(LAMBDA (j) : re(s,i,j))

%
mae(s,i): nonnegative_real = max(de(s,i),mre(s,i))
```

Figure 6.8: Maximum error of agent i within its out-going channels and within agent states.

where `mbe` is the maximum error of agent i within the set of packets in transit on its outgoing channels, and `mae` is the maximum error of agent i within agent states. We refer to Figure 6.8 for definitions of `mbe` and `mae`.

6.4.2 Proof of Correctness PVS meta-theory

In this Section we present the `Proof of Correctness` meta-theory. This theory has a set of input parameters, an assuming clause environment and the proof of correctness of the generic message-passing multi-agent system.

6.4.2.1 Inputs and Assumptions

Inputs to this meta-theory are:

```
ASSUMING
  % L1
  inverse_exist: ASSUMPTION inv?(A)

  % L2
  diag_entry:    ASSUMPTION FORALL(i: Index): A(i, i) = 1

  % L3
  diag_dominant: ASSUMPTION dd?(A)

  % L4
  strictly_diag_dominant: ASSUMPTION sdd?(A)
ENDASSUMING
```

Figure 6.9: Assumptions on matrix A.

- `N` of type `posnat`, storing the number of agents in the system;
- `A` of type `Matrix`, storing matrix A;
- `b` of type `Vector`, storing vector b;
- `x0` of type `Vector`, storing the initial position vector $x0$;
- `xstar` of type `Vector`, storing the solution of the system of equations, i.e $A^{-1} \cdot b$;
- `ancs` of type `I -> list[I]`, storing the structure of the rooted forest used in the proof of correctness.

We encode the assumptions of the meta-theory using the PVS assumption environment. This environment facilities the access of properties within the meta-theory and obligate users of our library to discharge them. The assuming clause contains assumptions on the structure of the matrix, on structure of the rooted forest and on solution vector.

Assumptions on matrix A are presented in Figure 6.9 and correspond to conditions L1-4. They have been encoded in PVS using predicates defined in `Matrix` meta-theory; the definition and implementation of these predicates has been presented in Section 6.2.

We encode the arbitrary strictly diagonally dominant rooted forest using the function `ancs`; this function maps each input agent i to the complete path from i to a rooted node of the forest. This path is stored as a PVS list of identifiers. The user checks the validity of the forest data structure by discharging certain assumptions on it. These assumptions are reported in Figure 6.10.

We briefly discuss these assumptions:

- Assumption `root_` ensures that root vertices are strictly diagonally dominant;
- Assumption `edge_` ensures that an agent and its parent are connected in the underlying communication graph;

```
ASSUMING
        root_: ASSUMPTION root?(i) IMPLIES sdd?(A,i)
        edge_: ASSUMPTION
                not_root?(i) IMPLIES edge(A,i,parent(i))/=0
        path_1: ASSUMPTION
                not_root?(i) IMPLIES ancs(i)=cons(parent(i),ancs(parent(i)))
        path_2: ASSUMPTION
                not_root?(i) AND not_root?(j) AND member(j,ancs(i))
                IMPLIES member(parent(j),ancs(i))
        no_repetition:
        ASSUMPTION not_root?(i) IMPLIES NOT member(i,ancs(i))
        induct_: ASSUMPTION
                (FORALL (i|root?(i)) : predic(i)) AND
                (FORALL (j|not_root?(j)): predic(parent(j)) IMPLIES predic(j))
                IMPLIES (FORALL(k:I) : predic(k))
ENDASSUMING
```

Figure 6.10: Assumptions on the forest of trees.

- Assumption `path_1,path_2` and `no_repetition` ensure that the input graph is a forest, i.e. it has no cycles;
- Assumption `induct_` defines an induction scheme along the forest; if some property holds at the roots of the forest, and, given a node, we prove that it holds at the node, given that it holds for its parent, then we can safely derive that the property holds for all nodes of the forest.

As stated above, the solution vector `xstar` of the system of equations satisfies the fixed point assumption.

As future work, we would remove `ancs` and `xstar` as input parameters of the theory. This requires implementing a Deep-First Search Algorithm for constructing a forest from the communication graph and implementing matrix inversion algorithms. In this case, assumptions on `ancs` and `xstar` would be properties of these two data-structure to prove.

6.4.2.2 Proof of Correctness Theorems

Reasoning about system convergence requires the analysis of the system throughout an arbitrary execution. Our responsibility is to show that

- the error of the system does not increase, and that
- it eventually decreases by a lower-bounded amount.

Using the diagonally dominant assumption on `A`, we can prove the first condition:

```
not_incr_error: LEMMA enabled(a, s) IMPLIES me(s) >= me(trans(a, s))
```

To prove the second condition, as discussed in Section 5.1.3, we use an arbitrary forest, encoded in PVS by the input function `ancs`.

The proof requires to define a factor α by which the system will eventually decrease. Given the rooted forest, for each node of the forest we recursively define the quantity `p_value`. We prove that this value is positive and (strictly) upper bounded by 1. The factor α is the maximum of these quantities. In PVS,

```
alpha: real = max(LAMBDA (i:I): p_value(i))
```

We use extensively these two lemmas about α:

```
alpha_all: LEMMA FORALL(i: Index): p_value(i) <= alpha
alpha_ex:  LEMMA EXISTS(j: Index): alpha = p_value(j)
```

Using induction on the forest (see Assumption `induct_` in Figure 6.10), we prove that the maximum error of the system eventually decreases by α. Assuming that the error of the system is upper-bounded by W, the base case prove that the error of the roots of the tree eventually decreases by α. From there, we prove, assuming that the error of all ancestors of a node is upper-bounded by $W \cdot \alpha$, that eventually the error of the node is upper bounded by the same quantity.

6.5 Framework Discussion

In this section, we offer commentary on our experience using PVS. Our library consists of over 200 lemmas, and approximately 8700 proof steps. We took advantages of PVS pre- and user-defined types for modeling the system and the proof of correctness.

Vectors, matrices and forests were used extensively throughout our library. Developing a sufficient infrastructure based around these structures consumed about 15% of our effort (with respect to the number of proof steps). The PVS NASA libraries provided some relief, but modeling diagonally dominant matrices and proving lemmas on products of matrices and vectors forced us to extend them. Although NASA does provide a representation of trees and forests, their recursive implementation made proving properties we required very difficult. Unlike the NASA implementation, where trees are traversed starting from the leaves, we needed to prove properties on the tree starting from the root and induct over the structure as well. For this reason, we preferred to represent the forest using function `ancs` and ensure the needed properties using assumptions. The end-user is required to discharge these assumptions.

We managed the proof of convergence by breaking it into smaller lemmas. This allowed us to tackle small proofs where the goal was to show that eventually a specific property holds. For example, proving that the error of the target position of an agent eventually decreases by the constant factor

α; then proving that its error in the outgoing channels eventually decreases by this factor; and finally showing that its error stored in the state of the remaining agents eventually decreases by this factor. Using this collection of sublemmas, we were able to prove that eventually the maximum error of each agent decreases by the factor α.

Our libraries did not introduce new PVS proof strategies; the system-defined strategies, such as `grind` and `induct-and-simplify` were sufficient. Future work includes investigation onto how our libraries can take advantage of the proof strategy capabilities of PVS.

6.6 Verification of the Linear Robot Pattern Formation Protocol in PVS

In this Section, we prove correctness of the linear robot pattern formation protocol discussed in Section 5.3 using the tool presented in this Chapter. The system consists of $N+1$ robots, with robots 0 and N fixed throughout the execution. Agent i, with $0 < i < N$, communicates with its immediate neighbors, $i-1$ and $i+1$. Upon receiving a message from $i-1$,$i+1$, i computes its new position as the average of the left and right received values and moves towards it. As shown in Section 5.3.2, this system can be expressed as a multi-agent system solving a system of linear equations of the form $A \cdot x = b$.

In order to use the tool for proving convergence, we implement a new theory, import and instantiate appropriately the main theory of the tool and discharge its assumptions. We next discuss the structure of the theory describing the linear robot pattern formation system; this theory can be downloaded from [58] and it is called `Linear System Agents`.

6.6.1 Parameters

Parameters of the theory are:

- `N`, storing the number of agents,
- `lx`, storing the initial position of agent 0, and
- `rx`, storing the initial position of agent N.

6.6.2 PVS Instantiations

In this theory, we encode in PVS matrix A, vector b, vector of initial positions $x0$, vector of final positions $\hat{x}$ and spanning forest $\mathbb{F}$, defined in Section 5.3.2. Initial robot positions are stored in the PVS `init` vector; without loss of generality, we assume that initial position of agent i, with $0 < i < N$, is zero. In PVS,

```
init: Vector = LAMBDA (r):
    IF (r=0) THEN lx
    ELSIF (r=N) THEN rx
    ELSE 0
    ENDIF
```

Matrix A and vector b are encoded in PVS as follows:

```
A: Matrix = LAMBDA (r,c):
    IF (r=c) THEN 1 % main diagonal entries
    ELSIF (r>0 AND r<N AND (c=r-1 OR c=r+1))
        THEN -0.5 % secondary diagonal entries
    ELSE 0
    ENDIF
```

and

```
b: Vector = LAMBDA (r):
    IF (r=0 OR r=N) THEN init(r)
    ELSE 0
    ENDIF
```

As discussed in Section 5.3.2, matrix A is a tri-diagonal matrix having value 1 along the main diagonal, and value -0.5 on the secondary diagonals (with the exception of rows 0 and N, which have 0 on the secondary diagonals) and b is a vector having only two non-zero entries (entry 0 storing the initial position of agent 0 and entry N storing the initial position of agent N).

Final robot positions are stored in the `xstar` vector; in PVS,

```
xstar: Vector = LAMBDA (r):
    init(0) * (N-r) / N + init(N) * r / N
```

As discussed in Section 5.3, the goal of the robots is to form an equi-spaced line with extremes `lx` and `rx`.

As discussed in Section 5.3.3, we encode in PVS the following spanning forest of the communication graph rooted at strictly diagonally dominant nodes:

```
ancs(i:I): RECURSIVE list[I] =
    IF (i=0 OR i=N) THEN null
    ELSE cons(i-1,ancs(i-1)) ENDIF
    MEASURE (i)
```

This forest consists of two trees. The first one is rooted at 0 and contains all nodes except N. In this tree, the parent of node i, with $0 < i < N$, is $i - 1$. The other tree consists of only node N.

6.6.3 Proving Correctness of the Protocol

In order to prove the correctness of this system in PVS, the `Linear System Agents` imports `Proof of Correctness` meta-theory and instantiates its parameters. As discussed in Section 6.4.2.1, parameters of `Proof of Correctness` theory are the number of agents in the system, matrix A, vector b, vector of initial positions, vector of final positions and a spanning forest of the communication graph rooted at strictly diagonally dominant nodes. The import clause in the `Linear System Agents` theory becomes

```
IMPORTING ProofofCorrectness[N, A, b, init, ancs, xstar]
```

When importing this meta-theory, the system generates 11 TCCs. These TCCs correspond to the assumptions of the `Proof of Correctness` meta-theory and encode properties of A, $\mathbb{F}$ and $\hat{x}$ needed to be discharged (see Section 6.4.2.1). We discuss them in the next subsection.

6.6.4 Discharging Library Assumptions

In Table 6.1, we present a summary of our effort in discharging the assumptions of the tool. Almost 57% of our effort went into proving the invertibility property of A, the remaining 43% was equally divided into proving the remaining assumptions of A, the assumptions on the forest and the fixed-point assumption on the solution vector. Our total effort can be estimated into 1230 proof steps.

Assumptions	**Proof Steps**	**Pct.**
Invertibility (L1)	702	57.0%
Normalized (L2) Weakly Diagonally Dominant (L3) Strongly Diagonally Dominant (L4)	190	15.4%
Structure of Forest (Figure 6.10)	170	13.8%
Fixed point property of `xstar` (Equation 5.2)	168	13.7%
Total	1230	

Table 6.1: Number of proof steps needed for discharging the assumptions of the tool.

We next discuss the invertibility property of A. Assumption L1 has been the most difficult to discharge, because it required us to explicitly construct the inverse of the matrix and prove that it is both left and right inverse. To this end, we used the method outlined in [67] for constructing the inverse. This inverse is a function of the number of agents. The method first computes the principal minors of matrix A,

```
minor(i: Index): real =
```

```
  IF (i = N) THEN expt(1/2, N-1) * N
  ELSE expt(1/2,i) * (i+1)
  ENDIF
```

where `expt` is the standard exponential function, defined in the PVS prelude. Using this function it computes the determinant of A:

```
det: real = minor(N)
```

This method then computes the sequence $\{\phi_i\}$ as follows:

```
phi(i: Index): real =
  IF (N=2) THEN 1
  ELSIF (i==0) THEN expt(1/2, N-3) * (N-2)
  ELSE expt(1/2, N-1-i) * (N-i)
  ENDIF
```

If finally defines A^{-1}:

```
invA: Matrix =
  LAMBDA(r,c: Index):
    IF (r=c) THEN diag_inv(r)
    ELSIF (r>c) THEN l_prod(r,c)
    ELSE r_prod(r,c)
    ENDIF
```

where

```
diag_inv(i: Index): real =
  (IF (i=0) THEN 1 ELSE minor(i-1) ENDIF) *
  (IF (i=N) THEN 1 ELSE phi(i+1) ENDIF) /
  det
```

and

```
l_prod(i,j: Index): real =
  (IF even?(i+j) THEN 1 ELSE -1 ENDIF) *
  (IF (i>j) THEN
    IF (i=N) THEN 0
    ELSE expt(-0.5, i-j)
    ENDIF
  ELSE 0
  ENDIF) *
```

```
  (IF (j=0) THEN 1 ELSE minor(j-1) ENDIF) *
  (IF (i=N) THEN 1 ELSE phi(i+1) ENDIF) /
  det
```

and

```
r_prod(i,j: Index): real =
  (IF even?(i+j) THEN 1 ELSE -1 ENDIF) *
  (IF (i>j) THEN
    IF (i=0) THEN 0
    ELSE expt(-0.5, j-i)
    ENDIF
  ELSE 0
  ENDIF) *
  (IF (j=0) THEN 1 ELSE minor(i-1) ENDIF) *
  (IF (i=N) THEN 1 ELSE phi(j+1) ENDIF) /
  det
```

After constructing the inverse of A, we prove lemmas about the product of matrix A and its inverse and show that the product matrix is an identity matrix.

Chapter 7

Properties of Automata in the Presence of Exogenous Inputs

In this Chapter, we present theorems about systems in the presence of exogenous inputs, which represent changes to the environment in which the system operates. We present sufficient conditions that guarantee that the system converges to a desired state when the environment does not change (i.e. no exogenous inputs); however, when the environment changes the desired system state may also change and therefore convergence to a changing desired state may not be possible. We give sufficient conditions that ensure that the distance between the actual state and the desired state is eventually bounded. We apply these results to the class of automata presented in Chapter 5. The automata in this class solve systems of linear equations.

In Section 7.1 we present assumptions on the automaton in the absence of exogenous inputs and on the automaton of the exogenous inputs. In Section 7.2 we show some properties of the executions of the automaton in the presence of exogenous inputs. In Section 7.3 we prove the main result of this Chapter. We show that if the assumptions presented in Section 7.1 hold, then the exogenous automaton is bounded. In Section 7.4 we apply these results to the class of automata solving systems of linear equations. Finally, in Section 7.5 we discuss the main result presented in this Chapter and relate them to the current literature.

Throughout this Chapter, we denote by $\mathcal{A}$ an automaton modeling a system in the absence of exogenous inputs, by $\bar{\mathcal{A}}$ an automaton modeling the exogenous inputs and by $\mathcal{A}_{exog}$ the exogenous automaton, i.e. the automaton modeling the system in the presence of exogenous inputs. We refer to Section 2.4 for the definitions of the exogenous automaton $\mathcal{A}_{exog}$ and the automaton of the exogenous inputs $\bar{A}$. We recall that $\mathcal{A}, \bar{\mathcal{A}}$ and $\mathcal{A}_{exog}$ operate over the same state space; we denote this state space by S. We denote the components of $\mathcal{A}$ by $(S, S0, A, E, T)$, the set of equilibrium states of $\mathcal{A}$ by $\hat{S}$ and the components of $\bar{A}$ by $(S, S0, \bar{A}, \bar{E}, \bar{T})$. Automaton $\mathcal{A}_{exog}$ is constructed using the procedure in Section 2.4.

7.1 Assumptions

In this Section, we present assumptions on the system without external inputs and on the external inputs. Later in this Chapter, we prove that if these conditions hold then the distance between the actual and time-varying desired states of the system is bounded. We make the following assumptions:

Assumptions.

N1. *S is closed under $+$ operator, i.e.*

$$\forall s, \bar{s} \in S \ : \ s + \bar{s} \in S$$

N2. *transition function T of $\mathcal{A}$ is additive with respect to S, i.e.*

$$\forall s, \bar{s} \in S, \forall a \in A, \forall t \in [0, \tau_{s+\bar{s},a}] \ : \ T(s + \bar{s}, a)(t) = T(s, a)(t) + T(\bar{s}, a)(t)$$

N3. *transition function $\bar{T}$ of $\bar{A}$ is defined as*

$$\forall s \in S, \ \forall a \in \bar{A}, \ \forall t \in [0, \tau_{s,a}], \ : \ \bar{T}(s, a)(t) = s + v_{s,a,t}$$

where $v_{s,a,t} \in S$

N4. *given function $g : S \to \hat{S}$, $\exists \alpha$, with $0 \leq \alpha < 1$, such that $\mathcal{A}$ converges linearly to g with rate α,*

N5. *the error function $e : S \to \mathbb{R}_{\geq 0}$, defined as $e(s) = d(s, g(s))$ is sub-additive, i.e.*

$$\forall s, \bar{s} \in S \ : \ e(s + \bar{s}) \leq e(s) + e(\bar{s})$$

N6. *$\exists C \geq 0$ such that $\forall s0 \in S0$, $e(s0) \leq C$ and $\forall s \in S, \forall a \in \bar{A}, \forall t \in [0, \tau_{s,a}]$, $e(v_{s,a,t}) \leq C$*

We next discuss these Assumptions. Assumption N1 requires the state space S to be closed under $+$ operator. As an example, we consider the state space of the shared-state Line-Up automaton presented in Section 2.1.2. In this example, the state space is $\mathbb{R}^{N+1}$ and it is closed under $+$ operator, defined as addition between vectors. Assumption N2 requires the transition function of $\mathcal{A}$ to be additive with respect to the state space of $\mathcal{A}$. This assumption defines a specific structure for the transition function; it requires that $\forall s, \bar{s} \in S$, $a \in A$, the time durations of a in state s and in state $\bar{s}$ are upper bounds to the time duration of action a in state $s + \bar{s}$. Formally, $\tau_{s+\bar{s},a} \leq \tau_{s,a}$ and $\tau_{s+\bar{s},a} \leq \tau_{\bar{s},a}$. For example, the transition function of the shared-state Line-Up automaton presented in Section 2.1.2 is additive with respect to the state space $\mathbb{R}^{N+1}$. In this example, actions

are discrete. Assumption N3 requires a specific behaviour of the exogenous inputs injected in the systems. These inputs model quantities that can be added to or subtracted from the current state of the system. Throughout this Chapter, $\forall s \in S$, $\forall a \in \bar{A}$, we denote by $v_{s,a,t}$ the input injected at time t when executed the exogenous action a in state s. As an example, we consider the exogenous inputs defined in Section 2.4 for the shared-state Line-Up automaton presented in Section 2.1.2. These exogenous inputs corresponds to the action *move* defined in Equation 2.5 and Equation 2.6. When this action is executed on state $s \in \mathbb{R}^N$, it adds to s the time-varying quantity $v \cdot t$, with v constant and $t \in [0, \tau_{s,move}]$. Assumption N4 requires that for all states $s \in S$ the automaton in the absence of exogenous inputs with initial state s converges linearly to $g(s)$, with $g : S \to \hat{S}$. We refer to Section 3.4 for the definition of convergence of an automaton to a function. The constant α is such that the error of the system starts in $e(s)$ and eventually decreases to $\alpha \cdot e(s)$, then to $\alpha^2 \cdot e(s)$ and, thus, converging to 0. We refer to Section 3.4 for the definition of linear convergence. Assumption N5 requires that function e defined on the state space S is sub-additive. We recall that d is a distance function on the state space S. This assumption restricts the possible behaviour of function g. In order to show that the automaton in the presence of exogenous inputs is bounded, this assumption requires that function g together with function d define a sub-additive function e. For example, if function g is additive and d is a norm function on S, then function e is sub-additive. Assumption N6 requires that the error of the initial system and the error of the exogenous inputs are uniformly bounded by C. Together with Assumption N4, we have that the system starting in any initial state of $\mathcal{A}$ or in any $v_{s,a,t}$ ($\forall s \in S$, $\forall a \in A$, $\forall t \in [0, \tau_{s,a}]$) converges linearly to $g(s)$ with rate α and error bounded by C.

7.2 Properties of Executions of Exogenous Automata

In this Section, we present an upper-bound on the error of the executions of the automaton in the presence of exogenous inputs.

We denote by π_{exog} an execution fragment of $\mathcal{A}_{exog}$. By construction, π_{exog} can be finite or infinite and, as discussed in Section 2.2.2, it can start while executing the first action of the fragment (see Figure 2.9), or, in case of finite fragment, it can stop before completing the execution of the last action of the fragment (see Figure 2.8).

We denote by π the projection of π_{exog} on $\mathcal{A}$. This is an execution fragment where actions of π_{exog} in $\bar{A}$ are treated as no-op operations, i.e. their execution does not change the state of the system. The initial state of the π is equal to the initial state of π_{exog}. Consider the case when an exogenous input action a ($a \in \hat{A}$) is the first action of π_{exog}. Suppose that π_{exog} starts while executing action a; by construction, $\exists s \in S, t \in [0, \tau_{s,a}]$ such that the execution of action a in π_{exog} corresponds to $T(s, a)[t, \tau_{s,a}]$. In this case, we have that the post-state of the execution of action a

in π is s, i.e. $\pi(\tau_{s,a} - t) = s$.

Given an action $a \in \pi_{exog}$, with $a \in \bar{A}$, we denote by $\pi_{exog,a}$ the suffix of π_{exog} starting from the pre-state of the execution of action a in π_{exog}. If a is the first action of π_{exog} and the system starts while executing action a, we have that $\pi_{exog,a}.fstate = \pi_{exog}.fstate$.

We denote by π_a a new execution fragment that has as initial state the exogenous input injected at the end of the execution of a in $\pi_{exog,a}$. The fragment π_a executes all actions of $\mathcal{A}$ in $\pi_{exog,a}$ in the same order as $\pi_{exog,a}$. Consider the case when $a \in \hat{A}$ is the last action of π_{exog}. Suppose that π_{exog} ends before completing action a, i.e. there exists $s \in S$, $t \in (0, \tau_{s,a})$, such that $\pi_{exog}.lstate = T(s,a)(t)$. In this case the fragment π_a has 0 time duration and consists only of the input injected at state $T(s,a)(t)$.

We next show an upper bound on the error of a finite execution fragment of $\mathcal{A}_{exog}$.

Lemma 29. *Given a finite execution fragment π_{exog} of $\mathcal{A}_{exog}$,*

$$e(\pi_{exog}.lstate) \leq e(\pi.lstate) + \sum_{\substack{a \in \pi_{exog} \\ a \in \bar{A}}} e(\pi_a.lstate)$$

Proof. We fix an arbitrary finite execution fragment π_{exog} of $\mathcal{A}_{exog}$. We denote by s' the final state of π_{exog}, i.e. $s' = \pi_{exog}.lstate$. Using Assumption N3 and Assumption N2, we have that

$$s' = \pi.lstate + \sum_{\substack{a \in \pi_{exog} \\ a \in \bar{A}}} \pi_a.lstate$$

Using Assumption N5, we have that

$$\begin{aligned} e(s') &= e\left(\pi.lstate + \sum_{\substack{a \in \pi_{exog} \\ a \in \bar{A}}} \pi_a.lstate\right) \\ &\leq e(\pi.lstate) + \sum_{\substack{a \in \pi_{exog} \\ a \in \bar{A}}} e(\pi_a.lstate) \end{aligned}$$

□

Before computing the bound on the error of an infinite execution, we introduce the concept of epochs. By Assumption N4, there exists a sequence of consecutive time intervals, called epochs, such that $\forall s \in S$ the error of the automaton $(S, \{s\}, A, E, T)$ at the end of epoch k, with $k \geq 0$, is bounded by $C_s \cdot \alpha^{k+1}$, with $C_s = e(s)$. For each epoch k, we denote by $\mathbb{L}(k)$ the length of epoch k.

We next consider infinite executions. Given the sequence of epochs, we denote by $\pi[k]$, $k \geq 0$, the prefix of π ending at the end of epoch k; we denote by $\pi_{exog}[k]$ the corresponding fragment in π_{exog}.

Lemma 30. *Let π_{exog} be an infinite execution of $\mathcal{A}_{exog}$. Then, for all epoch k, with $k \geq 0$,*

$$e(\pi_{exog}[k].lstate) \leq (\alpha^{k+1} \cdot C) + \left(C \cdot \mathbb{L} \cdot \sum_{j=0}^{k} \alpha^{k-j} \right)$$

where $\mathbb{L} = \max_{j \in \{0,\ldots,k\}} \mathbb{L}(j)$.

Proof. Using Lemma 29, the error of π_{exog} at the end of epoch k can be bounded as follows:

$$e(\pi_{exog}[k].lstate) \leq e(\pi[k].lstate) + \sum_{\substack{a \in \pi_{exog}[k] \\ a \in \bar{A}}} e(\pi_a[k].lstate) \tag{7.1}$$

Using Assumption N4 and Assumption N6, the error of $\pi[k].lstate$ is upper bounded as follows

$$e(\pi[k].lstate) \leq \alpha^{k+1} \cdot C$$

The summation term of Equation 7.1 can be partitioned into epochs:

$$\sum_{\substack{a \in \pi_{exog}[k] \\ a \in \bar{A}}} e(\pi_a[k].lstate) \leq \sum_{j=0}^{k} \sum_{\substack{a \in epoch(j) \\ a \in \bar{A}}} e(\pi_a[k].lstate)$$

Using Assumption N4 and Assumption N6, we have that

$$e(\pi_a[k].lstate) \leq \alpha^{k-j} \cdot C$$

Hence,

$$\sum_{j=0}^{k} \sum_{\substack{a \in epoch(j) \\ a \in \bar{A}}} e(\pi_a[k].lstate) \leq \sum_{j=0}^{k} \alpha^{k-j} \cdot C \cdot \mathbb{L}(j)$$

where $\mathbb{L}(j)$ is an upper bound on the number of adversary actions in epoch j. Hence,

$$e(\pi_{exog}[k].lstate) \leq \left(\alpha^{k+1} \cdot C\right) + \left(C \cdot \mathbb{L} \cdot \sum_{j=0}^{k} \alpha^{k-j} \right)$$

where $\mathbb{L} = \max_{j=0,\ldots,k} \mathbb{L}(j)$. □

7.3 Properties of the Exogenous Automaton

In this Section, we present the main result of the Chapter. Under the Assumptions presented in Section 7.1, we prove the exogenous automaton is bounded with respect to some function g.

Theorem 31. *If Assumptions N1-6 hold, then $\mathcal{A}_{exog}$ is bounded with respect to g.*

Proof. Let π_{exog} an infinite execution of $\mathcal{A}_{exog}$. Using Lemma 30, we have that

$$e(\pi_{exog}[k].lstate) \leq \alpha^{k+1} \cdot C + C \cdot \mathbb{L} \cdot \sum_{j=0}^{k} \alpha^{k-j}$$

Taking the limit of the last quantity as k goes to infinity, we obtain that the system is bounded with respect to g with constant

$$L = \frac{1}{1-\alpha} \cdot C \cdot \mathbb{L}$$

□

We notice that the value of L in the proof of Theorem 31 is not a strict upper bound. We can construct executions of the system where the distance between the current state and the corresponding desired state is L.

Assume that $\forall s \in S, \forall a \in \bar{A}, \forall t \in [0, \tau_{s,a}]$, the state $v_{s,a,t}$ injected when executing action a in state s at time t is an equilibrium state, i.e. $v_{s,a,t} \in \hat{S}$. This extra condition leads to the following stronger result:

Theorem 32. *If Assumptions N1-6 hold and*

N7. $\forall s \in S, a \in \bar{A}, \forall t \in [0, \tau_{s,a}]$, $v_{s,a,t} \in \hat{S}$,

then $\mathcal{A}_{exog}$ is 0-bounded with respect to g.

Proof. Let π_{exog} an infinite execution of $\mathcal{A}_{exog}$. Using Assumption N7, Equation 7.1 reduces to

$$e(\pi_{exog}[k].lstate) \leq e(\pi[k].lstate)$$

since $\forall a \in \pi[k]$, with $a \in \bar{A}$, $e(\pi_a[k].lstate) = 0$. The quantity $e(\pi[k].lstate)$ converges to 0 as k goes to infinity, by Assumption N4. □

This theorem ensures that the system in the presence of exogenous automaton converges to $\hat{S}$, even if the system is driven by non-zero exogenous inputs.

7.4 Solving Systems of Linear Equations in the Presence of Exogenous Inputs

In this Section, we present properties of the class of systems discussed in Chapter 5 in the presence of exogenous inputs. This class consists of decentralized schemes whose goal is solving systems of equations of the form $A \cdot x = b$. For example, the systems presented in Figure 1.3 and Figure 1.6

can be expressed as message-passing multi-agent systems whose goal is to solve a system of linear equations in the presence of exogenous inputs. We focus on shared-state multi-agent systems. We construct a generic exogenous automaton for this class and derive conditions on the exogenous inputs that guarantee the automaton in presence of exogenous inputs to be bounded. We first discuss exogenous automata for multi-agent systems with discrete actions; then, we consider exogenous automata for multi-agent systems with timed actions. We conclude the Section with a discussion of message-passing schemes for solving systems of linear equation in the presence of exogenous inputs. We refer to Section 5.1 for a detailed discussion of shared-state multi-agent systems solving systems of linear equations and to Section 5.2 for a discussion of message-passing systems.

7.4.1 Solving Systems of Linear Equations with Discrete Actions

In this Section, we present properties of a generic exogenous automaton for shared-state systems with discrete actions solving systems of linear equations.

7.4.1.1 Exogenous Automaton

In this Section, we describe the generic exogenous automaton for shared-state systems with discrete actions. This automaton, denoted by $\mathcal{A}_{exog,\mathcal{D}} = (S_{exog,\mathcal{D}}, S0_{exog,\mathcal{D}}, A_{exog,\mathcal{D}}, E_{exog,\mathcal{D}}, T_{exog,\mathcal{D}})$, combines the generic discrete automaton $\mathcal{A}_{\mathcal{D}} = (S_{\mathcal{D}}, S0_{\mathcal{D}}, A_{\mathcal{D}}, E_{\mathcal{D}}, T_{\mathcal{D}})$ presented in Section 5.1.2 and a generic exogenous input automaton $\bar{\mathcal{A}} = (S_{\mathcal{D}}, S0_{\mathcal{D}}, \bar{A}_{\mathcal{D}}, \bar{E}_{\mathcal{D}}, \bar{T}_{\mathcal{D}})$ using the procedure presented in Section 2.4.

In this Section, we consider an extension of the automaton $\mathcal{A}_{\mathcal{D}}$ defined in Section 5.1.2. We assume that $\mathcal{A}_{\mathcal{D}}$ explicitly models vector b in its state space. This is because we allow the exogenous input automaton $\bar{\mathcal{A}}$ to modify vector b. A state $s \in S_{\mathcal{D}}$ becomes of a pair, where the first component of the pair is vector x and the second component is vector b. A state $s \in S_{\mathcal{D}}$ is of the form $s = (x, b)$; we refer to the first component of s as $s.x$ and to the second component of s by $s.b$. The automaton $\mathcal{A}_{\mathcal{D}}$ does not modify component b of the state. This component is set to the input parameter b, initially. Actions of the system does not change it, they can only change component x of the state. The proof of convergence of $\mathcal{A}_{\mathcal{D}}$ of Section 5.1.3 remains valid. A state $s_{\mathcal{D}}$ is an equilibrium state of $\mathcal{A}_{\mathcal{D}}$ if the x component of the state is the solution of the system of linear equations $A \cdot x = s_{\mathcal{D}}.b$, i.e. $s_{\mathcal{D}}.x = A^{-1} \cdot s_{\mathcal{D}}.b$. By construction of the action set, for all vectors of initial guess, $\mathcal{A}_{\mathcal{D}}$ converges to the equilibrium state $\hat{s}_{\mathcal{D}} = (A^{-1} \cdot b, b)$, where b is an input parameter. We denote by $\hat{S}_{\mathcal{D}}$ the set of equilibrium states of $\mathcal{A}_{\mathcal{D}}$.

7.4.1.2 Properties of the Exogenous Automaton

We next discuss some properties of the exogenous automaton $\mathcal{A}_{exog,\mathcal{D}}$. Some of these properties are derived from properties of the discrete automaton $\mathcal{A}_{\mathcal{D}}$.

We first show that the state space $S_{\mathcal{D}}$ is closed under addition, where the addition operation between states is defined as follows:

$$\forall s_{\mathcal{D}}, \bar{s}_{\mathcal{D}} \in S_{\mathcal{D}} \ : \ s_{\mathcal{D}} + \bar{s}_{\mathcal{D}} = (s_{\mathcal{D}}.x + \bar{s}_{\mathcal{D}}.x, s_{\mathcal{D}}.b + \bar{s}_{\mathcal{D}}.b)$$

The addition between two states defines a pair, where the first component of the pair is the sum of the x component of the two states, and the second component of the pair is the sum of the b component of the states.

Lemma 33. *The state space $S_{\mathcal{D}}$ is closed under addition.*

Proof. It follows since $\mathbb{R}^N$ is a vector space. □

We next show that the transition function $T_{\mathcal{D}}$ of $\mathcal{A}_{\mathcal{D}}$ is additive.

Lemma 34. *$T_{\mathcal{D}}$ is additive with respect to $S_{\mathcal{D}}$.*

Proof. Our goal is to prove that $\forall s_{\mathcal{D}}, \bar{s}_{\mathcal{D}} \in S_{\mathcal{D}}$, $\forall a_{\mathcal{D}} = le_i$,

$$T_{\mathcal{D}}(s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}, a_{\mathcal{D}}) = T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}}) + T_{\mathcal{D}}(\bar{s}_{\mathcal{D}}, a_{\mathcal{D}})$$

Fix two arbitrary states $s_{\mathcal{D}}, \bar{s}_{\mathcal{D}} \in S_{\mathcal{D}}$ and an arbitrary action $a_{\mathcal{D}} = le_i \in A_{\mathcal{D}}$. By construction, we have that action $a_{\mathcal{D}}$ does not modify component b of the state. Hence,

$$\begin{aligned} T(s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}, a_{\mathcal{D}}).b &= (s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}).b \\ &= s_{\mathcal{D}}.b + \bar{s}_{\mathcal{D}}.b \\ &= T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}}).b + T_{\mathcal{D}}(\bar{s}_{\mathcal{D}}, a_{\mathcal{D}}).b \end{aligned}$$

where the first inequality holds by construction of action $a_{\mathcal{D}}$; the second inequality, by definition of the addition operation; and the third inequality holds by construction of action $a_{\mathcal{D}}$.

We next consider component x of the state. By construction of action $a_{\mathcal{D}} = le_i$, only component x of agent i is modified. Hence, for all $j \in \{1, \ldots, N\}$ with $j \neq i$, we have that

$$\begin{aligned} T(s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}, a_{\mathcal{D}}).x(j) &= (s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}).x(j) \\ &= s_{\mathcal{D}}.x(j) + \bar{s}_{\mathcal{D}}.x(j) \\ &= T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}}).x(j) + T_{\mathcal{D}}(\bar{s}_{\mathcal{D}}, a_{\mathcal{D}}).x(j) \end{aligned}$$

We next consider the i-th component of $T(s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}, a_{\mathcal{D}}).x$. In this case, we have that

$$\begin{aligned}
T_{\mathcal{D}}(s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}, a_{\mathcal{D}}).x(i) &= (s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}).b(i) - \sum_{j \neq i} A(i,j) \cdot (s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}).x(j) \\
&= s_{\mathcal{D}}.b(i) + \bar{s}_{\mathcal{D}}.b(i) - \sum_{j \neq i} A(i,j) \cdot (s_{\mathcal{D}}.x(j) + \bar{s}_{\mathcal{D}}.x(j)) \\
&= \left(s_{\mathcal{D}}.b(i) - \sum_{j \neq i} A(i,j) \cdot s_{\mathcal{D}}.x(j) \right) + \left(\bar{s}_{\mathcal{D}}.b(i) - \sum_{j \neq i} A(i,j) \cdot \bar{s}_{\mathcal{D}}.x(j) \right) \\
&= T_{\mathcal{D}}(s_{\mathcal{D}}, a_{\mathcal{D}}).x(i) + T_{\mathcal{D}}(\bar{s}_{\mathcal{D}}, a_{\mathcal{D}}).x(i)
\end{aligned}$$

where the first inequality follows from definition of action $a_{\mathcal{D}}$; the second inequality follows from construction of addition operator; the third inequality rewrites the previous inequality; and the last inequality follows from definition of the transition function. $\square$

We next define an additive function $g : S_{\mathcal{D}} \to \hat{S}_{\mathcal{D}}$. It is defined as follows: $\forall s_{\mathcal{D}} \in S_{\mathcal{D}}$,

$$g(s_{\mathcal{D}}) = (A^{-1} \cdot s_{\mathcal{D}}.b, s_{\mathcal{D}}.b)$$

This function maps a state $s_{\mathcal{D}}$ into the solution of the system of equations $A \cdot x = s_{\mathcal{D}}.b$. We next show that function g is additive.

Lemma 35. *Function g is additive.*

Proof. We next prove that $\forall s_{\mathcal{D}}, \bar{s}_{\mathcal{D}} \in S_{\mathcal{D}}$,

$$g(s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}) = g(s_{\mathcal{D}}) + g(\bar{s}_{\mathcal{D}})$$

Fix an arbitrary pair of states $s_{\mathcal{D}}, \bar{s}_{\mathcal{D}} \in S_{\mathcal{D}}$. We have that the following chain of equalities holds

$$\begin{aligned}
g(s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}) &= (A^{-1} \cdot (s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}).b, (s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}).b) \\
&= (A^{-1} \cdot (s_{\mathcal{D}}.b + \bar{s}_{\mathcal{D}}.b), s_{\mathcal{D}}.b + \bar{s}_{\mathcal{D}}.b) \\
&= (A^{-1} \cdot s_{\mathcal{D}}.b, s_{\mathcal{D}}.b) + (A^{-1} \cdot \bar{s}_{\mathcal{D}}.b, +\bar{s}_{\mathcal{D}}.b) \\
&= g(s_{\mathcal{D}}) + g(\bar{s}_{\mathcal{D}})
\end{aligned}$$

where the first inequality follows by construction of function g; the second inequality follows by construction of the addition operator; the third inequality follows from construction of the state space and last inequality follows from definition of g. $\square$

We next define the error function $e : S_{\mathcal{D}} \to \mathbb{R}_{\geq 0}$ as the infinity norm of the distance between the

input state and corresponding equilibrium state computed using function g: $\forall s_{\mathcal{D}} \in S_{\mathcal{D}}$,

$$e(s_{\mathcal{D}}) = ||s_{\mathcal{D}}.x - g(s_{\mathcal{D}}).x||_{\infty}$$

This is because, by construction, $s_{\mathcal{D}}.b = g(s_{\mathcal{D}}).b$. This function generalizes the function e defined in Section 5.1.2. We next show that function e is sub-additive.

Lemma 36. *Function e is sub-additive.*

Proof. Our goal is to show that $\forall s_{\mathcal{D}}, \bar{s}_{\mathcal{D}} \in S_{\mathcal{D}}$,

$$e(s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}) \leq e(s_{\mathcal{D}}) + e(\bar{s}_{\mathcal{D}})$$

Fix states $s_{\mathcal{D}}, \bar{s}_{\mathcal{D}} \in S_{\mathcal{D}}$. We have that the following chain of inequalities holds:

$$\begin{aligned} e(s_{\mathcal{D}} + \bar{s}_{\mathcal{D}}) &= ||s_{\mathcal{D}} + \bar{s}_{\mathcal{D}} - g(s_{\mathcal{D}} + \bar{s}_{\mathcal{D}})||_{\infty} \\ &= ||s_{\mathcal{D}} + \bar{s}_{\mathcal{D}} - g(s_{\mathcal{D}}) - g(\bar{s}_{\mathcal{D}})||_{\infty} \\ &\leq ||s_{\mathcal{D}} - g(s_{\mathcal{D}})||_{\infty} + ||\bar{s}_{\mathcal{D}} - g(\bar{s}_{\mathcal{D}})||_{\infty} \\ &\leq e(s_{\mathcal{D}}) + e(\bar{s}_{\mathcal{D}}) \end{aligned}$$

where the first equality follows by definition of function e; the second equality follows by Lemma 35; the third inequality follows by the triangle equality for the infinity norm and the last inequality follows by construction of function e. □

7.4.1.3 Bounded Exogenous Automaton

In this Section, we show that the exogenous automaton in bounded with respect to function g defined in Section 7.4.1.2.

Theorem 37. *If $\bar{\mathcal{A}}$ satisfies Assumptions N3 and N6, then $\mathcal{A}_{exog,\mathcal{D}}$ is bounded with respect to function g.*

Proof. This Theorem follows from Theorem 31. Specifically, Assumption N1 follows from Lemma 33, Assumption N2 from Lemma 34, Assumption N5 from Lemma 36, Assumption N4 from Theorem 23, Assumptions N3 and N6 hold by assumption of the Theorem. □

Furthermore,

Theorem 38. *If $\bar{\mathcal{A}}$ satisfies Assumptions N3, N6 and N7, then $\mathcal{A}_{exog,\mathcal{D}}$ is 0-bounded with respect to function g.*

Proof. This Theorem follows from Theorem 32. □

7.4.1.4 Discussion

In this Section, we discuss Theorem 37 and Theorem 38 in in the presence of specific exogenous inputs.

We first consider an exogenous input automaton $\bar{\mathcal{A}}_b$ whose actions can only modify vector b. This exogenous input automaton models a multi-agent system where vector b is time-varying, i.e. the final configuration of the system and of the protocol executed by the agents changes with time. The set of actions of $\bar{\mathcal{A}}_b$ consists of a single action b_update. This action is always enabled, i.e. $\bar{E}_b(s_{\mathcal{D}}, b\_update) = true, \forall s_{\mathcal{D}} \in S_{\mathcal{D}}$. When it is executed in state $s_{\mathcal{D}} \in S_{\mathcal{D}}$, it adds to the component b of $s_{\mathcal{D}}$ the constant vector $\hat{b}$, and does not change the component x, i.e.

$$\bar{T}_b(s_{\mathcal{D}}, b\_update) = (s_{\mathcal{D}}.x, s_{\mathcal{D}}.b + \hat{b})$$

The post-state of this action can be represented as the sum of the states $s_{\mathcal{D}}$ and v_{b_update}, with v_{b_update} being the pair of vectors $(0, \hat{b})$. By construction of the state space $S_{\mathcal{D}}$, $v_{b\_update} \in S_{\mathcal{D}}$. By construction, action b_update satisfies assumption N3. We assume that $e(v_{b\_update})$ is bounded by C, where, by definition,

$$e(v_{b\_update}) = ||A^{-1} \cdot \hat{b}||_\infty$$

Hence, $\bar{\mathcal{A}}_b$ satisfies Assumption N6. We next show that the automaton in the presence of this specific exogenous input is bounded.

Theorem 39. *If $\bar{\mathcal{A}}_b = (S_{\mathcal{D}}, S0_{\mathcal{D}}, \{b\_update\}, \bar{E}_b, \bar{T}_b)$, then $\mathcal{A}_{exog,\mathcal{D}}$ is bounded with respect to function g.*

Proof. This Theorem follows from Theorem 37. Specifically, the value of the constant L is bounded by:

$$L \leq \frac{1}{1-\alpha} \cdot \mathbb{L} \cdot ||A^{-1} \cdot \hat{b}||_\infty$$

with α defined in Chapter 5. □

We notice that we can construct executions of the system where the upper bound on the value of constant L in the proof of Theorem 39 is reached.

Given the automaton $\bar{\mathcal{A}}_b$, the automaton $\mathcal{A}_{exog,\mathcal{D}}$ is 0-bounded with respect to function g if and only if $||A^{-1} \cdot \hat{b}||_\infty = 0$. By construction of matrix A, this condition holds if and only if $\hat{b}(i) = 0$ for all $i \in \{1, \ldots, N\}$, or equivalently there are no inputs injected into the system.

We next discuss this exogenous input automaton in the case of the linear robot pattern formation multi-agent system discussed in Section 5.3. We assume that $\hat{b}$ has all entries equal to 0, with the exception of entries 0 and N that are positive. In this case, action b_update corresponds to agents 0 and N moving with a constant velocity. In this special case, Theorem 39 ensures that $\mathcal{A}_{exog,\mathcal{D}}$ is

bounded with respect to g. Furthermore, the constant L can be bounded as:

$$L \leq 2^{N-1} \cdot \mathbb{L} \cdot \max\{\hat{b}(0), \hat{b}(N)\}$$

since $A^{-1} \cdot \hat{b} = \tilde{b}$ where

$$\begin{aligned}
\tilde{b}(0) &= \hat{b}(0) \\
\tilde{b}(1) &= -0.5 \cdot \hat{b}(0) \\
\tilde{b}(i) &= 0 \quad \forall i, 1 < i < N-1 \\
\tilde{b}(N-1) &= -0.5 \cdot \hat{b}(N) \\
\tilde{b}(N) &= \hat{b}(N)
\end{aligned}$$

by construction of matrix A. As shown in the formula, the rate of growth of the system is exponential in the size of the system and linear in the epoch size and in the leader velocities. We can construct executions where the error of the system is exponentially large.

We next consider a different exogenous input automaton $\bar{\mathcal{A}}_x$ whose actions of the exogenous input automaton can only change the x component of the state. This exogenous input automaton models a multi-agent system where agents change their value due to some external conditions. The set of actions of $\bar{\mathcal{A}}_b$ consists of a single action x_update. This action is always enabled, i.e. $\bar{E}_x(s_{\mathcal{D}}, x\_update) = true, \forall s_{\mathcal{D}} \in S_{\mathcal{D}}$. When it is executed in state $s_{\mathcal{D}} \in S_{\mathcal{D}}$, it adds to the component x of $s_{\mathcal{D}}$ the constant vector $\hat{x}$, i.e.

$$\bar{T}_x(s_{\mathcal{D}}, x\_update) = (s_{\mathcal{D}}.x + \hat{x}, s_{\mathcal{D}}.b)$$

The post-state of this action is the sum of the state $s_{\mathcal{D}}$ and the state v_{x_update}, with v_{x_update} being the pair of vectors $(\hat{x}, 0)$. By construction, action x_update satisfies Assumption N3. The error of v_{x_update} is given by

$$e(v_{x\_update}) = ||\hat{x}||_\infty$$

We assume that the maximum component of $\hat{x}$ is bounded. Hence, $\bar{\mathcal{A}}_x$ satisfies Assumption N6. We next prove bounded-ness of $\mathcal{A}_{exog,\mathcal{D}}$.

Theorem 40. *If $\bar{\mathcal{A}}_x = (S_{\mathcal{D}}, S0_{\mathcal{D}}, \{x\_update\}, \bar{E}_x, \bar{T}_x)$, then $\mathcal{A}_{exog,\mathcal{D}}$ is bounded with respect to function g.*

Proof. This Theorem follows from Theorem 37. Specifically, the value of the constant L is bounded by:

$$L \leq \frac{1}{1-\alpha} \cdot \mathbb{L} \cdot ||\hat{x}||_\infty$$

with α defined in Chapter 5. $\square$

We notice that, given the automaton $\bar{\mathcal{A}}_x$, the automaton $\mathcal{A}_{exog,\mathcal{D}}$ is 0-bounded with respect to function g if and only if $||\hat{x}||_\infty = 0$. By construction of vector $\hat{x}$, this condition holds if and only if $\hat{x}(i) = 0$ for all $i \in \{1, \ldots, N\}$, or equivalently there are no inputs injected into the system.

In the special case of the linear robot pattern formation multi-agent system, we have that action *x_update* corresponds to follower agents moving with constant velocities. Their movement can be due to exogenous factors. In this special case, Theorem 40 holds with constant L bounded as follows:

$$L \leq 2^{N-1} \cdot \mathbb{L} \cdot ||\hat{x}||_\infty$$

As expected, the bound on the convergence grows linearly with the exogenous inputs injected in the system.

7.4.2 Solving Systems of Linear Equations with Dynamics

In this Section, we present a generic exogenous automaton modeling a shared-state system with dynamics in the presence of exogenous inputs. The shared-state system in the absence of exogenous inputs has been discussed in Section 5.1.4. This Section generalizes Section 7.4.1.

7.4.2.1 Exogenous Automaton

The exogenous automaton $\mathcal{A}_{exog,dyn}$ combines the automaton $\mathcal{A}_{dyn}$ with explicit arbitrary dynamics presented in Section 5.1.4 and a generic exogenous input automaton $\bar{\mathcal{A}}$.

We explicitly model vector b in the state space of $\mathcal{A}_{dyn}$. As discussed in Section 7.4.1, we extend the state of the system because we want to model exogenous inputs that can change both the state space of the agents and vector b. A state $s \in S_{dyn}$ is a triple (x, z, b), where $s.x$ stores the current state of the system, $s.z$ its destination state and $x.b$ the vector b. Component b is set to the input parameter b, initially; actions of the automaton $\mathcal{A}_{dyn}$ do not change it. Hence, Theorem 24 still holds.

An equilibrium state $\hat{s}_{dyn}$ is of the form:

$$\hat{s}_{dyn}.x = \hat{s}_{dyn}.z = \left(A^{-1} \cdot \hat{s}_{dyn}.b\right)$$

We denote by $\hat{S}_{dyn}$ the set of equilibrium states of $\mathcal{A}_{dyn}$.

7.4.2.2 Properties of the Exogenous Automaton

We next discuss some properties of $\mathcal{A}_{exog,dyn}$. These properties generalize properties of the shared-state discrete automaton presented in Section 7.4.1.2. By construction, the state space is closed

under addition. Given two states $s_{dyn}, \bar{s}_{dyn} \in S_{dyn}$, the state $s_{dyn} + \bar{s}_{dyn}$ is defined as

$$s_{dyn} + \bar{s}_{dyn} = (s_{dyn}.x + \bar{s}_{dyn}.x, s_{dyn}.z + \bar{s}_{dyn}.z, s_{dyn}.b + \bar{s}_{dyn}.b)$$

In this state, component x is the sum of component x in the two states; similarly for component b and component z.

Under specific assumptions of function f describing the dynamics of the agents, the transition function T_{dyn} is additive. We refer to Section 5.1.4 for the definition of T_{dyn}.

Lemma 41. *If*

O1. $\forall s \in S_{dyn}$, $\forall a \in \mathcal{A}_{dyn}$, *function* $f_{s,a}$ *is additive with respect to* S_{dyn},

then T_{dyn} *is additive with respect to* S_{dyn}.

Proof. The proof is similar to the proof of Lemma 34 and not reported. □

We extend function g of Section 7.4.1.2 to S_{dyn}; function $g : S_{dyn} \to \hat{S}_{dyn}$ is defined as: $\forall s_{dyn} \in S_{dyn}$,

$$g(s_{dyn}) = (A^{-1} \cdot s_{dyn}.b, A^{-1} \cdot s_{dyn}.b, s_{dyn}.b)$$

This function maps a state s_{dyn} into a state where both x component and z component stores the solution of the system of equations $A \cdot x = s_{dyn}.b$. By construction, function g is additive.

Lemma 42. *Function* g *is additive.*

Proof. The proof is similar to the proof of Lemma 35 and not reported. □

We next define the error function $e : S_{dyn} \to \mathbb{R}_{\geq 0}$, $\forall s_{dyn} \in S_{dyn}$,

$$e(s_{dyn}) = ||s_{dyn}.x - g(s_{dyn}).x||_\infty + ||s_{dyn}.z - g(s_{dyn}).z||_\infty$$

By construction, $s_{dyn}.b = g(s_{dyn}).b$, for this reason is not reported in the sum. This function generalizes the function e defined in Section 7.4.1.2 and it is additive.

Lemma 43. *Function* e *is sub-additive.*

Proof. The proof is similar to the proof of Lemma 36 and not reported. □

7.4.2.3 Bounded Exogenous Automaton

In this Section, we discuss the bounded-ness property of $\mathcal{A}_{exog,dyn}$ with respect to function g. We have that

Theorem 44. *If* $\bar{\mathcal{A}}$ *satisfies Assumptions N3 and N6, and* $\mathcal{A}_{dyn}$ *satisfies Assumptions F3-4 and Assumption O1, then* $\mathcal{A}_{exog,dyn}$ *is bounded with respect to function* g.

Proof. This Theorem follows from Theorem 31. Specifically, Assumption N1 follows by construction, Assumption N2 from Lemma 41 with Assumption O1, Assumption N5 from Lemma 43, Assumption N4 from Theorem 23, Assumptions N3 and N6 hold by assumption of the Theorem. □

Assumptions F3-4 ensure convergence of the automaton in the absence of exogenous inputs.

7.4.3 Solving Systems of Linear Equations via Message-Passing

In this Section, we discuss the applicability of the results of this Chapter to message-passing multi-agent systems. In Section 5.2, we have detailed the structure of the generic message-passing automaton $\mathcal{A}_{mp} = (S_{mp}, S0_{mp}, A_{mp}, E_{mp}, T_{mp})$; this automaton models a message-passing multi-agent system with bounded delay $\mathcal{B}$ and explicit arbitrary dynamics solving a system of linear equations.

As presented in Section 5.2, a state of $\mathcal{A}_{mp}$ is a function from $[-\mathcal{B}, 0]$ to S_{dyn}. In these systems, the automaton modeling the exogenous inputs can change any entry of the state. These exogenous input may model noise in the communication. For example, the automaton $\bar{\mathcal{A}}_x$ models a communication medium where messages may be corrupted. Given $s_{mp}, \bar{s}_{mp} \in S_{mp}$, the state $s_{mp} + \bar{s}_{mp}$ is a function from $[-\mathcal{B}, 0]$ to S_{dyn} defined as $\forall t \in [-\mathcal{B}, 0]$, $s_{mp} + \bar{s}_{mp}(t) = s_{mp}(t) + \bar{s}_{mp}(t)$. By construction, S_{mp} is closed under $+$ operator. By construction of T_{mp}, it follows that transition function T_{mp} is additive. We do not define function g for $\mathcal{A}_{mp}$; instead, we use the linear function g of $\mathcal{A}_{dyn}$ defined in Section 7.4.2. Using the asynchronous view relation, we construct function e as follows. For all $s_{mp} \in S_{mp}$, $e(s_{mp})$ is the maximum of the errors of its asynchronous views, i.e.

$$e(s_{mp}) = \max_{s_{dyn} \in \mathcal{H}(s_{mp})} e(s_{dyn})$$

These properties of $\mathcal{A}_{mp}$ ensure that Assumptions N1, N2, N5 and N4. Hence, we can derive that the automaton $\mathcal{A}_{exog}$ is bounded with respect to function g.

7.5 Discussion

In this Section, we discuss the main results of this Chapter and relate them to the current literature.

Our results apply to systems that, in the absence of exogenous inputs, execute additive protocols (see Assumption N2) and converge linearly with rate α (see Assumption N4). The error function of the system is sub-additive (see Assumption N5) and the exogenous inputs injected into the system are uniformly bounded quantities added to the state (see Assumption N3 and N6). For example, a linear protocol that estimates a linear statistics of the system, such as linear schemes for computing the average of a system, satisfies these assumptions.

In this Chapter, $\mathcal{A}$ represents an arbitrary automaton with timed actions. Automaton $\mathcal{A}$ can model both shared-state multi-agent systems and message-passing multi-agent systems. Further-

more, these systems can have fixed or time-varying network topologies. In Section 7.4, we apply these results to the class of systems solving systems of linear equations. We notice that, by construction of the exogenous inputs, we have that actions of $\bar{A}$ can modify the state of all agents in the system. This behavior is different from the behavior of actions in $\mathcal{A}$ where an action can only modify the state of a single agent. In the case of message-passing systems the inputs injected into the system correspond to noise in the communication, for example, they can model forged or corrupted message. Hence, in message-passing systems in the presence of exogenous inputs, messages may be lost, duplicated, delayed, received out-of-order or corrupted. Under this extremely weak communication medium we cannot ensure convergence of the system.

By construction of the automaton in the presence of exogenous inputs, execution fragments of $\mathcal{A}_{exog}$ consist of sequences of states and timed actions where the actions belong to $\mathcal{A} \cup \bar{A}$. In this model, actions of $\mathcal{A}$ and actions of $\bar{A}$ are sequentially executed. We can generalize the automaton in the presence of exogenous inputs and model a general system in the presence of exogenous inputs where actions of the agents are executed concurrently with the exogenous input actions. Concurrency is modeled using the transition function. The concurrent execution of an action $a \in \mathcal{A}$ and an action of $\bar{a} \in \bar{A}$ corresponds to the execution of the two actions independently; the state of the concurrent execution of a and $\bar{a}$ is the sum of the state obtained by executing action a and the exogenous input injected when executing action $\bar{a}$.

Our work extends previous work on multi-agent systems in the presence of exogenous inputs such as [62, 63, 65, 26, 68, 69]. In [62], authors provide conditions on the exogenous inputs for shared-state concurrent systems with discrete actions and fixed network topology. These systems execute distributed linear schemes. Results presented in this Chapter and published in [55] extend [62] to a more general class of multi-agent systems. We allow for systems with timed-actions operating over an unreliable communication medium. In [65], authors present a specific multi-agent system able to track the average of time-varying quantities. The work in [26] investigates multi-agent systems consisting of iterative linear schemes for computing statistics over time-varying topologies. This work assumes bounded exogenous inputs with bounded derivatives. In [68, 69], authors focus on discrete shared-state concurrent multi-agent systems. They present a family of systems whose goal is tracking the average of time-varying quantities; the class of exogenous input functions investigated in their work include polynomial and periodic functions.

Chapter 8

Conclusions

In this Chapter, we summarize the main results obtained in this Thesis and discuss ideas for future research.

8.1 Thesis Contributions

We developed theory to support verification of multi-agent systems operating on very general environments. This theory combined nondeterministic models of communication with multi-agent systems where agents can change their states continuously.

We introduced a general automaton model and used it to represent a very general class of multi-agent systems. We allowed for continuous dynamics, unreliable message-passing communication and time-varying goal configurations. For example, in the case of unreliable communication, the model allowed for lost, delayed, duplicated, or received out-of-order messages. We considered both multi-agent systems where only one agent at a time can change its state and multi-agent systems where multiple agents may change their states concurrently.

Models in this Thesis used fairness from temporal logic. They assumed that an agent is never partitioned from the system. In case of unreliable communication, the fairness requirement translated into assuming that infinitely many messages sent from one set of agents to the others get through eventually.

We focused primarily on convergence and stability properties. We provided conditions on the systems that ensure stability and convergence. A system converges to a desired goal configuration if it gets arbitrary close to it, although it may never actually reach it. In our theory, proofs of stability and convergence can be verified mechanically.

We studied robustness properties of multi-agent systems in which the environment is time-varying and affects the goal configuration of the system. In this case, we derived bounds on the distance between the actual and desired trajectory of the system. Such bounds require that fairness constraints hold within a bounded time interval called epoch.

8.2 Summary

In Chapter 2 we have presented the automaton with timed action model. This model is used to formally describe the structure and behaviour of multi-agent systems, along with the structure and behavior of exogenous inputs. We have introduced a new notion of fairness for multi-agent systems, which require that the multi-agent system is never permanently partitioned in non-communicating sub-systems. We have shown that this notion of fairness is weaker than weak fairness.

In Chapter 3 we have discussed the notion of equilibrium states for automata with timed actions. We have modeled stability and asymptotical stability properties of these equilibrium states. We have provided conditions on the structure of the Lyapunov function which guarantee stability and convergence. We have defined a novel property for multi-agent systems in the presence of exogenous inputs, which ensures that the multi-agent system is eventually-always close to the set of equilibrium states, i.e. it is bounded.

In Chapter 4 we have modeled shared-state and message-passing systems using the automaton with timed action model. Stability and convergence properties of message-passing systems have been derived from the stability and convergence properties of the corresponding shared-state systems. These conditions ensure that the message-passing system converge to the same equilibrium state reached by its shared-state counterpart.

In Chapter 5 we have proven correctness for a general class of multi-agent systems, whose goal is to solve a system of linear equations. This class includes both shared-state and message-passing systems. We have shown their correctness using the results of Chapter 4. For example, the message-passing versions of Gauss and Gauss-Seidel methods belong to this class.

In Chapter 6 we have provided a novel verification framework for message-passing multi-agent systems with dense state space. This framework allows verification of multi-agent systems whose goal is to solve systems of linear equations, where messages may be lost, delayed or received out-of-order. The framework consists of a library of PVS meta-theories.

In Chapter 7 we have studied multi-agent systems in the presence of exogenous inputs, and provided conditions that ensure that the system is bounded. These conditions require the protocol of the agents to be linear and the exogenous inputs to be uniformly bounded quantities added to the agents. We have applied this results to the class of system presented in Chapter 5.

8.3 Future Work

There are several directions which we can follow to extend and improve the results of this thesis. We next outline some of them.

Multi-agent solutions to systems of non-linear equations. In Chapter 5 we have presented shared-state and message-passing multi-agent systems whose goal is to solve systems of linear equations. In a future continuation of this work, we would like to investigate multi-agent systems whose goal is to solve non-linear systems of equations. This class would include iterative decentralized message-passing schemes where agent i is responsible for computing the value of the i-th variable using the i-th non-linear equation of the system as updating rule. We would like to derive conditions on the Jacobian of the matrix that ensure convergence. Figure 8.1 presents an example of a multi-agent system whose goal is to solve a system of non-linear equations. This specific system converges to the solution of the system of non-linear equations.

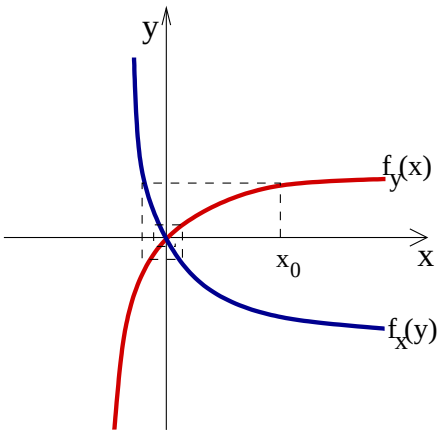


Figure 8.1: Pictorial representation of a multi-agent system whose goal is to solve a system of non-linear equations. This system consists of two equations, these are $y = f_y(x)$ and $x = f_x(y)$. The corresponding multi-agent system consists of two agents x and y. Upon receiving a message m from y, agent x moves evolves its current state towards $f_x(m)$. This multi-agent system converges to the solution of the system of non-linear equations. For example, the dashed line represents a converging execution of the multi-agent system. This execution starts in state $(x_0, 0)$ and alternates the execution of the updating rule of the two agents.

Multi-agent dynamic message-passing games with continuous dynamics. Classical game theory studies equilibria of dynamic games where agents, called players, alternate their moves. In these games, only one player moves at a time and all agents have full information about the system. It would be interesting to investigate equilibrium properties of dynamic games where agents have continuous movements and operate over an unreliable communication medium. In this new class of games, players may move concurrently and have partial information on the system. For example, we would like to investigate potential games [48]; in this class of games, there is a global function, called potential, that summarizes the incentive of all players to change their strategy. In the case of potential games, we can show that if the potential function is strictly concave, then the potential message-passing game with continuous dynamics converges to the same set of equilibria of the corresponding classical potential game (see Figure 8.2).

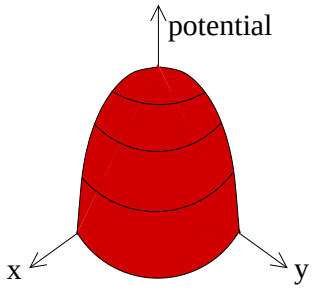


Figure 8.2: Pictorial representation of a two-player potential game. In this game, the potential function is concave. This game converges to the set of Nash equilibria when agents have continuous dynamics and communicate over an unreliable communication medium.

Multi-agent message-passing optimization. We would like to study distributed optimization problems where agents communicate over an unreliable communication medium. Those include, for example, optimization problems arising in electricity demand-response markets. In these problems, the system consists of N agents. These agents consume some resource; the total amount C of resource is assumed to be constant. The amount of resource consumed by agent i is denoted by $x(i)$. The price p of the resource changes with the total amount of resource consumed, i.e. $p = f(x, C)$, for some function f. Similarly, the amount of resource consumed by agent i depends on the price of the resource, i.e. $x(i) = g_i(p)$ for some g_i. The goal of this system is to compute the equilibrium price. This price may be computed using the following iterative algorithm, at round n

$$\begin{aligned} p^{n+1} &= f(x^n, C) \\ x^{n+1}(i) &= g_i(p^{n+1}) \quad \forall i \in \{1, \ldots, N\} \end{aligned}$$

where x^n denotes the amount of resources consumed at round n, and p^n denotes the price of the resources at round n. We would like to consider the corresponding distributed optimization problem, where agents communicate via message-passing. If agents communicate via an unreliable communication medium, then the market may compute the price of the resource using old information regarding the amount of resource consumed by the agents and the agents may decide on the amount of resource to consume using an old price. The iterative algorithm becomes:

$$\begin{aligned} p^{n+1} &= f((x(1)^{m_1}, \ldots, x(N)^{m_N}), C) \\ x^{n+1}(i) &= g_i(p^m) \quad \forall i \in \{1, \ldots, N\} \end{aligned}$$

where $m \leq n+1$ and $\forall i \in \{1, \ldots, N\}$, the value $m_i \leq n$.

Robustness of multi-agent systems in the presence of exogenous inputs. In this Thesis, we have introduced the notion of bounded error as a measure of robustness of multi-agent systems in

the presence of exogenous inputs, where the goal configuration of the system is a function of the initial configuration and of the exogenous inputs injected into the system. This notion of robustness can be used to model systems able to track time-varying quantities. For example, as shown in Figure 1.3 and Figure 1.6, it can be used to model a system able to track a time-varying pattern configuration. In a future continuation of this work, we would like to investigate other measures of robustness of multi-agent systems in presence of exogenous inputs. For example, we would like to investigate robustness properties of systems whose goal configuration is static, i.e. depending only on the initial configuration. In particular, we would like to give necessary and sufficient conditions on the system in the absence of exogenous inputs that ensure convergence of the corresponding system in the presence of exogenous inputs. Such a notion of robustness can be used to model systems where agents can lie about their current state by sending false signals. For example, in the case of multi-agent pattern formation systems, they may send messages containing locations that they have never visited. For the case of linear protocols, as shown in Figure 8.3, we are able to show that the multi-agent system in the presence of exogenous inputs converges under specific assumptions on the inputs.

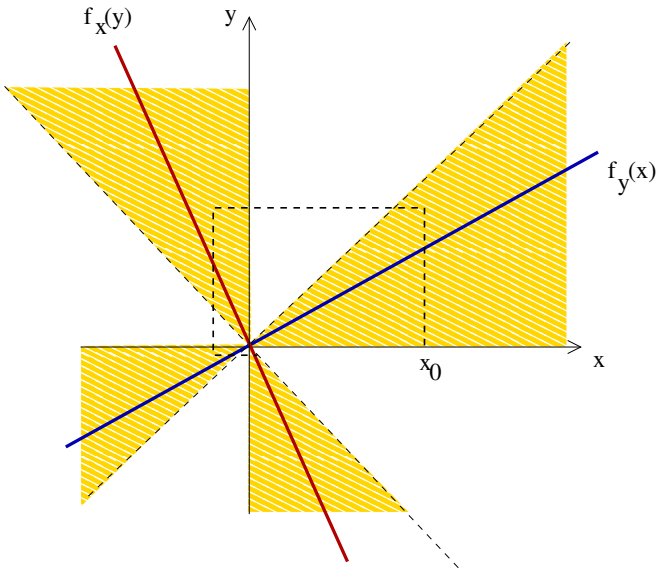


Figure 8.3: Pictorial representation of a multi-agent system where the protocol of the agents is linear. This system consists of two agents, x and y. Agent x has protocol $f_x(y)$ and agent y has protocol $f_y(x)$. In this specific example, upon receiving a message containing a value x_0 from agent x, agent y can send to agent x any value in the interval $[0, x_0)$; the shaded area represents this region for all choices of x_0. Similarly, upon receiving a message containing a value y_0 from agent y, agent x can send any value in the interval $[0, y_0)$. If agents exchange values falling within the shaded areas, they will eventually converge to the equilibrium state $(0, 0)$. For example, the dashed line represents a converging execution of the multi-agent system.

Verification Framework. The framework presented in Chapter 6 allows verifying multi-agent systems that solve systems of linear equations where messages may be lost, delayed or received

out-of-order. It specializes the automaton and proofs presented in Chapter 4 to message-passing systems. We would like to construct a more general verification framework, where one can encode (1) the generic shared-state system, (2) the corresponding shared-state system with sliding window, and (3) the proofs of Theorem 13 and Theorem 14. Theorem 13 requires that the Lyapunov function satisfies G1-2; Theorem 14 requires that the Lyapunov function satisfies H1-2. Here, the Lyapunov function would be encoded as an input of the framework, while conditions G1-2 and H1-2 would be encoded as PVS assumptions. The end-user of the framework would provide the Lyapunov function and discharge the Assumptions of the library.

Bibliography

[1] R. Alur, and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994. 37

[2] M. Archer. TAME: Using PVS strategies for special-purpose theorem proving. *Annals of Mathematics and Artificial Intelligence*, vol. 29, no. 4, pp. 139–181, 2000. 12, 111

[3] M. Archer, C. Heitmeyer, and S. Sims. TAME: A PVS interface to simplify proofs for automata models. In *Proceedings of the 1st International Workshop on User Interfaces for Theorem Provers* (UITP'98), 1998. 12, 111

[4] M. Archer, H. Lim, N. Lynch, S. Mitra, and S. Umeno. Specifying and proving properties of timed I/O automata using Tempo. *Journal of Design Automation for Embedded Systems*, vol. 2, no. 1-2, 2008. 12, 111

[5] C. Baier, and J.P. Katoen. *Principles of Model Checking.* MIT Press, Cambridge, Mass., 2008. 15

[6] P. Bauer, M. Sichitiu, R. Istepanian, and K. Prematne. The mobile patient: Wireless distributed sensor networks for patient monitoring and care. In *Proceedings of the 2000 IEEE EMBS International Conference on Information Technology Applications in Biomedicine*, pp. 17–21, 2000. 4

[7] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL in 1995. In *Proceedings of the 2nd International Workshop on Tools and Algorithms for Construction and Analysis of Systems* (TACAS'96), pp. 431–434, 1996. 13

[8] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Athena Scientific, 1997. 12, 87

[9] V.D. Blondel, J.M. Hendrickx, A. Olshevsky, and J.N. Tsitsiklis. Convergence in multiagent coordination, consensus, and flocking. In *Proceedings of the Joint 44th IEEE Conference on Decision and Control and European Control Conference* (CDC-ECC), pp. 3387 3392, 2005. 14

[10] J. Buhl, D.J.T. Sumpter, I.D. Couzin, J. Hale, E. Despland, E. Miller, and S.J. Simpson. From disorder to order in marching locusts. *Science*, pp. 312–406, 2006. 4

[11] L. Bulwahn, A. Krauss, and T. Nipkow. Finding lexicographic orders for termination proofs in Isabelle/HOL. In *Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics*, (TPHOLs'07), vol. 4732, LNCS, pp. 38–53, 2007. 13

[12] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: application driver for wireless communications technology. In *Proceedings of the 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, pp. 20–41, 2001. 4

[13] K.M. Chandy. Reasoning about continuous systems. *Science of Computer Programming*, vol. 14, pp. 117–132, 1990. 36

[14] K.M. Chandy, B. Go, S. Mitra, C. Pilotto, and J. White. Verification of distributed systems with local-global predicates. *Formal Aspect of Computing*, 2010. 11, 12, 32, 55, 110, 120

[15] K.M. Chandy and J. Misra. Parallel Program Design. Addison-Wesley, 1988. 21

[16] K.M. Chandy, S. Mitra, and C. Pilotto. Convergence verification: from shared memory to partially synchronous systems. In *Proceedings of the 5th International Conference on Formal Modeling and Analysis of Times Systems* (FORMATS'08), vol. 5215, LNCS, pp. 217-231, 2008. 12, 87, 88

[17] S. Chatterjee and E. Seneta. Towards consensus: some convergence theorems on repeated averaging. *Journal of Applied Probability*, vol. 14, no. 1, pp. 89–97, 1977. 14

[18] D. Chazan, and W. Miranker. Chaotic relaxation. *Linear algebra and its Applications*, vol. 2, no. 2, pp. 199-222, 1969. 12, 110

[19] S. Clavaski, M. Chaves, R. Day, P. Nag, A. Williams, and W. Zhang. Vehicle networks: achieving regular formation. In *Proceedings of the 2003 American Control Conference* (ACC'03), 2003. 6

[20] I.D. Couzin, J. Krause, N.R. Franks, and S.A. Levin. Effective leadership and decision-making in animal groups on the move. *Nature*, pp. 433–516, 2005. 4

[21] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, vol. 7, no. 2, pp. 279–301, 1989. 5

[22] M.H. DeGroot. Reaching a consensus. *Journal of American Statistical Association*, vol. 69, no. 345, pp. 116–121, 1974. 14

[23] R. D'Andrea, and R. Murray. The RoboFlag Competition. In *Proceedings of the 2005 American Control Conference* (ACC'05), pp. 650–655, 2003. 4

[24] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of ACM*, vol. 35, no. 2, pp. 288–323, 1988. 87

[25] J.A. Fax, and R.M. Murray. Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1465–1476, 2004. 6

[26] R.A. Freeman, P. Yang, and K.M. Lynch. Stability and convergence properties of dynamic average consensus estimators. In *Proceedings of the IEEE Conference on Decision and Control* (CDC'06), pp. 398–403, 2006. 14, 145

[27] P. Frey, R. Radhakrishnan, H.W. Carter, P.A. Wilsey, and P. Alexander. A formal specification and verification framework for time warp-based parallel simulations. *IEEE Transition on Software Engineering*, vol. 28, no. 1, pp. 58–78, 2002. 13

[28] D. Gabay and H. Moulin. On the uniqueness and stability of Nash equilibria in non-cooperative games. In *Applied Stochastic Control of Econometrics and Management Science*, pp. 271–293, North-Holland, 1980. 12, 110

[29] H. Gottliebsen. Transcendental functions and continuity checking in PVS. In *Proceedings of the 13th International Conference on Theorem Proving in Higher Order Logics* (TPHOLs'00), vol. 1869, LNCS, pp. 197–214, 2000. 13

[30] J. Harrison. *Theorem Proving with the Real Numbers*. Springer-Verlag, 1998. 13

[31] M. Hendriks. Model checking the time to reach agreement. In *Proceedings of the 3rd International Conference on Formal Modeling and Analysis of Timed Systems* (FORMATS'05), pp. 98–111, 2005. 13

[32] T.A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science* (LICS'96), pp. 278–292, 1996. An extended version appeared in *Verification of Digital and Hybrid Systems* (M.K. Inan, R.P. Kurshan, eds.), vol. 170, pp. 265–292, 2000. 36, 37

[33] P.B. Jackson. Total-correctness refinement for sequential reactive systems. In *Proceedings of the 13th International Conference on Theorem Proving in Higher Order Logics* (TPHOLs'00), vol. 1869, LNCS, pp. 320-337, 2000. 13

[34] A. Jadbabaie, J. Lin, and A.S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, 2003. 6

[35] D. Kaynar, N. Lynch, S. Mitra, and S. Garland. *TIOA Language*. MIT Computer Science and Articial Intelligence Laboratory, Cambridge, MA, 2005. 13

[36] D.K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures in Computer Science. Morgan Claypool, 2006. 13, 37, 111

[37] D. Lester. NASA Langley PVS library. http://shamesh.larc.nasa.gov/fm/ftp/larc/PVS-library/pvslib.html 13, 111, 112, 115

[38] H. Lim, D. Kaynar, N.A. Lynch, and S. Mitra. Translating timed I/O automata specifications for theorem proving in PVS. In *Proceedings of the 3rd International Conference on Formal Modelling and Analysis of Timed Systems* (FORMATS'05), vol. 3829, LNCS, Springer-Verlag, 2005. 13

[39] A.M. Lyapunov. *Stability of Motion*. Academic Press, 1966. 42

[40] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., USA, 1996. 54

[41] N.A. Lynch, R. Segala and F.W. Vaandrager. Hybrid I/O automata. *Information and Computation*, vol. 185, no. 1, pp. 105-157, 2003. 37

[42] N.A. Lynch, and M.R. Tuttle. An introduction to Input/Output automata. *CWI-Quarterly*, vol. 2, no. 3, pp. 219–246, 1989. 12, 111

[43] D.G. Luenberger. *Introduction to Dynamic Systems: Theory, Models, and Applications*. John Wiley and Sons, Inc., New York, 1979. 42

[44] S. Maharaj, and J. Bicarregui. On the verification of VDM specification and refinement with PVS. In *Proceedings of the 12th International Conference on Automated Software Engineering* (ASE'97), pp. 280, 1997. 13

[45] S. Mitra. A verification framework for hybrid systems. PhD thesis. Massachussetts Institute of Technology, 2007. 13, 111

[46] S. Mitra and M. Archer. PVS strategies for proving abstraction properties of automata. *Electronic Notes in Theoretical Computer Science*, vol. 125, no. 2, pp. 45–65, 2005. 13, 111, 116

[47] S. Mitra, and K.M. Chandy. A formalized theory for verifying stability and convergence of automata in PVS. In *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics* (TPHOLs'08), vol. 5170, LNCS, pp. 230–245, 2008. 55

[48] D. Monderer. Potential Games. *Games and Economic Behavior*, vol. 14, pp. 124–143, 1996. 148

[49] R. Olfati-Saber. Distributed Kalman filter with embedded consensus filters. In *Proceedings of the Joint 44th IEEE Conference on Decision and Control and European Control Conference* (CDC-ECC'05), pp. 8179–8184, 2005 6

[50] R. Olfati-Saber, J. Fax, and R. Murray. Consensus and cooperation in networked multi-agent systems. In *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007. 88

[51] R. Olfati-Saber, and R.M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004. 88

[52] R. Olfati-Saber, and J.S. Shamma. Consensus filters for sensor networks and distributed sensor fusion. In *Proceedings of the Joint 44th IEEE Conference on Decision and Control and European Control Conference* (CDC-ECC'05), pp. 6698–6703, 2005. 6

[53] S. Owre, J.M. Rushby, and N. Shankar. PVS: A prototype verification system. In *Proceedings of the 11th International Conference on Automated Deduction* (CADE'92), vol. 607, LNAI, pp. 748–752, 1992. 12

[54] C. Pilotto, K.M. Chandy, and R. McLean. Networked sensing systems for detecting people carrying radioactive material. In *Proceedings of the 5th International IEEE Conference on Networked Sensing Systems* (INSS 2008), 2008. 4

[55] C. Pilotto, K.M. Chandy, and J. White. Consensus on asynchronous communication networks in presence of external input. In *Proceedings of the 49th IEEE Conference on Decision and Control* (CDC'10), pp. 3838–3844, 2010. 14, 145

[56] C. Pilotto, and J. White. Verification of faulty message-passing systems with continuous state space in PVS. In *Proceedings of the 2nd NASA Formal Methods Symposium*, 2010. 13

[57] C. Pilotto, and J. White. Towards a verification framework for faulty message passing systems. To Appear in *Innovations in Systems and Software Engineering*, pp.1–10, 2011. 13

[58] C. Pilotto, and J. White. Infospheres project. `www.infospheres.caltech.edu/nfm`, 2010.

[59] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pp. 46-57, 1977. 12, 111, 125

[60] J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Application*, pp. 99–97, 2002. 10

[61] W. Ren. Consensus strategies for cooperative control of vehicle formations. *IET Control Theory & Applications*, vol. 1, no. 2, pp. 505–512, 2007. 4

[62] W. Ren, and R.W. Beard. Dynamic Consensus Seeking in Distributed Multi-agent Coordinated Control. Technical Report, Brigham Young University, available at `citeseer.ist.psu.edu/ren03dynamic.html`, 2003. 6

[63] W. Ren, and R.W. Beard. Consensus of Information Under Dynamically Changing Interaction Topologies. In *Proceedings of the 2004 American Control Conference* (ACC'04), pp. 4939–4944, 2004. 14, 145

[64] W. Ren, R.W. Beard, and E.M. Atkins. A survey of consensus problems in multi-agent coordination. In *Proceedings of the 2005 American Control Conference* (ACC'05), pp. 1859–1864, 2005. 14, 145

[65] D.P. Spanos, R. Olfati-Saber, and R.M. Murray. Dynamic consensus on mobile networks. In *Proceedings of the IFAC World Congress*, 2005. 14

14, 145

[66] J.N. Tsitsiklis. On the stability of asynchronous iterative processes. *Theory of Computing Systems*, vol. 20, no. 1, pp. 137–153, 1987. 11, 12, 55, 87

[67] R.A. Usmani. Inversion of Jacobi's tridiagonal matrix. *Computers & Mathematics with Applications*, vol 27, no. 8, pp. 59–66, 1994. 106, 107, 127

[68] M. Zhu, and S. Martinez. Dynamic average consensus on synchronous communication networks. In *Proceedings of the 2008 American Control Conference* (ACC'08), pp. 4382–4387, 2008. 14, 145

[69] M. Zhu, and S. Martinez. Discrete-time dynamic average consensus. *Automatica*, vol. 46, no. 2, pp. 322-329, 2010. 14, 145

[70] Robocup, `www.robocup.org` 4

[71] Roboflag, `roboflag.mae.cornell.edu` 4

[72] Tempo toolset, version 0.2.2 beta, January 2008. `http://www.veromodo.com/`. 111