FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Temporal-Epistemic Logic in Byzantine Message-Passing Contexts

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Patrik Fimml
Matrikelnummer 1027027

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.-Prof. Dr. Ulrich Schmid

Wien, 28. Oktober 2017

Patrik Fimml	Ulrich Schmid

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Temporal-Epistemic Logic in Byzantine Message-Passing Contexts

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Patrik Fimml
Registration Number 1027027

to the
Faculty of Informatics at Technische Universität Wien

Advisor: Univ.-Prof. Dr. Ulrich Schmid

Vienna, October 28, 2017

Patrik Fimml	Ulrich Schmid

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Patrik Fimml
Jupiterstrasse 4
8032 Zürich
Schweiz

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. Oktober 2017

Patrik Fimml

Abstract

In this thesis, we use temporal-epistemic logic to analyze asynchronous message-passing systems with up to f Byzantine node failures. We introduce the class of contexts Γ_{bamp}, a model for Byzantine asynchronous message-passing systems that allows us to apply temporal-epistemic logic. We propose an intuitive interpretation of knowledge for Byzantine-faulty agents. We use temporal-epistemic logic to prove that with up to f Byzantine-faulty agents, an agent can initially only gain knowledge by receiving messages from at least $f + 1$ agents.

We seek to apply temporal-epistemic logic to the analysis of fault-tolerant distributed clock synchronization. For this purpose we introduce Firing Rebels without Relay, a weaker problem related to clock synchronization. We apply our model to prove necessary and sufficient knowledge for the actions of agents in Firing Rebels without Relay. Based on necessary knowledge, we derive a lower bound on communication between agents.

This work was supported through project RiSE/SHiNE (S11405) of the Austrian Science Fund (FWF).

Kurzfassung

In dieser Diplomarbeit verwenden wir temporal-epistemische Logik, um asynchrone Message-Passing-Systeme mit bis zu f Byzantinisch fehlerhaften Rechenknoten zu analysieren. Wir stellen die Kontextklasse Γ_{bamp} vor, ein Modell für asynchrone Message-Passing-Systeme mit Byzantinischen Fehlern, in dem wir temporal-epistemische Logik verwenden können. Wir stellen eine anschauliche Möglichkeit vor, wie Wissen für Byzantinisch-fehlerhafte Akteure definiert werden kann. Mittels temporal-epistemischer Logik zeigen wir, dass ein Akteur in einem System mit bis zu f Byzantinisch-fehlerhaften Akteuren zu Beginn nur dann Wissen gewinnen kann, wenn er Nachrichten von mindestens $f + 1$ Akteuren erhält.

Wir versuchen, fehlertolerante verteilte Uhrensynchronisation mittels epistemischer Logik zu untersuchen. Zu diesem Zweck führen wir Firing Rebels ohne Relay ein, ein einfacheres, aber mit Uhrensynchronisation verwandtes Problem. Mit unserem Modell beweisen wir notwendiges und hinreichendes Wissen, mit dem Akteure in Firing Rebels ohne Relay agieren können. Davon ausgehend leiten wir eine untere Schranke für die Kommunikation zwischen Akteuren ab.

Diese Arbeit wurde durch das Projekt RiSE/SHiNE (S11405) des Österreichischen Wissenschaftsfonds (FWF) unterstützt.

Contents

1 Introduction **1**
1.1 Related Work . . . 3
1.2 Outline and Major Contributions . . . 4

2 Introduction to Epistemic Logic **5**
2.1 Possible Worlds . . . 6
2.2 S5 Models . . . 9
2.3 Group Knowledge . . . 9
2.4 Muddy Children Puzzle . . . 10

3 Temporal-Epistemic Logic in Distributed Systems **12**
3.1 Runs . . . 13
3.2 Systems . . . 14
3.3 Histories . . . 15
3.4 Message-Passing Systems . . . 17
3.5 Protocols . . . 18
3.6 Contexts . . . 19
3.7 Liveness Conditions . . . 22
3.8 Generating Systems from Protocols . . . 22
3.9 Non-Excluding Contexts . . . 23
3.10 Temporal-Epistemic Logic over Runs: Syntax . . . 23
3.11 Temporal-Epistemic Logic over Runs: Semantics . . . 24

4 Failure-Free Asynchronous Message-Passing Contexts **26**
4.1 The Class Γ_{amp} . . . 27
4.2 Transition Relation . . . 28
4.3 Liveness Conditions . . . 30
4.4 Recording Context . . . 30
4.5 Relation to Message-Passing Systems . . . 31
4.6 Lock-Step Synchronous Systems . . . 31

5 Byzantine Asynchronous Message-Passing Contexts **33**
5.1 Knowledge of Byzantine-Faulty Agents 34
5.2 The Class Γ_{bamp} 37
5.3 Transition Relation 38
5.4 Liveness Conditions 40
5.5 Recording Context 41
5.6 Non-Excluding Context 41
5.7 Properties in Temporal-Epistemic Logic 41
5.8 Knowledge Gain 43

6 Temporal-Epistemic Analysis of Clock Synchronization **47**
6.1 Clocks and Ensembles 48
6.2 Algorithm for Tick Generation 49
6.3 Firing Squad 51
6.4 Firing Rebels with/without Relay 52
6.5 Properties in Temporal-Epistemic Logic 54
6.6 Necessary Knowledge 56
6.7 Necessary Communication 58
6.8 Sufficient Knowledge 59

7 Conclusion and Outlook **63**
7.1 Future Work 64

Bibliography **66**

List of Symbols **69**

CHAPTER 1

Introduction

In distributed computing systems, several processing nodes—*agents*— communicate with each other to solve a given distributed computing problem, coordinating certain decisions or actions. Given a problem statement, how soon and under what circumstances can an agent perform some action X without potentially violating correctness criteria for the system—for example, when can a consensus algorithm decide on a certain value, or when can a tick generation algorithm advance the clock value? An engineer might argue informally that "the agent can perform action X once it *knows* that Y has happened", or "no algorithm can solve the problem earlier, because the agent *cannot know* whether X or Y has happened". Knowledge and indistinguishability are intuitive concepts for reasoning about distributed systems.

Indistinguishability arguments and combinatorial reasoning are frequently used in proofs for problems in Distributed Computing, often in an ad-hoc manner. Epistemic logic—logic with a concept of *knowledge*—can be a natural fit for these kinds of arguments and has been applied to distributed systems before. Epistemic logic is a modal logic that enables statements such "agent X knows that Y has happened". In semantics for epistemic logic, knowledge is usually defined through indistinguishability. Research has used the potential of epistemic logic to make arguments about distributed systems simpler, more rigorous, or provide additional insights on the fundamental properties of a given distributed computing problem [FHMV03, DM90, BM14].

Temporal logic is logic extended by a notion of time; it enables statements such as "from now on, X holds forever" or "at some point in the future, Y holds". Temporal and epistemic operators can be combined to form temporal-epistemic logic.

Most published results that apply epistemic logic to distributed systems either assume a failure-free system, or if they talk about Byzantine failures, they restrict themselves to a *crash/omission failure assumption*, e.g. [DM90, HMW01, CGM14].

In this work, we investigate how a Byzantine failure assumption without restrictions and the knowledge-based approach play together, and take first steps towards the analysis of fault-tolerant clock synchronization using temporal-epistemic logic.

Fault-tolerant clock synchronization has been studied extensively at the Institute of Computer Engineering at TU Wien, particularly in weak asynchronous system models without absolute time bounds [WS09, RS11]. Ultimately, we would like to see a knowledge-based characterization of clock synchronization, in the hope that it provides new insights on about clock synchronization in different system models. We would like to understand in which system models clock synchronization is achievable at all, and why. This thesis aims to provide a starting point for future work in this direction.

Research for this work began in March 2016. Based on a January 2017 draft of this thesis, Prosperi, Kuznets and Schmid [PKS17] started further research in the area of temporal-epistemic logic for Byzantine-faulty message-passing systems. We briefly mention a few of their ideas in the final version of this thesis.

This work received support through project RiSE/SHiNE (S11405) of the Austrian Science Fund (FWF), which we gratefully acknowledge.

1.1 Related Work

The following publications are related to this thesis, grouped by topic:

Temporal-epistemic logic. The foundation for our analysis of distributed systems using temporal-epistemic logic is laid by Fagin, Halpern, Moses and Vardi [FHMV03], the revised edition of [FHMV95], which collects material from several earlier papers by the authors such as [HM90]. Fagin et al. define a formal model of distributed systems that allows the evaluation of temporal-epistemic formulas, which we present in Chapter 3. Van Ditmarsch, van der Hoek and Kooi [DHK08] give a modern introduction to epistemic logic isolated from its application to distributed systems from which we borrow some notation.

Knowledge and communication. Ben-Zvi and Moses [BM14] extend Lamport's *happened-before* relation [Lam78] to derive a link between nested knowledge and communication in distributed systems. Assume an asynchronous, failure-free system with bounded message delivery times. In this system model, they define a simple problem called *Ordered Response*. They derive *necessary knowledge* that agents need to obtain to solve Ordered Response, and find that it is nested knowledge. Generalizing these findings, they show that a certain communication pattern—a *centipede*—must always occur when agents obtain such nested knowledge.

Knowledge and failures. Variants of the consensus problem have been studied using epistemic logic under a crash/omission failure assumption. In particular, *simultaneous Byzantine agreement* was investigated by Moses and Tuttle [MT88] and Dwork and Moses [DM90]. *Eventual* Byzantine agreement has been studied using epistemic logic by Halpern, Moses and Waarts [HMW01], Neiger and Bazzi [NB99], and Castañeda, Gonczarowski and Moses [CGM14]. Ruben Michel [Mic89] combines a restricted Byzantine failure assumption with category theory. In contrast to our model, Michel allows agents to only have corrupted state, but not to send a combination of messages that appears in no legal state.

Clock synchronization. Fault-tolerant clock synchronization is a well-studied problem in distributed systems literature, e.g. see [AW04]. Our analysis uses an algorithm by Widder and Schmid [WS09], a variation of Srikanth and Toueg [ST87]. Widder and Schmid [WS09] and Robinson and Schmid [RS11] have studied fault-tolerant clock synchronization under particularly weak assumptions on relative end-to-end delay. We are not aware of published results on the application of epistemic logic to fault-tolerant clock synchronization.

1.2 Outline and Major Contributions

This thesis is laid out as follows:

Chapter 2 introduces and motivates an abstract notion of *knowledge* for readers unfamiliar with epistemic logic. We define knowledge based on indistinguishability of worlds. We briefly explore group knowledge and introduce the Muddy Children Puzzle as an example application showing some of the intricacies of nested knowledge.

Chapter 3 explains how temporal-epistemic logic can be applied to distributed systems, and we introduce *runs, systems, protocols,* and *contexts* [FHMV03]. We define a semantics for temporal-epistemic logic over runs that forms the foundation for arguments and proofs in later chapters.

Chapter 4 introduces Γ_{amp}, the class of *failure-free asynchronous message-passing contexts*, a model for distributed systems that enables reasoning using temporal-epistemic logic. We explain the model and explain limitations that we encountered trying to model lock-step synchronous systems.

Chapter 5 extends Γ_{amp} to Γ_{bamp}, the class of *Byzantine asynchronous message-passing contexts.* We discuss how the knowledge of Byzantine-faulty agents in our model can be understood intuitively. We prove a basic theorem about knowledge gain in Byzantine asynchronous message-passing contexts.

Chapter 6 introduces *Firing Rebels*, a variant of Firing Squad [BL87]. We argue that Firing Rebels is a stepping stone towards an analysis of fault-tolerant clock synchronization using epistemic logic. We use our definition of Byzantine asynchronous message-passing contexts to analyze necessary and sufficient knowledge for agents to solve Firing Rebels without Relay. We derive a basic statement about necessary communication to solve Firing Rebels without Relay.

Chapter 7 summarizes our results, explains difficulties that we encountered, and explores directions for further research.

The appendix lists our references, along with a glossary of mathematical symbols used in this thesis.

CHAPTER 2

Introduction to Epistemic Logic

What does it mean to *know* something? As a first approximation, philosophy literature has discussed the question whether or not knowledge can be described as "justified, true belief". In everyday speech, we assign an intuitive meaning to a sentence such as "Bob knows that $2 + 2 = 4$". What is the exact semantic meaning of such a sentence, and can we find a formalization for it?

The field of *epistemic logic* provides formal tools for reasoning about knowledge. What it calls knowledge is an abstraction, an idealization of human knowledge. We will later see how this formal notion of knowledge allows us to reason about the information states of agents in distributed computing systems.

In this chapter, we discuss the *Possible-Worlds* model [Hin62, FHMV03], which enables us to reason about knowledge in a static world. We will see how knowledge defined this way shares some properties with *human* knowledge as understood in everyday language use, justifying the terminology. We introduce different notions of *group knowledge*. We then turn to the *Muddy Children Puzzle* as a well-known example demonstrating the application of epistemic logic.

Chapter 3 builds upon the notion of knowledge developed here, but introduces a separate model based on *runs* and *systems* [HM90, FHMV03].

2.1 Possible Worlds

The *possible-worlds* model formalizes a notion of knowledge. In this chapter, we consider abstract agents and their knowledge, removed from a specific application domain. Chapter 3 introduces a model specifically for distributed computing systems.

In the possible-worlds model, *agents* are actors within a *global state* or *world*. For example, in distributed systems as will be introduced in Chapter 3, agents represent *processes* or *processors*. However, an agent might just as well represent an idealized human, as in the Muddy Children Puzzle (Section 2.4).

As a convention, we use the letters $i, j, \ldots$ to denote agents. Often, we consider n agents and label them i_1 through i_n, giving rise to the following definition:

Definition 2.1. $\mathcal{A} = \{i_1, i_2, \ldots, i_n\}$ *is the* ***set of agents****.*

We want to consider different scenarios of how the agents $\mathcal{A}$ behave and in particular, what information they have available. We capture this using the notion of a *global state* that agents find themselves in. This global state can be considered as being one of multiple possible *worlds* for these agents.

Definition 2.2. *G is the set of global states (set of all possible worlds).*

Right now, we do not make any assumptions about the shape of a global state $s \in G$. The global states should be considered as abstract objects. Only when we consider a specific scenario, we might specify the shape of global states in detail.

A global state s (a world s) represents the state of the agents $\mathcal{A}$ at a specific point in time, a "snapshot" of the agents and their information. We will soon see that we can reason about the state of agents $\mathcal{A}$ using propositional logic, extended with knowledge operators K_i. When doing so, we want to make statements about certain *facts* about the world s. How do we denote these in a formula? At the most basic level, we need some *atomic propositions* to express facts that hold in a given world:

Definition 2.3. Π *is the set of atomic propositions. Every $p \in \Pi$ is an atomic proposition (a propositional atom).*

Π may be countably infinite. The actual propositions are specific to each individual scenario. For example, atomic propositions could be "agent 2 has the ace of spades on its hand" or "child 5 has mud on its forehead" (see Section 2.4). We will assign identifiers for such propositions as needed. The choice of atomic propositions defines what statements we can make about a world s, and hence Π is chosen depending on the specific arguments we want to make about worlds or systems.

We require that an omniscient observer can determine the truth value of an atomic proposition by examining the global state s. In other words, atomic propositions can be either true or false in a given world, and cannot change within this world. This

allows us to define an interpretation function (truth assignment function, valuation) as follows:

Definition 2.4. *An **interpretation function** $\pi : G \mapsto (\Pi \mapsto \text{Bool})$ assigns a truth value to each atomic proposition $p \in \Pi$, depending on the global state $s \in G$ over which it is evaluated.*

Note that π has the state s as a parameter, and is function-valued (it returns another function). Every $\pi(s)$ is an interpretation (truth assignment) in the sense of propositional logic. $\pi(s)(p)$ is the truth value of atomic proposition p in state s. We use $\top$ and $\bot$ to denote *true* and *false* respectively, and write the set of Boolean values as $\text{Bool} = \{\top, \bot\}$.

Throughout this thesis, we will not explicitly define the interpretation function π, but assume that π is clear from the context. We will specify the meaning of atomic propositions as we introduce them.

So far we have discussed facts about the world as they can be examined by an omniscient observer. What about individual agents? An agent i may have limited information about the global state. Multiple *global* states, as distinguishable by an omniscient observer, may result in the same *local* state for agent i, such that the agent considers several global states possible. In other words, these global states are indistinguishable to i with its current information.

The ability to rule out certain global states is what we use to define knowledge of an agent. To formalize this intuition, we can define a Kripke model as follows, following [FHMV03] with some notation borrowed from [DHK08]:

Definition 2.5. *Let G be a set of global states, π an interpretation function over Π, and R_1 through R_n relations on global states, $R_i \subseteq G^2$. Then*

$$M = (G, \pi, R_1, \dots, R_n)$$

*is a **Kripke model** over Π for n agents. R_i is the **accessibility relation** of agent i.*

The key part here are the accessibility relations R_i. $(s,t) \in R_i$ means that in global state s, agent i considers it possible that the global state is actually t. Throughout this thesis, we assume that R_i is an equivalence relation (reflexive, symmetric, transitive); this is the $\mathcal{S5}$ class of models [DHK08].

With R_i as an equivalence relation, $(s,t) \in R_i$ has the intuitive meaning that the global states s and t are *indistinguishable* to agent i. The notion of indistinguishability is central to our notion of knowledge. Consider a formula φ. In any given world s, φ either holds or doesn't hold. We interpret "knowledge" that an agent i has as follows:

Definition 2.6. *We say that* ***agent i knows*** $\boldsymbol{\varphi}$ *when, in all worlds that i considers possible,* φ *holds.*

This can be formalized as follows. We write $K_i\,\varphi$ for "agent i knows that formula φ holds". By enriching propositional logic with operators K_i, we get the following language:

Definition 2.7 (Syntax)**.** *Let* $\mathcal{L}_{PW}$ *be the language generated by the following BNF specification:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\,\varphi$$

where the symbol p *is any atomic proposition* $p \in \Pi$*, and* i *is any agent* $i \in \mathcal{A}$*.*

Note that other propositional operators such as $\vee$ or $\rightarrow$ can be rewritten using $\wedge$ and $\neg$, and are not explicitly introduced here. We will revisit this topic in Section 3.10.

The language $\mathcal{L}_{PW}$ allows us to formulate statements about the knowledge of agents in a certain world s. Given a formula $\varphi \in \mathcal{L}_{PW}$, we want to check whether it holds, relative to a Kripke model M and a fixed world s. This is captured in the following semantics for $\mathcal{L}_{PW}$.

Definition 2.8 (Semantics)**.** *A formula* $\varphi \in \mathcal{L}_{PW}$ *is evaluated with regard to a Kripke model* $M = (G, \pi, R_1, R_2, \ldots, R_n)$ *and a global state (world)* $s \in G$ *as follows:*

$$(M, s) \models p \quad \text{iff} \quad \pi(s)(p) = \top. \tag{2.1}$$

$$(M, s) \models \psi \wedge \psi' \quad \text{iff} \quad (M, s) \models \psi \text{ and } (M, s) \models \psi'. \tag{2.2}$$

$$(M, s) \models \neg\psi \quad \text{iff} \quad (M, s) \not\models \psi. \tag{2.3}$$

$$(M, s) \models K_i\,\psi \quad \text{iff} \quad (M, t) \models \psi \text{ for all } t \text{ with } (s, t) \in R_i. \tag{2.4}$$

This is our first definition of semantics for K_i. We will later see a different semantics that is tailored to distributed systems and that incorporates time (Section 3.10). With Definition 2.8, an agent i is ascribed knowledge of φ if it has sufficient information to rule out all worlds in which φ does not hold. This information is encoded in the relation R_i as well as the actual state s as seen by an omniscient observer.

Halpern and Moses [HM90] aptly remark that "Knowledge is just the dual of possibility [...]". This is not only an intuitive summary of the definition of knowledge we just gave, but also true in a technical sense: Along with K_i, it is common to introduce the dual operator $\hat{K}_i\,\varphi = \neg\, K_i \neg\,\varphi$. The intuitive meaning of $\hat{K}_i\,\varphi$ is "agent i considers φ possible".

2.2 S5 Models

The class (collection) of all Kripke models where all R_i are equivalence relations is denoted as $\mathcal{S5}$, which we will also call *epistemic* models [DHK08]. All models considered in this thesis are epistemic models. Epistemic models have the following properties:

Proposition 2.9. *The following hold for any Kripke model M, world s and any formulas $\varphi, \psi \in \mathcal{L}_{PW}$:*

$$\begin{array}{ll} \text{(K)} & M, s \models K_i (\varphi \to \psi) \to (K_i \varphi \to K_i \psi) \\ \text{(T)} & M, s \models K_i \varphi \to \varphi \\ \text{(4)} & M, s \models K_i \varphi \to K_i K_i \varphi \\ \text{(5)} & M, s \models \neg K_i \varphi \to K_i \neg K_i \varphi \end{array}$$

These equations are modeled after axioms in deductive systems for epistemic logic, where (K) corresponds to the distribution axiom, (T) to the truth axiom, (4) to positive introspection and (5) to negative introspection. These are axioms in the axiom system **S5**, which we will not explore in this work as we restrict ourselves to semantic (rather than syntactic) reasoning.

These equations show that with Definition 2.8, the operator K_i actually possesses some properties that we also expect of human knowledge. For example, (T) intuitively states that only true things can be known. (4) states that agents know what they know, while (5) states that agents know what they don't know. As [DHK08] remark, the assumption that agents know what they don't know is rather strong, and a good example for how the properties of K_i deviate from typical assumptions about human knowledge.

A more in-depth treatment of axiom systems and syntactic reasoning for epistemic logic is given by [FHMV03] and [DHK08]. Throughout this thesis, we will limit ourselves to reasoning on the semantic level.

2.3 Group Knowledge

We have defined knowledge for a single agent i. What if we have a group G of agents, $G \subseteq \mathcal{A}$? What can we say about their knowledge? Can agents in such a group gain information by pooling their knowledge?

Halpern and Moses [HM90] introduce four notions of group knowledge:

Definition 2.10 (Notions of group knowledge)**.** *For a group $G \subseteq \mathcal{A}$, we define the following knowledge operators:*

- $D_G \varphi$ *— distributed knowledge: if an agent knew everything that each member of G knows, it would know φ,*

- $S_G\,\varphi$ — *someone in G knows* φ*,*
- $E_G\,\varphi$ — *everyone in G knows* φ*, and*
- $C_G\,\varphi$ — φ *is common knowledge in G, that is,* $E_G^k\,\varphi$ *holds for arbitrary k.*

The subscript "G" is usually omitted when $G = \mathcal{A}$*.*

Note that $E_G^k\,\varphi$ is an abbreviation for $\overbrace{E_G\,E_G \cdots E_G}^{k \text{ times}}\varphi$. Interestingly, while $K_i\,\varphi \rightarrow K_i\,K_i\,\varphi$ holds by Proposition 2.9, $E_G\,\varphi \rightarrow E_G\,E_G\,\varphi$ does not always hold. This also means that in general, $C_G\,\varphi$ is a stronger statement than $E_G^k\,\varphi$ for any given k.

Clearly, S_G and E_G could equivalently be specified in $\mathcal{L}_{PW}$ using just K_i, disjunctions and conjunctions. On the other hand, D_G and C_G are, in a sense, new notions of knowledge. We can extend our semantics as follows:

$$(M,s) \models S_G\,\psi \quad \text{iff} \quad (M,s) \models K_i\,\psi \text{ for some } i \in G. \tag{2.5}$$

$$(M,s) \models E_G\,\psi \quad \text{iff} \quad (M,s) \models K_i\,\psi \text{ for all } i \in G. \tag{2.6}$$

$$(M,s) \models C_G\,\psi \quad \text{iff} \quad (M,s) \models E_G^k\,\psi \text{ for } k = 1, 2, \ldots \tag{2.7}$$

$$(M,s) \models D_G\,\psi \quad \text{iff} \quad (M,t) \models \psi \text{ for all } t \text{ with } (s,t) \in \bigcap_{i \in G} R_i. \tag{2.8}$$

2.4 Muddy Children Puzzle

The Muddy Children Puzzle is a popular example for the kind of reasoning that epistemic logic can capture. The following account is from [Bar81], cited after [HM90]:

> Imagine n children playing together. The mother of these children has told them that if they get dirty there will be severe consequences. So, of course, each child wants to keep clean, but each would love to see the others get dirty. Now it happens during their play that some of the children, say k of them, get mud on their foreheads. Each can see the mud on others but not on his own forehead. So, of course, no one says a thing. Along comes the father, who says, "At least one of you has mud on your head", thus expressing a fact known to each of them before he spoke (if $k > 1$). The father then asks the following question, over and over: "Can any of you prove you have mud on your head?" Assuming that all the children are perceptive, intelligent, truthful, and that they answer simultaneously, what will happen?

In the possible-worlds model introduced in this chapter, the children can be considered *agents*; the father could also be considered an agent, or an entity external to the system. We will stick with the latter interpretation, only considering children to be

agents. As the father makes his announcements and asks questions, the knowledge of the children changes, which we can model as a series of Kripke models M. We will use the following atomic propositions to reason about this problem:

- p_i means that the i-th child has mud on its forehead ($1 \leq i \leq n$), and
- q means that $k \geq 1$, i.e., at least one kid has mud on its forehead.

Note that q captures exactly the father's initial statement, while the father's repeated question, "Can any of you prove you have mud on your head?", is essentially asking whether $K_i\, p_i$ for some i.

What will happen? [Bar81] notes that with k muddy children, all children will answer "no" the first $k-1$ times. When the father asks his question for the k-th time, the k muddy children will answer "yes".

Let us consider the case when exactly one child has a muddy forehead, i.e., $k = 1$. The only muddy child, say i, will see only clear foreheads; thus considering $k = 0$ and $k = 1$ possible. But with the father's announcement, $k = 1$ remains as the only possibility, and $K_i\, p_i$ holds. Similarly, for $k \geq 2$ muddy children, assume that the muddy children would have answered "yes" to the $(k-1)$-th question or earlier if there were less than k muddy children. Then all muddy children, who see $k-1$ muddy foreheads, can conclude that their foreheads must be muddy.

As [Bar81] remarks, in the case of $k \geq 2$, the father's initial announcement of $k \geq 1$ is not providing any new information. But, assuming that the children still consider $k = 1$ possible, the children cannot solve the puzzle without this announcement!

What is going on from the perspective of epistemic logic? Consider the situation with two muddy children, $k = 2$. Before the father's announcement, everyone knows that $k \geq 1$, i.e., $E\, q$ holds. But not everyone knows that everyone knows this, i.e., $E^2\, q$ does not hold. On the other hand, since everyone can observe the father's announcement, and everyone can observe this, ..., we have $C\, q$ after the father's announcement. The effect of the father's announcement is to establish common knowledge.

[HM90] argue that to solve the muddy children puzzle, at least $E^k\, q$ is required, whereas $E^{k-1}\, q$ is not sufficient. It is the father's announcement that provides this required level of knowledge (and it even provides a stronger condition, namely *common* knowledge).

Other puzzles lend themselves to an analysis using epistemic logic: e.g., "Consecutive Numbers" [DHK08, ex. 2.4] or "Three-player card game" [DHK08, ex. 4.2]. The framework of epistemic logic can also be applied to distributed computing systems, as we will see in the next chapter.

CHAPTER 3

Temporal-Epistemic Logic in Distributed Systems

In Chapter 2, we introduced the possible-worlds model and a formal definition of knowledge. So far we have evaluated formulas over one specific, static world. To describe distributed systems, we would like to argue not only about one world in isolation, but about a time sequence of worlds, or even multiple, related time sequences.

A problem in distributed systems is typically characterized by a *problem statement* and a *system model*. The system model describes what is also called the *setting* or *environment* that agents find themselves in. What we refer to as agents are also called *processors*, *processes*, or *nodes* depending on literature. The system model is an abstraction of the real-world processing and networking characteristics of computing systems. At a very fundamental level, system models are called *synchronous* or *asynchronous*, depending on whether agents have access to a shared notion of time (a shared clock). If the system model allows erroneous computations or communication, the *failure assumption* specifies the remaining correctness guarantees, typically as an upper bound on the number of failures of a certain kind. To ensure progress, *liveness conditions* are another part of the system model.

A system going through a possibly infinite sequence of states is called an *execution* [AW04] or *run* [HM90]. A run is typically the result of executing an algorithm in the given system. When all conditions on system behavior, such as the failure assumption or liveness conditions are fulfilled in a run, such a run is called *admissible* [AW04]. The goal is then often to analyze which properties hold in all admissible runs of an algorithm, or to find an algorithm that guarantees certain properties in all admissible runs.

In this chapter, we introduce *runs*, *systems*, *protocols* and *contexts* [HM90,

FHMV03] as a foundation for reasoning about distributed systems using temporal-epistemic logic. In Chapter 4, we see how this framework applies to asynchronous message-passing systems in particular. In Chapter 5, we extend this further to a Byzantine [LSP82] failure assumption.

Throughout this chapter, we assume a discrete time domain, $t \in \{0, 1, \ldots\}$. This is not in conflict with a treatment of asynchronous systems: one can think of the time base being stretched by an arbitrary factor, or t progressing simply whenever a message arrives. t does not necessarily progress at the same speed as real time or at constant speed.

3.1 Runs

Runs and *systems* are a formalization of a “time-series of worlds” and enables analysis of distributed systems using temporal-epistemic logic. *Runs* and *systems* were proposed in [HF89, HM90] and appeared slightly refined in [FHMV03]. Runs and systems are used to analyze distributed systems for example in [HM90], which examines common knowledge in distributed systems, and in [BM14], which investigates necessary communication structures for certain actions (*centipedes*). Runs and systems are the foundation for the higher-level concepts of *protocols* [HF89] and *contexts* [FHMV95, FHMV03].

As in Section 2.1, we consider a set of n agents $\mathcal{A} = \{i_1, i_2, \ldots, i_n\}$. In the context of distributed systems, agents are nodes (processes, processors) on some sort of computing network. Each agent can be thought of as executing a local state machine. In addition, we will consider the *environment* as a “pseudo-agent”. The state of the environment can encode external influences on the system, for example, input, wall-clock time, randomness, failures, etc. The set of agents $\mathcal{A}$ contains all regular agents, but not the environment.

In this section, we introduce runs over abstract state domains. We later introduce *message-passing systems*, where local states are *histories*.

Definition 3.1. *Consider a snapshot of n agents at a given point in time. Let s_e be the state of the environment, and s_i be the state of agent i for all agents $1 \leq i \leq n$. Then,*

$$s = (s_e,\, s_1,\, s_2,\, \ldots,\, s_n)$$

is the global state s. For each agent i, L_i is the ***domain of local states*** *of agent i, L_e is the domain of local states of the environment, and G is the* ***domain of global states****, with $s_i \in L_i$, $s_e \in L_e$, $s \in G$, and*

$$G = L_e \times L_1 \times L_2 \times \cdots \times L_n.$$

In Section 2.1, we introduced an abstract definition of global states and called them

worlds. For runs and systems, we assume a specific shape of the global state, namely that it is composed of the local states of individual agents. This is a natural way of modeling distributed computing systems.

When a computing system executes an algorithm, a time-sequence of global states can be observed. In the terms of [AW04], this would be called an *execution*. In the framework of runs and systems, this is called a *run*. Following [FHMV03], we define runs as follows:

Definition 3.2. *A* ***run*** $r : \mathbb{N} \mapsto G$ *represents one specific execution of a system in the history of time.* $r(t)$ *maps a point in time to a global state of the system.*

Per Definition 3.1, a global state $s \in G$ consists of the state of the environment s_e and the local states of agents $s_1, \ldots, s_n$. Analogous to this notation, we split $r(t)$ into per-agent components, such that for all $r(t) \in G$,

$$r(t) = (r_e(t),\, r_1(t),\, r_2(t),\, \ldots,\, r_n(t)).$$

$r_i : \mathbb{N} \mapsto L_i$ is the sequence of local states of agent i in run r.

Throughout this thesis, the domain of t is $\mathbb{N} = \{0, 1, 2, \ldots\}$, so $r_i(t)$ is the state of agent i in run r at time t. This is the notation that we will most frequently use, such as when we will define the knowledge operator over runs. Note that for two agents i and j, $r_i(t)$ and $r_j(t)$ refer to the same run r (r_i and r_j are not independent objects).

3.2 Systems

A set of related runs—for example, all runs that correspond to executions of a given algorithm in a computing system—is called a *system* [FHMV03]:

Definition 3.3. *A* ***system*** $\mathcal{R}$ *over* G *is a* ***set of runs****. We say that a run* $r \in \mathcal{R}$ *is a run of system* $\mathcal{R}$*.*

Note that *system* is a technical term here. A system $\mathcal{R}$ in the sense of Definition 3.3 can represent a distributed computing system, or an abstraction for the behavior of a group of people. We might also be able to come up with a system $\mathcal{R}$ in the sense of Definition 3.3 that does not resemble any such real-world group of agents. To prevent confusion, we will write “the system $\mathcal{R}$” as a fixed expression when referring to a system in the sense of Definition 3.3.

A system $\mathcal{R}$ is really just *any set of runs* with a common set of agents and domain of global states, and this set a priori does not have any additional properties that make it a “system”.

Assume that we want to talk about a run r at some time t. Then we might not only be interested in the global state $r(t)$, but also the earlier and later states in this run. [FHMV03] introduce the following terminology:

Definition 3.4. *A **point** in system $\mathcal{R}$ is a pair (r, t) such that $r \in \mathcal{R}$ and t is in the domain of r.*

Technically, (r, t) is a pair of a run and a time, whereas $r(t)$ is just a global state without connection to r or t. We will often not make this distinction and write (r, t) and $r(t)$ interchangeably.

3.3 Histories

We have defined runs and systems over abstract domains for states. In order to model message-passing systems[1] using runs and systems, we define *histories.* We consider systems with *perfect recall* [FHMV03], that is, systems where agents remember all of their past states. In Section 3.10, we will use the local state of agents to define epistemic knowledge. There are only a few ways how the local state of agents can change, and thus how agents can gain knowledge in a message-passing system:

- performing a local state transition (computation),
- sending a message,
- receiving a message, or
- interactions with the environment (inputs, passage of time).

No other communication or synchronization primitives exist.

In line with these constraints, we model the local state of agents in such a system with histories. We extend the definition of Fagin et al. [FHMV03] to allow *sets* of events at each time step and include *external events* [BM14]:

Definition 3.5. *Fix an agent i. Let Σ_i be its set of initial states, Int_i its set of internal actions, Ext_i its set of external actions, and Msgs a set of messages. These domains are arbitrary. Then a **history** h over $\Sigma_i, \mathrm{Int}_i, \mathrm{Ext}_i, \mathrm{Msgs}$ is a sequence*

$$h = \langle s_{i0}, E_1, E_2, \ldots, E_m \rangle,$$

where $\langle s_{i0} \rangle \in \Sigma_i$ is the initial state of agent i, m is arbitrary ≥ 0, and E_1 through E_m are sets of events on agent i as per Definition 3.6.

[1] Here, "message-passing system" refers to computers on a network, not a system $\mathcal{R}$ in the sense of Definition 3.3. However, Definition 3.7 will also use the term "message-passing system" to refer to a system $\mathcal{R}$ with certain properties. This ambiguity is a bit unfortunate, but we want to stay true to the terminology introduced by [FHMV03].

For ease of exposition, we assume that any message $M \in \text{Msgs}$ is unique, and that it is sent only once. This is an innocent assumption, since starting from a system where messages do not have this property, we can make messages unique by attaching a sequence number for our analysis (possibly resulting in an infinite set of messages), an approach taken by [PKS17].

Sequence notation. We denote histories as (ordered) sequences. When h is a sequence, we will write $h' = h : E$ to indicate that h' equals h with the sequence element E appended, i.e. $\langle s_{i0}, E_1 \rangle : E_2 = \langle s_{i0}, E_1, E_2 \rangle$. We blur the line between sequences and sets when the intended meaning should be clear, in particular we will write $E_1 \in h$ for "E_1 is an element of sequence h".

Definition 3.6. *Fix an agent i. An* ***event on agent i*** *($e \in E \in h$) is one of the following with its intended meaning, for any $j \in \mathcal{A}$, $M \in \text{Msgs}$, $a \in \text{Int}_i$, $b \in \text{Ext}_i$:*

- $\text{internal}(i, a)$*: Agent i performs internal action a (a local computing step).*
- $\text{send}(i, j, M)$*: Agent i sends message M to agent j.*
- $\text{recv}(j, i, M)$*: Agent i receives message M from agent j.*
- $\text{external}(i, b)$*: External action b occurs on agent i.*

We define the arguments of send and recv here such that both $\text{send}(a, b, M)$ and $\text{recv}(a, b, M)$ refer to a message M from a to b.

Note that i is fixed in Definitions 3.5 and 3.6, i.e., the history of agent i can only contain events pertaining to agent i. s_{i0} is a single object, but E_1 through E_m are *sets* of events. The intuition is that events in the same set E occur simultaneously. This makes it possible to model e.g. lock-step synchronous systems or atomic broadcasts in this framework.

How do histories of agents evolve over time in a message-passing system? Informally, the history of agent i starts out with the initial state only ($m = 0$). When one or multiple events occur, a set of events E is appended to the history. In a lock-step synchronous system, such a set of events might occur at every time step for every agent. In an asynchronous system, this might not be the case. Hence, the length of the history is not necessarily linked to the elapsed time.

3.4 Message-Passing Systems

We now consider systems $\mathcal{R}$ where the local states of agents are histories and progress in such a way that they record all past events that have occurred on an agent. When a system $\mathcal{R}$ fulfills these constraints, Fagin et al. [FHMV03] call $\mathcal{R}$ a *message-passing system*:

Definition 3.7. *A system $\mathcal{R}$ is a* ***failure-free message-passing system*** *when all of the following hold for any run $r \in \mathcal{R}$, any time t and any $i, j \in \mathcal{A}$:*

(MP1) $r_i(t)$ *is a history over* $\Sigma_i, \mathrm{Int}_i, \mathrm{Ext}_i, \mathrm{Msgs}$.

(MP2) *When* $\mathrm{recv}(i, j, M)$ *has occurred in the history* $r_j(t)$*, a corresponding event* $\mathrm{send}(i, j, M)$ *has occurred in the history* $r_i(t)$.

(MP3) $r_i(0)$ *consists only of the initial state of agent i.*
$r_i(t+1)$ *equals* $r_i(t)$*, optionally extended by one set of events.*

(MP1) limits what kind of knowledge agents can have, since $r_i(t)$ is the local state of agent i. (MP2) guarantees causality of message delivery, and in fact also that messages cannot be forged or corrupted. (MP3) describes how histories evolve, and it establishes *perfect recall*, i.e., agents always remember information they have had in the past.

[FHMV03] define an additional property for guaranteed eventual message delivery ("reliability") that they do not require for all message-passing systems:

(MP4) *When* $\mathrm{send}(i, j, M)$ *has occurred in the history* $r_i(t)$*,* $\exists\, t' \geq t$ *s.t.* $\mathrm{recv}(i, j, M)$ *has occurred in the history* $r_j(t')$.

Limitations of Definition 3.7. In a sense, Definition 3.7 places an "upper bound" on a system $\mathcal{R}$: (MP1) through (MP3) ensure that $\mathcal{R}$ does not encode any "magic" actions that cannot occur in real-world message-passing systems. In other words, every run r of a message-passing system $\mathcal{R}$ could occur in a real-world message-passing system. However, there is no "lower bound" on $\mathcal{R}$: runs that could in fact occur in a real-world message-passing system are not guaranteed to be present in $\mathcal{R}$.

Why is this a problem? Assume that in a system $\mathcal{R}$, two messages M_1 and M_2 are sent. In $\mathcal{R}$, message M_1 is always delivered earlier than M_2. Further assume that this is not always the case in real-world message-passing systems. But then an algorithm that works fine in $\mathcal{R}$ might not work in a real-world system!

Hence, knowing that $\mathcal{R}$ is a message-passing system is not sufficient for analysis without more guarantees about which runs are in $\mathcal{R}$. To improve on this, we take a look at *protocols and contexts* [FHMV95, FHMV03].

3.5 Protocols

Given a distributed algorithm and a system model, we would like a system $\mathcal{R}$ where each run $r \in \mathcal{R}$ represents an admissible execution of the given algorithm in the given system model. This then allows us to connect reasoning about a system $\mathcal{R}$ using epistemic logic to properties of the distributed system that $\mathcal{R}$ models.

Definition 3.7 only specifies properties for *individual runs* within a message-passing system $\mathcal{R}$. Knowledge of agents is defined relative to *all possible runs*, however. To construct the set of all possible runs, Fagin et al. [FHMV03] introduce *protocols* and *contexts*.

A given protocol and context induce a set of runs representing the execution of this protocol in the given context. Protocols represent a *distributed algorithm*. The context represents the *setting* in which an algorithm is executed; it effectively encodes the *system model* or the *adversary*.

We first introduce protocols and contexts as an abstract mechanism to generate a set of runs. In Section 4.1, we will apply this framework to message-passing systems in particular.

A *protocol* for an agent i specifies its behavior, the *algorithm* that an agent executes. We need a domain for *actions* that an agent can take:

Definition 3.8. Act_i *is the set of* ***actions*** *of agent* i*.* Act_e *is the set of actions of the environment.*

For agents in message-passing systems, an action could be a local state transition (an internal action) combined with a number of send events as per Definition 3.6. We will revisit this in Section 4.1. An action is essentially something that an agent *requests* to perform, but it is up to the context whether the behavior specified by that action actually takes place. This is because the context specifies whether and how the action affects the global state, as we will see in Definition 3.13.

We require that an agent chooses its actions exclusively based on its local state. Following [FHMV03], we define protocols as follows:

Definition 3.9. *Consider an agent* i*. Let* L_i *be its set of local states and* Act_i *its set of actions. Then*

$$P_i : L_i \mapsto 2^{\mathrm{Act}_i}$$

is a ***protocol for agent*** $\boldsymbol{i}$*. Analogously,* $P_e : L_e \mapsto 2^{\mathrm{Act}_e}$ *is a protocol for the environment.*

A protocol maps each local state to a *set* of actions, encoding nondeterministic choice. In this thesis, protocols for agents are always deterministic, i.e., $|P_i(s_i)| = 1$. The protocol of the environment can be nondeterministic.

Often, we want to reason about the protocols of *all* the agents combined:

Definition 3.10. *$P = (P_1, P_2, \dots, P_n)$ is the **joint protocol** of agents 1 through n.*

Notably, the protocol P_e of the environment is not part of P. While P is controlled by the algorithm designer, P_e is usually fixed as part of the problem statement. For this practical reason, P_e is not included in P, but rather part of the context.

3.6 Contexts

Intuitively, a context represents the system model in which a distributed algorithm is executed. A joint protocol P and a context γ together yield a set of runs $\mathcal{R}$. We will later say that $\mathcal{R}$ represents P in γ. $\mathcal{R}$ corresponds to all possible executions of P in the system model defined by the context γ.

Definition 3.11. *A **context** γ is a tuple $\gamma = (L, \mathrm{Act}, G_0, \tau, \Psi, P_e)$, where*

- *$L = (L_e, L_1, \dots, L_n)$ are domains of local states, which induce the domain of global states $G = L_e \times L_1 \times \cdots \times L_n$,*
- *$\mathrm{Act} = (\mathrm{Act}_e, \mathrm{Act}_1, \dots, \mathrm{Act}_n)$ are domains of actions for protocols in γ,*
- *$G_0 \subseteq (\Sigma_e \times \Sigma_1 \times \cdots \times \Sigma_n) \subseteq G$ is the set of initial global states,*
- *$\tau : (\mathrm{Act}_e \times \mathrm{Act}_1 \times \cdots \times \mathrm{Act}_n) \mapsto (G \mapsto G)$ is the transition function,*
- *Ψ is a liveness condition (Section 3.7), and*
- *P_e is the protocol of the environment.*

Similar to regular agents, the environment e has a local state and executes a (possibly nondeterministic) protocol. P_e yields a set of actions for each state.

For each global state, the protocols P_i of agents and the protocol of the environment P_e each yield a set of actions. Depending on the system model that we encode in γ, some actions of agents influence the state of other agents or the environment. Such a "non-local" action could be sending a message, or writing a shared memory variable. In some system models, the environment can make agents behave erroneously, crash, or perform stuttering steps instead of their desired action. To capture these possibilities, we always consider global states and global state transitions. In particular, we do not view actions in isolation, but instead consider the effect of *joint actions* on the global state:

Definition 3.12. *A **joint action** $\vec{\alpha}$ is a tuple*

$$\vec{\alpha} = (\alpha_e, \alpha_1, \alpha_2, \dots, \alpha_n),$$

where $\alpha_i \in \mathrm{Act}_i$ is the action of agent i and $\alpha_e \in \mathrm{Act}_e$ is the action of the environment.

Definition 3.13. *The* ***transition function***

$$\tau : \mathrm{Act}_e \times \mathrm{Act}_1 \times \cdots \times \mathrm{Act}_n \;\mapsto\; (G \;\mapsto\; G)$$

defines a global transition relation for each joint action. It specifies the effects of a joint action on the global state.

The intuition, as visualized in Figure 3.1, is that each run in a context γ starts out in an initial global state $r(0)$. Depending on the state, the protocol of every agent specifies a set of actions that it wants to perform; similarly for the environment. The Cartesian product of these sets of actions yields a number of joint actions, of which exactly one joint action $\vec{\alpha}$ is chosen nondeterministically. The transition function τ describes how the joint action $\vec{\alpha}$ changes the global state. A formal definition follows in Definition 3.15.

τ encodes whether agents find themselves in a message-passing or a shared-memory system, as well as what failures they are subject to.

In particular, τ defines the influence that P_e, the protocol of the environment, can have on the system. Restrictions on the environment (on the adversary) can be modeled either as a property of τ or as a property of P_e. Throughout this thesis, we encode desired properties in τ and leave P_e unrestricted. For example, in Section 5.2 we will specify τ such that P_e can make up to f agents behave in a faulty way. We will see a different choice of τ for the failure-free case in Section 4.1.

Observe that $\tau(\vec{\alpha})$ is a *function* from G to G. While $\vec{\alpha}$ is chosen nondeterministically, given $\vec{\alpha}$, the following global state is uniquely determined. In other words, given an initial state $r(0)$ and a sequence of joint actions, the resulting state is uniquely determined. This will become important when we introduce *recording contexts* in Definition 4.5.

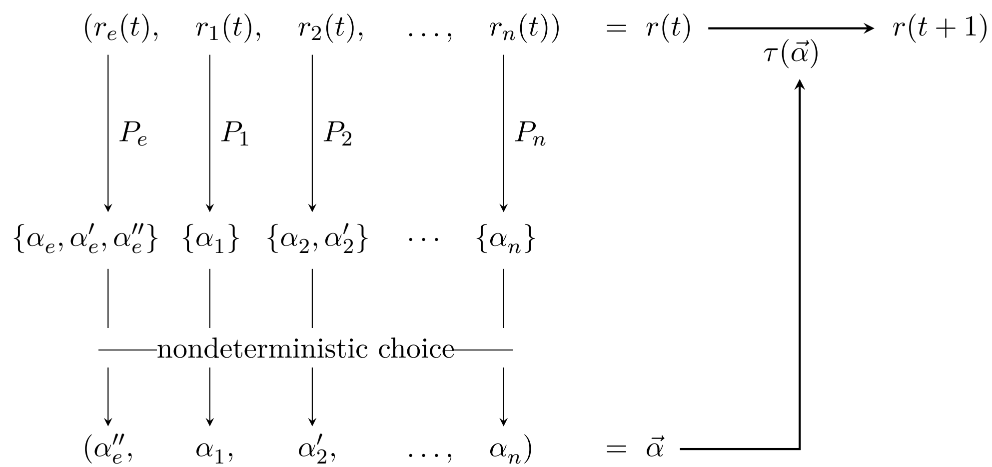

Figure 3.1: The evolution of global states in a context γ, from $r(t)$ to $r(t+1)$. For each agent, the protocol P_i specifies a set of actions based on its *local* state. One action from each set is picked nondeterministically to form the joint action $\vec{\alpha}$. τ defines a transition relation on global states, depending on $\vec{\alpha}$. In this example, P_e and P_2 are nondeterministic protocols, whereas we will later assume deterministic protocols (except for P_e).

3.7 Liveness Conditions

In a given system model, we want to guarantee safety properties: in any prefix of a run, nothing bad happens. For example, no message is delivered before it was sent. We can encode this in the transition function τ.

Often, we also want to argue that eventually, something good happens [AW04]. Hence contexts can require runs to satisfy liveness conditions, which we specify as a predicate Ψ.

Definition 3.14. *The **liveness condition** Ψ is a predicate over runs r.*

Ψ can be interpreted interchangeably as a function mapping each run to a Boolean value, or simply as a set of runs. We denote the trivial condition that allows all runs as *True*. To enforce that Ψ should specify liveness conditions rather than arbitrary conditions, we require that contexts are *non-excluding* in Definition 3.17.

In [FHMV03], Fagin et al. informally define conditions *Rel*, *Fair* and *FS*, where *Rel* guarantees eventual ("reliable") message delivery, *Fair* guarantees eventual delivery of messages sent infinitely often, and *FS* guarantees a fair schedule, i.e., every agent is allowed to perform actions infinitely often. We will revisit liveness conditions in Sections 4.3 and 5.4.

3.8 Generating Systems from Protocols

We have defined protocols and contexts, and argued that a joint protocol P and a context γ yield a set of runs $\mathcal{R}$. Formally:

Definition 3.15. *Consider a joint protocol P and a context $\gamma = (L, \mathrm{Act}, G_0, \tau, \Psi, P_e)$. A run r is **weakly consistent with P in γ** when*

$$r(0) \in G_0, \quad \text{and} \tag{3.1}$$

$$\begin{aligned} \text{for all } t,\ \exists\, \vec{\alpha} = (\alpha_e, \alpha_1, \ldots \alpha_n)\ \text{ s.t. }\ & (r(t), r(t+1)) \in \tau(\vec{\alpha}) \\ \text{and }\ & \alpha_e \in P_e(r(t)) \\ \text{and }\ & \alpha_i \in P_i(r(t)) \quad \forall i \in \mathcal{A}. \end{aligned} \tag{3.2}$$

*r is **(strongly) consistent with P in γ** when it also satisfies*

$$r \in \Psi. \tag{3.3}$$

A run consistent with P in γ starts out in one of the initial states specified by the context. All subsequent states result from applying the global state transition for some joint action $\vec{\alpha}$ to the previous state. The joint action contains actions as specified by the protocol of each agent and the environment. Since agents can specify

a *set* of possible actions in general, there can be multiple choices of $\vec{\alpha}$ in a single state. Finally, to be *strongly* consistent, the run also needs to satisfy the admissibility condition.

Definition 3.16. *Consider a joint protocol P and a context γ. Let $\mathcal{R}$ be the set of all runs consistent with P in γ. Then $\mathcal{R}$ is the* ***system representing P in γ****, also written as $\hat{\mathcal{R}}(P, \gamma)$.*

To get a shorter notation, we will often write $\mathcal{R} = \hat{\mathcal{R}}(P, \gamma)$ to indicate that $\mathcal{R}$ is the system representing P in γ.

3.9 Non-Excluding Contexts

We introduced the notion of weak consistency for runs that do not necessarily satisfy the admissibility condition. Based on this, Fagin et al. [FHMV03] introduce *non-excluding contexts* to restrict Ψ to liveness conditions, rather than arbitrary conditions that could also be encoded in the transition function τ.

Definition 3.17. *A context γ is* ***non-excluding*** *when, for an arbitrary joint protocol P and all times t, the following holds: Given r weakly consistent with P in γ, there is a strongly consistent r' s.t. $r'(t') = r(t')$ for all $t' \leq t$.*

In other words, non-excluding contexts guarantee that any prefix of a weakly consistent run can always be extended to become a strongly consistent run. We only consider non-excluding contexts throughout this thesis.

3.10 Temporal-Epistemic Logic over Runs: Syntax

In Definition 3.1, we specified that every global state is composed of the local states of agents and the state of the environment. From an agent's point of view, two global states are distinguishable exactly when the local state of i differs in these two states. This intuition yields an indistinguishability relation over all global states, with which we could build a Kripke model (in fact, an $\mathcal{S}5$ model) over the global states G. This would be a natural way of defining epistemic modal operators over runs. Rather than mapping runs to Kripke models, however, we will define the knowledge operator directly over runs (in Definition 3.21).

Since runs encode a time sequence of global states, given a certain point in a run, an omniscient observer can argue about the past and future global states of the entire run r. This allows us to also introduce temporal modal operators over runs.

We will use the formal language introduced here throughout the remainder of this thesis. In contrast to [FHMV03], we will not consider group knowledge for runs and systems, and we will not introduce the next-state operator $\bigcirc$. This is done

for ease of exposition, as we never need them in our later arguments. It should be possible to add them in a straightforward way.

Our definitions build upon those of Section 2.1, where we considered the Possible-Worlds framework. In this section, we will also define the temporal modal operators $\Box$ ("holds globally from now on") and $\Diamond$ ("holds eventually"). Furthermore, we will introduce shorthand notations for formulas that hold at all times in a run, and for formulas that hold across all runs of a system.

As introduced in Section 2.1, $\mathcal{A}$ denotes the set of agents and Π denotes the set of atomic propositions (propositional atoms). First, we specify the language that we will use for temporal-epistemic formulas throughout the rest of this thesis:

Definition 3.18. *$\mathcal{L}$ is the language generated by the following BNF specification:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\, \varphi \mid \Box\varphi$$

where the symbol p is any atomic proposition $p \in \Pi$, and i is any agent $i \in \mathcal{A}$.

Note that other propositional operators such as $\vee$ or $\rightarrow$ can be rewritten using $\wedge$ and $\neg$. Additionally, $\Diamond\varphi$ can be rewritten as $\neg\Box\neg\varphi$. For ease of exposition, we only include a minimum set of operators in $\mathcal{L}$. Throughout this thesis, when a formula uses other operators, the formula should be treated as a shorthand for its equivalent formula in $\mathcal{L}$.

3.11 Temporal-Epistemic Logic over Runs: Semantics

We can now define how formulas in $\mathcal{L}$ are evaluated over runs, and start by assigning truth values to atoms:

Definition 3.19. *An **interpretation function** $\pi : G \mapsto (\Pi \mapsto Bool)$ assigns a truth value to each atomic proposition $p \in \Pi$, depending on the global state $s \in G$ over which it is evaluated.*

When applied to runs, $\pi(r(t))(p)$ is the truth value of atomic proposition p in run r at time t (in global state $r(t)$). This is analogous to Definition 2.5 in the Possible-Worlds framework. Again, our choice of atomic propositions and π defines which statements we can make about runs and systems. We will often not explicitly mention π, although formally, [FHMV03] introduce the following object:

Definition 3.20. *An **interpreted system** is the pair $(\mathcal{R}, \pi)$ of a system $\mathcal{R}$ and an interpretation function π.*

On the language $\mathcal{L}$, we can now define a semantics to evaluate formulas over runs and systems, extending Definition 2.8.

We define knowledge using the fact that global states are indistinguishable to an agent i whenever its local state is the same, thus equality of the local state of i replaces the indistinguishability relation R_i from Definition 2.8.

We introduce the temporal operator $\Box$ ("globally"), which specifies that something holds in a state and all later states. Its dual operator, $\Diamond$ ("eventually"), specifies that something holds eventually, in *some* later state.

Our notion of knowledge in a run is always linked to a system $\mathcal{R}$, as the system $\mathcal{R}$ defines which states an agent considers possible. Some formulas may hold in *every* possible system, for which we introduce a shorthand notation. Finally, we often want to make statements about something that holds throughout a run, or even throughout all runs in a system $\mathcal{R}$. For these cases, we also introduce shorthand notations.

Definition 3.21. *A formula $\varphi \in \mathcal{L}$ is evaluated with regard to an interpreted system $(\mathcal{R}, \pi)$, a run r and a time t as follows:*

$$(\mathcal{R}, r, t) \models p \quad \text{iff} \quad \pi(r(t))(p) = \top. \tag{3.4}$$

$$(\mathcal{R}, r, t) \models \neg\varphi \quad \text{iff} \quad (\mathcal{R}, r, t) \not\models \varphi. \tag{3.5}$$

$$(\mathcal{R}, r, t) \models \varphi \wedge \psi \quad \text{iff} \quad (\mathcal{R}, r, t) \models \varphi \text{ and } (\mathcal{R}, r, t) \models \psi. \tag{3.6}$$

$$(\mathcal{R}, r, t) \models K_i\, \varphi \quad \text{iff} \quad (\mathcal{R}, r', t') \models \varphi \text{ for all } r' \in \mathcal{R} \text{ s.t. } r'_i(t') = r_i(t). \tag{3.7}$$

$$(\mathcal{R}, r, t) \models \Box\varphi \quad \text{iff} \quad (\mathcal{R}, r, t') \models \varphi \text{ for all } t' \geq t. \tag{3.8}$$

For properties of a given run that hold in every possible system $\mathcal{R}$ (system-independent properties), we also write

$$(r, t) \models \varphi \quad \text{iff} \quad (\mathcal{R}, r, t) \models \varphi \text{ for all } \mathcal{R} \ni r. \tag{3.9}$$

For properties of runs resp. systems that hold at any point in time (global properties), we also write

$$(\mathcal{R}, r) \models \varphi \quad \text{iff} \quad (\mathcal{R}, r, t) \models \varphi \text{ for all } t \geq 0. \tag{3.10}$$

$$\mathcal{R} \models \varphi \quad \text{iff} \quad (\mathcal{R}, r, t) \models \varphi \text{ for all } r \in \mathcal{R},\ t \geq 0. \tag{3.11}$$

$$r \models \varphi \quad \text{iff} \quad (\mathcal{R}, r, t) \models \varphi \text{ for all } \mathcal{R} \ni r,\ t \geq 0. \tag{3.12}$$

We will use this semantics throughout the rest of this thesis.

CHAPTER 4

Failure-Free Asynchronous Message-Passing Contexts

In Chapter 3, we have introduced *protocols and contexts* as a generic way of generating a set of *runs* (a *system*). We have further introduced the language $\mathcal{L}$ that includes epistemic and temporal operators, and defined a semantics to evaluate a formula $\varphi \in \mathcal{L}$ over a given run in a given system.

In this chapter, we examine contexts that model failure-free asynchronous message-passing systems (failure-free a.m.p. systems). In such systems, agents have a local state that is not shared with other agents. Agents do not have a notion of time. Agents can send messages to each other, which get delivered eventually and intact, but not necessarily timely or in order. The environment decides when agents make computing steps, but agents are never stuck forever (fair schedule). Neither agents nor messages are subject to failures.

We denote this class of contexts as Γ_{amp} and write $\gamma \in \Gamma_{\text{amp}}$ for contexts in this class. This is a refinement of the *message-passing contexts* introduced by Fagin et al. [FHMV03] and based on the "examples for contexts" they give.

In contrast to [FHMV03], we allow environment actions α_e that specify more than one action for an agent at a time, with an eye towards a later extension of this model towards synchronous systems. In particular, our model can capture simultaneous delivery of multiple messages. In truly asynchronous systems, the difference might be negligible: when agents do not have access to the time t available to the omniscient observer, they cannot distinguish simultaneous events from consecutive ones.

We focus on failure-free a.m.p. systems in this chapter, but keep an eye on how other system models could be captured using similar definitions. In Section 4.6, we highlight potential pitfalls for an extension to lock-step synchronous systems. In

Chapter 5, we extend the class Γ_{amp} introduced here to Γ_{bamp}, a class of contexts that allows Byzantine failures.

4.1 The Class Γ_{amp}

A context γ is specific to a set of agents $\mathcal{A}$ and local states L. To reason about contexts independently from the number of agents and from their local states, we introduce the class Γ_{amp} of contexts which guarantees certain properties relevant for the analysis of failure-free a.m.p. systems:

Definition 4.1 (Class of failure-free a.m.p. contexts)**.** *$\gamma = (L, \text{Act}, G_0, \tau, \Psi, P_e)$ is a* ***failure-free asynchronous message-passing context,*** *or $\boldsymbol{\gamma \in \Gamma_{\text{amp}}}$, iff all of the following hold:*

$$s_i \in L_i \iff s_i \textit{ is a history over some arbitrary } \Sigma_i, \text{Int}_i, \text{Ext}_i, \textit{ and } \text{Msgs}, \textit{ (Definition 3.5)} \tag{4.1}$$

$$s_e \in L_e \iff s_e = \langle s_{e0},\ \vec{\alpha}_1,\ \vec{\alpha}_2,\ \ldots,\ \vec{\alpha}_m \rangle, \textit{ where } \langle s_{e0}\rangle \in \Sigma_e,\ m \geq 0, \textit{ and } \vec{\alpha}_1 \textit{ through } \vec{\alpha}_m \textit{ are joint actions,} \tag{4.2}$$

$$\alpha_i \in \text{Act}_i \iff \alpha_i \subseteq \{\text{internal}(i, a_{ik}) \mid a_{ik} \in \text{Int}_i\} \cup \{\text{send}(i, j, M) \mid j \in \mathcal{A}, M \in \text{Msgs}\}, \tag{4.3}$$

$$\alpha_e \in \text{Act}_e \iff \alpha_e \subseteq \{\text{go}(i) \mid i \in \mathcal{A}\} \cup \{\text{deliver}(i, j, M) \mid i, j \in \mathcal{A}, M \in \text{Msgs}\} \cup \{\text{trigger}(i, b) \mid i \in \mathcal{A}, b \in \text{Ext}_i\}, \tag{4.4}$$

$$\forall s, s' \in G_0.\ s \neq s' \implies s_e \neq s'_e, \tag{4.5}$$

$$\tau = \hat{\tau}_{\text{amp}}(L), \qquad \textit{(Definition 4.2)} \tag{4.6}$$

$$\Psi = \mathit{Rel} \wedge \mathit{FS}. \tag{4.7}$$

Equation (4.1) requires that the local states of agents are histories; in other words, agents remember all received messages and their past actions. $\hat{\tau}_{\text{amp}}$ specifies that these events are added to the history of an agent as they occur. Equation (4.2) requires that, similar to histories, the states of the environment are a sequence of the initial state s_{e0} of the environment, and the joint global actions taken in the past. $\hat{\tau}_{\text{amp}}$ specifies that the joint action taken is added to the state of the environment for each global state transition.

Equation (4.3) specifies the shape of actions of agents, and Equation (4.4) specifies the shape of actions of the environment. Each action is a set of sub-actions. In Section 4.2, we define how the actions of agents and the environment influence the evolution of the global state through $\hat{\tau}_{\text{amp}}$.

Equation (4.5) requires that the environment can distinguish between all initial global states. Together with $\hat{\tau}_{\text{amp}}$, this makes γ a recording context, as we show in Theorem 4.6. Equation (4.6) specifies that global states evolve according to $\hat{\tau}_{\text{amp}}$, which we define in Section 4.2. Equation (4.7) specifies liveness conditions for γ, namely Eventual Delivery and Fair Schedule, which we define in Section 4.3.

Remark. This definition of Γ_{amp} places very strict requirements on the context γ. A possible generalization [PKS17] is to allow sets of states and the transition relation to be a *subset* of what we currently require. We do not pursue this here for easier reasoning about the contexts in Γ_{amp}.

4.2 Transition Relation

In a context γ, the transition relation τ defines how the global state evolves in response to joint actions. Definition 4.1 requires that failure-free a.m.p. contexts follow a canonical transition relation $\hat{\tau}_{\text{amp}}$, which we define here.

Actions of agents. In Γ_{amp}, the protocol P_i of agent i can propose a set of sub-actions in each step, with the following desired meaning:

- $\text{internal}(i, a)$: Agent i performs internal action a (a local computing step).
- $\text{send}(i, j, M)$: Agent i sends message M to agent j.

This is a subset of the events that can occur on agents per the definition of histories (Definition 3.6). We will specify $\hat{\tau}_{\text{amp}}$ such that the environment can influence whether and how these events actually occur. In this way, actions of agents are a "proposal", a desired action.

Actions of the environment. In Γ_{amp}, the protocol P_e of the environment can specify a set of sub-actions in each step to influence evolution of the global state, with the following desired meaning:

- $\text{go}(i)$: i performs its action α_i (send and internal actions). Conversely, we specify that i performs a stuttering step when $\text{go}(i) \notin \alpha_e$.
- $\text{deliver}(i, j, M)$: j receives message M from i.
- $\text{trigger}(i, b)$: External action b occurs on i.

A single environment action can specify an arbitrary number of sub-actions (including zero) for each agent.

Attention! An individual action α_i/α_e is a *set of sub-actions* (internal, send, go, ...) that occur simultaneously. *Sets of actions*, in turn, can be specified by protocols to model nondeterminism (Definition 3.9).

Recall that $q : x$ denotes the sequence q with the element x appended. s_i denotes the local state of agent i in global state s. We can define $\hat{\tau}_{\text{amp}}$ as follows to get the desired semantics of actions:

Definition 4.2. *Fix the domains of local states $L = (L_e, L_1, \ldots, L_n)$. Then $\hat{\tau}_{\text{amp}}(L)$ is a transition function s.t. for all global states $s, s' \in G$ and joint actions $\vec{\alpha}$,*

$$(\vec{\alpha}, (s, s')) \in \hat{\tau}_{\text{amp}}(L)$$

iff all of the following hold for all $i \in \mathcal{A}$:

$$s'_e = s_e : \vec{\alpha} \tag{4.8}$$

$$s'_i = \begin{cases} s_i & \textit{if } \text{go}(i) \notin \alpha_e \textit{ and } R = \emptyset \\ s_i : R & \textit{if } \text{go}(i) \notin \alpha_e \textit{ and } R \neq \emptyset \\ s_i : (\alpha_i \cup R) & \textit{if } \text{go}(i) \in \alpha_e \end{cases} \tag{4.9}$$

$$\begin{aligned} \textit{where } R = \{&\text{recv}(j, i, M) \mid \text{deliver}(j, i, M) \in \alpha_e\} \cup \\ &\{\text{external}(i, b) \mid \text{trigger}(i, b) \in \alpha_e\}. \end{aligned}$$

$$\forall\, \text{deliver}(i, j, M) \in \alpha_e\colon\ \text{send}(i, j, M) \in E \textit{ for some } E \in s_i. \tag{4.10}$$

Transitions occur between $r(t)$ and $r(t+1)$ as per Definition 3.15. States are labelled s and s', since the *possible* transitions allowed by $\hat{\tau}_{\text{amp}}$ do not necessarily appear in any run r.

Equation (4.8) states that the state of environment keeps a record of all joint actions, making $\hat{\tau}_{\text{amp}}$ a recording context (Theorem 4.6).

Equation (4.9) assigns meaning to actions of the environment. $\text{go}(i)$ determines whether agent i performs its desired action (α_i can contain $\text{internal}(i, \ldots)$ and $\text{send}(i, \ldots)$ events). R are the messages and external events that an agent receives. When the environment specifies $\text{deliver}(j, i, M)$, a receive event is appended to the history of i. Similarly, when the environment specifies $\text{trigger}(i, b)$, an external action is appended to the history of i. In the absence of $\text{go}(i)$ and message deliveries, i does not make progress, and does not even notice the occurrence of a step. In other words, in this case, the agent is oblivious to the passage of time. The agent's internal action, messages, and external events are added to its state simultaneously, which i can only process in its next step.

Equation (4.10) states that only messages that have been sent earlier are delivered.

Throughout this thesis, we consider the protocol $P_e(s_e) := \text{Act}_e$ where the environment chooses any action nondeterministically. Given a global state s, it could specify delivery of a message that was never sent, or other inconsistent behavior. However, a joint action $\vec{\alpha}$ that contains such an action α_e is filtered out by the fact that no state s' exists s.t. $(\vec{\alpha}, (s, s')) \in \hat{\tau}_{\text{amp}}(L)$.

4.3 Liveness Conditions

We require that a.m.p. systems guarantee the liveness conditions Reliable Message Delivery and Fair Schedule. We formalize them as follows:

Definition 4.3. *A run r provides Reliable Message Delivery, $r \in Rel$, iff*

$$\begin{aligned}&\textit{for all } i, j \in \mathcal{A},\ M \in \mathrm{Msgs}, \textit{ and times } t,\\ &\quad \exists\, E.\ \mathrm{send}(i,j,M) \in E \in r_i(t)\\ &\implies \exists\, E', t' \geq t.\ \mathrm{recv}(i,j,M) \in E' \in r_j(t').\end{aligned}$$

Definition 4.4. *A run r provides Fair Schedule, $r \in FS$, iff*

$$\begin{aligned}&\textit{for all } i \in \mathcal{A} \textit{ and times } t,\\ &\exists\, t' \geq t.\ r_e(t') = q : \vec{\alpha} \textit{ and } \mathrm{go}(i) \in \alpha_e.\end{aligned}$$

In Section 5.4, we generalize *Rel* to allow Byzantine failures. In Section 5.7, we translate these liveness conditions to temporal-epistemic logic.

4.4 Recording Context

Fagin et al. [FHMV03] call a context that records all joint actions in the state of the environment a *recording context*. Together with a suitable transition relation τ, this allows P_e to distinguish between all global states. We base our definition directly on this property:

Definition 4.5. *A context γ is a* ***recording context*** *when, for all joint protocols P, runs r, r' consistent with P in γ, and times t, t',*

$$r(t) \neq r'(t') \implies r_e(t) \neq r'_e(t').$$

This captures the intuition that the environment is "omniscient". In particular, the environment can choose a different action for each global state.

Theorem 4.6. *Every context $\gamma \in \Gamma_{\mathrm{amp}}$ is a recording context.*

Proof. Assume $r(t) \neq r'(t')$. We show that $r_e(t) \neq r'_e(t')$.

Case $t \neq t'$: By Equation (4.8), the state of the environment grows by exactly one element in each time step, hence $r(t) \neq r'(t')$.

Case $t = t'$: Proof by induction on t.
Base case: $t = t' = 0$. By Equation (4.5) and our assumption, $r_e(0) \neq r'_e(0)$.
Induction step: Assume the induction hypothesis holds for $t-1$. When $r(t-1) \neq$

$r'(t-1)$, the claim follows by Definition 4.2. When $r(t-1) = r'(t-1)$, assume for contradiction that $r(t) \neq r'(t)$ and the claim does not hold, i.e.,

$$r_e(t) = r'_e(t)$$
$$r_e(t-1) : \vec{\alpha} = r'_e(t-1) : \vec{\alpha}' \qquad \text{by Definition 4.2.}$$

But then

$$r(t) = \tau(\vec{\alpha})(r(t-1)) = \tau(\vec{\alpha}')(r'(t-1)) = r'(t).$$

This contradicts our assumption that $r(t) \neq r'(t)$, hence the claim must hold. □

4.5 Relation to Message-Passing Systems

As [FHMV03] note, we can relate failure-free message-passing contexts to message-passing systems:

Theorem 4.7. *In a failure-free a.m.p. context $\gamma \in \Gamma_{\text{amp}}$, every system $\mathcal{R} = \hat{\mathcal{R}}(P, \gamma)$ is a message-passing system (Definition 3.7).*

Proof sketch. We argue that $\mathcal{R}$ must satisfy (MP1) to (MP3). (MP1) requires that every $r_i(t)$ is a history, which is guaranteed by Equation (4.1).

For (MP2), let $e = \text{recv}(j, i, M)$ be a receive event such that $e \in E \in r_i(t)$. Since initial states never include events and states are histories, E must have been appended to the history at some point. By Equation (4.9), we can find $t' \leq t, \vec{\alpha}$ s.t. $(r(t'-1), r(t')) \in \hat{\tau}_{\text{amp}}(\vec{\alpha})$ and $e \in (\alpha_i \cup R)$. Since e is a recv event and agent actions are never recv events by Equation (4.3), we know that $e \in R$. By the definition of R and Equation (4.10), we can find j, E' s.t. $\text{send}(j, i, M) \in E' \in r_j(t')$. Since $t' \leq t$ and $r_j(t)$ is a history, $\text{send}(j, i, M) \in E' \in r_j(t)$.

(MP3) follows directly from Equations (4.1), (3.1) and (4.9). □

4.6 Lock-Step Synchronous Systems

We have introduced Γ_{amp} as the class of failure-free a.m.p. contexts. Can lock-step synchronous systems be modeled in a similar fashion?

For illustration purposes, consider a simple system where agent i sends a message $\langle\text{ping}\rangle$, to which agent j responds with $\langle\text{pong}\rangle$. In a lock-step synchronous system, when i sends $\langle\text{ping}\rangle$ in round 1, j would see it in round 2 and send $\langle\text{pong}\rangle$ as its response still in round 2. The end-to-end-delay, measured from sending a message to an action depending on it, is 1.

When modeling the same exchange using a failure-free a.m.p. context $\gamma \in \Gamma_{\text{amp}}$, the closest we can get is a run like this, due to the way we defined $\hat{\tau}_{\text{amp}}$:

$$\left.\begin{array}{ll} r_i(0) = \langle s_{i0} \rangle & \alpha_i = \{\text{send}(i, j, \langle \text{ping} \rangle)\} \\ r_j(0) = \langle s_{j0} \rangle & \alpha_j = \emptyset \\ r_e(0) = \langle s_{e0} \rangle & \alpha_e = \{\text{go}(i), \text{go}(j)\} \end{array}\right\} \vec{\alpha}_0$$

$$\left.\begin{array}{ll} r_i(1) = \langle s_{i0}, \{\text{send}(i, j, \langle \text{ping} \rangle)\} \rangle & \alpha_i = \emptyset \\ r_j(1) = \langle s_{j0} \rangle & \alpha_j = \emptyset \\ r_e(1) = \langle s_{e0}, \vec{\alpha}_0 \rangle & \alpha_e = \{\text{go}(i), \text{go}(j), \text{deliver}(i, j, \langle \text{ping} \rangle)\} \end{array}\right\} \vec{\alpha}_1$$

$$\left.\begin{array}{ll} r_i(2) = \langle s_{i0}, \{\text{send}(i, j, \langle \text{ping} \rangle)\} \rangle & \alpha_i = \emptyset \\ r_j(2) = \langle s_{j0}, \{\text{recv}(i, j, \langle \text{ping} \rangle)\} \rangle & \alpha_j = \{\text{send}(j, i, \langle \text{pong} \rangle)\} \\ r_e(2) = \langle s_{e0}, \vec{\alpha}_0, \vec{\alpha}_1 \rangle & \alpha_e = \{\text{go}(i), \text{go}(j)\} \end{array}\right\} \vec{\alpha}_2$$

Here, agent j can only act at $t = 2$ on the message sent at $t = 0$, which means that state transitions in r cannot directly represent lock-step rounds. There are a few ways to resolve this:

- Introduce quiescent ticks for communication, stretching a round over two global states in a run r. This makes the model more complex.
- Modify $\hat{\tau}_{\text{amp}}$, such that the environment can specify "deliver" for messages already before it sees them in s_e. This goes against the idea that the protocol P_e can "willfully influence the evolution of runs, P_e would have to blindly specify delivery of all possible messages at every point in time.
- Modify the definition of contexts to change the role of P_e. The separation between τ and P_e is a bit artificial to begin with; in this thesis, we always use $P_e = \text{Act}_e$ and encode all desired properties in τ. One could eliminate P_e, and replace s_e with a "trace" of the actions that have occurred that is maintained by τ. Or, keeping P_e, P_e could be given "early access" to the actions of agents in the current transition. One could view this as the transition function operating in multiple phases [PKS17].

We do not further investigate lock-step synchronous systems here and leave this aspect up to future work.

CHAPTER 5

Byzantine Asynchronous Message-Passing Contexts

In the analysis of fault-tolerant distributed systems, the *failure assumption* defines which failures an algorithm must tolerate. We will consider the communication links between agents to be a perfect (failure-free), fully-connected point-to-point network. The agents in this network, however, may misbehave in a particularly malicious way, namely according to a *Byzantine* failure assumption [LSP82].

With f Byzantine node failures, up to f faulty agents may transmit arbitrary messages at arbitrary times. In particular, they can send erroneous and contradictory messages to different agents, i.e., a combination of messages that a correct agent would never send in a run of the system. This means that the apparent behavior of Byzantine-faulty agents need not correspond to their actual local state, or in fact any legal state at all. This is in contrast to [Mic89].

With such a permissive failure assumption, the idea is to capture a very broad range of real-world scenarios. A more restricted failure assumption, such as the assumption of crash failures, may not always be justified in a real-world scenario. Consequently, Attiya and Welch write, “If a system designer is not sure exactly how errors will be manifested, a conservative assumption is that they will be Byzantine” [AW04, p. 123].

In Chapter 4, we introduced the class Γ_{amp} of failure-free a.m.p. contexts. We extend this to allow Byzantine node failures, and get the class Γ_{bamp} of Byzantine a.m.p. contexts.

In Sections 3.10 and 3.11 we introduced a language $\mathcal{L}$ and a corresponding semantics to make statements about runs. In this chapter, we describe properties of Γ_{bamp} using temporal-epistemic logic, and derive a result about knowledge gain in Byzantine a.m.p. contexts.

In Chapter 6, we will then use properties of Γ_{bamp} to analyze necessary and sufficient knowledge in *Firing Rebels*.

5.1 Knowledge of Byzantine-Faulty Agents

What does it mean to say "agent i knows φ" ($K_i\,\varphi$) when i is Byzantine-faulty?

A naive attempt at answering this question might go like this: At a given point (r,t), a Byzantine-faulty agent i can pretend that $K_i\,\varphi$, $K_i\,\neg\varphi$, and $\neg\,K_i\,\varphi$ all hold at the same time (in communication with different neighbors). But does this mean that all of these formulas hold? Byzantine agents are also allowed to have "targeted" behavior. But does this mean that agent i is omniscient, that is, $\varphi \to K_i\,\varphi$? Or does it mean that K_i cannot be defined in a meaningful way at all?

One resolution of this apparent paradox is to make the knowledge of an agent conditional on its correctness. For example, [HMW01] employ a logic of *belief*[2] for crash failures, where many statements are of the form $K_i\,(\text{correct}_i \to \varphi)$, meaning "$i$ knows that when it is correct, φ holds." Interestingly, this means that the truth value of this statement may change when i becomes faulty, which seems unintuitive at first. We will later see a formal definition of the atom correct_i.

A priori, we would like an intuition also for statements that are not of the form $K_i\,(\text{correct}_i \to \varphi)$. But what should $K_i\,\varphi$ mean when i is Byzantine-faulty?

Simulation with Byzantine-faulty links. The thought model employed in this thesis is based the idea that a system with Byzantine-faulty agents can be transformed by a simulation as illustrated in Figure 5.1. Under this simulation, we treat Byzantine-faulty agents *as if they were correct agents*, but with Byzantine-faulty outgoing links.

Observe that the two situations are indistinguishable to all other agents in the system, because messages over the outgoing links are the only thing they can observe. But in the scenario depicted in Figure 5.1b, knowledge of agent i has a well-defined intuitive meaning: $K_i\,\varphi$ holds if agent i has sufficient evidence for φ. In other words, a correct agent in the place of i can conclude from its local history that φ must hold.

We thus propose that when dealing with a Byzantine-faulty agent i like in 5.1a, we will ascribe the same knowledge to it as to the agent in 5.1b.

Examples. What are the consequences of defining knowledge of Byzantine-faulty agents in this way? Consider a system of four agents $\mathcal{A} = \{i,j,k,l\}$, where a single agent may become Byzantine-faulty during any run ($f = 1$). Assume that in the system $\mathcal{R}$, j, k and l run the following protocol: when they receive a message, they echo the message back to the sender immediately. Further assume that $\text{Msgs} = \{1,2,3\}$ are all valid messages that i could send.

[2] *Belief* is a technical term here, and as introduced in [HMW01] its semantics can be quite different from colloquial "belief".

Figure 5.1: Simulation of a Byzantine-faulty agent i by a correct agent with Byzantine-faulty outgoing links. (a) The agent as a whole is considered faulty, with an a priori unclear meaning of knowledge of i. (b) We pretend that i is a correct agent, but its outgoing links behave in a Byzantine-faulty way.

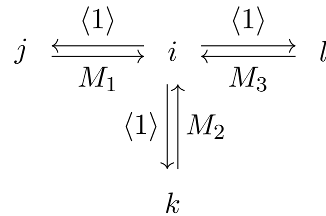

Figure 5.2: A system of $n = 4$ agents, with j, k, l as *echo* nodes and up to $f = 1$ Byzantine node failures. Edges indicate communication as observed by agent i. In different runs, agent i can observe different values depending on whether any of the agents are faulty.

	r_1	r_2	r_3
What i sends to j, k, l:	$\langle 1 \rangle$	$\langle 1 \rangle$	$\langle 1 \rangle$
What i receives from j (M_1):	$\langle 1 \rangle$	$\langle 1 \rangle$	$\langle 1 \rangle$
What i receives from k (M_2):	$\langle 1 \rangle$	$\langle 2 \rangle$	$\langle 1 \rangle$
What i receives from l (M_3):	$\langle 1 \rangle$	$\langle 3 \rangle$	$\langle 3 \rangle$
Does i know that i is correct?	no	no	no
Does i know that i is faulty?	no	yes	no

Table 5.1: Three runs of the system in Figure 5.2, from the perspective of agent i. First, i sends 1 to all neighbors. Then, i receives a reply from them. Based on the replies, can i conclude (with $f = 1$) that its own outgoing links are behaving in a faulty way?

We consider three different runs as listed in Table 5.1, where i sends 1 to its neighbors. The communication in these runs is illustrated in Figure 5.2. We observe: In run r_1, all agents appear to be correct. Since a Byzantine-faulty agent may pretend to be correct for an arbitrary amount of time, i cannot infer anything about its own correctness. In r_2, k and l send inconsistent messages. But $f = 1$, thus in fact the messages that k and l received must already have been corrupted. Hence i can infer that i must be faulty. In r_3, agent i can infer that i or l are faulty, but not which of the two.

This example illustrates that with our proposed thought model, agents can know that they are faulty. Of course, this is a purely technical artifact: when they are faulty, they cannot reliably communicate this "knowledge" to other agents. In some runs, agents might also learn that they are correct, by cooperating to identify f faulty agents (but they cannot rely on f agents showing faulty behavior in other runs).

We define Γ_{bamp} with this model in mind in the next section.

Remarks. The idea of treating Byzantine-faulty agents as if they were correct with Byzantine-faulty outgoing links was originally inspired by the concept of *honest announcements* in Pucella and Sadrzadeh [PS10]. Honest announcements are in turn based on earlier concepts of *honesty* [HF85, PR03]. Whenever an agent makes a honest announcement of φ, it is not only required that φ is true, but also that the sender *knows* it is true. In other words, with honest announcements, a sender cannot make announcements that are true merely "by accident".

In our thought model, the knowledge of Byzantine-faulty agents evolves just like that of correct agents. The crucial difference is that Byzantine-faulty agents are not required to be *honest*: they may send arbitrary messages, in particular even ones that are not backed by their local knowledge. Note that if we allow Byzantine-faulty agents (resp. their outgoing links) to violate message timing assumptions, they are actually more powerful than mere dishonest announcements.

The observation that state corruption (faultiness of an agent) and faultiness of links is indistinguishable to neighboring nodes has also been made in [BCG+07, Section 5.2] and [Lam01].

5.2 The Class Γ_{bamp}

In Chapter 4, we introduced the class Γ_{amp}. We extend this definition to allow Byzantine node failures as follows:

Definition 5.1 (Class of Byzantine a.m.p. contexts)**.** *$\gamma = (L, \text{Act}, G_0, \tau, \Psi, P_e)$ is a* ***Byzantine asynchronous message-passing context*** *with f node failures, or* $\boldsymbol{\gamma \in \Gamma_{\text{bamp}}(f)}$*, iff all of the following hold:*

$$s_i \in L_i \iff s_i \textit{ is a history over some arbitrary } \Sigma_i, \text{Int}_i, \text{Ext}_i, \textit{ and } \text{Msgs}, \textit{ (Definition 3.5)} \tag{5.1}$$

$$s_e \in L_e \iff s_e = \langle s_{e0},\ \vec{\alpha}_1,\ \vec{\alpha}_2,\ \ldots,\ \vec{\alpha}_m \rangle, \textit{ where } \langle s_{e0} \rangle \in \Sigma_e,\ m \geq 0, \textit{ and } \vec{\alpha}_1 \textit{ through } \vec{\alpha}_m \textit{ are joint actions,} \tag{5.2}$$

$$\alpha_i \in \text{Act}_i \iff \alpha_i \subseteq \{\text{internal}(i, a_{ik}) \mid a_{ik} \in \text{Int}_i\} \cup \{\text{send}(i, j, M) \mid j \in \mathcal{A}, M \in \text{Msgs}\}, \tag{5.3}$$

$$\alpha_e \in \text{Act}_e \iff \alpha_e \subseteq \{\text{go}(i) \mid i \in \mathcal{A}\} \cup \{\text{deliver}(i, j, M) \mid i, j \in \mathcal{A}, M \in \text{Msgs}\} \cup \{\text{trigger}(i, b) \mid i \in \mathcal{A}, b \in \text{Ext}_i\} \cup \{\text{fail}(i) \mid i \in \mathcal{A}\}, \tag{5.4}$$

$$\forall s, s' \in G_0.\ s \neq s' \implies s_e \neq s'_e, \tag{5.5}$$

$$\tau = \hat{\tau}_{\text{bamp}}(L, f), \qquad \textit{(Definition 5.3)} \tag{5.6}$$

$$\Psi = \textit{EDel} \wedge \textit{FS}. \tag{5.7}$$

This mirrors Definition 4.1 (Γ_{amp}), which explains the equations in detail. The differences are: Γ_{bamp} is parameterized by the number f of Byzantine node failures allowed. Equation (5.4) allows an action fail(i) for the environment. Equation (5.6) specifies a transition relation which deviates from $\hat{\tau}_{\text{amp}}$ and introduces Byzantine behavior, see Section 5.3. Equation (5.7) specifies *EDel* rather than *Rel*, see Section 5.4.

5.3 Transition Relation

We introduce $\hat{\tau}_{\text{bamp}}$, the transition relation for Γ_{bamp}, mirroring $\hat{\tau}_{\text{amp}}$ with some adjustments. Refer to Section 4.2 for a detailed description of actions and their interaction with the transition relation.

We introduce a new action $\text{fail}(i)$, other actions are identical to Section 4.2:

Actions of agents. In Γ_{bamp}, agents can propose the following sub-actions:

- $\text{internal}(i, a)$: Agent i performs internal action a (a local computing step).
- $\text{send}(i, j, M)$: Agent i sends message M to agent j.

Actions of the environment. In Γ_{bamp}, the environment can specify the following sub-actions:

- $\text{go}(i)$: i performs its action α_i (send and internal actions). Conversely, we specify that i performs a stuttering step when $\text{go}(i) \notin \alpha_e$.
- $\text{deliver}(i, j, M)$: j receives message M from i.
- $\text{trigger}(i, b)$: External action b occurs on i.
- $\text{fail}(i)$: i is marked Byzantine-faulty.

A single environment action can specify an arbitrary number of sub-actions (including zero) for each agent.

To model Byzantine behavior, $\hat{\tau}_{\text{bamp}}$ will allow the environment to deliver arbitrary messages, i.e., even messages that have never been sent. To simplify later arguments, the environment must explicitly mark an agent i faulty when it imposes such behavior in Γ_{bamp}.

We introduce $\text{Failed}(s)$ as the set of faulty agents in global state s:

Definition 5.2. *Fix a global state $s \in G$. Then*

$$\text{Failed}(s) := \{i \in \mathcal{A} \mid \exists\, \vec{\alpha} \in s_e.\ \text{fail}(i) \in \alpha_e\}.$$

We will define $\hat{\tau}_{\text{bamp}}$ such that s_e has the full history of joint actions, thus $\text{Failed}(r(t))$ includes all agents marked as faulty in the past in r. Implicitly, $\text{Failed}(s) = \emptyset$ for all initial states $s \in G_0$.

We define the transition function such that the environment can fake messages from agents marked as faulty. We do not allow more than f agents to be marked as faulty.[3]

Definition 5.3. *Fix domains of local states $L = (L_e, L_1, \ldots, L_n)$ and the number of Byzantine node failures f. Then $\hat{\tau}_{\text{bamp}}(L, f)$ is a transition function s.t. for all global states $s, s' \in G$ and joint actions $\vec{\alpha}$,*

$$(\vec{\alpha}, (s, s')) \in \hat{\tau}_{\text{bamp}}(L, f)$$

iff all of the following hold for all $i \in \mathcal{A}$:

$$s'_e = s_e : \vec{\alpha} \tag{5.8}$$

$$s'_i = \begin{cases} s_i & \text{if } \mathrm{go}(i) \notin \alpha_e \text{ and } R = \emptyset \\ s_i : R & \text{if } \mathrm{go}(i) \notin \alpha_e \text{ and } R \neq \emptyset \\ s_i : (\alpha_i \cup R) & \text{if } \mathrm{go}(i) \in \alpha_e \end{cases} \tag{5.9}$$

$$\text{where } R = \{\mathrm{recv}(j, i, M) \mid \mathrm{deliver}(j, i, M) \in \alpha_e\} \cup \{\mathrm{external}(i, b) \mid \mathrm{trigger}(i, b) \in \alpha_e\}.$$

$$\forall\, \mathrm{deliver}(i, j, M) \in \alpha_e : \quad i \notin \mathrm{Failed}(s) \implies \mathrm{send}(i, j, M) \in E \text{ for some } E \in s_i. \tag{5.10}$$

$$|\mathrm{Failed}(s')| \leq f \tag{5.11}$$

$$\mathrm{fail}(i) \in \alpha_e \implies i \notin \mathrm{Failed}(s) \tag{5.12}$$

Identical to Definition 4.2, transitions occur between $r(t)$ and $r(t+1)$ as per Definition 3.15. States are labelled s and s', since the *possible* transitions allowed by $\hat{\tau}_{\text{bamp}}$ do not necessarily appear in any run r.

Identical to Definition 4.2, Equation (5.8) states that the state of environment keeps a record of all joint actions, making $\hat{\tau}_{\text{bamp}}$ a recording context (proof omitted, similar to Theorem 4.6).

Identical to Definition 4.2, Equation (5.9) assigns meaning to actions of the environment. $\mathrm{go}(i)$ determines whether agent i performs its desired action (α_i can contain $\mathrm{internal}(i, \ldots)$ and $\mathrm{send}(i, \ldots)$ events). R are the messages and external events that an agent receives. When the environment specifies $\mathrm{deliver}(j, i, M)$, a receive event is appended to the history of i. Similarly, when the environment specifies $\mathrm{trigger}(i, b)$, an external action is appended to the history of i. In the absence of $\mathrm{go}(i)$ and message deliveries, i does not make progress, and does not even notice the occurrence of a step. In other words, in this case, the agent is oblivious to the passage of time. The agent's internal action, messages, and external events are added to its state simultaneously, which i can only process in its next step.

[3] The restriction to f faulty agents could also be enforced in the protocol of the environment P_e. We chose to capture this property in τ and leave P_e unconstrained.

Equation (5.10) states that any message received from a correct agent i was actually sent by agent i, thus providing causality and intactness for correct agents. Conversely, arbitrary behavior can take place on the outgoing links of Byzantine-faulty nodes. Equation (5.11) states that only up to f agents are marked faulty (initially, no agent is faulty). Equation (5.12) states that an agent can only be marked faulty once.

In Γ_{bamp}, we assume $P_e(r(t)) := \text{Act}_e$ throughout this thesis.

Remarks. With this transition function, Byzantine failures correspond to a one-time $\text{fail}(i)$ action of the environment, only after which the environment is allowed to forge and corrupt messages appearing to come from i at will. Why did we choose this approach? A different possibility would be "implicit" signaling of failures, where an agent is regarded as faulty as soon as the environment specifies actions that are not normally allowed. One could also require that the environment yield $\text{fail}(i)$ in every step throughout the rest of the run for faulty agents.

Our approach of explicitly flagging failed agents makes it easier to argue about statements like "all correct agents perform action X", since an omniscient observer can distinguish whether agent i is correct or only acting like it is correct at any given time. The fact that we allow regular $\text{go}(i), \text{deliver}(j, i, M)$ actions after $\text{fail}(i)$ corresponds to our intuition of evolving the local state like for correct agents. This detail also allows us to state some liveness conditions without any special treatment of faulty agents, as we will see in Section 5.4.

5.4 Liveness Conditions

We motivated liveness conditions in Section 3.7 and introduced *Rel* (reliable delivery) and *FS* (fair schedule) in Section 4.3.

We apply *FS* in Γ_{bamp} without modification, i.e., we require infinitely many $\text{go}(i)$ actions from the environment. This includes faulty agents.

We cannot directly apply *Rel* since it prohibits faked messages. We modify it to accommodate Byzantine behavior, and choose the term *eventual delivery* (*EDel*) rather than "reliable" delivery for this generalization.

Definition 5.4 (Eventual message delivery)**.** *A run r provides Eventual Delivery, $r \in EDel$, iff*

$$\begin{aligned} &\textit{for all } i, j \in \mathcal{A},\ M \in \text{Msgs}, \textit{ and times } t, \\ &\qquad \exists\, E.\ \text{send}(i, j, M) \in E \in r_i(t) \textit{ and } i \notin \text{Failed}(r(t)) \\ &\quad \Longrightarrow \exists\, E', t' \geq t.\ \text{recv}(i, j, M) \in E' \in r_j(t'). \end{aligned}$$

With our definition, when agent i is marked faulty at time t, messages already in transit from i cannot be lost or corrupted. This is not a problem in Γ_{bamp}, as the adversary could always mark agent i as faulty at an earlier time.

5.5 Recording Context

Theorem 5.5. *Every context $\gamma \in \Gamma_{\text{bamp}}$ is a recording context.*

We introduced recording contexts in Section 4.4, and the proof is analogous to Theorem 4.6.

5.6 Non-Excluding Context

We introduced non-excluding contexts in Section 3.9.

Theorem 5.6. *Every context $\gamma \in \Gamma_{\text{bamp}}$ is a non-excluding context.*

Proof sketch. Intuitively, $\hat{\tau}_{\text{bamp}}$ is specified in such a way that starting from any state $r(t)$, the environment can always choose actions such that r satisfies *EDel* and *FS*. □

5.7 Properties in Temporal-Epistemic Logic

We introduced a language $\mathcal{L}$ of temporal-epistemic formulas and corresponding semantics over runs in Sections 3.10 and 3.11.

In this section, we introduce "special" atoms that allow us to reason about runs in a context $\gamma \in \Gamma_{\text{bamp}}$ using temporal-epistemic logic. In particular, we want to be able to express properties like Eventual Delivery in $\mathcal{L}$. Rather than extending $\mathcal{L}$, we introduce atomic propositions with a predefined meaning. In other words, in the remainder of this thesis, we restrict interpretations π to ones that assign the desired meaning to these predefined atoms.

[FHMV03] describe *stable* formulas, which we extend by the notion of 0-stable formulas:

Definition 5.7. *A formula φ is **1-stable** (stable) in a system $\mathcal{R}$ when*

$$\mathcal{R} \models \varphi \to \Box\varphi.$$

*A formula φ is **0-stable** when $\neg\varphi$ is 1-stable.*

Predefined atoms. To reason about runs of a Byzantine a.m.p. context $\gamma \in \Gamma_{\text{bamp}}$ in $\mathcal{L}$, we introduce a number of predefined atomic propositions. Their intuitive meaning at time t in a run r is as follows:

- correct_i: i is a correct agent at time t (may become faulty later).
- $\text{msg-sent}(i, j, M)$: i has sent M to j.
- $\text{msg-recvd}(i, j, M)$: j has received M from i.
- $\text{ext-occurred}(i, b)$: i has observed external action b.
- $\text{env-action}(a)$: The most recent action of the environment includes a.

Here, msg-sent, msg-recvd and ext-occurred refer to the current and all past states of r, hence they are 1-stable. We use a function-like notation for convenience and readability, but syntactically, all objects of the form $\text{msg-sent}(i, j, M)$ are individual atomic propositions. To give atoms their desired meaning, we require the following for all interpretation functions π considered in contexts $\gamma \in \Gamma_{\text{bamp}}$:

Definition 5.8 (Predefined Atoms)**.** *Let $\mathcal{R}$ be a system consistent with a protocol P in a Byzantine a.m.p. context $\gamma \in \Gamma_{\text{bamp}}$. Then an interpretation π must satisfy:*

$$
\begin{aligned}
(\mathcal{R}, r, t) &\models \text{correct}_i && \textit{iff} \quad i \notin \text{Failed}(r(t)) \\
(\mathcal{R}, r, t) &\models \text{msg-sent}(i, j, M) && \textit{iff} \quad \exists E.\ \text{send}(i, j, M) \in E \in r_i(t) \\
(\mathcal{R}, r, t) &\models \text{msg-recvd}(i, j, M) && \textit{iff} \quad \exists E.\ \text{recv}(i, j, M) \in E \in r_j(t) \\
(\mathcal{R}, r, t) &\models \text{ext-occurred}(i, b) && \textit{iff} \quad \exists E.\ \text{external}(i, b) \in E \in r_j(t) \\
(\mathcal{R}, r, t) &\models \text{env-action}(a) && \textit{iff} \quad r_e(t) = q : \vec{\alpha} \textit{ and } a \in \alpha_e.
\end{aligned}
$$

This allows us to state properties of Γ_{bamp} using temporal-epistemic logic:

Theorem 5.9. *Consider a run r consistent with a protocol P in $\gamma \in \Gamma_{\text{bamp}}$. Then for all $i, j \in \mathcal{A}$, $M \in \text{Msgs}$,*

$$r \models \neg\,\text{correct}_i \to \Box\,\neg\,\text{correct}_i \tag{5.13}$$
$$r \models \text{msg-sent}(i, j, M) \to \Box\,\text{msg-sent}(i, j, M) \tag{5.14}$$
$$r \models \text{msg-recvd}(i, j, M) \to \Box\,\text{msg-recvd}(i, j, M) \tag{5.15}$$

$$r \models \text{msg-sent}(i, j, M) \to K_i\,\text{msg-sent}(i, j, M) \tag{5.16}$$
$$r \models \text{msg-recvd}(i, j, M) \to K_j\,\text{msg-recvd}(i, j, M) \tag{5.17}$$

$$r \models\ |\{i \in \mathcal{A} \mid \neg\,\text{correct}_i\}|\ \le f \tag{5.18}$$
$$r \models \text{correct}_i \to (\text{msg-recvd}(i, j, M) \to \text{msg-sent}(i, j, M)) \tag{5.19}$$

$$r \models \text{correct}_i \to (\text{msg-sent}(i, j, M) \to \Diamond\,\text{msg-recvd}(i, j, M)). \tag{5.20}$$
$$r \models \Diamond\,\text{env-action}(\text{go}(i)). \tag{5.21}$$

Proof sketch. Claim (5.13) follows from Definition 5.8 and Equation (5.8). Claims (5.14)–(5.15) follow from Definition 5.8 and Equation (5.9). Claims (5.16)–(5.17) hold because the atoms are defined using the local state of i, hence agent i always knows their truth value. Claim (5.18) follows from Equation (5.11). Claim (5.19) follows from Equation (5.10). Claims (5.20)–(5.21) follow from the liveness conditions *EDel* and *FS*. ☐

5.8 Knowledge Gain

Ben-Zvi and Moses [BM14] introduce *syncausality* to describe how agents can gain knowledge about external actions in a failure-free system with time bounds on message delivery. Syncausality provides a foundation for *centipedes* [BM14] that describe necessary communication structures for certain actions in such a system.

In a similar spirit, we prove that in order to gain knowledge in any Byzantine a.m.p. context $\gamma \in \Gamma_{\text{bamp}}$, an agent needs to have received messages from at least $f+1$ agents. This is a basic result on necessary communication structures for knowledge gain in Byzantine a.m.p. systems.

Our formal proof uses the following construction of indistinguishable runs for agent i in states where i has received messages from less than f other agents:

Lemma 5.10. *Fix a run r consistent with P in $\gamma \in \Gamma_{\text{bamp}}$. Fix a time t and an agent $i \in \mathcal{A}$. Assume a set of agents S exists s.t.*

$$|S| \leq f, \text{ and} \tag{5.22}$$

$$S \supseteq \{j \in \mathcal{A} \mid \text{recv}(j, i, M) \in E \in r_i(t) \text{ and } j \neq i\}. \tag{5.23}$$

Then we can find a run r' consistent with P, and a time t', s.t.

$$r'_i(t') = r_i(t) \tag{5.24}$$

$$r'_j(t') = r_j(0) \quad \text{for all } j \in \mathcal{A},\ j \neq i \tag{5.25}$$

$$Failed(r'(t')) = S. \tag{5.26}$$

Equations (5.22) and (5.23) state that agent i has received messages from at most f other agents. Equations (5.25) and (5.26) state that in the constructed run r' at time t', all agents other than i are in their initial states, and the Byzantine-faulty agents are exactly the agents in S. It follows that i is correct, and no agent other than i sent any messages or observed any external actions. Equation (5.24) states that agent i cannot distinguish between $r(t)$ and $r'(t')$.

Proof. By Theorem 5.6, γ is a non-excluding context, thus we can extend any weakly consistent run to a strongly consistent run. We show the existence of a weakly consistent run with properties (5.24)–(5.26) by induction on t.

Base case: $t = 0$. Then $r_i(t) = r_i(0) = \langle s_{i0} \rangle$. Choose r' such that $r'(0) = r(0)$ and $r'_e(1) = r_e(0) : \{\text{fail}(j) \mid j \in S\}$, and r' weakly consistent with P in γ. r' exists by $r(0) \in G_0$ and Equation (5.9). Then $r'(1)$ satisfies (5.24)–(5.26).

Induction step: Assume the hypothesis holds for $r(t-1)$. Distinguish two cases. Case 1: $r_i(t) = r_i(t-1)$. The claim follows trivially from the induction hypothesis. Case 2: $r_i(t) = r_i(t-1) : E$. In other words, one set of events E was appended to $r_i(t)$. By the induction hypothesis and substituting $t' = t'' - 1$, we can find r' and t'' s.t.

$$\begin{aligned} r'_i(t''-1) &= r_i(t-1) \\ r'_j(t''-1) &= r_j(0) \quad \text{for all } j \in \mathcal{A},\ j \neq i \\ \mathit{Failed}(r'(t''-1)) &= S. \end{aligned} \tag{5.27}$$

E are the events added from $r_i(t-1)$ to $r_i(t)$. Partition E by event type s.t.

$$E = E_{\text{internal}} \cup E_{\text{send}} \cup E_{\text{recv}} \cup E_{\text{external}}.$$

By $\hat{\tau}_{\text{bamp}}$, r being consistent, (5.27), and (5.23) for $r(t)$ and S, these subsets must fulfill

$$E_{\text{internal}} \cup E_{\text{send}} = \begin{cases} \emptyset & \text{or} \\ \alpha_i, & \text{where } \alpha_i \in P_i(r'_i(t''-1)) = P_i(r_i(t)) \end{cases} \tag{5.28}$$

$$E_{\text{recv}} \subseteq \{\text{recv}(j, i, M) \mid j = i \text{ or } j \in \text{Failed}(r'(t''-1))\} \tag{5.29}$$

The two cases of Equation (5.28) correspond to the absence or presence of go(i) in the previous action of the environment respectively.

We now construct an action of the environment such that only i makes a step, but the resulting run is identical to $r(t)$ for i. Choose $\alpha_e = \alpha_{e1} \cup \alpha_{e2} \cup \alpha_{e3}$ in accordance with E s.t.

$$\begin{aligned} \alpha_{e1} &= \begin{cases} \emptyset & \text{if } E_{\text{internal}} \cup E_{\text{send}} = \emptyset \\ \{\text{go}(i)\} & \text{otherwise} \end{cases} \\ \alpha_{e2} &= \{\text{deliver}(j, i, M) \mid \text{recv}(j, i, M) \in E_{\text{recv}}\} \\ \alpha_{e3} &= \{\text{trigger}(i, b) \mid \text{external}(i, b) \in E_{\text{external}}\} \end{aligned}$$

By (5.28), (5.29), $r_i(t''-1) = r'_i(t-1)$ and $\hat{\tau}_{\text{bamp}}$, there is a run r'' s.t.

$$r''(t''-1) = r'(t''-1) \text{ and}$$
$$(r''(t''-1), r''(t'')) \in \tau(\alpha_e),$$

that is, we can extend $r'(t''-1)$ by applying α_e to get $r''(t'')$. By choice of α_e and $\hat{\tau}_{\text{bamp}}$,

$$r''_i(t'') = r'_i(t''-1) : E = r_i(t-1) : E = r_i(t),$$

and $r''(t'')$ satisfies (5.24)–(5.26). □

With Lemma 5.10, we can prove the following theorem:

Theorem 5.11. *Let $\mathcal{R} = \hat{\mathcal{R}}(P, \gamma)$ with $\gamma \in \Gamma_{\text{bamp}}$. Fix a run $r \in \mathcal{R}$, time t, agent $i \in \mathcal{A}$ and formula $\varphi \in \mathcal{L}$. Assume that*

$$(\mathcal{R}, r, t) \models \text{correct}_i \wedge K_i\,\varphi, \text{ and} \tag{5.30}$$

$$\text{for all } r' \in \mathcal{R}, t' \text{ with } \begin{cases} r'_i(t) = r_i(t) & \text{and} \\ r'_j(t) = r_j(0) & \text{for all } j \in \mathcal{A}, j \neq i, \end{cases}$$
$$\text{it holds that } (\mathcal{R}, r', t') \models \text{correct}_i \rightarrow \neg\varphi. \tag{5.31}$$

Then

$$(\mathcal{R}, r, t) \models |\{j \in \mathcal{A} \mid \text{msg-recvd}(j, i, M) \text{ and } j \neq i\}| \geq f + 1. \tag{5.32}$$

Equation (5.30) states that agent i is correct and knows φ in $r(t)$. Equation (5.31) states that φ never holds when all other agents are in their initial states, in other words, φ is some formula that depends on the state of other agents.

When these assumptions hold, Equation (5.32) guarantees that agent i has received messages from at least $f+1$ distinct agents in $r(t)$.

Proof. For the sake of contradiction, assume that for some $r \in \mathcal{R}$, t and $i \in \mathcal{A}$,

(1) $(\mathcal{R}, r, t) \models \text{correct}_i$

(2) $(\mathcal{R}, r, t) \models K_i\,\varphi$

(3) $(\mathcal{R}, r, t) \not\models |\{j \in \mathcal{A} \mid \text{msg-recvd}(j, i, M) \text{ and } j \neq i\}| \geq f + 1.$

By (2), for all $r'(t')$ s.t. $r'_i(t') = r_i(t)$,

(4) $(\mathcal{R}, r', t') \models \varphi.$

Let S be the set of agents that sent messages to i in $r(t)$, i.e.,

$$\begin{aligned} S &= \{j \in \mathcal{A} \mid (\mathcal{R}, r, t) \models \text{msg-recvd}(j, i, M) \text{ and } j \neq i\} \\ &= \{j \in \mathcal{A} \mid \text{recv}(j, i, M) \in E \in r_i(t) \text{ and } j \neq i\}. \end{aligned}$$

By (3), $|S| \leq f$. By Lemma 5.10, we can find $r' \in \mathcal{R}$ and t' such that

(5) $\quad r'_i(t') = r_i(t)$

(6) $\quad r'_j(t') = r_j(0) \quad \text{for all } j \neq i$

(7) $\quad i \notin \text{Failed}(r'(t'))$.

By (5) and (6), Equation (5.31) holds for $r'(t')$. Thus

(8) $\quad (\mathcal{R}, r', t') \models \text{correct}_i \to \neg\varphi \qquad$ by (5.31)

(9) $\quad (\mathcal{R}, r', t') \models \text{correct}_i \qquad$ by (7)

(10) $\quad (\mathcal{R}, r', t') \models \neg\varphi \qquad$ by (8) and (9).

But this contradicts (4), since $r'_i(t') = r_i(t)$ by (5). Hence, the claim holds. $\square$

CHAPTER 6

Temporal-Epistemic Analysis of Clock Synchronization

In Chapter 3, we described runs and systems [FHMV03] and a semantics to evaluate formulas in temporal-epistemic logic over runs. In Chapter 5, we introduced Byzantine asynchronous message-passing contexts, a formalization of some of their properties in temporal-epistemic logic, and proved Theorem 5.11 as a basic result on *knowledge gain* in Byzantine a.m.p. contexts. In this chapter, we discuss how Clock Synchronization in Byzantine a.m.p. systems can be analyzed using temporal-epistemic Logic.

Clock synchronization is a popular problem in distributed systems research. We discuss clock synchronization in general and a clock synchronization algorithm by Srikanth and Toueg [ST87] that tolerates Byzantine failures, in a variant by Widder and Schmid [WS09]. The core of this algorithm provides *tick generation*, a problem closely related to clock synchronization. Tick generation primitives can be used in Systems-on-Chip to provide synchronized local clocks for different functional units [FS12], for example.

We relate tick generation to Firing Squad [BL87], and introduce two variants of the problem, *Firing Rebels with/without Relay,* that share some properties with both Firing Squad and tick generation. Firing Rebels is essentially a simplified, time-free variant of one round of tick generation, which in turn can serve as a basis for clock synchronization.

In the spirit of *centipedes* [BM14], we look for necessary communication structures in *Firing Rebels without Relay*, based on Theorem 5.11. In particular, we show that an agent in Firing Rebels without Relay needs to establish specific knowledge before performing its action, giving a lower bound on knowledge. We further show that this knowledge is actually sufficient to act on, giving an upper bound on knowledge.

While Firing Rebels without Relay is a deliberately simplified problem, we hope

that our results can be eventually be extended to the larger problem of Clock Synchronization, and can serve as a starting point for others looking to apply epistemic logic to distributed systems with Byzantine failures.

6.1 Clocks and Ensembles

Based on the treatment of clock synchronization in Attiya and Welch [AW04, Ch. 13] and Kopetz [Kop11], we establish some basic terms and definitions. Following [AW04] and [DFP+14], we will consider idealized, continuous clocks for a high-level introduction, even though clocks in a real-world computing system output values from a discrete domain.

Definition 6.1. *A* ***clock*** C_i *is a function* $C_i(t) : \mathbb{R} \mapsto \mathbb{R}$. $C_i(t)$ *is the* ***clock reading*** *at real time* t. *Typically, we consider* $t \geq 0$. *A* ***clock ensemble*** $\{C_1, \ldots, C_n\}$ *is a set of related clocks.*

For example, if every agent i has its own clock C_i available, the clocks C_i of all agents together form a clock ensemble.

Due to physical limitations, hardware clocks in any real system *drift* over time. A simple model is that clocks stay within a *linear envelope* of real time [AW04]:

Definition 6.2. *A clock* C_i *has* ***drift*** ρ *when it satisfies, for all* t, t' *s.t.* $t' \geq t$*:*

$$\frac{1}{1+\rho}(t'-t) \quad \leq \quad C_i(t') - C_i(t) \quad \leq \quad (1+\rho)(t'-t).$$

This is illustrated in Figure 6.1.

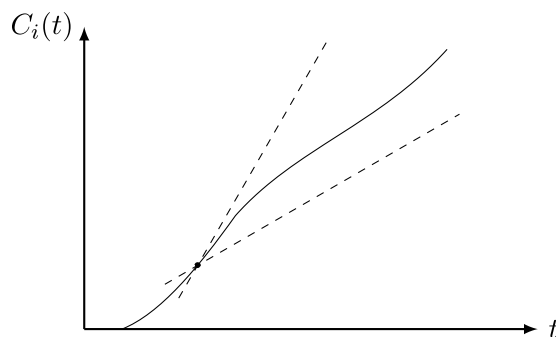

Figure 6.1: Bounded Drift. Starting from any point in time t, the clock C_i may not range outside the linear envelope defined by lines of slope $(1+\rho, \frac{1}{1+\rho})$.

The imperfections of clock ensembles in real systems can be roughly described using two characteristics [Kop11]:

- **Accuracy** describes how much a clocks differs from a reference clock, or from real time. An ensemble has accuracy α when all clocks satisfy $|C_i(t) - t| \leq \alpha$.
- **Precision** describes how much clocks differ from each other within an ensemble. An ensemble has precision π when all pairs of clocks satisfy $|C_i(t) - C_j(t)| \leq \pi$.

Hardware clocks are imperfect and might drift away from each other over time. A clock synchronization algorithm can provide *adjusted clocks* with stronger precision and/or accuracy guarantees. Assuming a communication network between agents that provides certain guarantees on end-to-end delay, agents can exchange clock values and accelerate or slow down their adjusted clocks to improve these properties.

Internal and external synchronization [Kop11]. The desire for accuracy and precision translates to two sub-problems of clock synchronization. *External clock synchronization* keeps adjusted clocks close to a reference clock, addressing accuracy (and to some extent, precision). In real-world systems, a reference clock could take the form of a GPS receiver or a networked time server external to the system.

Internal clock synchronization keeps the clocks of an ensemble close together, addressing precision. Depending on the application, agents in a distributed system might find an offset from real time acceptable as long as all the clocks of all agents are still in approximate agreement with each other.

In the remainder of this thesis, we deal exclusively with internal clock synchronization.

6.2 Algorithm for Tick Generation

We consider an algorithm by Srikanth and Toueg [ST87] for internal clock synchronization with Byzantine failures. With unauthenticated messages, the algorithm can tolerate up to f Byzantine node failures in a system of $n \geq 3f + 1$ nodes. We discuss the core of the algorithm in a variant by Widder and Schmid [WS09].

By regularly exchanging messages between agents, the algorithm generates events that occur almost simultaneously on all correct agents, which we will call *ticks*. On top of its hardware clock, each agent keeps a series of adjusted clocks. In the original algorithm, each tick initiates a new *resynchronization epoch*, where a new adjusted clock is started with an initial value that is shared across all agents. This clock becomes the new output clock of the algorithm. In this manner, the clock provided by the algorithm is periodically re-synchronized, and clocks are brought "closer together" at the start of each resynchronization epoch. By this mechanism, the adjusted clocks can satisfy precision guarantees [ST87, AW04, WS09].

```
1   k := 0
2   send ⟨tick 0⟩ to all [once]
    /* catch-up rule */
3   if received ⟨tick l⟩ from f + 1 distinct nodes:
4       while k < l:
5           step()
    /* advance rule */
6   if received ⟨tick k⟩ from n − f distinct nodes:
7       step()
8   step():
9       k := k + 1
10      perform action k
11      send ⟨tick k⟩ to all [once]
```

Algorithm 6.1: Algorithm for tick generation, after [WS09], for $n \geq 3f + 1$. This is the core of the algorithm by [ST87].

Given that the communication network guarantees an end-to-end delay within $[0; \delta]$, the algorithm guarantees that clocks are at most 2δ time units apart at the start of each resynchronization period [WS09].

Tick generation. The repeated generation of ticks is the core of the clock synchronization algorithm, shown in Algorithm 6.1. Instead of building adjusted clocks of a finer granularity on top of these almost-simultaneous ticks, ticks generated from a similar algorithm can be used on the hardware level to directly drive a clock [FS12].

Timing assumptions. The timing assumptions for messages sent over the communication network influence how well the algorithm can perform, i.e., how close together ticks are. [ST87] assume that the communication network guarantees an end-to-end-delay within $[0, \delta]$. However, Algorithm 6.1 also works in system models with weaker timing assumptions, such as the Theta Model [WS09] or the Asynchronous Bounded-Cycle Model (ABC Model) [RS11].

Our model for Byzantine a.m.p. systems as introduced in Chapter 5 does not include timing guarantees at all. It turns out, however, that we can define a problem related to tick generation that does not reference time.

6.3 Firing Squad

In Section 6.2, we introduced Algorithm 6.1 as the foundation of clock synchronization in Byzantine a.m.p. systems. Algorithm 6.1 generates approximately synchronized rounds, where each tick separates one round from the next one. We consider one such round in isolation, i.e., consider what happens from tick k to tick $k+1$. The idea is that one isolated round will be easier to analyze. We call this *one-shot* tick generation.

One-shot tick generation bears similarities to a distributed systems problem called *Firing Squad* [BL87]. Firing Squad assumes a lock-step synchronous system with up to f Byzantine node failures. The environment can "trigger" each agent with a START event. START events are external to the system and can occur at an arbitrary time. Each agent can emit a FIRE event.

Strict Byzantine Firing Squad [BL87] requires the following:

Definition 6.3. *A system is consistent with **Strict Byzantine Firing Squad** when all runs satisfy:*

- *Correctness: If at least $f+1$ correct agents observe* START*, at least one correct agent fires eventually.*
- *Unforgeability: If a correct agent fires, a correct agent has observed* START*.*
- *Agreement: If a correct agent fires in round r, all correct agents fire in round r.*

Informally, we require that when $f+1$ START events have occurred on correct agents, all correct agents must emit FIRE simultaneously in some future round.

To understand some subtleties of this specification, consider a run r where START occurs on k correct agents. Observe that the following situations can arise:

- $k = 0$. By Unforgeability, no correct agent may fire.
- $1 \leq k \leq f$. Either all correct agents fire, or none. If one correct agent fires, by Agreement, all correct agents must fire.
- $f+1 \leq k$. By Correctness and Agreement, all correct agents must fire.

Definition 6.3 allows a choice of "all or none" in the case of $1 \leq k \leq f$, and this choice is necessary: No agent shall fire when $k = 0$, but all agents shall fire when $k \geq f+1$. But depending on the behavior of up to f Byzantine-faulty agents, a run r with $1 \leq k \leq f$ could be indistinguishable from a run r' with $k = 0$, or indistinguishable from a run r'' with $k = f+1$. In the worst case, this indistinguishability could hold for all agents! Hence, any sound specification must allow this choice.

Correctness in Definition 6.3 requires that $f+1$ *correct* agents observe START. An alternative specification could require that $2f+1$ agents, not necessarily correct, must observe START before the specification guarantees that all correct agents fire.

Burns and Lynch [BL87] show that an algorithm for Byzantine Firing Squad can be derived from an Eventual Byzantine Agreement (consensus) algorithm with bounded termination time.

6.4 Firing Rebels with/without Relay

In Section 6.3, we argued that *Firing Squad* bears similarities to a single round of tick generation in Algorithm 6.1. Byzantine Firing Squad is essentially a time-free, lockstep-synchronous variant of one-shot tick generation. Consider the following variation on Firing Squad:

Definition 6.4. *A system $\mathcal{R}$ is consistent with* ***Firing Rebels*** *when all runs satisfy:*

- *Correctness: If at least $f + 1$ correct agents observe* START*, all correct agents fire eventually.*
- *Unforgeability: If a correct agent fires, a correct agent has observed* START.

$\mathcal{R}$ is consistent with ***Firing Rebels with Relay*** *when runs also satisfy:*

- *Relay: If a correct agent fires, all correct agents fire eventually.*

Compared to Definition 6.3, we give up the Agreement property, which allows the problem to be solved in asynchronous systems. To exclude trivial solutions in the absence of Relay, we strengthen Correctness. Finally, the Relay property is essentially an asynchronous variant of Agreement that guarantees “all-or-none” behavior.

Algorithm 6.2 shows an algorithm that solves Firing Rebels with Relay, based on Algorithm 6.1. An agent fires when it receives its own ⟨fire⟩ message, which we explain in Section 6.5. We claim (without proof):

Proposition 6.5. *Algorithm 6.2 solves Firing Rebels with Relay.*

A run of this algorithm is visualized in Figure 6.2.

Since the specification of Firing Rebels does not mention time, we can express it as a protocol in Γ_{bamp} as introduced in Chapter 5. Despite being time-free, Firing Rebels closely resembles one-shot tick generation.

The Relay property is tricky to analyze using temporal-epistemic logic. As a first step, we analyze consider Firing Rebels without Relay in this thesis. Figure 6.2 illustrates a case where the Relay property makes a significant difference.

```
1   if observed START:
2       send ⟨echo⟩ to all
3   if received ⟨echo⟩ from f + 1 distinct agents:
4       send ⟨echo⟩ to all
5   if received ⟨echo⟩ from 2f + 1 distinct agents:
6       send ⟨fire⟩ to self
```

Algorithm 6.2: Algorithm for Firing Rebels with Relay, based on Algorithm 6.1 for tick generation [ST87, WS09]. We model the action FIRE as a message that an agent sends to itself, as explained in Section 6.5.

```
1   if observed START:
2       send ⟨echo⟩ to all
3       FIRE
4   if received ⟨echo⟩ from f + 1 distinct agents:
5       FIRE
```

Algorithm 6.3: Algorithm for Firing Rebels (without Relay).

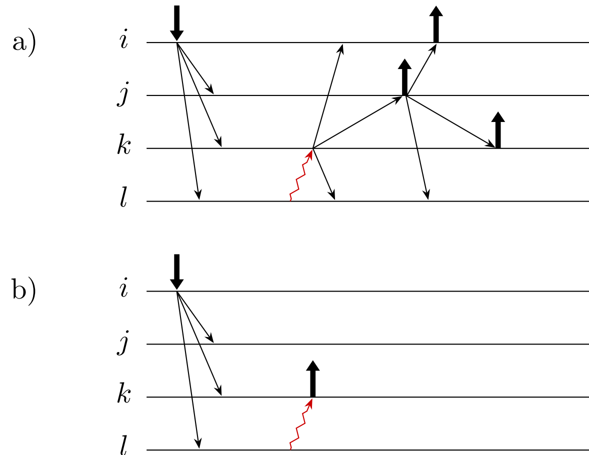

Figure 6.2: A possible run of a) Algorithm 6.2, solving Firing Rebels with Relay, and b) Algorithm 6.3, solving Firing Rebels. Horizontal lines represent agents in time. Thick down and up arrows represent START and FIRE actions respectively, thin arrows represent ⟨echo⟩ messages. Wavy arrows indicate messages forged by the environment, here agent l is Byzantine-faulty. The environment triggers START on agent i, i.e., on exactly one agent. By the specification, correct agents may either fire or not fire. In a), echo messages guarantee that when one correct agent fires, all correct agents fire (Relay property). In b), no such guarantee exists.

Dropping the Relay property allows a situation where only a subset of all correct processors fire. This allows for the simpler Algorithm 6.3, and we claim:

Theorem 6.6. *Algorithm 6.3 solves Firing Rebels without Relay.*

Proof sketch. Informally, we show Correctness as follows. When $f+1$ correct agents observe START, every such agent broadcasts $\langle\text{echo}\rangle$ by the algorithm. By eventual delivery, every correct agent eventually receives $f+1$ distinct $\langle\text{echo}\rangle$ messages. By the algorithm, every correct agent fires eventually.

For Unforgeability, observe that a correct agent i has two possibilities to emit FIRE. In line 3, the agent has observed START by itself. In line 5, it has received $\langle\text{echo}\rangle$ from $f+1$ agents, thus at least one such agent is correct by the failure assumption. By the algorithm, this agent has observed START itself or received $\langle\text{echo}\rangle$ from $f+1$ agents, in which case we repeat our argument until we arrive at a correct agent that has observed START. $\square$

In the following section, we formalize the properties of the Firing Rebels specification. Based on this, we will examine what kind of knowledge is at play in protocols that solve Firing Rebels.

6.5 Properties in Temporal-Epistemic Logic

In Sections 3.10 and 3.11, we introduced semantics for temporal-epistemic logic formulas $\varphi \in \mathcal{L}$ over runs and systems. In Chapter 5, we introduced the class of Byzantine a.m.p. contexts Γ_{bamp} and properties of such contexts. In Section 6.4, we introduced Firing Rebels.

In this section, we discuss how we can formulate properties of the Firing Rebels problem using temporal-epistemic logic.

When representing Algorithm 6.3 in Γ_{bamp}, we model START and FIRE as follows:

- START is an external action on each agent $i \in \mathcal{A}$: START $\in \text{Ext}_i$.
- FIRE is a message that each agent i sends to itself: FIRE $\in$ Msgs.

We model FIRE as a message, in order to allow the environment to fake the FIRE action for Byzantine-faulty agents per $\hat{\tau}_{\text{bamp}}$. This highlights a limitation of the way we defined Γ_{bamp}: The environment cannot fake internal actions of agents, nor do agents have separate actions for outputs that could be faked, similar to external actions that represent inputs.

Extending Definition 5.8, we define the following atomic propositions:

$$(\mathcal{R}, r, t) \models \text{start}_i \quad \text{iff} \quad \exists E.\ \text{external}(i, \textsc{start}) \in E \in r_i(t) \tag{6.1}$$

$$(\mathcal{R}, r, t) \models \text{fire}_i \quad \text{iff} \quad \exists E.\ \text{recv}(i, i, \textsc{fire}) \in E \in r_i(t) \tag{6.2}$$

$$(\mathcal{R}, r, t) \models \text{ready}_i \quad \text{iff} \quad \exists E.\ \text{send}(i, i, \textsc{fire}) \in E \in P_i(r_i(t)) \tag{6.3}$$

The intuitive meaning of these atoms is:

- start_i states that START has occurred on agent i.
- fire_i states that agent i has fired, which we model as the receipt of a FIRE message from itself.
- ready_i states that agent i is *ready to fire*, that is, it will send a FIRE message as soon as the environment allows it with a $\text{go}(i)$ action.

The truth value of start_i can only be changed by the environment, namely through a $\text{trigger}(i, \text{START})$ action. We introduce ready_i here because we will need it for the analysis of sufficient knowledge in Section 6.8. We can show the following:

Lemma 6.7. *Fix a protocol P for Firing Rebels, and a context $\gamma \in \Gamma_{\text{bamp}}$. Then the following hold for $\mathcal{R} = \hat{\mathcal{R}}(P, \gamma)$:*

$$\begin{aligned}
\mathcal{R} &\models \text{start}_i \to \Box\, \text{start}_i \\
\mathcal{R} &\models \text{fire}_i \to \Box\, \text{fire}_i \\
\mathcal{R} &\models \text{start}_i \to K_i\, \text{start}_i \\
\mathcal{R} &\models \text{fire}_i \to K_i\, \text{fire}_i \\
\mathcal{R} &\models \text{ready}_i \to K_i\, \text{ready}_i
\end{aligned}$$

Proof sketch. The claims hold because (6.1)–(6.3) depend only on the local state of agent i, and local states in Γ_{bamp} are histories. $\square$

With these atomic propositions, we can state the properties required in Firing Rebels as follows. Note that since $\mathcal{A}$ is a finite domain, all formulas can be expressed in $\mathcal{L}$, even though we use a set-based shorthand notation here.

Correctness:

$$\mathcal{R} \models \big|\{i \in \mathcal{A} \mid \text{correct}_i \wedge \text{start}_i\}\big| \geq f + 1 \quad \to \quad \Diamond \bigwedge_{i \in \mathcal{A}} (\text{correct}_i \to \text{fire}_i) \tag{6.4}$$

Unforgeability:

$$\mathcal{R} \models \bigvee_{i \in \mathcal{A}} (\text{correct}_i \wedge \text{fire}_i) \quad \to \quad \bigvee_{i \in \mathcal{A}} (\text{correct}_i \wedge \text{start}_i) \tag{6.5}$$

Relay:[4]

$$\mathcal{R} \models \bigvee_{i \in \mathcal{A}} (\text{correct}_i \wedge \text{fire}_i) \quad \to \quad \Diamond \bigwedge_{i \in \mathcal{A}} (\text{correct}_i \to \text{fire}_i) \tag{6.6}$$

[4]We mention Relay here for the sake of completeness, but the remainder of this work deals with Firing Rebels without Relay.

6.6 Necessary Knowledge

In Section 5.8, we have shown a theorem on knowledge gain in Γ_{bamp}. In Section 6.5, we established basic properties of Firing Rebels in temporal-epistemic logic. Similar to how Ben-Zvi and Moses establish a lower bound on communication structures in Ordered Response [BM14], we aim to find such a bound in Firing Rebels, as a first step towards an epistemic analysis of clock synchronization.

Ben-Zvi and Moses [BM14] talk informally about *necessary* and *sufficient* knowledge. Castañeda, Gonczarowski and Moses give a formal definition of *necessary knowledge* [CGM14].

The concept is simple, but there are some peculiarities involved in specifying when a formula φ is necessary knowledge. In particular, can the formula be different for each agent i? In this work, we assume that each agent i has some formula φ_i as necessary knowledge. Formally:

Definition 6.8. *Fix a context $\gamma \in \Gamma_{\text{bamp}}$, an agent $i \in \mathcal{A}$ and a formula $\varphi_i \in \mathcal{L}$. When for all protocols P consistent with Firing Rebels and $\mathcal{R} = \hat{\mathcal{R}}(P, \gamma)$,*

$$\mathcal{R} \models (\text{correct}_i \wedge \text{fire}_i) \to K_i\, \varphi_i, \tag{6.7}$$

then φ_i is ***necessary to be known*** *at agent i to fire.*

In other words, if we can show (6.7) for a formula φ_i without any restrictions on the protocol P, we know that $K_i\, \varphi_i$ is a necessary condition for agent i to FIRE, i.e., no protocol can solve the problem without agent i attaining knowledge of φ_i. This is a stronger statement than $\mathcal{R} \models (\text{correct}_i \wedge \text{fire}_i) \to \varphi_i$: φ_i might only be seen by an omniscient observer without being known to i, and i can only act on its local knowledge.

Theorem 6.9. *Fix a context $\gamma \in \Gamma_{\text{bamp}}$ and an agent $i \in \mathcal{A}$. Then the following is necessary to be known at each agent $i \in \mathcal{A}$ to fire:*

$$\varphi_i \;:=\; \text{correct}_i \to \bigvee_{j \in \mathcal{A}} \text{start}_j\,.$$

Proof. For the sake of contradiction, assume that there is a context $\gamma \in \Gamma_{\text{bamp}}$, protocol P that solves Firing Rebels without Relay, $\mathcal{R} = \hat{\mathcal{R}}(P, \gamma)$, $r(t)$, and $i \in \mathcal{A}$, s.t.

(1) $$(\mathcal{R}, r, t) \not\models (\text{correct}_i \wedge \text{fire}_i) \to K_i\,(\text{correct}_i \to \bigvee_{j \in \mathcal{A}} \text{start}_j).$$

Then

(2) $(\mathcal{R}, r, t) \models \text{correct}_i \wedge \text{fire}_i$ by (1)

(3) $(\mathcal{R}, r, t) \not\models K_i\,(\text{correct}_i \rightarrow \bigvee_{j\in\mathcal{A}} \text{start}_j)$ by (1)

By the semantics of K_i, we can find $r'(t')$ s.t. $r'_i(t') = r_i(t)$ and

(4) $(\mathcal{R}, r', t') \not\models \text{correct}_i \rightarrow \bigvee_{j\in\mathcal{A}} \text{start}_j$ by (3)

(5) $(\mathcal{R}, r', t') \models \text{correct}_i$ by (4)

(6) $(\mathcal{R}, r', t') \not\models \bigvee_{j\in\mathcal{A}} \text{start}_j$ by (4)

(7) $(\mathcal{R}, r, t) \models K_i\ \text{fire}_i$ by (2) and Lemma 6.7

(8) $(\mathcal{R}, r', t') \models \text{fire}_i$ by $r'_i(t) = r_i(t)$

(9) $(\mathcal{R}, r', t') \models \text{correct}_i \wedge \text{fire}_i$ by (5) and (8)

(10) $(\mathcal{R}, r', t') \models \bigvee_{j\in\mathcal{A}} (\text{correct}_j \wedge \text{fire}_j)$ by (9) and $\vee$

By assumption, P solves Firing Rebels and thus Equation (6.5) holds:

(11) $(\mathcal{R}, r', t') \models \bigvee_{j\in\mathcal{A}} (\text{correct}_j \wedge \text{fire}_j) \rightarrow \bigvee_{j\in\mathcal{A}} (\text{correct}_j \wedge \text{start}_j)$

(12) $(\mathcal{R}, r', t') \models \bigvee_{j\in\mathcal{A}} (\text{correct}_j \wedge \text{start}_j)$ by (10) and (11)

(13) $(\mathcal{R}, r', t') \models \bigvee_{j\in\mathcal{A}} \text{start}_j$ by (12)

which contradicts (6). Hence, the theorem holds. □

In Section 6.8, we will show that φ_i is not only necessary, but also sufficient to be known at agent i to fire. Note that one might intuitively find φ_i too weak: When start_j holds on a faulty agent, φ_i also becomes true! However, agent i cannot distinguish whether start_j holds or not for a faulty j, and hence $K_i\,\varphi_i$ cannot hold in this case.

6.7 Necessary Communication

How can agent i gain knowledge of φ_i? We show that it needs to receive at least $f+1$ messages from other agents:

Theorem 6.10. *Let P represent Algorithm 6.3, $\gamma \in \Gamma_{\text{bamp}}$, and $\mathcal{R} = \hat{\mathcal{R}}(P, \gamma)$. Then*

$$(\mathcal{R}, r, t) \models \text{correct}_i \wedge \neg\,\text{start}_i \wedge K_i\,(\text{correct}_i \rightarrow \bigvee_{j \in \mathcal{A}} \text{start}_j) \tag{6.8}$$

implies

$$(\mathcal{R}, r, t) \models |\{j \in \mathcal{A} \mid \text{msg-recvd}(j, i, M) \text{ and } j \neq i\}| \geq f+1. \tag{6.9}$$

Proof. Assume (6.8) holds. Then

(1)	$(\mathcal{R}, r, t) \models \neg\,\text{start}_i$	by (6.8)
(2)	$(\mathcal{R}, r, t) \models K_i\,\neg\,\text{start}_i$	by (6.1)

Then, for all $r'(t')$ s.t. $r'_i(t') = r_i(t)$ and $r'_j(t) = r_j(0)$ for all $j \neq i$,

(3)	$(\mathcal{R}, r', t') \models \neg\,\text{start}_i$	by (2)
(4)	$(\mathcal{R}, r', t') \models \neg\,\text{start}_j$	for all $j \neq i$ by $r'_j(t') = r_j(0)$ and (6.1)
(5)	$(\mathcal{R}, r', t') \models \neg \bigvee_{j \in \mathcal{A}} \text{start}_j$	by (3) and (4)

By (5) and $\neg\psi \implies \neg\varphi \vee (\varphi \wedge \neg\psi) \iff \neg\varphi \vee \neg(\neg\varphi \vee \psi) \iff \varphi \rightarrow \neg(\varphi \rightarrow \psi)$,

(6) $\quad (\mathcal{R}, r', t') \models \text{correct}_i \rightarrow \neg(\text{correct}_i \rightarrow \bigvee_{j \in \mathcal{A}} \text{start}_j)$

Since (6.8) holds, and (6) holds for all $r'(t')$ with the above conditions, we can apply Theorem 5.11:

(7) $\quad (\mathcal{R}, r, t) \models |\{j \in \mathcal{A} \mid \text{msg-recvd}(j, i, M) \text{ and } j \neq i\}| \geq f+1.$ □

6.8 Sufficient Knowledge

Can agent i fire, based solely on knowledge of φ_i? Consider Algorithm 6.4, a knowledge-based full-information protocol [FHMV03] based on φ_i.

We show that Algorithm 6.4 solves Firing Rebels without Relay:

Theorem 6.11. *Algorithm 6.4 provides Unforgeability.*

Proof sketch. We need to show

$$(1) \qquad (\mathcal{R}, r, t) \models \bigvee_{i\in\mathcal{A}} (\mathrm{correct}_i \wedge \mathrm{fire}_i) \rightarrow \bigvee_{i\in\mathcal{A}} (\mathrm{correct}_i \wedge \mathrm{start}_i)$$

For the sake of contradiction, assume that for some i,

$$(2) \qquad (\mathcal{R}, r, t) \models \mathrm{correct}_i \wedge \mathrm{fire}_i$$

$$(3) \qquad (\mathcal{R}, r, t) \not\models \bigvee_{j\in\mathcal{A}} (\mathrm{correct}_j \wedge \mathrm{start}_j)$$

Further, assume that

$$(4) \qquad (\mathcal{R}, r, t) \not\models \mathrm{start}_i$$

(otherwise, the argument is trivial). Let $S = \{j \in \mathcal{A} \mid (\mathcal{R}, r, t) \models \mathrm{start}_j\}$. By (3),

$$(5) \qquad (\mathcal{R}, r, t) \models \mathrm{start}_j \rightarrow \neg\,\mathrm{correct}_j \qquad \text{for all } j \in \mathcal{A}$$

$$(6) \qquad (\mathcal{R}, r, t) \models \neg\,\mathrm{correct}_j \qquad \text{for all } j \in S.$$

By (6) and $\hat{\tau}_{\mathrm{bamp}}$, $|S| \leq f$. Then, analogous to the construction of Lemma 5.10 (proof omitted), we can find $r'(t')$ such that

$$(7) \qquad r'_i(t') = r_i(t)$$

$$(8) \qquad r'_j(t') = r_j(0) \quad \text{for all } j \neq i$$

$$(9) \qquad i \notin \mathrm{Failed}(r'(t')).$$

```
1  on each state change of agent i:
2      send local state to all
3      if K_i (correct_i → ⋁_{j∈A} start_j):
4          FIRE
```

Algorithm 6.4: Knowledge-based program for Firing Rebels (full-information protocol).

Then

(10) $\quad (\mathcal{R}, r', t') \not\models \bigvee_{j \in \mathcal{A}} \text{start}_j$ $\quad$ by (4), (8), (6.1)

(11) $\quad (\mathcal{R}, r', t') \models \text{correct}_i$ $\quad$ by (9)

But by (2) and the algorithm,

(12) $\quad (\mathcal{R}, r, t) \models K_i\,(\text{correct}_i \rightarrow \bigvee_{j \in \mathcal{A}} \text{start}_j)$

Since $r'_i(t') = r_i(t)$,

(13) $\quad (\mathcal{R}, r', t') \models \text{correct}_i \rightarrow \bigvee_{j \in \mathcal{A}} \text{start}_j$

(14) $\quad (\mathcal{R}, r', t') \models \bigvee_{j \in \mathcal{A}} \text{start}_j$ $\quad$ by (11)

But this contradicts (10). Hence, the algorithm provides Unforgeability. □

Theorem 6.12. *Algorithm 6.4 provides Correctness.*

Proof sketch. We need to show

(1) $\quad (\mathcal{R}, r, t) \models \left|\{i \in \mathcal{A} \mid \text{correct}_i \wedge \text{start}_i\}\right| \geq f + 1 \rightarrow \Diamond \bigwedge_{i \in \mathcal{A}} (\text{correct}_i \rightarrow \text{fire}_i).$

Let $S = \{i \in \mathcal{A} \mid (\mathcal{R}, r, t) \models \text{correct}_i \wedge \text{start}_i\}$ and assume

(2) $$|S| \geq f + 1.$$

By the definition of start_i, for all $i \in S$,

(3) $$\exists E.\ \text{external}(i, \textsc{start}) \in E \in r_i(t)$$

By the algorithm, each agent repeatedly broadcasts its local state. For each $i \in S$, let M_i denote the message that contains $r_i(t)$. By the algorithm and Eventual Delivery, there is a time $t^+ \geq t$ s.t. for arbitrary $i \in S$ and arbitrary $j \in \mathcal{A}$,

(4) $\quad (\mathcal{R}, r, t^+) \models \text{msg-recvd}(i, j, M_i)$

(5) $\quad (\mathcal{R}, r, t^+) \models K_j\, \text{msg-recvd}(i, j, M_i)$ $\quad$ by (5.17)

Thus for all $r'(t')$ s.t. $r'_j(t') = r_j(t^+)$, and $i \in S$,

(6)	$(\mathcal{R}, r', t') \models \text{msg-recvd}(i, j, M_i)$	
(7)	$(\mathcal{R}, r', t') \models \text{correct}_i \rightarrow \text{msg-sent}(i, j, M_i)$	by $\hat{\tau}_{\text{bamp}}$
(8)	$(\mathcal{R}, r', t') \models \text{correct}_i \rightarrow \text{start}_i$	by def. of M_i

By $|S| \geq f + 1$, there is an agent $i \in S$ s.t.

(9)	$(\mathcal{R}, r', t') \models \text{correct}_i$	
(10)	$(\mathcal{R}, r', t') \models \text{correct}_i \wedge \text{start}_i$	by (8), (9)

We can weaken this to

(11) $$(\mathcal{R}, r', t') \models \bigvee_{l \in \mathcal{A}} \text{start}_l$$

And in particular, for our arbitrary $j \in \mathcal{A}$ chosen earlier,

(12) $$(\mathcal{R}, r', t') \models \text{correct}_j \rightarrow \bigvee_{l \in \mathcal{A}} \text{start}_l$$

Since this holds in arbitrary $r'(t')$ s.t. $r'_j(t') = r_j(t^+)$,

(13) $$(\mathcal{R}, r, t^+) \models K_j \left(\text{correct}_j \rightarrow \bigvee_{i \in \mathcal{A}} \text{start}_i\right)$$

Then, by the algorithm,

(14) $$(\mathcal{R}, r, t') \models \text{correct}_j \rightarrow \text{ready}_j$$

By Fair Schedule and Eventual Delivery, every agent that is ready to fire is firing eventually. Let $t^* > t^+$ be a time when this has happened for every agent. Then

(15) $$(\mathcal{R}, r, t^*) \models \text{correct}_j \rightarrow \text{fire}_j$$

Since $j \in \mathcal{A}$ was arbitrary,

(16) $$(\mathcal{R}, r, t^*) \models \bigwedge_{i \in \mathcal{A}} (\text{correct}_i \rightarrow \text{fire}_i)$$

(17) $$(\mathcal{R}, r, t) \models \Diamond \bigwedge_{i \in \mathcal{A}} (\text{correct}_i \rightarrow \text{fire}_i) \qquad \text{by } t^* > t$$

Hence, the algorithm provides Correctness. □

Since Algorithm 6.4 solves Firing Rebels (without Relay), we can conclude that

$$\varphi_i \; = \; \text{correct}_i \to \bigvee_{j \in \mathcal{A}} \text{start}_j$$

is indeed sufficient knowledge for agent i to fire.

CHAPTER 7

Conclusion and Outlook

In this thesis, we introduced a formal model for temporal-epistemic logic in Byzantine Asynchronous Message-Passing systems, namely the class of contexts Γ_{bamp}. We presented a non-contradictory intuitive interpretation of the knowledge of faulty agents in our model. We showed limitations of the model regarding its application to message-passing systems.

We used temporal-epistemic logic to show that in Γ_{bamp}, agents can only gain knowledge when they have received messages from more than $f + 1$ agents. This is a result about Γ_{bamp}, but more importantly an example of how temporal-epistemic logic can be applied to reason about Byzantine message-passing systems.

We have investigated the link between clock synchronization and tick generation, and introduced the Firing Rebels problem as a simplification. Using temporal-epistemic logic, we characterized Firing Rebels without Relay using knowledge formulas. We proved that in Firing Rebels without Relay, knowledge of a certain formula is both a necessary and sufficient condition for the FIRE action of agents. We extended this result to a lower bound on communication for any protocol solving Firing Rebels without Relay in Γ_{bamp}.

As a foundation, we have summarized the existing framework of Protocols and Contexts [FHMV03] and laid it out in a coherent way, aimed at readers from a distributed systems background and not necessarily familiar with epistemic logic.

The analysis of Firing Rebels also serves an indicator for the practical usefulness of the model specified by Γ_{bamp} and its limitations. We point out possible improvements later in this chapter.

We encountered several obstacles in the course of our work dealing with Byzantine-faulty systems:

- The way of thinking about the knowledge of Byzantine-faulty agents proposed in Section 5.1 was central for our first formalization of Γ_{bamp}.

- The complexity of analyzing clock synchronization using epistemic logic has led us to the definition of Firing Rebels as a stripped-down version of the problems that we are actually interested in.
- In the specification of Firing Rebels, we discovered that we had no good way to model agent actions that the environment can fake. We decided to let agents send messages to themselves for this purpose, so that Firing Rebels could be accommodated in the existing model.

7.1 Future Work

In this section, we describe possibilities for future work, based on the experiences we made in the application of temporal-epistemic logic to Byzantine message-passing systems. Prosperi, Kuznets and Schmid [PKS17] build their work on some of our findings and notably introduce a different, more generic system model.

Limitations of our system model. Our model Γ_{bamp} is a first shot at bringing temporal-epistemic logic and Byzantine failures together. We see two main issues:

In the specification of Firing Rebels, we let agents send messages to themselves to represent actions that the environment can fake. In an improved model, agents could have explicit "outputs" that the environment can fake, similar to how *external actions* represent "inputs". One could also model inputs and outputs as messages sent from and to the environment respectively. Roman Kuznets has proposed another alternative: like a "powerful psychic", the environment could fake internal actions [PKS17, Kuz]. Agents would "know" about the fake actions, but be unable to communicate this to other agents.

Throughout our thesis, we never use the fact that the protocol of the environment P_e can specify specific actions; instead, we always let it propose the set of all actions, of which a nondeterministic choice is then made. A simplified model could remove P_e from the specification, and encode all restrictions in τ, as we basically did already for $\hat{\tau}_{\text{amp}}$ and $\hat{\tau}_{\text{bamp}}$. For reasoning, it might be desirable to keep a trace of which actions happened similar to the state of the environment s_e, but this could also be encoded directly in τ. One could also explore a model in the style of *executions* [AW04], where unlike runs, not only the global state is tracked but also the transitions between states.

Knowledge gain. Our knowledge gain theorem (Theorem 5.11) is very simple and only states that, *at some time*, an agent must have heard from $f + 1$ neighbors. It would be interesting to see an extension that allows statements about what communication is necessary *after a certain event has occurred.* This gets more complicated since, when agent i has heard from $f + 1$ neighbors, it might already

know which ones are faulty and which ones are correct. *Past causal cones* [BM14] could be a starting point. Based on this, it might be possible to find necessary communication structures similar to the centipedes of [BM14].

Time bounds. Our framework does not include any timing guarantees on messages. Ben-Zvi and Moses [BM14] propose a failure-free model with time bounds that is compatible with epistemic logic. Epsilon-Common Knowledge [HM90], a variant of Common Knowledge with timing uncertainties, also goes in this direction. For an analysis of clock synchronization, a modal operator dealing with time bounds would be desirable. Ideally, weaker timing guarantees such as can be found in the Theta model [WS09] or the Asynchronous Bounded-Cycle Model [RS11] should also have a representation in this logic.

Firing Rebels with Relay. Our treatment of Firing Rebels deals with the variant *without Relay*, which is much simpler to analyze. It would be interesting to see whether a similar analysis about necessary and sufficient knowledge can be performed on Firing Rebels *with* Relay, which is essentially a time-free version of the consistent broadcast primitive of Srikanth and Toueg [ST87].

Logics to directly represent messages. This thesis only applies temporal-epistemic logic by reasoning on the semantic level. Pucella and Sadrzadeh [PS10] propose a logic with axioms for announcements. Possibly, a logic could be defined that allows syntactic reasoning based on axioms for Byzantine-faulty message-passing systems.

Bibliography

[AW04] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. Wiley, 2nd edition, 2004.

[Bar81] Jon Barwise. Scenes and other situations. *Journal of Philosophy*, 78(7):369–397, 1981. Cited after [HM90].

[BCG+07] Martin Biely, Bernadette Charron-Bost, Antoine Gaillard, Martin Hutle, André Schiper, and Josef Widder. Tolerating corrupted communication. In *Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing*, pages 244–253. ACM, 2007.

[BL87] James E. Burns and Nancy A. Lynch. The Byzantine Firing Squad problem. In *Advances in Computing Research, Vol. 4: Parallel and Distributed Computing*, pages 147–161. JAI Press, 1987.

[BM14] Ido Ben-Zvi and Yoram Moses. Beyond Lamport's happened-before: On time bounds and the ordering of events in distributed systems. *J. ACM*, 61(2):13, 2014.

[CGM14] Armando Castañeda, Yannai A. Gonczarowski, and Yoram Moses. Unbeatable consensus. In *DISC 2014, 28th International Symposium, Proceedings*, pages 99–106. Springer, 2014.

[DFP+14] Danny Dolev, Matthias Függer, Markus Posch, Ulrich Schmid, Andreas Steininger, and Christoph Lenzen. Rigorously modeling self-stabilizing fault-tolerant circuits: An ultra-robust clocking scheme for systems-on-chip. *Journal of Computer and System Sciences*, 80(4):860–900, 2014.

[DHK08] Hans van Ditmarsch, Wiebe van der Hoek, and Barteld Kooi. *Dynamic Epistemic Logic*. Springer, 2008.

[DM90] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, 1990.

[FHMV95] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[FHMV03] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge*. MIT Press, first paperback edition, 2003.

[FS12] Matthias Függer and Ulrich Schmid. Reconciling fault-tolerant distributed computing and systems-on-chip. *Distributed Computing*, 24:323–355, 2012.

[HF85] Joseph Y. Halpern and Ronald Fagin. A formal model of knowledge, action, and communication in distributed systems: preliminary report. In *Proceedings of the 4th Annual ACM symposium on Principles of Distributed Computing*, pages 224–236. ACM, 1985.

[HF89] Joseph Y. Halpern and Ronald Fagin. Modelling knowledge and action in distributed systems. *Distributed computing*, 3(4):159–177, 1989.

[Hin62] Jaakko Hintikka. *Knowledge and Belief*. Cornell University Press, Ithaca, NY, USA, 1962. Cited after [FHMV03].

[HM90] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *J. ACM*, 37(3):549–587, 1990.

[HMW01] Joseph Y. Halpern, Yoram Moses, and Orli Waarts. A characterization of eventual Byzantine agreement. *SIAM J. Computing*, 31(3):838–865, 2001.

[Kop11] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, 2nd edition, 2011.

[Kuz] Roman Kuznets. Personal communication.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[Lam01] Butler W. Lampson. The ABCD's of Paxos, 2001. Presented at the 20th Annual ACM Symposium on Principles of Distributed Computing. http://research.microsoft.com/en-us/um/people/blampson/65-ABCDPaxos/Acrobat.pdf.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[Mic89] Ruben Michel. A categorical approach to distributed systems expressibility and knowledge. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing*, pages 129–143. ACM, 1989.

[MT88] Yoram Moses and Mark R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3(1–4):121–169, 1988.

[NB99] Gil Neiger and Rida A. Bazzi. Using knowledge to optimally achieve coordination in distributed systems. *Theoretical Computer Science*, 220(1):31–65, 1999.

[PKS17] Laurent Prosperi, Roman Kuznets, and Ulrich Schmid. Knowledge in Byzantine message-passing systems. Technical Report TUW-260549, Institute of Computer Engineering, Technische Universität Wien, 2017. To appear at http://publik.tuwien.ac.at/files/PubDat_260549.pdf.

[PR03] Rohit Parikh and Ramaswamy Ramanujam. A knowledge based semantics of messages. *Journal of Logic, Language and Information*, 12(4):453–467, 2003.

[PS10] Riccardo Pucella and Mehrnoosh Sadrzadeh. A runs-and-systems semantics for logics of announcements. In *Logic and the Foundations of Game and Decision Theory – LOFT 8: Revised selected papers*, pages 112–134. Springer, 2010.

[RS11] Peter Robinson and Ulrich Schmid. The Asynchronous Bounded-Cycle model. *Theoretical Computer Science*, 412(40):5580–5601, 2011.

[ST87] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *J. ACM*, 34(3):626–645, 1987.

[WS09] Josef Widder and Ulrich Schmid. The Theta-Model: achieving synchrony without clocks. *Distributed Computing*, 22(1):29–47, 2009.

List of Symbols

Symbol	Meaning
$\mathcal{A}$	The set of agents (nodes, processors).
Act_i	The set of actions of agent i.
$\vec{\alpha}$	A joint action.
α_i	The action of agent i.
e	The environment.
Ext_i	External actions of agent i.
G	The set of global states.
G_0	The set of initial global states, part of a context γ.
γ	A context.
Γ_{amp}	The class of failure-free asynchronous message-passing contexts.
Γ_{bamp}	The class of Byzantine asynchronous message-passing contexts.
$i, j, \ldots$	An agent.
Int_i	Internal actions of agent i.
K_i	The knowledge operator.
L_i	The set of local states of agent i.
P	A joint protocol.
P_i	The protocol of agent i.
Π	The set of atomic propositions.
π	An interpretation.
Ψ	Admissibility conditions, part of a context γ.
R_i	Possible-worlds relation of agent i.
$r(t)$	A run.
$\mathcal{R}$	A system (set of runs).
$\hat{\mathcal{R}}(P, \gamma)$	The system representing joint protocol P in context γ.
Σ_i	Initial states of agent i.
τ	Global transition function, part of a context γ.