Informatics

On the strongest message adversary for directed dynamic networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Stephan Felber
Matrikelnummer 01426418

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Ulrich Schmid

Wien, 9. Dezember 2021

Stephan Felber

Ulrich Schmid

Informatics

On the strongest message adversary for directed dynamic networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Stephan Felber
Registration Number 01426418

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Ulrich Schmid

Vienna, 9th December, 2021

Stephan Felber	Ulrich Schmid

Erklärung zur Verfassung der Arbeit

Stephan Felber

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, 9. Dezember 2021

Stephan Felber

Acknowledgements

I would like to thank my family for asking me countless times how my thesis is coming along and why it needs to have such a complicated title. I would like to thank Karishma for reading my introduction and I would like to thank my supervisor, Prof. Ulrich Schmid, for tirelessly correcting my comma errors in every draft (and the chance to write a thesis on an interesting topic, of course).

Kurzfassung

Inspiriert durch die Definition des Weakest Failure Detectors in Asynchronous Message Passing Systems begeben wir uns auf die Suche nach dem Strongest Message Adversary in Synchronous Message Passing Systems. Dafür müssen wir natürlich zuerst festlegen, was 'stronger' im Kontext von Message Adversaries überhaupt bedeutet. Ähnlich wie im Asynchrous Message Passing Model vergleichen wir Message Adversaries, indem wir einen Message Adversary auf einem anderen Message Adversary simulieren. Die formale Basis für diese Arbeit ist eine Erweiterung des HO-Models, das von Charron-Bost und Schiper in [CBS09] eingeführt wurde. Nach der formalen Definition einer Simulation eines Message Adversaries präsentieren wir einen Algorithmus der den Message Adversary STAR auf jedem Message Adversary, der eine Lösung für Consensus erlaubt, simulieren kann. Das impliziert, nach unserer Definition, dass der Message Adversary STAR ein Top Element in der induzierten Quasi-Ordnung ist. Wir führen weiter aus, dass es nicht nur einen 'strongest' Message Adversary gibt, sondern eine Menge an 'strongest' Message Adversaries, und charakterisieren diese. Danach besprechen wir die Beziehung der 'strongest' Message Adversaries zu dem Weakest Failure Detector und die Relation zu den paradoxen Resultaten aus [BRS$^+$18] die durch unsere verfeinerte theoretische Basis aufgelöst werden können. Schlussendlich definieren und beweisen wir eine speichereffizientere Reduktion von Multi-Valued Consensus auf Binary Consensus, die einen zentralen Baustein in unserer Message Adversary Simulation darstellt.

Abstract

Inspired by the chase for the weakest failure detector in asynchronous message passing systems, we look for the strongest message adversary in the synchronous setting. This first requires defining the notion of a 'stronger' message adversary and, similar to the asynchronous model, building a pre-order by simulating a message adversary on top of another message adversary. The formal basis for this thesis is an extended version of the HO-model introduced by Charron-Bost and Schiper in [CBS09]. After stating a validity criterion for simulated runs, i.e., a correct message adversary simulation, we present an algorithm that can simulate the message adversary STAR atop of any message adversary that allows to solve consensus. This implies that STAR is a top element in our relation and thus named a 'strongest' message adversary. However, rather than a single strongest message adversary, it turns out that there is a set of strongest message adversaries, which we characterize fully. After discussing the relation of a strongest message adversary to the weakest failure detector, we elaborate on a seemingly paradoxical result presented in [BRS$^+$18] and resolve it by means of our refined theoretical basis. Finally we provide and prove correct a more memory efficient reduction of multi-valued consensus to binary consensus, which forms a crucial building block in our message adversary simulation and also provides a practical example for the utility of message adversary simulations that might be of independent interest.

Contents

CHAPTER 1

Introduction

A century ago there were no computers. Fast forwarding to the first processors, we discover that back then most of the problems concerned just one processing unit at a time. Pretty soon after that we taught processors to talk to one another, introducing a variety of new problems just because processors could now *exchange* information. One of the most prominent problems turned out to be the seemingly trivial task of agreeing on some common value in a group of processors, similar to a group of friends trying to agree which pub they meet at tonight. While the choice of the pub in the group of friends might depend on the individual taste in beer, the outcome in the set of processes depends more on the environment, for example the timeliness of messages or the reliability of the communication channels.

In more scientific terms, we would call this area *distributed computing*, meaning multiple processing entities with some means of exchanging information together with a common task to solve. We usually refer to the processing entities as *processes* to avoid confusion with a processor, which today contains already multiple cores and each may execute a different program, whereas a *process* only executes one 'program'. Coincidentally, multicore processing already holds a distributed computing problem, namely keeping a consistent distributed memory state.

In order to study distributed systems, we abstract away the not so important details and focus on the core of the problems. Romantic relationships within the group of friends might not be important, but the means of communication is. Are they all standing in a circle and talking over each other, or are they communicating via postal mail with non reliable delivery? Finding the next pub arguing in a circle might crucially depend on the group synchronizing their communication in order to even understand each other. On the other hand, the postal exchange of information may silently not deliver a letter, and thus presents the group of friends with the entirely different challenge of deciding whether a message is late, or never arriving.

The group of friends using the lossy postal service might decide to pay extra for same day delivery, granting them security that if, at the end of the day, a letter did not arrive, it was lost. This is already really close to the *synchronous message passing* (abbreviated as $\mathcal{SMP}$) model used in this thesis. For sake of completeness assume that a second group of friends decides to ditch postal and every friend personally delivers every message to the doorstep and drops it there. This introduces the danger that, on their way to the next doorstep, a friend might find another pub they like together with an entirely different group of friends and suddenly, silently drop out of the decision process. As they obviously cannot inform their original group of friends about their decision to drop out, the original group of friends cannot distinguish between them just taking very long to deliver their messages, or them having found other friends (and a new pub). This second group of friends is very close to the *asynchronous message passing* (abbreviated as $\mathcal{AMP}$) model and [FLP85] famously proved that in general, they cannot find a common pub if already just one friend silently deserts to another group of friends.

One effort of making the decision process in asynchronous message passing possible again involved introducing oracles for every process, called failure detectors. For our second group of friends, this would mean every friend has an app granting them arbitrary information about the state of the group. A *weakest* app (in terms of information provided) still allowing a consensus to be reached is abbreviated as (Σ, Ω) and was already found in [CHT96].

1.1 Motivation, problem statement and results

This thesis builds on the analogy of the group of friends trying to coordinate via the lossy postal service. After abstracting away the whole social situation we are left with a set of processing entities (the friends), exchanging messages in rounds (where one day is one round) and an adversary (the postal service), which ultimately decides for each message whether it arrives or not. The so called message adversary usually consists of a set of infinite directed graph sequences $\mathcal{G}_1, \mathcal{G}_2, \ldots$ (one for each round), where a message arrives at its destination q if and only if the corresponding communication graph $\mathcal{G}_k$ in that round k contains an edge from the sender p to q. We can right away conclude that if the message adversary looses every message, the processes cannot reach an agreement and the consensus problem is unsolvable. This motivates restricting the message adversary in some way, for example in the amount of messages it may loose in each round, or identifying one process whose messages are guaranteed to arrive.

This raises the question of how much freedom for the message adversary is too much freedom, i.e., when is the adversary too strong for consensus to be solvable? We know adversaries which make solving consensus rather easy, for example, the message adversary which looses *no* messages, and we know adversaries which make consensus impossible. This suggest that there is some adversary in the middle which just allows consensus to be solved, but any 'strengthening' would make consensus unsolvable. The goal of this thesis is to rigorously define and find this *strongest* message adversary, building on and further developing the idea and general approach introduced by Schmid, Schwarz and Winkler in [SSW18]. Finding this *core* of making consensus possible allows us, for example, to reduce the postal service reliability to a minimum necessary.

For an initial inspiration, we can look over to asynchronous message passing, as the weakest failure detector for consensus has already been found there. This requires drawing relations between $\mathcal{AMP}$ and $\mathcal{SMP}$, which is usually done by trying to simulate an algorithm for one model via some simulator in the other model. Raynal and Stainer in [RS13] compared the weakest failure detector (Σ, Ω) in $\mathcal{AMP}$ to the message adversary $(\text{SOURCE}, \text{QUORUM})$ in a very fine-grained manner. Among other results, they showed that the failure detector Σ exactly captures the message adversary SOURCE, whereas Ω exactly captures the message adversary QUORUM. Despite their detailed results, it remained an open question whether any message adversary allowing consensus, and in particular a strongest message adversary for consensus, can simulate the weakest failure detector (Σ, Ω). In order to positively answer this question, $(\text{SOURCE}, \text{QUORUM})$ will have to be a strongest message adversary or, as it turns out, belong to the set of strongest message adversaries.

Finding a strongest message adversary requires us to define what 'stronger' means. More precisely, we need to relate two message adversaries in terms of their ability to allow solving consensus. Building on the predicate emulations introduced in the heard-of model in [CBS09], we define a *message adversary simulation*, as the simulation of some algorithm specified for message adversary MA' atop of some other message adversary MA via another simulation algorithm. We spend considerable efforts on rigorously specifying what makes a simulation correct, meaning that the simulated algorithm behaves as if 'it was run on the model directly', in this case MA' in $\mathcal{SMP}$. This formal definition of a simulation of message adversary MA in $\mathcal{SMP}$ atop of another message adversary MA' in $\mathcal{SMP}$ allows us to compare arbitrary message adversaries in terms of their problem solving capabilities. Utilizing this, we will identify the message adversary STAR, which simply consists of sequences of repeated star-graphs, as a top element in the preorder defined by the message adversary simulation relation. Additionally, we extend the set of strongest message adversaries to any message adversary that can simulate STAR.

Here we also encounter the paradoxon already spotted in [BRS+18]. There, the authors proved that the message adversary $\text{VSSC}(\infty)$ in $\mathcal{SMP}$ cannot simulate any algorithm on the failure detector Σ in $\mathcal{AMP}$, implying that the message adversary $\text{VSSC}(\infty)$ in $\mathcal{SMP}$ cannot simulate the weakest failure detector (Σ, Ω) in $\mathcal{AMP}$. On the other hand, they identified that $\text{VSSC}(\infty)$ *can* simulate any algorithm on STAR in $\mathcal{SMP}$, and that it is straight forward to simulate (Σ, Ω) in $\mathcal{AMP}$ on STAR, contradicting the previous conclusions. The reason for this paradoxon was traced back to the validity condition for simulated runs. We will sort of dance around this paradoxon, because the definition of a correct run in this thesis differs from the definition applied in [BRS+18]. This deals with the paradoxon by simply allowing Σ in $\mathcal{AMP}$ to be simulated atop of $\text{VSSC}(\infty)$ in $\mathcal{SMP}$, invalidating the first side of the contradiction.

Last, we will reduce multi-valued consensus to binary consensus. This reduction was already proven in Winklers PhD thesis [Win19], but we will reduce its space complexity from exponential to linear. This also provides a 'real-world' setting, where our definition of a message adversary simulation can be used to to prove the correctness of a simulated algorithm.

1.2 Related work

There are already a couple of publications about possibility and impossibility results for consensus in undirected dynamic networks [KOM11] and in directed dynamic networks [BRS12, BRS$^+$15, SWS16, BRS$^+$18, WSS16]. As already mentioned, Raynal and Stainer in [RS13] established many interesting equivalences between various failure detectors on $\mathcal{AMP}$ and message adversaries on $\mathcal{SMP}$, but restricted the applicability to so called task-solving algorithms. Similar to this thesis, they also compared two systems by simulating one on top of the other.

[KS06] introduce the *GIRAF* framework to compare the impact of various timeliness and failure detector assumptions on the runtime performance of consensus algorithms. [DLS88] examined the consensus problem in presence of partial synchrony, where the processes' speeds can differ, the message delay can be arbitrary (but bounded) or both. They showed that the partially synchronous model with $\Phi = 1$ and $\Delta \geq 1$ and the asynchronous model with the perfect failure detector are not equivalent in terms of problem solvability. Charron-Bost and Schiper introduced the *HO*-model in [CBS09], which this thesis uses and extends. Instead of focusing just on synchronous message passing, the *HO*-model is an abstraction that works for a large set of distributed systems allowing for easier comparison between different models.

As already mentioned, the weakest failure detector (Σ, Ω) for consensus was found in [CHT96] right after the concept of failure detectors was introduced in [CT96] by nearly the same authors. This inspired several publications in the chase for the weakest model still allowing the implementation of the failure detector Ω [ADGFT01, ADGFT03, ADGFT04, MR99, AFM$^+$04, MOZ05, HMSZ09, FR10]. Rajsbaum, Raynal and Travers showed in [RRT08] that failure detectors do not increase the solution power of the iterated immediate snapshot model over asynchronous read/write shared memory. A more general relation between various eventually synchronous models and asynchronous models with stabilizing failure detectors, which also considers efficiency of the algorithmic transformations, has been established by Biely et. al. in [BHDPW07]. Among other results, it establishes that Ω is essentially equivalent to models with an eventually timely source, as well as to eventual lock-step rounds. [JT08, CBHW10] shed some light on limitations of the failure detector abstraction in timing-based models of computation, which are relevant in our context.

Consensus in synchronous message passing networks was already mentioned in the seminal paper [SW89] by Santoro and Widmayer, generalizations have been published in [SWK09, CBS09, BSW11, CGP15, CBFN15]. The term message adversary was coined by Afek and Gafni in [AG13]. Whereas the message adversaries in all the work above are oblivious, meaning that they may choose the graph for a round arbitrarily from a fixed set of graphs, [BRS12] and some follow-up work [BRS$^+$15, SWS16] allows arbitrary sequences of communication graphs (that can also model stabilizing behavior, for example).

1.3 Outline

In Chapter 2, we introduce $\mathcal{SMP}$ using the formalism of the heard-of model and extend it with process crashes. We also formally define the consensus problem in a round based setting. Then we briefly introduce the $\mathcal{AMP}$ model. We proceed with Chapter 3, where we motivate our message adversary simulations and how we define a valid simulation. We present a couple of very simple examples motivating the following definitions. The chapter finishes with a formal definition of our message adversary simulations, which will be shown in Chapter 4 to induce a preorder on message adversaries. We present a message adversary simulation simulating STAR on any message adversary allowing consensus to be solved, establishing STAR as a strongest message adversary. In Chapter 5, we explore the relation between a strongest message adversary and a weakest failure detector. In Chapter 6, we circle back to the heard-of model and the predicate emulations defined there and relate our results to the results presented in [SSW18]. We finish this thesis with Chapter 7, where we formally establish an equivalence between multivalued and binary consensus in our model.

CHAPTER 2

Model of computation

2.1 Synchronous message passing ($\mathcal{SMP}$)

Following the notation of [RS13], the abbreviation we use for *synchronous message passing* systems is $\mathcal{SMP}_n[adv : \text{MA}]$. Here, n is the number of processes and MA is the *message adversary*, for example $(\text{SOURCE}, \text{QUORUM})$, or STAR to name a few.

The formal foundation of the model $\mathcal{SMP}_{n,f}$ used here is based on the formalism of the heard-of model, or *HO*model introduced in [CBS09], which is extended to account for process *crash failures*. We study a set of $n > 1$ processes $\{p_1, \dots, p_n\}$ with unique identifiers $\Pi = \{1 \dots n\}$ out of which up to f processes may crash at arbitrary points in time. Failed processes end up in a non-recoverable crashed-state, denoted by $\varnothing$, which we add to the process states. Non-failed processes may exchange messages from a set of messages M, which includes a placeholder $\perp$ indicating the empty message (i.e., no message received). Processes may fail at any time, in particular during the sending operation and therefore only reach a subset of the intended receivers in the round they fail. But we can assume that the sending operation of one message to a single recipient is atomic, i.e., we do not deal with corrupted (partially sent) messages.

Each process $p \in \Pi$ can be described as a 3 tuple

$$\langle states_p \cup \{\varnothing\}, init_p,\ \{(S_p^r, T_p^r) \mid r \geq 1\}\rangle$$

where $states_p$ is the set of states, $init_p \subseteq states_p$ is the set of initial states and $\{(S_p^r, T_p^r) \mid r \geq 1\}$ is the set of message-sending and state-transitioning functions, one per round $r \geq 1$. Round 0 exists and is usually called the *initial configuration*, but S_p^1 and T_p^1 are already responsible for message-sending and state-transitioning.

S_p^r, the message sending function for round r, maps a state and the intended receiver to a unique (possibly $\perp$) message

$$S_p^r : states_p \times \Pi \to M.$$

T_p^r, the state-transition function for round r, maps states and partial vectors of M (the incoming messages or $\perp$ if none, indexed by Π) to the next state

$$T_p^r : states_p \times (M^{\Pi}) \to states_p.$$

Note that S_p^r is responsible for the message p sends to itself, not T_p^r. The collection of processes is called an *algorithm* on Π, which is by definition deterministic. We call an algorithm *uniform* if it is not aware of n (the number of processes in the system), meaning that the same algorithm works for any arbitrary number of n. Conversely, a *non-uniform* algorithm is dependent on the number of processes.

An algorithm tries to solve a given problem by sending messages between processes (one message from every process to every process in every round), but is impaired by the *message adversary* who ultimately decides which sent messages actually arrive at their intended destination. There are different approaches for modelling this behavior; we use a graph-based approach, whereas for example [CBS09] describes the message patterns via predicates.

A *message adversary* is a set of admissible *communication graph sequences* and a *communication graph sequence* σ is an infinite sequence $\sigma = (\mathcal{G}^r)_{r\geq 1}$ of directed graphs $\mathcal{G}^r = (\Pi,\ E_{\mathcal{G}^r})$ (one graph per round), where an edge $(i, j) \in E_{\mathcal{G}^r}$ means that process p_j hears from process p_i in round r. We generally assume that every process hears from itself in every round, which is equivalent to a self-edge for every process in every communication graph $\mathcal{G}^r$. An example for a message adversary would be the tournament message adversary $TOUR$ [RS13]: it consists of the set of all communication graph sequences where, in any round, for any two processes p and q, either the message from p to q or the message from q to p is suppressed, but not both.

Note that we can translate any communication graph sequence into a *heard-of collection*, as defined in [CBS09], and vice versa:

$$HO(p, r) = \{q \mid (q, p) \in E_{\mathcal{G}^r}\} \tag{2.1a}$$
$$E_{\mathcal{G}^r} = \{(p, q) \mid p \in HO(q, r)\} \tag{2.1b}$$

This translation only works assuming that every process sends a message to all other processes in every round. Otherwise, a process not appearing in an HOset could either be due to the message adversary suppressing the communication, or because the algorithm did not send anything in that round. In the case of an algorithm that selectively sends messages, this could be fixed by introducing an empty message $m_{empty} \neq \perp$, indicating that the link is alive, but the process has nothing to say.

Computations in the HO-model consist of infinitely many rounds, which are communication-closed layers in the sense that any message sent in a round can be received only in that round. In each round r, any non-crashed process p first applies S_p^r to the current state $\mathtt{s}_p$ and 'emits' the messages to be sent. After that, p 'collects' all incoming messages from $HO(p,r)$ (the in-neighbors of p in $\mathcal{G}_r$, see Eq. (2.1a)) and applies T_p^r to its current state together with the received messages to compute its next state $\mathtt{s}_p^{'}$.

We call the collection of the round r states

$$C^r = (\mathtt{s}_1, \dots, \mathtt{s}_n), \quad C^r \in (states_1 \cup \{\varnothing\} \times \dots \times states_n \cup \{\varnothing\}) \tag{2.2}$$

at the end of the round (meaning, after application of the state-transition function) the *round r configuration.*

In order to describe the failure pattern for a given run, we define a failure pattern function $F(r)$ (similar to $F(t)$ in [CT96]) mapping a round r to the set of processes which crashed up to that round. If a process p fails in round r', it neither sends any messages in any round $r > r'$ nor does it apply any state transition function T_p^r for $r \geq r'$. However *in* its failing round r', process p may send just a subset of the intended messages before it crashes. We therefore introduce a new failure characteristic function H_F that defines, for each failing process p, which messages it sends in its failing round r':

$$\begin{aligned} F:&\ \mathbb{N} \to 2^{\Pi} \\ H_F:&\ faulty(F) \to 2^{\Pi} \end{aligned} \tag{2.3}$$

We assumed processes cannot crash in their initial state, meaning $F(0) = \emptyset$, and that process failures are non-recoverable, implying $F(r) \subseteq F(r+1)$. We call the set $faulty(F) = \bigcup_{r\in\mathbb{N}} F(r)$ the faulty processes and we call the set $\Pi \setminus faulty(F)$ the correct processes. We also use the discrete derivative $f = \frac{\partial F}{\partial r} = \frac{\partial F}{\partial r} : \mathbb{N} \to 2^{\Pi}$ to map a round r to the processes failing in round r.

Similar to the heard-of model in [CBS09], we call the tuple $(\mathcal{A}, \mathrm{MA})$ a *communication-graph machine* (or shorter *CG-machine*), where $\mathcal{A}$ is an algorithm on Π and MA is a message adversary for Π. A *run* of a CG-machine is a sequence of configurations, one per round, and is completely determined by an initial configuration

$$C^0 \in init_1 \times \dots \times init_n \ ,$$

a communication graph sequence $\sigma \in \mathrm{MA}$, a failure pattern F and a failure characteristic H_F.

Note that [CBS09] uses σ to denote a run, but we already assigned it to communication graph sequences, so here we will use ϵ to denote a run given C^0, $\sigma \in \mathrm{MA}, F, H_F$ and $(\mathcal{A}, \mathrm{MA})$:

$$\epsilon = (C^0, C^1, C^2, \dots) = (C^r)_{r\geq 0} = (\mathcal{A}, \mathrm{MA})_{C^0,\sigma,F,H_F} \tag{2.4}$$

We denote a synchronous message passing system where up to f processes may fail as $\mathcal{SMP}_{n,f}$, but abbreviate the special case with $f = 0$ as just $\mathcal{SMP}_n = \mathcal{SMP}_{n,0}$ with $F(r) = \emptyset$ and $H_F(p) = \emptyset$ for all $p \in \Pi$. If we are reasoning about $\mathcal{SMP}_n$, then we will abbreviate the CG-machine from Eq. (2.4) as $(\mathcal{A}, \mathrm{MA})_{C^0,\sigma}$.

For readability, we define the helper function $\mathcal{M}_p^r(q)$ (in Eq. (2.5)), which provides the message process p receives from process q in round r by combining the message sending function S_q^r with the communication graph σ^r, the failure pattern F and the failure characteristic H_F:

$$\mathcal{M}_p^r(q) = \begin{cases} S_q^r(\epsilon_q^{r-1},\ p) & \text{if } (q,p) \in E_{\sigma^r} \wedge [q \notin F(r) \vee q \in f(r) \wedge p \in H_F(q)] \\ \perp & \text{otherwise.} \end{cases} \tag{2.5}$$

A run ϵ is constructively defined as follows:

$$\epsilon_p^r = \begin{cases} T_p^r(\epsilon_p^{r-1},\ (\mathcal{M}_p^r(1),\ \mathcal{M}_p^r(2),\ \dots\ \mathcal{M}_p^r(n))\ ) & \text{if } r > 0 \wedge p \notin F(r) \\ C_p^0 & \text{if } r = 0 \\ \varnothing & \text{if } p \in F(r)\ . \end{cases} \tag{2.6}$$

We use ϵ_p^r to represent p's state C_p^r after round r in ϵ.

We then define the set of all possible runs $\mathcal{E}_{\mathcal{A},\text{MA}}$ of a given CG-machine $(\mathcal{A}, \text{MA})$ as:

$$\begin{aligned} \mathcal{E}_{\mathcal{A},\text{MA}} = \ \{(\mathcal{A},\text{MA})_{C^0,\sigma,F,H_F} \mid\ & C^0 \in init_1 \times \dots \times init_n, \\ & \sigma \in \text{MA}, \\ & F : \mathbb{N} \to 2^{\Pi}, \\ & H_F : faulty(F) \to 2^{\Pi}\} \ \subseteq \Omega \end{aligned} \tag{2.7}$$

where F and H_F conform to Eq. (2.3), and

$$\Omega = states_1 \cup \{\varnothing\} \times \dots \times states_n \cup \{\varnothing\}$$

contains all possible configurations of the algorithm $\mathcal{A}$, and

$$\Omega = \Omega \times \Omega \times \dots$$

denotes the set of all possible runs of $\mathcal{A}$.

As we later deal with pseudo-code definitions, we can also understand a state C_p^r as a function mapping variables to their values at the end of that round. Formally, if we want to refer to the value of the variable y of process p in round r of run ϵ, we would write it as $\epsilon_p^r.y$, but if it is clear which run ϵ we are talking about, we will use just y_p^r. Additionally, we identify the special vector IN_p^r in every state, which maps each process q to the message p received from q in round $r \geq 1$:

$$\text{IN}_p^r = (\mathcal{M}_p^r(1),\ \mathcal{M}_p^r(2),\ \dots\ \mathcal{M}_p^r(n)) \tag{2.8}$$

For studying information propagation in a sequence of communication graphs $(\mathcal{G}^r)_{r\geq 1}$, the *compound graph* is a very useful concept:

Definition 1 (Compound Graph) *Given two graphs $\mathcal{G} = \langle V, E\rangle$ and $\mathcal{G}' = \langle V, E'\rangle$ with the same vertex-set V, the* compound graph *$\mathcal{G} \circ \mathcal{G}'$ is defined as $\langle V, E''\rangle$, where*

$$(p, q) \in E'' \iff \exists r : (p, r) \in E \land (r, q) \in E'.$$

A compound graph over multiple rounds captures a possible message chain from a process to another process. We use this concept to capture information exchange over multiple rounds.

Definition 2 *By $p \overset{r,r'}{\rightsquigarrow} q$, we express the fact that the communication graph sequence permits a chain of messages from process p to process q starting in round r and ending in round r'. Formally*

$$p \overset{r,r'}{\rightsquigarrow} q \iff (p, q) \in E_{\mathcal{G}^{r+1} \circ \dots \circ \mathcal{G}^{r'}}.$$

With the definition of a run, we now know how to execute an algorithm under any message adversary, failure pattern and characteristic, but we don't know whether said algorithm does what we expect of it. *Problems* (as we encounter them here), essentially map a run of a given algorithm to *true* or *false* depending on whether the given run solved the problem or not. Formally, we will describe a problem through a set of *correct* runs $\mathcal{P}$, where a run is contained in $\mathcal{P}$ if and only if it solves the problem represented by $\mathcal{P}$. We will characterize a problem through a propositional formula ϕ, where a run ϵ solves a problem iff it satisfies said formula $\epsilon \models \phi$. Thus the set of admissible runs can be defined as:

$$\mathcal{P} = \{\epsilon \in \Omega^\omega \mid \epsilon \models \phi\} \tag{2.9}$$

where Ω^ω is the set of all possible runs of all algorithms, i.e., CG-machines. See for example Definition 3 for how the consensus problem is specified.

We say that an algorithm $\mathcal{A}$ solves a problem $\mathcal{P}$ under MA, if the set of all possible runs is contained in $\mathcal{P}$:

$$\mathcal{E}_{\mathcal{A},\mathrm{MA}} \subseteq \mathcal{P}$$

On the other hand, we say that a problem is *impossible* to solve under some model $\mathcal{SMP}_{n,f}[adv : \mathrm{MA}]$, if there is no deterministic algorithm $\mathcal{A}$ such that $\mathcal{E}_{\mathcal{A},\mathrm{MA}} \subseteq \mathcal{P}$. For example, every problem that requires at least some communication among the processes is impossible under the *unrestricted* message adversary ∞, as the sequence $(\mathcal{G}, \mathcal{G}, \dots)$ where the only edges of $\mathcal{G}$ are self-loops, is also in ∞.

2.2 Consensus on $\mathcal{SMP}$

In this thesis, we will primarily focus on the consensus problem tailored for $\mathcal{SMP}_{n,0}$, meaning that we do not account for process failures.

Every process $p \in \Pi$ has an input value $x_p^0 \in V$ from some finite domain V that excludes the special symbol $\perp$, and a decision variable y_p, which is initially $y_p = \perp$. We use the term multi-valued consensus to stress that V is not restricted (but finite) and binary consensus when $|V| = 2$. After a finite number of rounds, all correct processes need to irrevocably decide on a common value from V. We define three propositional formulas, each representing a different aspect of the consensus problem.

(V) Validity: If a process decided on v, then v is the input value to some process:

$$\phi_V := \forall r > 0, \forall p \in \Pi, \forall v \in V : y_p^r = v \rightarrow \exists q \in \Pi : x_q^0 = v \tag{2.10}$$

(A) Agreement: All processes irrevocably decide on the same value v:

$$\phi_A := \forall r, r' > 0, \forall p, q \in \Pi : y_p^r \neq \perp \wedge y_q^{r'} \neq \perp \rightarrow y_p^r = y_q^{r'} \tag{2.11}$$

(T) Termination: All processes eventually irrevocably decide or fail:

$$\phi_T := \forall p \in \Pi, \exists r > 0, \forall r' > r : y_p^{r'} \neq \perp \tag{2.12}$$

Note that we cover irrevocability with agreement and termination $\phi_A \wedge \phi_T$, whereas [CBS09] uses a fourth condition specifying that no process may change its value once decided. With the formal description of validity, agreement and termination in place, we can state the consensus problem:

Definition 3 (Consensus) *Assuming a set Ω^ω representing all possible runs of all algorithms on n processes, we define Σ to contain all traces solving consensus:*

$$\Sigma = \{\epsilon \in \Omega^\omega \ |\epsilon \models \phi_V \wedge \phi_A \wedge \phi_T\} \tag{2.13}$$

with ϕ_V, ϕ_A and ϕ_T as defined in Eq. (2.10)*, Eq.* (2.11) *and Eq.* (2.12)*, respectively.*

Given an actual algorithm $\mathcal{A}$ and a messages adversary MA, we require $\mathcal{E}_{(\mathcal{A},\text{MA})} \subseteq \Sigma$. An equivalent definition would be:

$$\forall \epsilon \in \mathcal{E}_{\mathcal{A},\text{MA}} : \epsilon \models \phi_V \wedge \phi_A \wedge \phi_T$$

which simply postulates that each possible run produced by $\mathcal{A}$ under MA satisfies the consensus properties and thus all runs are a subset of the consensus problem and $\mathcal{A}$ solves consensus.

2.3 Asynchronous message passing ($\mathcal{AMP}$)

The section on $\mathcal{AMP}_{n,f}$ will not be as exhaustive as the section on $\mathcal{SMP}_{n,f}$, as we will not use it in such detail. We assume a finite set of messages M and a distributed system consisting of $n = |\Pi|$ processes, where each process p is again a 3-tuple

$$\langle states_p \cup \{\tilde{\varnothing}\}, init_p, (S_p, T_p)\rangle.$$

$states_p$ is the set of states and $init_p \subseteq states_p$ is the set of initial states. The function T_p

$$T_p : states_p \times A \times 2^M \to states_p$$

maps a state, a failure detector output and a set of received messages to a new state, where A is the failure detector co-domain. The function S_p

$$S_p : states_p \times A \times 2^M \times \Pi \to M$$

maps a state, a failure detector output, a set of received messages, and a recipient to a message to be sent to that recipient. Note that usually just one transition-and-message-sending function is used, but this separation allows for easier modelling.

We assume a global clock with domain $\mathbb{N} = \{0, 1, 2, \dots\}$, which is completely invisible to the processes. Processes in $\mathcal{AMP}$, as in $\mathcal{SMP}_{n,f}$, may crash at any point in time, after which they assume the crash state $\tilde{\varnothing}$. A failure pattern $F : \mathbb{N} \to 2^\Pi$ maps a time t to the set of processes which have already failed up to t. Similarly, we define $crashed(F) = \bigcup_{t\to\infty} F(t)$ as the set of crashed processes and $correct(F) = \Pi \backslash crashed(F)$ as the set of correct processes.

Each process may have access to the read-only output of a local failure detector, which provides the process with additional information about the system [CT96]. Failure detectors are sets of *failure detector histories* $H : \mathbb{N} \times \Pi \to A$, mapping processes and a point in time to some arbitrary value in A, which is usually Π or 2^Π. A weakest failure detector for solving consensus in wait-free environments, where up to $f = n - 1$ processes may crash, is the pair (Σ, Ω) as shown in [CT96].

Definition 4 *The quorum failure detector Σ consists of all histories $H : \mathbb{N} \times \Pi \to 2^\Pi$ where, given the failure pattern F, the following holds*

$$\forall p, q \in \Pi, \forall t, t' \in \mathbb{N} : H(p, t) \cap H(q, t') \neq \emptyset \tag{2.14}$$

and

$$\exists t' \in \mathbb{N}, \forall t > t', \forall p \in \Pi : H(p, t) \subseteq correct(F). \tag{2.15}$$

Definition 5 *The eventual leader failure detector Ω consists of all histories $H : \mathbb{N} \times \Pi \to \Pi$ where, given the failure pattern F, the following holds*

$$\exists q \in correct(F), \exists t' \in \mathbb{N}, \forall t > t', \forall p \in \Pi : H(p, t) = q. \tag{2.16}$$

The weakest failure detector (Σ, Ω) is the combination $H : \mathbb{N} \times \Pi \to 2^\Pi \times \Pi$, of the histories of Σ and Ω.

We represent the states of all processes at a point in time $t \in \mathbb{N}$ in a configuration $\tilde{C}^t$, with

$$\tilde{C}^t = ((\mathsf{s}_1^t, \mathsf{transit}_1^t), \dots, (\mathsf{s}_n^t, \mathsf{transit}_n^t))$$

where $\mathsf{transit}_p$ holds all messages currently in transit for process p, i.e., all messages which will eventually arrive at p. Note that $\mathsf{transit}$ is not part of p's state and is thus invisible to the process.

The adversary is responsible for the specific failure pattern F, the specific failure detector history H of the given failure detector and for advancing each process, i.e., scheduling its steps and choosing the received messages. We model the latter with an infinite sequence of events called a schedule $\tilde{\sigma} : \mathbb{N} \to \Pi \times 2^M$

$$\tilde{\sigma} = (\mathrm{step}(p, \mathrm{IN}_p^1),\ \mathrm{step}(q, \mathrm{IN}_q^2),\ \dots)$$

where $\tilde{\sigma}^1 = \mathrm{step}(p, \mathrm{IN}_p^1)$ denotes that process p receives the messages in the set $\mathrm{IN}_p^1 \subseteq M$ and takes a step at time 1, possibly reading its failure detector $H(p, 1)$. The set of received messages IN_p^1 has to be a subset of the messages currently in transit for p, i.e., $\mathrm{IN}_p^1 \subseteq C_p^0.\mathsf{transit}$ and can be the empty set. Similar to $\mathcal{SMP}$, a step at time t takes the $t-1$ configuration $\tilde{C}^{t-1}$ and maps it to $\tilde{C}^t$.

A schedule has to be *sound*, meaning a crashed process $p \in F(t)$ does not take any steps after time t and processes cannot take an infinite number of steps in a finite time. A schedule also has to be *admissible*, meaning that all correct processes take an *infinite* number of steps and all messages in transit eventually arrive at *correct* recipients.

For readability, we define the helper function $\mathcal{M}_p^t(q)$ (in Eq. (2.17)), which provides the messages process q sends to process p at time t, i.e., the message which is placed in p's transit queue at time t:

$$\mathcal{M}_p^t(q) = S_q(\tilde{\epsilon}_q^{t-1}.\mathsf{s},\ H(q, t),\ \mathrm{IN},\ p). \tag{2.17}$$

Given a failure pattern F, a failure history H, a schedule $\tilde{\sigma}$ and some initial configuration $\tilde{C}^0$, we define a *run* as a sequence of configurations $(\tilde{C}^t)_{t\geq 0}$ with $\tilde{C}^t = (\tilde{\epsilon}_1^t, \dots, \tilde{\epsilon}_n^t)$ where:

$$\begin{aligned} t > 0 : \tilde{\epsilon}_p^t &= \begin{cases} (T_p(\tilde{\epsilon}_p^{t-1}.\mathsf{s},\ H(p,t),\ \mathrm{IN}),\ \tilde{\epsilon}_p^{t-1}.\mathsf{transit} \setminus \mathrm{IN}) & \text{if } \tilde{\sigma}^t = \mathit{step}(p, \mathrm{IN}) \\ (\quad \tilde{\epsilon}_p^{t-1}.\mathsf{s}, & \tilde{\epsilon}_p^{t-1}.\mathsf{transit} \cup \{\mathcal{M}_p^t(q)\}) \quad \text{if } \tilde{\sigma}^t = \mathit{step}(q, \mathrm{IN}) \\ (\tilde{\varnothing}, \emptyset) & \text{if } p \in F(t) \end{cases} \\ \tilde{\epsilon}_p^0 &= \tilde{C}_p^0. \end{aligned}$$

By $AMP_{n,f}[fd : (\Sigma, \Omega)]$, we denote the asynchronous system with n processes of which up to f may crash, enriched with the failure detector (Σ, Ω). This implies that any failure pattern F fulfills $|\mathit{crashed}(F)| \leq f$ and that any corresponding failure detector history H is in $H \in (\Sigma, \Omega)$.

Given an algorithm $\mathcal{A}$, we denote the produced run as

$$\tilde{\epsilon} = (\mathcal{A}, (\Sigma, \Omega))_{F,H,\tilde{\sigma},\tilde{C}^0}$$

and in a similar fashion we write the set of all runs as $\tilde{\mathcal{E}}_{\mathcal{A},(\Sigma,\Omega)}$ without further formalization.

CHAPTER 3

Message Adversary Simulations

Although we introduced $\mathcal{SMP}_{n,f}$ with up to f failing processes in the previous chapter, we will focus on just $\mathcal{SMP}_n = \mathcal{SMP}_{n,0}$ in this chapter. We can think of message adversary simulations as: given an algorithm $\mathcal{A}$ that solves the problem Σ in $\mathcal{SMP}_m[adv : \text{MA}_A]$, we would like to solve Σ in $\mathcal{SMP}_n[adv : \text{MA}_B]$ as well. Instead of rewriting $\mathcal{A}$ to produce a solution under MA_B, we could simulate MA_A on top of MA_B, and let $\mathcal{A}$ solve Σ there. We then take the solution $\mathcal{A}$ delivered under the simulated MA_A and return it for Σ on MA_B.

We assume that the simulated message adversary contains as many processes as the underlying message adversary: $m = n$. We also set that the simulated algorithm has an identical set of id's Π, but we distinguish simulation from simulator by writing $\mathbf{p}$ for a simulated process and p for its simulator. The notation already hints that each process p will simulate $\mathbf{p}$, which also entails that p has to pass the problem-related part of its initial state to $\mathbf{p}$. This one-to-one mapping immediately guarantees that any problem for $\mathcal{SMP}_n[adv : \text{MA}_B]$ is also a problem for the simulated $\mathcal{SMP}_n[adv : \text{MA}_A]$ and vice versa for its solutions.

We will continue denoting the message adversary of the simulated system with MA_A and the message adversary of the underlying system as MA_B. The simulated algorithm is denoted as $\mathcal{A}$, while the simulating algorithm is denoted as $\mathcal{B}$. Similar to process id's, we refer to a round number in the simulated system by $\mathbf{r}$ (also called a *macro* round), while a round in the simulating system is denoted by r (also called a *micro* round). The round r communication graph of the simulator is $(\mathcal{G}^r)_{r\geq 1} \in \text{MA}_B$, while we denote the simulated round $\mathbf{r}$ communication graph as $(\boldsymbol{\mathcal{G}}^{\mathbf{r}})_{\mathbf{r}\geq 1} \in \text{MA}_A$.

3.1 Simple message adversary simulations

First, we need the actual simulation algorithm $\mathcal{B}$ running on MA $_B$, which will simulate $\mathcal{A}$ on MA $_A$. The simulating part itself is not that complicated: recalling the definition of an execution (see Eq. (2.6)), all it takes is the vector of input messages that are to be delivered to $\mathcal{A}$ in the current macro round r and the current state to get to the next state. So a very simple simulation would be to just hand-over all messages received in each round to $\mathcal{A}$ and send all messages produced by its message-sending function:

Algorithm 1: Arguably simplest message adversary simulation, given algorithm $\mathcal{A}$ with $p_i = \langle states_i, init_i, \{(S_i^{\mathsf{r}},\ T_i^{\mathsf{r}}) | \mathsf{r} \geq 1\}\rangle$; and $\mathsf{s}_i^0 \in init_i$; code for process p_i.

```
1 Initially, let $\mathsf{r} := 1$, $\mathsf{simState} := \mathsf{s}_i^0$
  **Loop over micro rounds** $r = 1, 2, \dots$:
2 send $S_i^{\mathsf{r}}(\mathsf{simState}, j)$ to $j$
3 receive all messages as $\mathsf{simMsgIn}$
4 $\mathsf{simState} \leftarrow T_i^{\mathsf{r}}(\mathsf{simState}, \mathsf{simMsgIn})$
5 $\mathsf{r} \leftarrow \mathsf{r} + 1$
```

Of course, this simple simulation works only if MA $_B$ = MA $_A$. The initial value for $\mathsf{simState}$ is abbreviated as $\mathsf{s}_i^0 \in init_i$, some valid initial state of algorithm $\mathcal{A}$ for the simulated process p_i, and the macro round r is equal to the micro round r.

A more sophisticated simulation algorithm is the d-collect simulation: it tags each message m the simulated process sends with the source id i and destination id j: (i, m, j) and then collects all tagged messages in one big message, which is broadcast to everybody. All received sets of tagged messages $\{M_j, M_k, \dots\}$ are merged together and broadcast again. After d rounds the algorithm picks all the tagged messages intended for the simulated process and simulates the next step.

In principle, this is the same as if each process was a router and the routing strategy is to simply flood the network where each message had a time to live of d micro rounds. In formal terms, d-collect contracts the communication graphs of the last d rounds $\mathcal{G}^r \dots \mathcal{G}^{r+d-1}$ into one dense communication graph $\boldsymbol{\mathcal{G}}^{\mathsf{r}}$ and presents that to $\mathcal{SMP}_n[adv : MA_A]$:

$$\boldsymbol{\mathcal{G}}^{\mathsf{r}} = \bigcup_{l=r}^{r+d-1} \mathcal{G}^l \circ \dots \circ \mathcal{G}^r$$

Although it is not a very useful algorithm, d-collect exemplifies a case where the simulated round r is not equal to the simulator round r anymore:

$$\mathsf{r} = \left\lfloor \frac{r-1}{d} \right\rfloor + 1$$

Thus, any macro round has a corresponding micro round where the step from $\mathsf{r} - 1$ to r happens, but not (necessarily) vice versa. In the case of d-collect, the correspondence

Algorithm 2: d-collect message adversary simulation, given algorithm $\mathcal{A}$ with $p_i = \langle states_i, init_i, \{(S_i^{\mathbf{r}}, T_i^{\mathbf{r}}) | \mathbf{r} \geq 1\}\rangle$; and $\mathbf{s}_i^0 \in init_i$; code for process p_i.

```
1 Initially, let $\mathsf{r} := 1$, $\mathsf{simState} := \mathsf{s}_i^0$,
    $\mathsf{collectMsgSet} := \{(i, S_i^1(\mathsf{simState}, 1), 1), \dots, (i, S_i^1(\mathsf{simState}, n), n)\}$
  Loop over micro rounds $r = 1, 2, \dots$:
2 send collectMsgSet to all
3 receive $M_j = \{(k, m, l), \dots\}$ from $j$ in $\mathsf{recvMsgSet} = \{M_1, \dots, M_n\}$
4 $\mathsf{collectMsgSet} \leftarrow \mathsf{collectMsgSet} \cup \{(k, m, l) \mid (k, m, l) \in M_j, M_j \in \mathsf{recvMsgSet}\}$
5 if $r \bmod d = 0$ then
6     $\mathsf{simMsgIn} \leftarrow (m_1, \dots, m_n)$ where $(j, m_j, i) \in \mathsf{collectMsgSet}$
7     $\mathsf{simState} \leftarrow T_i^{\mathsf{r}}(\mathsf{simState}, \mathsf{simMsgIn})$
8     $\mathsf{r} \leftarrow \mathsf{r} + 1$
9     $\mathsf{collectMsgSet} \leftarrow \{(i, S_i^{\mathsf{r}}(\mathsf{simState}, 1), 1), \dots, (i, S_i^{\mathsf{r}}(\mathsf{simState}, n), n)\}$
```

between macro and micro rounds is fixed to a parameter d, but that's not always the case.

In the end, we don't care *when* a macro step happens, as long as it *happens*. Different simulating processes could even take macro steps at different micro rounds, as long as macro rounds are communication closed, meaning that messages sent in round $\mathbf{r}$ only arrive in round $\mathbf{r}$.

Concluding with this informal introduction, we can start formalizing the concepts introduced.

3.2 A formal definition of a message adversary simulation

We require each process p to maintain the following variables with the described semantics:

- $\mathsf{simState}_p^r$: The state of the simulated processor $\mathbf{p}$ at simulator p at the end of micro round r.
- r_p^r: The macro round r_p^r the simulated process $\mathbf{p}$ is currently in at the end of micro round r.

We already modeled the difference between micro rounds and macro rounds; what is missing is a characterization of the micro rounds r where a step in the macro round r happens. Recall that, in one micro round, the macro round may only advance by one, so each macro round has one unique micro round where it is incremented.

We define a mapping between a macro round and the micro round where it was incremented in Definition 6. Intuitively, given macro round $\mathbf{x}$ we collect all micro rounds r where the macro round is equal to the desired macro round plus one $\{r \mid \epsilon_p^r.\mathbf{r} = \mathbf{x}+1\}$ and choose the smallest micro round in that set. Note that if we want the state (at the end) of round r, we look at the state in round $r+1$ because the state transition to the next state $r \to r+1$ happens at the end of round r, i.e., the state for round r is actually valid during round $r+1$, as shown in Fig. 3.1.

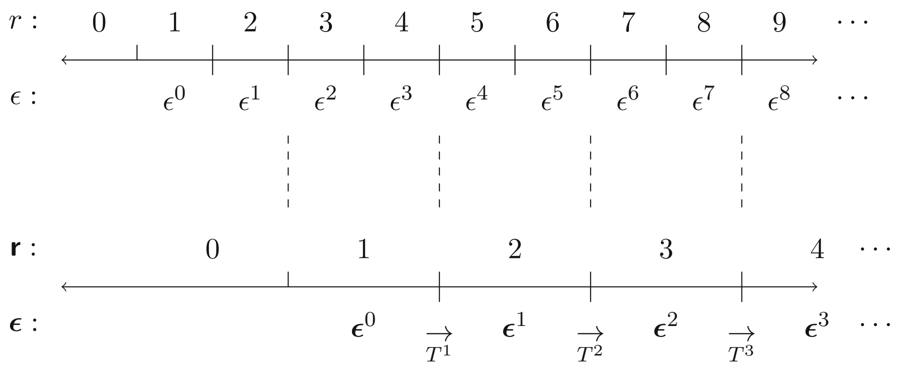

Figure 3.1: The correspondence of rounds and valid states between a run ϵ simulating the simulated run $\boldsymbol{\epsilon}$ of Algorithm 2 with $d = 2$. As an example, we want to access the state in ϵ where the simulated state of the simulated round 2 is valid. By Definition 6 we are looking for $\rho_p^\epsilon(2) = \inf\{6, 7\}$, as both $C^6.\mathbf{r} = 3$ and $C^7.\mathbf{r} = 3$, and both configurations indeed hold $\boldsymbol{\epsilon}^2$. The choice for the smallest is arbitrary.

Definition 6 (Simulated round number) *The function $\rho_p^\epsilon : \mathbb{N} \to \mathbb{N}$ that maps a macro round to the micro round where its state becomes valid in the run ϵ at simulated process p is defined by:*

$$\rho_p^\epsilon(x) = \{\boldsymbol{x} \to \inf(\ \{r \mid \epsilon_p^r.\mathbf{r} = \boldsymbol{x}+1\}\ )\} \tag{3.1}$$

Note that $\rho_p^\epsilon(0) = 0$ as required.

The following Definition 7 features a lot of $\boldsymbol{\epsilon}$'s and ϵ's in tiny subscripts, so lets first look at it intuitively. We have some run of a simulator ϵ, of which we know that it wants to simulate some other algorithm. This means that every process maintains a simulated state simState and a simulated round number $\mathbf{r}$. So what we want to do is take the simulated states simState of all simulated processes $\mathbf{p}$ at the same macro round and build a configuration out of them. As a macro round may correspond to multiple micro rounds, we choose the smallest micro round (through the definition of $\rho_p^\epsilon(x)$) as a representative for that macro round. What is left is to build a sequence of configurations, which gives us a simulated run $\boldsymbol{\epsilon}$ conforming to the definition of a run in Eq. (2.4). Note that the simulated initial state $\boldsymbol{\epsilon}^0$ of the simulated algorithm $\mathcal{A}$ is not the responsibility of the simulator and needs to be set beforehand as part of the configuration of the simulation.

Definition 7 (Simulated Run) *We define a* simulated run $\boldsymbol{\epsilon}$, *atop of the run* ϵ, *as the composition of its simulated states using Definition 6:*

$$(\boldsymbol{\epsilon}^{\mathsf{r}})_{\mathsf{r}\geq 0} = \ ((\mathsf{simState}_1^{\rho_1^\epsilon(\mathsf{r})},\ \ \mathsf{simState}_2^{\rho_2^\epsilon(\mathsf{r})},\ \ \ldots,\ \mathsf{simState}_n^{\rho_n^\epsilon(\mathsf{r})}))_{\mathsf{r}\geq 0} \tag{3.2}$$

Definition 8 formalizes the simulated communication graphs construction via heard-of sets. The set $\boldsymbol{\epsilon}^{\mathsf{r}}_{\mathbf{p}}$. IN contains the messages from the simulated processes that the simulated process $\mathbf{p}$ heard of in round r, which are precisely those messages that were not dropped by the simulated message adversary. All the others have been suppressed and so we place an edge from $\mathbf{q}$ to $\mathbf{p}$ in the simulated communication graph if and only if $\mathbf{p}$ heard of $\mathbf{q}$ in round r, i.e., the message received from $\mathbf{q}$ is not the empty message $\perp$:

Definition 8 (Simulated communication graph) *The* simulated communication graph $\boldsymbol{\mathcal{G}}^{\mathsf{r}} = (\Pi, \boldsymbol{E}^{\mathsf{r}})$, *indicating which messages reached their destination in macro round* r *in the simulated run* $\boldsymbol{\epsilon}$ *of Definition 7, is defined by using the set of received messages* $\boldsymbol{\epsilon}^{\mathsf{r}}_{\boldsymbol{p}}$. *IN of* $\boldsymbol{p}$*:*

$$\boldsymbol{E}^{\mathsf{r}} = \{(\boldsymbol{q}, \boldsymbol{p}) \mid \boldsymbol{p}, \boldsymbol{q} \in \Pi : \boldsymbol{\epsilon}^{\mathsf{r}}_{\boldsymbol{p}}.\,IN(\boldsymbol{q}) \neq \perp\}. \tag{3.3}$$

What is left is the definition of a message adversary simulation, given the algorithm $\mathcal{B}_{\mathcal{A}}$ (i.e., the simulator $\mathcal{B}$ parameterized with the algorithm $\mathcal{A}$) that produces the run ϵ. Conceptually, it is rather straight-forward: a run ϵ of the simulator $\mathcal{B}$ correctly simulated $\mathcal{A}$ if $(\boldsymbol{\epsilon}^{\mathsf{r}})_{\mathsf{r}\geq 0}$ according to Definition 7 is identical to the run produced by $\mathcal{A}$ on a 'real' system. By requiring that $\rho_p^\epsilon(x)$ has to be defined for all x, we ensure that macro rounds are only incremented once in a micro round (Definition 6). Since we know the implied simulated communication graph sequence (Definition 8) and the simulated initial state $(\boldsymbol{\epsilon}^0)$, we simply build a reference run and compare it to the simulated run (Definition 7).

Definition 9 (Message adversary simulation) *Consider any algorithm* $\mathcal{A}$ *for the message adversary* MA_A, *for any initial state* $\boldsymbol{\epsilon}^0$. *A CG-machine* $(\mathcal{B}_{\mathcal{A}}, \mathrm{MA}_B)$ *is a* message adversary simulation, *if any run* $\epsilon \in (\mathcal{B}_{\mathcal{A}}, \mathrm{MA}_B)$ *satisfies the following properties:*

1. *First, we require that* $\rho_p^\epsilon(\mathsf{r})$ *is defined and strictly monotonic for all* $\mathsf{r} \in \mathbb{N}$ *on all processes* p*:*

$$\xi := \forall \mathsf{r} \in \mathbb{N} : \forall p \in \Pi :\ \ \rho_p^\epsilon(\mathsf{r}+1) > \rho_p^\epsilon(\mathsf{r}) \geq 0 \tag{3.4}$$

2. *Second, we postulate that the simulated communication graph sequence* $\boldsymbol{\mathcal{G}}$ *implied by the simulated run* $\boldsymbol{\epsilon}$ *conforms to the message adversary specification* MA_A, *i.e., it is a valid communication graph sequence:*

$$\phi := (\boldsymbol{\mathcal{G}}^{\mathsf{r}})_{\mathsf{r}\geq 0} \in \mathrm{MA}_A \tag{3.5}$$

3. *Finally, we require the simulated run $\boldsymbol{\epsilon}$ to be a valid run of algorithm $\mathcal{A}$ on* MA_A*: the run produced by the CG-machine when started in the simulated initial state $\boldsymbol{\epsilon}^0$ with the simulated communication graph sequence $\boldsymbol{\mathcal{G}}$ has to be equal to the simulated run $\boldsymbol{\epsilon}$:*

$$\psi := (\boldsymbol{\epsilon}^{\mathsf{r}})_{\mathsf{r}\geq 0} = \ (\mathcal{A}, \mathrm{MA}_A)_{\boldsymbol{\epsilon}^0, \boldsymbol{\mathcal{G}}} \tag{3.6}$$

$\mathcal{B}_\mathcal{A}$ *on* MA_B *is a message adversary simulation for* $\mathcal{A}$ *on* MA_A*, denoted by* $\mathrm{MA}_A \triangleright_{\mathcal{B}_\mathcal{A}} \mathrm{MA}_B$*, if every run of* $(\mathcal{B}_\mathcal{A}, \mathrm{MA}_B)$ *models* ξ*,* ϕ *and* ψ*:*

$$\mathrm{MA}_A \triangleright_{\mathcal{B}_\mathcal{A}} \mathrm{MA}_B \Leftrightarrow \ \forall \epsilon \in \mathcal{E}_{\mathcal{B}_\mathcal{A}, \mathrm{MA}_B} : \ \epsilon \models \xi \wedge \phi \wedge \psi \tag{3.7}$$

An aspect we did not formalize is the case of simulated initial state $\boldsymbol{\epsilon}^0$ versus the initial state of $\mathcal{B}_\mathcal{A}$. We motivated message adversary simulations by reasoning that we could solve a problem by simulating another algorithm $\mathcal{A}$ to solve the problem for us. This requires the simulated algorithm to start from a valid initial state $\boldsymbol{\epsilon}^0$ (both with respect to the problem we want to solve, and with respect to $\mathcal{A}$), because if the simulated initial states are not valid, the result is not valid.

We did not formalize how to get from ϵ^0 to $\boldsymbol{\epsilon}^0$, because in the end the algorithm $\mathcal{B}_\mathcal{A}$ is responsible for that. If $\boldsymbol{\epsilon}^0$, which is implied by some ϵ^0, would lead to a non-valid solution of the simulated algorithm for the given problem (and thus a non-valid solution of the simulator), then the simulator is simply no solution to the problem. However, the definition of a message adversary simulation itself is not concerned with this.

CHAPTER 4

The Strongest Message Adversary for Consensus

In Chapter 3, we defined the concept of a message adversary simulation that allows us to run an algorithm $\mathcal{A}$ (specified to run on MA$_A$) on top of another message adversary it was not specified for (here, MA$_B$) via the simulator $\mathcal{B}$. Now let us set aside the simulated algorithm $\mathcal{A}$ together with the problem it is supposed to solve and focus on the fact that the message adversary MA$_B$ allowed the simulation of message adversary MA$_A$. This already seems like the relation between message adversaries we sought: we could say that MA$_A$ is *stronger* than MA$_B$, because MA$_B$ allows solving a problem via simulating MA$_A$. The intuition is that MA$_A$ allows for less messages to be exchanged than MA$_B$ and thus admits less information flow, while on the other hand MA$_B$ allows for more messages to be exchanged than necessary to solve the given problem.

We are still considering the special case of $f = 0$ failing processes. After specifying a relation using the definitions presented in Chapter 3, we provide a simulation for the simple message adversary STAR on $\mathcal{SMP}_{n,0}$. We then argue that STAR is a strongest message adversary for consensus: STAR allows to solve consensus and any other message adversary that allows to solve consensus can simulate STAR.

4.1 A relation for simulation power over message adversaries

First we have to tackle the problem that the Definition 9 of a message adversary simulation is bound to the actual algorithm $\mathcal{A}$ that is simulated. Thus, in order to define a relation just between message adversaries (via a simulator $\mathcal{B}$), we could say MA$_A$ is stronger than MA$_B$ if and only if for every $\mathcal{A}$ on MA$_A$ there is a simulator $\mathcal{B}$ on MA$_B$ such that $\mathcal{B}_{\mathcal{A}}$ is a message adversary simulation:

Definition 10 (Message adversary simulation-power relation) *Given two message adversaries* MA_A *and* MA_B*, we say* MA_A *is stronger than* MA_B*, denoted* $\mathrm{MA}_A \triangleright \mathrm{MA}_B$*, if*

$$\mathrm{MA}_A \triangleright \mathrm{MA}_B \Leftrightarrow \exists \mathcal{B} \text{ on } \mathrm{MA}_B : \forall \mathcal{A} \text{ on } \mathrm{MA}_A : \; \mathrm{MA}_A \triangleright_{\mathcal{B}_\mathcal{A}} \mathrm{MA}_B$$

Theorem 1 *The message adversary relation (Definition 10)* $\triangleright$ *is a pre-order.*

Proof *Reflexivity* is trivially satisfied, for example, by using the previously introduced Algorithm 1:

$$\mathrm{MA}_A \triangleright \mathrm{MA}_A$$

For *Transitivity*, we assume that

$$\mathrm{MA}_A \triangleright \mathrm{MA}_B \text{ and } \mathrm{MA}_B \triangleright \mathrm{MA}_C$$

with the message adversary simulations $\mathcal{B}$ and $\mathcal{C}$ for the message adversaries MA_B and MA_C respectively. Now we can simply simulate MA_A in the simulation of MA_B using $\mathcal{B}$ on top of $\mathcal{C}$ resulting in the CG-machine $(\mathcal{C}_{\mathcal{B}_\mathcal{A}}, \mathrm{MA}_C)$. Consequently we can simulate any algorithm $\mathcal{A}$ for MA_A on MA_C and

$$\mathrm{MA}_A \triangleright \mathrm{MA}_C$$

follows. □

4.2 A message adversary simulation for $\mathcal{SMP}$

With the message adversary relation in Definition 10, we have a means of relating message adversaries in a similar fashion as is done with failure detectors [CT96, JT08]. As already mentioned at the beginning of this chapter, our simulator simulates the message adversary STAR on every message adversary allowing consensus:

Definition 11 (The message adversary STAR**)** *Every admissible communication graph sequence has one distinct center process, which broadcasts to all other processes in every round. No other communication happens. Formally:*

$$\mathrm{STAR} = \{(\mathcal{G}^r)_{r\geq 1} \mid \exists p_c \in \Pi, \forall r \geq 1 : \; E_{\mathcal{G}^r} = \{(p_c, q) \mid q \in \Pi\}\}$$

The idea of the simulator is that every process p not only simulates the process $\mathbf{p}$ it is assigned to, but also the center process $\mathbf{p}_c$. This is possible, since the center process never receives any messages in STAR but only sends messages (and therefore $\mathbf{p}_c$'s state stays consistent on all simulators). So as soon as everybody agreed on the center process $\mathbf{p}_c$, the sequence of states of $\mathbf{p}_c$ in the run is completely determined and so are its sent messages. Thus, with the initial state of the center process $\mathbf{p}_c$, every simulator p can simulate the messages sent from the center $\mathbf{p}_c$ to its simulated process $\mathbf{p}$ without any actual communication from $\mathbf{p}_c$'s simulator to p.

The remaining challenge is to distribute, or rather agree, on the initial state of the center process $\mathbf{p}_c$. The wording already hints the solution, as we will simply run multivalued consensus on the simulated initial states: after a finite number of rounds, all the simulators will have agreed on the initial state of a common center process $\mathbf{p}_c$. Each process q then simulates both $\mathbf{p}_c$ and $\mathbf{q}$ in lockstep.

Algorithm 3: simStar$_{\mathcal{A}}$: Simulating algorithm
$\mathcal{A} = \langle states_i, init_i, \mathsf{r}, \{(S_i^{\mathsf{r}}, T_i^{\mathsf{r}}) \mid \mathsf{r} > 0\}\rangle$
for STAR, on top of a message adversary where a multi-valued consensus algorithm
$\mathcal{C} = \langle states'_i, init'_i, \mathsf{r}', \{(\mathcal{S}_i^{\mathsf{r}'}, \mathcal{T}_i^{\mathsf{r}'}) | \mathsf{r}' > 0\}\rangle$
exists, code for process i with $\mathbf{s}_i^0 \in init_i$.

```
1  Initially, let $\mathsf{r} := 1$, simState := $\mathsf{s}_0^i$, simCenterState := $\bot$, simCenterId := $\bot$, $\mathsf{r}' := 1$,
     simConsensusState.$x$ := (simState, $i$), simConsensusState.$y$ := $\bot$
   Loop over rounds $1, 2, \ldots$:
2  send $\mathcal{S}_i^{\mathsf{r}'}$(simConsensusState, $j$) to $j \in \Pi$
3  receive all messages as simConsensusMsgIn
4  simConsensusState $\leftarrow \mathcal{T}_i^{\mathsf{r}'}$(simConsensusState, simConsensusMsgIn)
5  if simConsensusState.$y \neq \bot \land \mathsf{r} = 1$ then
6      (simCenterState, simCenterId) $\leftarrow$ simConsensusState.$y$
7  $\mathsf{r}' \leftarrow \mathsf{r}' + 1$
8  if simConsensusState.$y \neq \bot$ then
9      simCenterMsgIn[simCenterId] $\leftarrow S^{\mathsf{r}}_{\mathsf{simCenterId}}$(simCenterState, simCenterId)
10     simCenterState $\leftarrow T^{\mathsf{r}}_{\mathsf{simCenterId}}$(simCenterState, simCenterMsgIn)
11     if simCenterId $\neq i$ then
12         simMsgIn[simCenterId]  $\leftarrow S^{\mathsf{r}}_{\mathsf{simCenterId}}$(simCenterState, $i$)
13         simMsgIn[$i$]          $\leftarrow S_i^{\mathsf{r}}$(simState, $i$)
14         simState $\leftarrow T_i^{\mathsf{r}}$(simState, simMsgIn)
15     else
16         simState $\leftarrow$ simCenterState
17     $\mathsf{r} \leftarrow \mathsf{r} + 1$
```

The algorithm does not just simulate $\mathcal{A}$ but also the consensus algorithm $\mathcal{C}$ in parallel. As soon as $\mathcal{C}$ reaches a consensus on the simulated initial state (and its process id), simStar starts simulating $\mathcal{A}$. In preparation of proving that simStar simulates STAR, we prove that simStar correctly simulates the consensus algorithm $\mathcal{C}$ on MA, with all the guarantees listed in Definition 3 of the consensus problem:

Theorem 2 simStar$_{\mathcal{C}}$ *is a message adversary simulation for the consensus algorithm* $\mathcal{C}$ *for* MA *on the underlying message adversary* MA*:*

$$\mathrm{MA} \rhd_{\mathrm{simStar}_{\mathcal{C}}} \mathrm{MA}$$

Proof We assume a run $\epsilon = (\mathrm{simStar}_{\mathcal{C}}, \mathrm{MA})_{C^0,\sigma}$ of Algorithm 3 simulating $\mathcal{A} = \mathcal{C}$ on the communication graph sequence $\sigma \in \mathrm{MA}$ of a message adversary MA that allows running $\mathcal{C}$, and alias the required variables:

$$\mathsf{simConsensusState} \to \mathsf{simState} \tag{4.1a}$$
$$\mathbf{r}' \to \mathbf{r} \tag{4.1b}$$

We can then apply Definition 6, Definition 7 and Definition 8 to define the simulated run $\boldsymbol{\epsilon}$ and the simulated communication graph sequence $\boldsymbol{\mathcal{G}}$.

1. $\rho_p^\epsilon(\mathbf{r}')$ is defined for all $\mathbf{r}' \geq 0$ and for all $p \in \Pi$, as Line 7 increments $\mathbf{r}'$ in every micro round, starting in micro round 1:

$$\rho_p^\epsilon(\mathbf{r}') = \mathbf{r}'$$

   This already defines the simulated run and the set of received messages as:

$$\boldsymbol{\epsilon}_{\mathbf{p}}^{\mathbf{r}'} = \epsilon_p^{\rho_p^\epsilon(\mathbf{r}')}.\mathsf{simConsensusState} \tag{4.2a}$$
$$\boldsymbol{\epsilon}_{\mathbf{p}}^{\mathbf{r}'}.\,\mathrm{IN} = \epsilon_p^{\rho_p^\epsilon(\mathbf{r}')}.\mathsf{simConsensusMsgIn},\ \ \mathbf{r}' \geq 1. \tag{4.2b}$$

2. Each process $\mathbf{p}$ receives the same messages in Line 4 as does its simulator p in Line 3 (because $\mathsf{simConsensusMsgIn}$ is not modified in between), and all messages process $\mathbf{q}$ intends to send to $\mathbf{p}$ are sent in Line 2, implying:

$$\boldsymbol{\mathcal{G}} = \sigma. \tag{4.3}$$

3. Third, we prove that in every macro round $\mathbf{r}'$ the configurations in the reference run $\xi = (\mathcal{C}, \mathrm{MA})_{\boldsymbol{\epsilon}^0,\boldsymbol{\mathcal{G}}}$ and the simulated run $\boldsymbol{\epsilon}$ are equal

$$\begin{aligned}\forall \mathbf{r}' > 0 : \mathcal{T}_p^{\mathbf{r}'}(\boldsymbol{\epsilon}^{\mathbf{r}'-1}, (\boldsymbol{\epsilon}^{\mathbf{r}'}.\,\mathrm{IN}(1),\ \ldots\ \boldsymbol{\epsilon}^{\mathbf{r}'}.\,\mathrm{IN}(n))\ ) =& \\ \boldsymbol{\epsilon}_p^{\mathbf{r}'} &= \xi_p^{\mathbf{r}'} \\ &= \mathcal{T}_p^{\mathbf{r}'}(\xi^{\mathbf{r}'-1}, (\mathcal{M}_p^r(1),\ \ldots\ \mathcal{M}_p^r(n))\ )\end{aligned} \tag{4.4}$$

   The base case $\boldsymbol{\epsilon}^0 = \xi^0$ holds by definition, for the step we assume Eq. (4.4) is satisfied. As $\mathsf{simConsensusState}$ is only written in Line 4, it is sufficient to show that any process $\mathbf{p}$ receives the same messages in the simulated $\boldsymbol{\epsilon}$ and in the reference run ξ:

$$\xi_{\mathbf{p}}^{\mathbf{r}'+1}.\mathrm{IN} = \boldsymbol{\epsilon}_{\mathbf{p}}^{\mathbf{r}'+1}.\mathrm{IN} = \epsilon_p^{\rho_p^\epsilon(\mathbf{r}'+1)}.\mathsf{simConsensusMsgIn},\ \mathbf{r}' \geq 0$$

By the induction hypothesis Eq. (4.4), the same set of messages is sent to $\mathbf{p}$ in both $\boldsymbol{\epsilon}^{\mathbf{r}'+1}$ (Line 2) and $\xi^{\mathbf{r}'+1}$. Since both $\boldsymbol{\epsilon}^{\mathbf{r}'}$ and $\xi^{\mathbf{r}'}$ use the same $\boldsymbol{\mathcal{G}}^{\mathbf{r}'+1}$, the same set of messages arrives in Line 3 and further in $\xi_{\mathbf{p}}^{\mathbf{r}'+1}.\mathrm{IN}$ and $\boldsymbol{\epsilon}_{\mathbf{p}}^{\mathbf{r}'+1}.\mathrm{IN}$. Thus we get $\xi_{\mathbf{p}}^{\mathbf{r}'+1} = \boldsymbol{\epsilon}_{\mathbf{p}}^{\mathbf{r}'+1}$ as needed.

□

simStar$_{\mathcal{C}}$ is hence indeed a message adversary simulation for the consensus algorithm $\mathcal{C}$ on MA, which allows us to postulate the consensus properties given in Definition 3 on the simulated state simCenterState. Now we can prove:

Theorem 3 simStar$_{\mathcal{C}}$ *is a message adversary simulation for any algorithm* $\mathcal{A}$ *for the message adversary* STAR*, on any message adversary* MA *that allows to solve (multivalued) consensus:*

$$\mathrm{STAR} \rhd_{\mathrm{simStar}_{\mathcal{A}}} \mathrm{MA}$$

implying:

$$\mathrm{STAR} \rhd \mathrm{MA}$$

Proof We assume a run $\epsilon = (\mathrm{simStar}_{\mathcal{A}}, \mathrm{MA})_{C^0,\sigma}$ of Algorithm 3 for $\mathcal{A}$ on the communication graph sequence $\sigma \in \mathrm{MA}$ of a message adversary MA allowing consensus.

We can directly apply Definition 6, Definition 7 and Definition 8 to define the simulated run $\boldsymbol{\epsilon}$ and the simulated communication graph sequence $\boldsymbol{\mathcal{G}}$.

1. Observe that $\mathbf{r}$ is incremented every micro round iff simConsensusState.y is not $\perp$. We know from Theorem 2 that simConsensusState adheres to the consensus properties, specifically termination (see Eq. (2.12)), implying that

$$\forall p \in \Pi, \exists r : \epsilon_p^r.\mathsf{simConsensusState}(y) \neq \perp$$

and therefore we can define

$$\rho_p^\epsilon(\mathbf{r}) = r_p^0 + \mathbf{r}$$

where r_p^0 is the smallest r on process p such that $\epsilon_p^r.\mathsf{simConsensusState}(y) \neq \perp$ holds.

2. As an intermediate result, we show that $\mathsf{simCenterState}$ and $\mathsf{simCenterId}$ are identical on each process in every macro round $\mathbf{r}$:

$$\begin{aligned}\forall \mathbf{r} \geq 0, \forall p, q \in \Pi : \epsilon_p^{\rho_p^\epsilon(\mathbf{r})}.\mathsf{simCenterState} &= \epsilon_q^{\rho_q^\epsilon(\mathbf{r})}.\mathsf{simCenterState} \wedge \\ \epsilon_p^{\rho_p^\epsilon(\mathbf{r})}.\mathsf{simCenterId} &= \epsilon_q^{\rho_q^\epsilon(\mathbf{r})}.\mathsf{simCenterId}\end{aligned} \tag{4.5}$$

The base case for $\rho_p^\epsilon(0) = r_p^0$ is the micro round r_p^0 where $\mathcal{C}$ *just* finished at p and Theorem 2 guarantees us agreement on each process q at Line 6 in r_q^0. In the same micro round, $\mathsf{simCenterState}$ is written again in Line 10 and assigned the evaluated state-transition function together with the message sent to itself. As this evaluation is exclusively parameterized by $\mathsf{simCenterState}$ and $\mathsf{simCenterId}$, of which we know equality over all processes at this point, we conclude equality holds after Line 10. Note that, in round r_p^0, the variable $\mathbf{r}$ is incremented and thus Line 6 is never executed again.

The induction step also follows from the previous reasoning: the transition function in Line 10 is evaluated with by-assumption equal states and thus maps to the same states on each process.

Theorem 2 also guarantees us validity, which in conjunction with the initial state of the simulated consensus algorithm implies that $\mathsf{simCenterState}$ is equal to some simulated initial state and $\mathsf{simCenterId}$ is always equal to the same simulated process id:

$$\begin{aligned}\exists \mathbf{p} \in \Pi, \forall \mathbf{r} \geq 1, \forall q \in \Pi : \epsilon_q^{\rho_q^\epsilon(\mathbf{r})}.\mathsf{simCenterState} &= \epsilon_p^{\rho_p^\epsilon(\mathbf{r})}.\mathsf{simState} \wedge \\ \epsilon_q^{\rho_q^\epsilon(\mathbf{r})}.\mathsf{simCenterId} &= \mathbf{p}\end{aligned}$$

3. Here we argue that the simulated communication graph sequence $\boldsymbol{\mathcal{G}}$ is a star with the process $\mathbf{p}_c = \mathsf{simCenterId}$ in the middle, i.e.,:

$$\begin{aligned}\exists \mathbf{p}_c \in \Pi, \forall \mathbf{r} > 0, \forall \mathbf{q} \in \Pi : \boldsymbol{\epsilon}_{\mathbf{q}}^{\mathbf{r}}.\,\mathrm{IN}[\mathbf{p}_c] &= S_{\mathbf{p}_c}^{\mathbf{r}}(\boldsymbol{\epsilon}_{\mathbf{p}_c}^{\mathbf{r}-1}, \mathbf{q}) \wedge \\ \boldsymbol{\epsilon}_{\mathbf{q}}^{\mathbf{r}}.\,\mathrm{IN}[\mathbf{q}] &= S_{\mathbf{q}}^{\mathbf{r}}(\boldsymbol{\epsilon}_{\mathbf{q}}^{\mathbf{r}-1}, \mathbf{q})\end{aligned} \tag{4.6}$$

holds on $\boldsymbol{\epsilon}$.

Assume a process $\mathbf{p}$ with $\mathsf{simCenterId} \neq i$ and observe that its next state is evaluated in Line 14 with exactly two received messages: one from the center process $\mathsf{simCenterState}$ and one from itself assembled in Line 13.

The state of the process $\mathbf{p}_c$ with $\mathsf{simCenterId} = i$ is assigned in Line 16 but evaluated in Line 10 with just the message from itself.

This corresponds to Eq. (4.6), and thus $\boldsymbol{\mathcal{G}}$ is a star with $\mathbf{p}_c = \mathsf{simCenterId}$ as the center process.

4. Finally, we prove that the simulated run $\boldsymbol{\epsilon}$ is equal to the reference run ξ:

$$(\boldsymbol{\epsilon}^{\mathbf{r}})_{\mathbf{r} \geq 0} = (\xi^{\mathbf{r}})_{\mathbf{r} \geq 0} = (\mathcal{A}, \mathrm{STAR})_{\boldsymbol{\epsilon}^0, \boldsymbol{\mathcal{G}}}$$

The base case $\boldsymbol{\epsilon}^0 = \xi^0$ holds by definition, the induction hypothesis is:

$$\boldsymbol{\epsilon}^{\mathbf{r}} = \xi^{\mathbf{r}}$$

The simulated state simState is only written once per round in either Line 14 or Line 16 (and, more specifically, evaluated in Line 10), which are mutually exclusive. Therefore it is sufficient to show that each process $\mathbf{p}$ receives the same set of messages in the reference and in the simulated run.

For process $\mathbf{p} \neq \mathbf{p}_c$ we know that in the reference run it receives exactly two messages, one from $\mathbf{p}_c$ and one from itself, $\mathbf{p}$. We already showed that

$$\epsilon_{\mathbf{p}}^{\rho_p^\epsilon(\mathbf{r})}.\mathsf{simCenterId} = \mathbf{p}_c$$

and

$$\epsilon_{\mathbf{p}}^{\rho_p^\epsilon(\mathbf{r})}.\mathsf{simCenterState} = \boldsymbol{\epsilon}_{\mathbf{p}_c}^{\mathbf{r}},$$

so the simulated received message from $\mathbf{p}_c$ in Line 13 is identical to the message $\mathbf{p}_c$ sends to $\mathbf{p}$ in the reference run (because the message sending function is deterministic and evaluated on identical states). Additionally, simMsgIn contains the message $\mathbf{p}$ sends to itself.

For $\mathbf{p} = \mathbf{p}_c$, the center of the star's simulated received messages are calculated in Line 9, its simState is assigned in Line 16, but effectively evaluated in Line 10. Thus, it is again sufficient to prove that the set of simulated received messages in the reference run $\xi_{\mathbf{p}_c}$ is exactly simMsgIn in Line 10, which is evident as both contain just the message from $\mathbf{p}_c$ to itself.

□

4.3 The strongest message adversaries for consensus

Theorem 2 and Theorem 3 imply that any message adversary MA that allows to solve consensus via some algorithm $\mathcal{C}$ immediately allows the simulation of the message adversary STAR. Furthermore, the proof of Theorem 3 was completely independent of the actual algorithm $\mathcal{A}$ being simulated, so we can claim that Algorithm 3 simulates any algorithm $\mathcal{A}$ for the message adversary STAR on any message adversary MA allowing consensus, hence:

$$\mathrm{STAR} \triangleright \mathrm{MA},$$

i.e., STAR is a strongest message adversary.

Formally, by Definition 10, specifically Eq. (3.5), we can also simulate any message adversary MA that fully contains STAR $\subseteq$ MA. This follows from the argument that any algorithm running on MA runs correctly on *any* communication graph sequence $\sigma \in$ MA. So in order to simulate some other algorithm on a different message adversary, it is sufficient to simulate just one communication graph sequence in MA.

As a consequence, every message adversary fully containing STAR is in the set of strongest message adversaries S:

$$\{\mathrm{MA} \in \Omega_\epsilon \mid \mathrm{STAR} \subseteq \mathrm{MA}\} \subseteq S, \tag{4.7}$$

assuming a set Ω_ϵ of all possible message adversaries that allow to solve consensus on a set of Π process identifiers. In particular, the combination (SOURCE, QUORUM) of the message adversaries SOURCE and QUORUM (introduced in Chapter 5), is in S.

S also contains any message adversary which can be simulated in STAR and allows consensus. Consider for example the message adversary STAR$_1$ containing just n communication graph sequences $G_{\mathrm{STAR},1,q}, q \in \Pi$, where $G_{\mathrm{STAR},1,q}$ contains a star in just the first round and no communication after that:

$$E_{G^1_{\mathrm{STAR},1,q}} = \{(q,p) \mid p \in \Pi\} \tag{4.8a}$$

$$r > 1 : E_{G^r_{\mathrm{STAR},1,q}} = \emptyset. \tag{4.8b}$$

Obviously, STAR$_1$ allows to solve consensus and STAR$_1$ can be simulated in STAR. This implies that STAR$_1$ can trivially simulate STAR via, for example, simStar and is therefore in S. There are a lot of other, similar message adversaries in S, so a more concise characterization of S is *any* MA *that allows consensus* and *can be simulated in* STAR.

Theorem 4 *The set of strongest message adversaries S is*

$$S = \{\mathrm{MA} \in \Omega_\epsilon \mid \mathrm{MA} \triangleright \mathrm{STAR}\}, \tag{4.9}$$

i.e., any message adversary simulate-able in STAR *and allowing consensus.*

An intuition for a strongest message adversary MA is not necessarily that it makes it difficult to write a consensus algorithm for. On the other hand, this does not mean that the message adversary with the most complicated consensus algorithm could not also be a strongest message adversary. Unfortunately, we do not have a metric for complicatedness of algorithms (yet). The goal of this thesis was hence isolating a minimum amount of information exchange necessary for solving consensus. An intuition for *strongest* message adversary would be any message adversary unrestricted enough to also just allow some one to all communication, which we identified as the minimum necessary for consensus.

Note that this does not imply that any message adversary containing the necessary minimum allows solving consensus. Think of the completely unrestricted message adversary, which trivially contains STAR, but does not allow consensus.

Similarly, not every message adversary allowing consensus to be solved necessarily contains a strongest message adversary. Consider the perfect message adversary, where the only communication graph sequence is just a sequence of fully connected graphs: it allows consensus to be solved but does not contain STAR (and is thus not a strongest message adversary). This also follows the intuition that the perfect message adversary allows *more* information exchange than necessary: reliable all-to-all communication is more than the minimum required for consensus and thus it is not a strongest message adversary.

CHAPTER 5

Message Adversary STAR and the Failure Detector (Σ, Ω)

Having specified the class of strongest message adversaries for consensus and its specific member STAR, we explore its relation to the weakest failure detector (Σ, Ω) for consensus in $\mathcal{AMP}$. Comparing two formal models with respect to their problem solving capabilities is usually done by simulating one model on top of the other, but so far we have only simulated different flavors of the same formal model, for example $\mathcal{SMP}_{n,0}[adv : \text{MA}]$ on $\mathcal{SMP}_{n,0}[adv : \text{MA}']$. Comparing $\mathcal{SMP}_{n,0}$ with $\mathcal{AMP}_{n,n-1}$ requires a new notion of a *correct simulation*, as processes in $\mathcal{AMP}_{n,n-1}$ may simply crash, while this is not allowed in $\mathcal{SMP}_{n,0}$.

[RS13], for example, tackled this issue by using a restricted definition of problems that works in both models. Problems turn into *tasks*, defined as an input/output relation Δ relating input vectors I to output vectors O. Each input vector contains one initial value per process and similarly each output vector O contains one accepted problem-solving value per process. The relation Δ then matches input vectors I to output vectors $O \in \Delta(I)$, with the semantics that if all processes adhere to I, the problem is solved iff for some $O \in \Delta(I)$ every process i either at some point matches $O[i]$ or i crashed. The important part with respect to faults is that processes may crash instead of matching the solution vector.

[RS13] also already showed an equivalence between the message adversary $(\text{SOURCE}, \text{QUORUM})$ in a synchronous setting and the weakest failure detector for consensus (Σ, Ω) in an asynchronous setting. As $(\text{SOURCE}, \text{QUORUM})$ is a strongest message adversary for consensus (and thus allows consensus to be solved), we expect STAR to be simulatable in (Σ, Ω) and thus confirm the results presented in [RS13].

5.1 Simulating STAR in (Σ, Ω)

We start by introducing the message adversary (SOURCE, QUORUM), which is composed by the intersection of the sets in Definition 12 and Definition 13.

Definition 12 *The message adversary* SOURCE *is the set of all communication graph sequences* $(\mathcal{G}^r)_{r>0}$*, where*

$$\exists p \in \Pi : \exists r' > 0 : \forall r \geq r' : \forall q \in \Pi : (p, q) \in E_{\mathcal{G}^r}.$$

I.e., there exists a process p *which eventually always reaches all other processes in every round.*

Definition 13 *The message adversary* QUORUM *is the set of all communication graph sequences* $(\mathcal{G}^r)_{r>0}$*, where*

$$\forall p, q \in \Pi : \forall r, r' > 0 : \{x \mid (x, p) \in E_{\mathcal{G}^r}\} \cap \{x \mid (x, q) \in E_{\mathcal{G}^{r'}}\} \neq \emptyset.$$

I.e., any two processes p*,* q*, share in any two rounds at least one common in-neighbor.*

We proceed with introducing a new notion of f*-crash compatible* message adversaries. The idea is that if, starting from some round r^0, a process q in $\mathcal{SMP}_{n,0}[adv : \text{MA}]$ is permanently suppressed in some $\sigma \in \text{MA}$ (meaning that all its outgoing messages are always dropped), then the rest of the processes $\Pi \setminus \{q\}$ do not care if q is dead or still running. So, given a concrete failure pattern F and a failure history H_F for $\mathcal{SMP}_{n,f}$, we might be able to restrict the message adversary MA to a subset MA_{F,H_F}, such that each communication graph sequence $\sigma \in \text{MA}_{F,H_F}$ exactly hides the process crashes in F and H_F, i.e., suppresses all messages from crashed processes once they crashed. Then, assuming consensus can be solved under MA on $\mathcal{SMP}_{n,0}[adv : \text{MA}]$, it can also be solved under $\text{MA}_{F,H_F} \subseteq \text{MA}$ on $\mathcal{SMP}_{n,f}[adv : \text{MA}_{F,H_F}]$.

Definition 14 *We call a message adversary* MA f*-crash compatible if, for each failure pattern* F *and corresponding failure characteristics* H_F*, the* F, H_F-masking set

$$\text{MA}_{F,H_F} = \{\mathcal{G} \in \text{MA} \mid \phi(\mathcal{G}, F, H_F)\}$$

is non-empty. We define ϕ *as*

$$\begin{aligned}&\phi(\mathcal{G}, F, H_F) := \forall p \in \Pi, \forall r \geq 1 : \\ &\quad ((p \in F(r) \land p \notin f = \frac{\partial F}{\partial r}(r)) \to (\forall q \in \Pi : (p, q) \notin E_{\mathcal{G}^r})\land \\ &\quad\quad ((p \in f = \frac{\partial F}{\partial r}(r)) \to (\forall q \in \Pi : (p, q) \in E_{\mathcal{G}^r} \to q \in H_F(p))),\end{aligned} \tag{5.1}$$

recall that $f = \frac{\partial F}{\partial r}(r)$ *is the discrete derivative of* $F(r)$*, i.e., maps a round* r *to the processes failing in round* r*.*

For example, the message adversary (SOURCE, QUORUM) is $(n-1)$-crash compatible: in any failure pattern F and any failure characteristic H_F, there is at least one process p_c which does not fail. As (SOURCE, QUORUM) contains STAR, we know that at least the star with the non-crashed process as the center is in the set MA_{F,H_F}, making it non-empty and (SOURCE, QUORUM) $(n-1)$-crash compatible.

This allows us to formalize the previous intuition in Theorem 5, where we relate the state sequences of all processes in a run on $\mathcal{SMP}_{n,f}$ to the same processes in $\mathcal{SMP}_{n,0}$ with the same communication graph sequence σ up to the round where they crash.

Theorem 5 *Given a failure pattern F, a failure characteristic H_F, any run ϵ*

$$\epsilon = (\mathcal{A}, \mathrm{MA}_{F,H_F})_{C^0,\sigma,F,H_F}$$

of any algorithm $\mathcal{A}$ on some $\sigma \in \mathrm{MA}_{F,H_F} \subseteq \mathrm{MA}$ in $\mathcal{SMP}_{n,f}[adv : \mathrm{MA}_{F,H_F}]$, of the f-crash compatible message adversary MA, has an equivalent run η

$$\eta = (\mathcal{A}, \mathrm{MA})_{C^0,\sigma}$$

with the same σ in crash-free $\mathcal{SMP}_n[adv : \mathrm{MA}]$, in the sense that all processes behave identical (same states in the same rounds) up to the round where they crash, i.e.,

$$\forall r > 0, \forall p \in \Pi : \epsilon_p^r = \eta_p^r \vee \epsilon_p^r = \varnothing. \tag{5.2}$$

Note that, as correct processes never crash, they exhibit the exact same state sequence $(\epsilon_p^r)_{r \geq 0}$ in $\mathcal{SMP}$ as in $\mathcal{SMP}_{n,f}$.

Proof Assume a failure pattern F, a failure characteristic H_F, a f-crash compatible message adversary implying some run $\epsilon = (\mathcal{A}, \mathrm{MA}_{F,H_F})_{C^0,\sigma,F,H_F}$ of any algorithm $\mathcal{A}$ on some $\sigma \in \mathrm{MA}_{F,H_F} \subseteq \mathrm{MA}$. We also assume a run $\eta = (\mathcal{A}, \mathrm{MA})_{C^0,\sigma}$ of the same algorithm on the same communication graph sequence σ and on identical initial states.

We prove the invariant in Eq. (5.2) by induction. The base case $r = 0$ holds, as we assumed identical initial states.

For the induction step we assume the invariant holds in some round $r-1$, already implying that every process sends the same messages in both runs ϵ and η, and we only have to show that in both runs the same messages arrive. Assume a message m from p arrives at q in round r in run ϵ, implying that $(p,q) \in E_\sigma$ and therefore the same message also arrives in η.

Now assume the message m from p arrives at q in round r in run η. By Eq. (5.1) we know that:

$$((p \in F(r) \wedge p \notin f = \frac{\partial F}{\partial r}(r)) \rightarrow ((p,q) \notin E_{\sigma^r})) \wedge ((p \in f = \frac{\partial F}{\partial r}(r)) \rightarrow ((p,q) \in E_{\sigma^r} \rightarrow q \in H_F(p)))$$

holds on σ, implying, as $(p,q) \in E_{\sigma^r}$, that p has not crashed up to round r, or just crashes in round r. As $p \notin F(r)$ implies that the message m also arrives in ϵ, we assume that $p \in f = \frac{\partial F}{\partial r}(r)$, which means, by the rightmost implication in the expression above, that $q \in H_F(p)$ and m arrives at q in this case as well. □

Having specified (SOURCE, QUORUM) and the concept of an f-crash tolerant message adversary, we cite the simulator from [RS13, Theorem 5] in Algorithm 4 and slightly adapt it to our notation. It simulates $\mathcal{SMP}_{n,0}[adv : (\text{SOURCE}, \text{QUORUM})]$ on $\mathcal{AMP}_{n,n-1}[fd : (\Sigma, \Omega)]$ by basically just waiting for all messages of the processes which are currently in the quorum and the process which is currently the leader. As both the leader and the processes in the quorum are eventually correct, their messages eventually arrive and the simulator makes a step. The resulting simulated communication graph sequence then lies in (SOURCE, QUORUM).

Algorithm 4: The simulator from [RS13, Theorem 5] (slightly adapted to match our notation), simulating $\mathcal{A} = \langle states_i, init_i, r_i, \{(S_i^{r_i}, T_i^{r_i}) \mid r_i > 0\}\rangle$ for $\mathcal{SMP}_{n,0}[adv : (\text{SOURCE}, \text{QUORUM})]$ on $\mathcal{AMP}_{n,n-1}[fd : (\Sigma, \Omega)]$. Here, qr_i is the quorum and ld_i is the current leader, provided by (Σ, Ω). Code for process p_i.

```
$r_i := 1$, $sim\_rec\_messages_i[1,\dots,n] := [\bot,\dots,\bot]$, $ls\_state_i := \mathbf{s}_i^0$
foreach $r > 0$ do
    $rec\_messages_i[r][1,\dots,n] \leftarrow [\bot,\dots,\bot]$
repeat forever
    $msgs\_to\_send_i[1,\dots,n] := [S_i^{r_i}(ls\_state_i, 1),\dots,S_i^{r_i}(ls\_state_i, n)]$
    foreach $j \in \Pi$ do
        send$((r_i, msgs\_to\_send_i[j]))$ to $p_j$
    repeat
        $cur\_qr_i \leftarrow qr_i$
        $cur\_ld_i \leftarrow leader_i$
    until $(\forall j \in cur\_qr_i \setminus \{i\} : rec\_messages_i[r_i][j] \neq \bot) \wedge$
          $(cur\_ld_i = i \vee rec\_messages[r_i][cur\_ld_i] \neq \bot)$
    foreach $j \in cur\_qr_i \cup cur\_ld_i$ do
        $sim\_rec\_messages_i[j] \leftarrow rec\_messages_i[r_i][j]$
    $ls\_state_i \leftarrow T_i^{r_i}(ls\_state_i, sim\_rec\_messages_i)$
    $r_i \leftarrow r_i + 1$
when $(r, m)$ received from $p_j$:  $rec\_msgs_i[r][j] \leftarrow m$.
```

Theorem 6 *For any simulated run ϵ of $\mathcal{A}$ in Algorithm 4, with the failure history F, the failure characteristic H_F, the simulated communication graph sequence $\mathcal{G} \in$ MA and an initial configuration C^0:*

$$\epsilon = (\mathcal{A}, (\text{SOURCE}, \text{QUORUM}))_{C^0, \mathcal{G}, F, H_F}$$

the communication graph sequence $\boldsymbol{\mathcal{G}}$ is in the F, H_F-masking set of (SOURCE, QUORUM)

$$\boldsymbol{\mathcal{G}} \in (\text{SOURCE}, \text{QUORUM})_{F,H_F}.$$

Proof We assume a run $\tilde{\epsilon} = (\tilde{C}^t)_{t \geq 0}$ of Algorithm 4 and define the function $\tilde{\rho}_p^{\tilde{\epsilon}} : \mathbb{N} \to \mathbb{N}$ mapping a round $x \geq 0$ to the time t where it was incremented:

$$\tilde{\rho}_p^{\tilde{\epsilon}}(x) = \{x \to \inf\{t \mid \tilde{C}_p^t.r = x + 1\}\},$$

which we abbreviate with $\tilde{\rho}_p(x)$ for readability. The simulated run $\boldsymbol{\epsilon}$ is then defined as

$$\boldsymbol{\epsilon}_p^r = C_p^r = \tilde{C}_p^{\tilde{\rho}_p(r)}.ls\_state.$$

We leave the proof of correctness, i.e., that

$$\boldsymbol{\epsilon} = (\mathcal{A}, (\text{SOURCE}, \text{QUORUM}))_{C^0, \boldsymbol{\mathcal{G}}, F, H_F}$$

to the authors in [RS13], and focus on the properties of the simulated communication graph sequence.

Let the function $\phi : \Pi \to \mathbb{N}$ map a process p to the time t where it last incremented its r_p (i.e., just before failing):

$$\phi(p) = \sup\{t \in \mathbb{N} \mid \tilde{C}_p^{t-1}.r + 1 = \tilde{C}_p^t.r\}$$

and denote the round where p crashes by $crash_p = \tilde{C}_p^{\phi(p)}.r$. Note that $\phi(p) = crash_p = \infty$ as required if p does not crash. The failure history F and the failure characteristic H_F of the simulated run $\boldsymbol{\epsilon}$ can then be specified as

$$\begin{aligned} F(r) : &\ \{q \in \Pi \mid r \geq \phi(q)\} \\ H_F(p) : &\ \{q \in \Pi \mid \tilde{C}_q^{\tilde{\rho}_q(crash_p)}.sim\_rec\_messages[p] \neq \perp\} \end{aligned} \tag{5.3}$$

By Line 14 the simulated communication graph sequence $(\boldsymbol{\mathcal{G}}^r)_{r \geq 1}$ is

$$\mathbf{E}^r = \{(p, q) \mid p, q \in \Pi : \tilde{C}_q^{\tilde{\rho}_q(r)}.sim\_rec\_messages[p] \neq \perp\}. \tag{5.4}$$

We now show that the simulated communication graph sequence $\boldsymbol{\mathcal{G}}$ is in the F, H_F-masking set of (SOURCE, QUORUM):

$$\boldsymbol{\mathcal{G}} \in (\text{SOURCE}, \text{QUORUM})_{F,H_F}.$$

Assume $\boldsymbol{\mathcal{G}} \notin (\text{SOURCE}, \text{QUORUM})_{F,H_F}$, then there exists an edge from a crashed process p to q in some simulated round r^0. We consider the first case where $p \in F(r^0)$ but $p \notin f = \frac{\partial F}{\partial r}(r^0)$, i.e., p crashed in some earlier round $\phi(p) < r^0$. This implies that p sends no message with the round tag r^0, contradicting Eq. (5.4), where q has received a message from p in the simulated round r^0. In the second case, where $p \in f = \frac{\partial F}{\partial r}(r^0)$, we know that $q \notin H_F(p)$ by assumption, implying

$$\tilde{C}_q^{\tilde{\rho}_q(crash_p)}.sim\_rec\_messages[p] = \perp$$

This again contradicts Eq. (5.4) where, by assumption, q has received a message from p in round $crash_p$.

□

We can conclude with Theorem 7, which claims that the weakest failure detector (Σ, Ω) in $\mathcal{AMP}$ can simulate the strongest message adversary STAR in $\mathcal{SMP}$ with respect to general task solvability.

Theorem 7 *Any task can be solved in the enriched message-passing model*

$$\mathcal{AMP}_{n,n-1}[fd : (\Sigma, \Omega)]$$

if it can be solved in the synchronous model

$$\mathcal{SMP}_{n,0}[adv : \text{STAR}].$$

Proof By [RS13, Theorem 5] together with Theorem 6, we know that Algorithm 4 simulates any algorithm $\mathcal{A}$ on the message adversary $(\text{SOURCE}, \text{QUORUM})_{F,H_F}$. I.e., the simulated run $\boldsymbol{\epsilon}$ of algorithm $\mathcal{A}$ simulated by Algorithm 4 is equal to

$$\boldsymbol{\epsilon} = (\mathcal{A}, (\text{SOURCE}, \text{QUORUM})_{F,H_F})_{C^0, \boldsymbol{\mathcal{G}}, F, H_F},$$

with F, H_F and $\boldsymbol{\mathcal{G}}$ depending on the particular run of Algorithm 4.

Because $\boldsymbol{\epsilon}$ is parameterised with the F, H_F masking set $(\text{SOURCE}, \text{QUORUM})_{F,H_F}$, Theorem 5 guarantees us identical execution of correct processes in $\boldsymbol{\epsilon}$ and in ϵ, where

$$\epsilon = (\mathcal{A}, (\text{SOURCE}, \text{QUORUM}))_{C^0, \mathcal{G}}$$

and $\mathcal{G} = \boldsymbol{\mathcal{G}}$, i.e., both runs use the same communication graph sequence.

This immediately grants us task solvability using any algorithm $\mathcal{A}'$ for STAR. Algorithm 4, on $\mathcal{AMP}_{n,n-1}[fd : (\Sigma, \Omega)]$, simulates Algorithm 3, on $\mathcal{SMP}_{n,n-1}[adv : (\text{SOURCE}, \text{QUORUM})_{F,H_F}]$. We know that the correct processes in Algorithm 3 behave as if they run on $\mathcal{SMP}_n[adv : (\text{SOURCE}, \text{QUORUM})]$, meaning they correctly simulate $\mathcal{A}'$ on STAR. As there are still processes crashing (silently), this implies task solvability but not problem solvability. This is because, in general, problems for $\mathcal{SMP}$ do not allow process crashes, but tasks do.

□

As expected at the beginning of this chapter, STAR may be simulated in (Σ, Ω) with respect to task solvability. STAR being a strongest message adversary, this result implies that any strongest message adversary is simulatable in (Σ, Ω), as it can be simulated in STAR.

CHAPTER 6

Consequences of our Results

6.1 Comparing message adversary simulations to other definitions

We already mentioned that message adversary simulations have been defined in [SSW18] already, but not as formal as in this thesis.

Definition 15 ([SSW18, Page 114]) *We say that A* emulates *(macro-)rounds $\rho \in \{1, 2, \dots\}$ of $\mathcal{SMP}_n[adv : M]$, if, in any run of the latter, the value of $NewHO_p^{(\rho)}$ computed at the end of macro-round ρ satisfies:*

(E1) $q \in NewHO_p^{(\rho)}$ iff $\mathsf{s}_q^{r_1 - 1} \rightsquigarrow \mathsf{s}_p^{r_k}$, i.e., if there exist an integer $\ell \in [1, k]$, a chain of $\ell + 1$ processes $p_0, p_1, \dots, p_\ell$ from $p_0 = q$ to $p_\ell = p$, and a subsequence of ℓ increasing round numbers $r_1, \dots, r_\ell$ in macro-round ρ such that, for any index i, $1 \leq i \leq \ell$, we have $p_{i-1} \in HO(p_i, r_i)$.

(E2) The collection $NewHO_p^{(\rho)}$ for all $p \in \Pi, \rho > 0$ satisfies M.

This definition already sounds very similar to the d-collect message adversary simulation in Algorithm 2 and, without proof, we claim that d-collect is also a message adversary emulation according to (E1) and (E2). We now examine the differences between the emulation presented in [SSW18] and the simulation in Definition 9, and we will argue that any run classified as a message adversary simulation according (E1) and (E2) is also valid message adversary simulation by Definition 9. This shows that our definition captures the definition in [SSW18] completely.

We start by identifying our required variables simState and $\mathbf{r}$ in a given run ϵ, which according to (E1) and (E2) is a message adversary simulation. As A emulates $\mathcal{A}$ [1], it

[1]The paper also uses $\mathcal{A}$ to denote the simulated algorithm

needs to keep $\mathcal{A}$'s state somewhere and we simply alias that to our simState. The variable $\mathbf{r}$ does not exist, but the paper uses the unspecified function $k(p, \rho)$ to map macro rounds ρ to micro rounds at process p:

$$r_1 = k(p, \rho - 1) + 1, r_2 = k(p, \rho - 1) + 2, \ldots, r_k = k(p, \rho)$$

The paper does not explicitly mention in which round the macro state is valid, so we simply fix $\rho_p^\epsilon(\mathbf{r}) = k(p, \mathbf{r})$.

Given a run ϵ satisfying (E2), we know that the communication graph $\boldsymbol{\mathcal{G}}$ built by the $\boldsymbol{\epsilon}_{\mathbf{p}}^{\rho_\epsilon(\mathbf{r})}.IN = NewHO_p^{k(p,\mathbf{r})}$ sets is a valid communication graph sequence in the message adversary M. This already fulfills our Definition 8, as $NewHO$ is the set of simulated received messages (i.e., IN), and thus satisfies Eq. (3.5) of Definition 9.

Next we need to show, by induction, that the simulated run $(\boldsymbol{\epsilon}_{\mathbf{p}}^{\mathbf{r}})_{\mathbf{r}\geq 0}$ produced by ϵ_p adheres to Eq. (3.6) of Definition 9. We assume that the simulated initial configuration $\boldsymbol{\epsilon}^0$ in ϵ is a valid initial configuration (i.e., $\boldsymbol{\epsilon}^0 \in init_1 \times \cdots \times init_n$ of the simulated algorithm $\mathcal{A}$) and we compare $\boldsymbol{\epsilon}$ to the reference run ξ given by $\xi = (\mathcal{A}, M)_{\boldsymbol{\epsilon}^0, \boldsymbol{\mathcal{G}}}$.

The induction step is, assuming for a round $\mathbf{r}$ that the configurations in our reference run $\xi^{\mathbf{r}}$ and the simulated run $\boldsymbol{\epsilon}^{\mathbf{r}}$ are equal, so are the configurations in round $\mathbf{r} + 1$. Assuming that the simulated process $\mathbf{p}$ in the simulated run $\boldsymbol{\epsilon}$ receives a message in macro round $\mathbf{r}$ from $\mathbf{q}$, we know by (E1) that there exists a message chain from $\mathbf{q}$ to $\mathbf{p}$ and the message lands in $NewHO_{\mathbf{p}}$. Therefore (by construction of the simulated communication graph of round $\mathbf{r}$) the message from $\mathbf{q}$ to $\mathbf{p}$ is not suppressed in the reference run ξ. On the other hand, since $\xi^{\mathbf{r}}$ and $\boldsymbol{\epsilon}^{\mathbf{r}}$ are equal by assumption, the reference run and the simulated run send the same messages and therefore the same messages arrive in $\xi^{\mathbf{r}}$ and in $\boldsymbol{\epsilon}^{\mathbf{r}}$. The state-transition function is deterministic and has to map to the same succeeding states:

$$\xi^{\mathbf{r}+1} = \boldsymbol{\epsilon}^{\mathbf{r}+1}$$

So if a run is a simulation according to (E1), (E2) it is also equal to our reference run ξ and thus a message adversary simulation according to Definition 9.

6.2 The strongest MA and the heard-of model

The heard-of model [CBS09] characterizes different systems only via so-called communication predicates and does not require definitions of message adversaries, crash-histories or any other model specific attributes. Communication predicates simply state which process hears from which other processes while performing a step at a given point in time, where time is defined as a global entity not available to the processes. This abstraction allows the comparison of many different systems, for example comparing the problem solving capabilities of some message adversary on $\mathcal{SMP}$ with some failure detector combined with a round based algorithm on $\mathcal{AMP}$.

This relation between predicates is established via predicate emulators. Similar to message adversary simulations presented here, they simulate a different predicate that specifies the simulated received messages of each simulated process. Our simulated message adversary STAR can be expressed in the heard-of model, as

$$\mathcal{P}_{\text{STAR}} = \exists p_c \in \Pi : \forall q \in \Pi : \forall r \geq 1 : HO(q,r) = \{p_c, q\}$$

like any other message adversary for $\mathcal{SMP}$. We thus originally intended to express our results in the heard-of model, but Algorithm 3 turned out *not* to be a valid predicate emulation.

This is due to criteria (E1) [CBS09, p. 55], which is essentially the same as condition (E1) in [SSW18] that we mention in Definition 15 in Section 6.1. It is a validity condition postulating that if a message arrives at the simulated process $\mathbf{p}$, sent from $\mathbf{q}$, then a message chain exists from the sending simulator q to the receiving simulator p, which is not necessarily the case in Algorithm 3.

The validity of *our* message adversary simulation is instead guaranteed by the equivalence of the simulated run to a reference run of a CG-machine, for the same initial states, with the identical communication graph sequence. This, as we argue in Section 6.1, also captures the criterion (E1), in the sense that any $\mathcal{SMP}$ algorithm that is a predicate emulation according to (E1) and (E2), is a message adversary simulation according to Definition 9, but not necessarily vice versa.

In order to turn Algorithm 3 into a proper predicate translation, [SSW18] modified (E1) to just requiring one message to reach all processes, thus acquiring 'initial knowledge' (and therefore being able to locally compute further messages). We will instead state the validity condition as *'any received message has to have been sent before'* and argue that Algorithm 3 is a predicate emulation according to (E1'), defined as

(E1') If $m \in \text{IN}_{\mathbf{p}}^{\mathbf{r}}$ then $m = \mathcal{M}_{\mathbf{p}}^{\mathbf{r}}(\mathbf{q})$.

This formalization requires the existence of two things: the set of received messages $\text{IN}_{\mathbf{p}}^{\mathbf{r}}$ in the simulated round $\mathbf{r}$ and the message receiving function $\mathcal{M}_{\mathbf{p}}^{\mathbf{r}}(\mathbf{q})$, i.e., the message $\mathbf{q}$ sends to $\mathbf{p}$ in the simulated round $\mathbf{r}$, have to be defined on each simulator. We argue that the two definitions required are present on any predicate emulation, but not necessarily easy to define.

Note that both validity conditions (E1') and the original (E1) do not imply a *correct* emulation, as in both cases the simulator could tamper with the simulated state during the simulation, making it non-correct. While the original (E1) allows a *valid* emulation to still be a *non-correct* emulation, i.e., an emulator could provide a forged message but still be valid because a message chain exists, (E1') is necessary for a correct emulation.

We already presented two simulators which provide IN and $\mathcal{M}$, the first is Algorithm 3 and the second is Algorithm 4 in Section 5.1 that simulates $\mathcal{SMP}$ in $\mathcal{AMP}$. For Algorithm 3, we can take the required definitions directly from the proof of Theorem 3, where we showed that it is a message adversary simulation, which, according to Section 6.1, already implies that it is a predicate emulation. For Algorithm 4, the (trivial) message receiving function is defined by

$$\mathcal{M}_{\mathbf{p}}^{\mathbf{r}}(\mathbf{q}) = S_{\mathbf{q}}^{\mathbf{r}}(\tilde{C}_q^{\rho_q(\mathbf{r})}.ls\_state, \mathbf{p})$$

and the set of received messages is defined by

$$\mathrm{IN}_{\mathbf{p}}^{\mathbf{r}} = \tilde{C}_q^{\rho_q(\mathbf{r})}.sim\_rec\_messages.$$

Last we show that Algorithm 4 fulfills (E2) in Definition 15 and our modified (E1'). Theorem 6 proved that the simulated communication graph sequence satisfies the predicate of $(\mathrm{SOURCE}, \mathrm{QUORUM})_{F,H_F}$, so we only need to show that a simulated received message was actually sent before. As Line 7 sends only messages from the simulated process, and Line 14 builds the simulated receive set exclusively out of received messages, any simulated received message has to have been sent before, implying (E1').

The modified validity condition (E1') captures Algorithm 3 but still allows some valid emulations to be non-correct, i.e., every emulation where the emulated state is modified in between some steps. An even better validity condition would be the configuration-by-configuration comparison to a reference run, which is exactly what a message adversary simulation does. This means that (E1') is, without loss of generality, necessary for a correct run, but not sufficient. In essence, the choice of the validity condition used is a trade-of between amount of work to prove it and the required formal correctness of the proven simulated run.

6.3 Paradoxes

[SSW18] pointed out a seemingly paradoxical result of our findings, as they proved that the message adversary VSSC(∞) (which allows to solve consensus) cannot simulate the failure detector Σ in $\mathcal{AMP}$ and hence cannot simulate the weakest failure detector (Σ, Ω) either. On the other hand, their results also showed that VSSC(∞) allows to simulate STAR and that it is trivial to simulate (Σ, Ω) in $\mathcal{AMP}$ atop of STAR, a contradiction! The problem could be traced back to the fact that the simulator for STAR does not necessarily provide a strongly correct process of the underlying system as the center process. We have not introduced strongly correct up to now, so we provide an informal definition here.

Definition 16 (Strongly correct process) *A process is called* strongly correct *iff its messages reach all other processes infinitely often.*

This implied that the simulation of (Σ, Ω) atop of STAR is not a valid simulation, as the set of simulated strongly correct processes (the center of the star) does not necessarily coincide with the set of strongly correct processes of the underlying system. This comes back to the initial validity condition (E1) in Definition 15, which implies that, if a simulated process transfers information, the simulator needs to transfer information.

This assumption is also crucial in their proof that VSSC(∞) cannot simulate the failure detector Σ, making it sort of the contradictory core of the previously mentioned paradox.

This thesis aimed at never assuming the validity condition (E1). As already mentioned, we establish validity of a simulation by comparison to a reference run on an equivalently parameterized CG-machine. Our simulation also does not necessarily provide a strongly correct process of the underlying system as the center of the star; the simulated algorithm is completely oblivious of that fact, making it correct. Algorithm 3 works because, by agreeing on a simulated process' state, it already agrees on any future message sent in the whole network as the network layout is fixed from the beginning.

This feature is arguably the beauty of Algorithm 3. Short lived stability in a system, just long enough to solve consensus, is sufficient to guarantee a forever property, namely 'center-to-all' communication. This could also be seen as the ability of the simulator to simply define a different simulated failure pattern than the underlying failure pattern, as the simulators can 'un-crash' a crashed simulator. It does require relinquishing the very intuitive assumption that repeated simulated communication requires repeated underlying communication, but, as the simulated algorithm obviously cannot tell the difference, it is without alternative.

CHAPTER 7

Multi-Valued Consensus from Binary Consensus

In this chapter, we extend the applicability of Theorem 3 to any message adversary allowing just binary consensus, by reducing multi-valued consensus to binary consensus in $\mathcal{SMP}adv : \text{MA}$. We achieve this with Algorithm 5, which solves multi-valued consensus given a binary consensus algorithm. Whereas such a reduction is known for classic synchronous distributed systems with byzantine faults [TC84], it has been introduced in the message adversary setting only in Winkler's recent PhD thesis [Win19]. We will adapt the algorithm presented there to our framework and decrease its exponential space complexity to just a linear one.

Algorithm 5 intuitively operates as follows: each process concurrently simulates $n+1$ instances $C_1, \ldots, C_{n+1}$ of the binary consensus algorithm and at the same time floods all initial values on the network. The inputs $I_1, \ldots, I_{n+1}$ of the $n+1$ consensus instances (interpreted as vectors), are configured in such a way that I_k and I_{k+1} differ by just one bitflip, as shown in Table 7.1. As the first instance starts with I_1, which only contains 0's, it has to decide on zero. Similarly, as the last instance similarly starts with I_{n+1}, which only contains 1's, it has to decide on one. Therefore there exists at least one pair of instances C_j, C_{j+1} where a single bitflip in the initial vectors I_j, I_{j+1} changed the decided value, implying that the message from the process which flipped that bit must have reached all other processes. As the initial multi-valued consensus values are piggybacked on each message from the binary consensus algorithm, we know that also the multi-valued consensus value had to reach every process and we simply choose that value.

The following lemma proves that Algorithm 5 is correct.

Lemma 1 *Binary consensus can be solved in a model* $\mathcal{SMP}_n[adv : \text{MA}]$ *if and only if*

Algorithm 5: MultiValuedConsensus: Multi-valued consensus for initial value v_i and decision variable y_i using a binary consensus algorithm $\mathcal{C} = \langle states_i, init_i, \{(S_i^{\mathbf{r}},\ T_i^{\mathbf{r}}) | \mathbf{r} \geq 1\}\rangle$ with the initial state $s_i^0 \in init_i$. Code for process p_i.

```
Initially, let $\mathbf{r} := 1$, $\mathsf{x} := v_i$, $\mathsf{setIn}[1, \dots, n] := \bot$, $\mathsf{setIn}[i] := \mathsf{x}$,
  $\mathsf{consensusState}[1 \dots n+1] := s_i^0$
$\mathsf{consensusState}[1 \dots i].\mathsf{x} := 0$
$\mathsf{consensusState}[i+1 \dots n+1].\mathsf{x} := 1$
$\mathsf{consensusMsgIn}[1 \dots n][1 \dots n+1] := \bot$
foreach $l \in \{1 \dots n\}, k \in \{1 \dots n+1\}$ do
    $\mathsf{consensusMsgOut}[l][k] \leftarrow S_i^1(\mathsf{consensusState}[k], l)$
Loop over rounds $1, 2, \dots$:
send $(\mathsf{setIn}, \mathsf{consensusMsgOut}[l])$ to $l$
receive $(\mathsf{setIn}_j, \mathsf{consensusMsgRecv}_j)$ from $j$
$\mathsf{consensusMsgIn}[1 \dots n][1 \dots n+1] := \bot$
foreach $(\mathsf{setIn}_j, \mathsf{consensusMsgRecv}_j)$ received from $j$ do
    foreach $l \in \{1 \dots n\} : \mathsf{setIn}_j[l] \neq \bot$ do
        $\mathsf{setIn}[l] \leftarrow \mathsf{setIn}_j[l]$      /* maintain array of known input values */
    $\mathsf{consensusMsgIn}[j] \leftarrow \mathsf{consensusMsgRecv}_j$
foreach $k \in \{1 \dots n+1\}$ do
    $\mathsf{consensusState}[k] \leftarrow$
      $T_i^{\mathbf{r}}(\mathsf{consensusState}[k], (\mathsf{consensusMsgIn}[1][k], \dots, \mathsf{consensusMsgIn}[n][k]))$
if $\forall k \in \{1 \dots n+1\} : \mathsf{consensusState}[k].y \neq \bot$ then
    for $k = 1$ to $n$ do
        if $\mathsf{consensusState}[k].y = 0 \wedge \mathsf{consensusState}[k+1].y = 1$ then
            $\mathsf{y}_i \leftarrow \mathsf{setIn}[k]$
            break
$\mathbf{r} \leftarrow \mathbf{r} + 1$
foreach $l \in \{1 \dots n\}, k \in \{1 \dots n+1\}$ do
    $\mathsf{consensusMsgOut}[l][k] \leftarrow S_l^{\mathbf{r}}(\mathsf{consensusState}[k], l)$
```

multi-valued consensus can be solved in $\mathcal{SMP}_n[adv : \mathrm{MA}]$.

We split the proof into two parts. The first part only shows that Algorithm 5 is a message adversary simulation for the binary consensus algorithm $\mathcal{C}$ for each of its $n+1$ instances. The second part then proves the multi-valued consensus property.

Proof We assume a run $\epsilon = (\mathrm{MultiValuedConsensus}, \mathrm{MA})_{C^0,\sigma}$ of Algorithm 5 simulating the $k \in \{1 \dots n+1\}$ instance of $\mathcal{C}$ on the communication graph sequence $\sigma \in \mathrm{MA}$ and alias the required variables:

$$\mathsf{consensusState}[k] \rightarrow \mathsf{simState}. \tag{7.1}$$

We can then apply Definition 6, Definition 7 and Definition 8 to define the simulated run $\boldsymbol{\epsilon}$ and the simulated communication graph sequence $\boldsymbol{\mathcal{G}}$.

1. $\rho_p^\epsilon(\mathsf{r})$ is defined for all $\mathsf{r} \geq 0$ and for all $p \in \Pi$ as Line 21 increments r in every micro round, starting in micro round 1:

$$\rho_p^\epsilon(\mathsf{r}) = \mathsf{r}$$

   This already defines the simulated run and the set of received messages for consensus instance k as:

$$\boldsymbol{\epsilon}_{\mathbf{p}}^{\mathsf{r}} = \epsilon_p^{\rho_p^\epsilon(\mathsf{r})}.\mathsf{consensusState}[k] \tag{7.2a}$$

$$\boldsymbol{\epsilon}_{\mathbf{p}}^{\mathsf{r}}.\,\mathrm{IN} = (\epsilon_p^{\rho_p^\epsilon(\mathsf{r})}.\mathsf{consensusMsgIn}[1][k],\ \ldots,\ \epsilon_p^{\rho_p^\epsilon(\mathsf{r})}.\mathsf{consensusMsgIn}[n][k]),\ \mathsf{r} \geq 1. \tag{7.2b}$$

2. Each process $\mathbf{p}_k \in \Pi$ in each consensus instance k receives messages from $\mathbf{q}_k \in \Pi$, to be used in Line 15, iff its simulator p receives $\mathsf{consensusMsgRecv}_q$ from q in Line 8 (because $\mathsf{consensusMsgIn}[q][k]$ is $\mathsf{consensusMsgRecv}_q[k]$ if a message from process q was received, else $\perp$), and all the intended messages from $\mathbf{q}_k$ to $\mathbf{p}_k$ are sent in Line 6 and Line 23, implying:

$$\boldsymbol{\mathcal{G}}_k = \sigma \tag{7.3}$$

   for each consensus instance k.

3. We prove that in every macro round r the configurations in the reference run $\xi = (\mathcal{C}, \mathrm{MA})_{\boldsymbol{\epsilon}^0, \boldsymbol{\mathcal{G}}}$ and the simulated run $\boldsymbol{\epsilon}$ of consensus instance k are equal:

$$\begin{aligned}\forall \mathsf{r} > 0 : \mathcal{T}_p^{\mathsf{r}}(\boldsymbol{\epsilon}^{\mathsf{r}-1}, (\boldsymbol{\epsilon}^{\mathsf{r}}.\,\mathrm{IN}(1),\ \ldots\ \boldsymbol{\epsilon}^{\mathsf{r}}.\,\mathrm{IN}(n))\ ) =&\\ \boldsymbol{\epsilon}_p^{\mathsf{r}} &= \xi_p^{\mathsf{r}}\\ &= \mathcal{T}_p^{\mathsf{r}}(\xi^{\mathsf{r}-1}, (\mathcal{M}_p^r(1),\ \ldots\ \mathcal{M}_p^r(n))\ )\end{aligned} \tag{7.4}$$

   The base case $\boldsymbol{\epsilon}^0 = \xi^0$ holds by definition, for the step we assume Eq. (7.4) is satisfied. As $\mathsf{consensusState}[k]$ is only written in Line 15, it is sufficient to show that any process $\mathbf{p}_k$ receives the same messages in the simulated $\boldsymbol{\epsilon}$ and in the reference run ξ:

$$\xi_{\mathbf{p}_k}^{\mathsf{r}+1}.\,\mathrm{IN} = \boldsymbol{\epsilon}_{\mathbf{p}_k}^{\mathsf{r}+1}.\,\mathrm{IN} = (\epsilon_p^{\rho_p^\epsilon(\mathsf{r}+1)}.\mathsf{consensusMsgIn}[1][k],\ \ldots,\ \epsilon_p^{\rho_p^\epsilon(\mathsf{r}+1)}.\mathsf{consensusMsgIn}[n][k])$$

   By the induction hypothesis Eq. (7.4), the same set of messages is sent to $\mathbf{p}_k$ in both $\boldsymbol{\epsilon}^{\mathsf{r}+1}$ and $\xi^{\mathsf{r}+1}$. Since both $\boldsymbol{\epsilon}^{\mathsf{r}}$ and ξ^{r} use the same $\boldsymbol{\mathcal{G}}^{\mathsf{r}+1}$, the same set of messages arrives in Line 8, lands in $\mathsf{consensusMsgIn}[*][k]$, and therefore also in $\xi_{\mathbf{p}_k}^{\mathsf{r}+1}.\,\mathrm{IN}$ and $\boldsymbol{\epsilon}_{\mathbf{p}_k}^{\mathsf{r}+1}.\,\mathrm{IN}$. Thus we get $\xi_{\mathbf{p}_k}^{\mathsf{r}+1} = \boldsymbol{\epsilon}_{\mathbf{p}_k}^{\mathsf{r}+1}$ as needed.

$$\begin{array}{cccccccc}
 & p_1 & p_2 & \cdots & p_{n-1} & p_n & \\
C_1 & 0 & 0 & \cdots & 0 & 0 & \to 0 \\
C_2 & 1 & 0 & \cdots & 0 & 0 & \\
C_3 & 1 & 1 & \cdots & 0 & 0 & \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\
C_n & 1 & 1 & \cdots & 1 & 0 & \\
C_{n+1} & 1 & 1 & \cdots & 1 & 1 & \to 1
\end{array}$$

Table 7.1: The initial values $I_1, \ldots, I_{n+1}$ of the $n+1$ consensus instances $C_1, \ldots, C_{n+1}$ on each of the n processes. Note that the matrix is not symmetric.

□

We proved that Algorithm 5 is a message adversary simulation for the binary consensus algorithm $\mathcal{C}$, inheriting the consensus properties validity, agreement and termination for the second part of our proof.

Proof We begin by observing that each process starts its binary consensus instances as shown in the matrix in Table 7.1 in Line 2 and Line 3. We have $n+1$ instances and, representing the initial values as an $n+1$ bit array starting from all 0's, we shift in a 1 from the right going to each next instance. By validity, we know that the first instance C_1 has to decide on 0, whereas the last instance C_{n+1} has to decide on 1, implying there is some instance j where C_j decides on 0 and C_{j+1} decides on 1.

As the initial values of instances C_j and C_{j+1} only differ in one value, namely the switch from a 0 to a 1 at process j, we know that there exists a chain of messages from process j to every other process $q \in \Pi$ in the communication graph sequence σ:

$$\forall q \in \Pi : \exists r, r' < r_q : j \overset{r,r'}{\rightsquigarrow} q$$

where q decides in round r_q.

Assume not and there exists a process $p \neq j$ such that there does not exists a chain of messages $j \overset{r,r'}{\rightsquigarrow} p$ for all $r, r' < r_p$. This implies that the set of processes which have a message chain to p before round r_p does not include j, i.e., any process, before sending a message along a chain to p, has not heard of process j. As the communication graph sequence σ is the same for the run of instances C_j and C_{j+1}, this implies that p receives the same set of messages in both instances up to round r_p and p ends up in the same state in round r_p in both instances. But this contradicts our assumption that instances C_j and C_{j+1} decide on a different value.

The algorithm maintains an array of all initial values of all processes in setIn. As there is a chain of messages from j to any process p and Algorithm 5 broadcasts its setIn in every round, the initial value of j will reach every process and is finally chosen in Line 19.

□

CHAPTER 8

Conclusions

This thesis started by introducing the synchronous message passing model along the lines of the heard-of model presented in [CBS09] and extended it with process crashes. We then defined CG-machines as the 'communication graph sequence'-equivalent to HO-machines and formulated a rigorous definition of a *message adversary simulation*. The validity of a simulation, called a *simulated run*, is given by its equivalence to a reference run on a CG-machine with identical parametrization, as we consider a simulation to be correct iff it cannot be distinguished from a normal run. This gave us the formal tools for defining a strongest message adversary STAR, via a message adversary simulation where we also simulate the center process in lock-step with the simulated process at each simulating process in the system. This is possible, as the underlying message adversary allows consensus to be solved and therefore the processes can agree on a simulated state, i.e., the state of the center process. It turned out that there is not just one strongest message adversary, but a set of strongest message adversaries, which we characterized in Theorem 4 as the set of message adversaries which can be simulated in STAR and allow consensus to be solved.

We then compared $\mathcal{SMP}adv : \text{STAR}$ with $\mathcal{AMP}fd : (\Sigma, \Omega)$, where (Σ, Ω) is the weakest failure detector for consensus. We concluded that they are equal with respect to task solvability which coincides with the results from [RS13]. Coming back to the heard-of model we briefly introduced predicate emulations and examined the relationship of our results to the conclusions in [CBS09]. As it turned out, our simulation is not a valid predicate emulation because our simulation simulates the center process alongside the simulated process and does not require any actual communication to take place. We thus introduced a new validity condition along the lines of *'if a message arrives, it was sent before'*, with the advantage of being more accurate but also more complicated.

Finally, we proved that multi-valued consensus can be efficiently reduced to binary consensus in the message adversary model, which extended our definition for the set of strongest message adversaries to all message adversaries allowing consensus to be solved.

8.1 Further Research

In Chapter 3, we introduced a couple of restrictions on message adversary simulations, which might be interesting to relax. For example, what happens if one drops the one-to-one mapping such that one simulator may simulate multiple processes, or if one allows multiple macro rounds in one micro round? One could also replace consensus with set-consensus and not simulate STAR but SP_UNIF, see [SSW18].

An obvious extension would be simulations on $\mathcal{AMP}$ where just the concept of comparing configurations to a reference run is kept. Does this result in the same conclusion as the weakest failure detector?

Glossary

(S_p^r, T_p^r) The message-sending function S and the state transitioning function T for process p at round r. 7

$(\mathcal{A}, \mathrm{MA})$ The CG-machine consisting of algorithm $\mathcal{A}$ together with the message adversary MA. 9

$(\boldsymbol{\mathcal{G}}^{\mathbf{r}})_{\mathbf{r}\geq 1}$ A simulated communication graph sequence. Each $\boldsymbol{\mathcal{G}}^r$ describes which messages arrive in the simulated round $\mathbf{r}$. 15

$(\mathcal{G}^r)_{r\geq 1}$ A communication graph sequence. Each $\mathcal{G}^r$ describes which messages arrive in round r. 2

C^r The round r configuration of a run ϵ in $\mathcal{SMP}$. 9

$F(r)$ The failure pattern. For each round it provides a set of processes which failed up to that round. 9

$HO(p, r)$ The heard-of set of process p in round r, defined in [CBS09]. 4

$H_F(p)$ The failure characteristic. For each process p it provides a set of processes which p reaches in its failing round. 9

M The set of messages an algorithm may send. 7

$\mathrm{MA}_A \rhd_{\mathcal{B}_\mathcal{A}} \mathrm{MA}_B$ This means that the simulator $\mathcal{B}$, running on MA_B, can simulate $\mathcal{A}$ running on MA_A. This implies that MA_A is stronger than MA_B. 20

$\Omega_\mathcal{A}^\omega$ The set of possible runs of algorithm $\mathcal{A}$. 10

$\Omega_\mathcal{A}$ The set of possible configurations of algorithm $\mathcal{A}$. 10

$\boldsymbol{\epsilon}$ A simulated run, usually in combination with an actual run ϵ of a simulator which defines the simulated run. 18

ϵ A run of an algorithm in $\mathcal{SMP}$. 9

Π The set of processes in the distributed system, usually $|\Pi| = n$. 7

$\mathcal{AMP}[fd : (\Sigma, \Omega)]$ The asynchronous message passing model with the failure detector (Σ, Ω). xiii

$\mathcal{E}_{\mathcal{A},\mathrm{MA}}$ The set of runs of algorithm $\mathcal{A}$ on $\mathcal{SMP}$ with the message adversary MA. 10

$\mathcal{M}_p^r(q)$ This is a helper function that maps to the message which process q sends to process p in round r. 10

$\mathcal{SMP}[adv : \mathrm{MA}]$ The synchronous message passing model with the message adversary MA. xiii

MA A message adversary. 3

QUORUM The message adversary QUORUM, see Definition 13. 3

SOURCE The message adversary SOURCE, see Definition 12. 3

STAR The message adversary STAR, see Definition 11. ix

VSSC(∞) A message adversary that allows solving consensus, specified in [BRS+18]. 3

simStar$_{\mathcal{A}}$ The simulator simStar simulating the algorithm $\mathcal{A}$ on the message adversary STAR. 23

$\mathbf{IN}_p^r$ The set of messages process p receives in round r. 10

$\tilde{\epsilon}$ A run of an algorithm in $\mathcal{AMP}$. 14

$\tilde{\varnothing}$ The state of a crashed process in $\mathcal{AMP}$. 12

$\varnothing$ The state of a crashed process in $\mathcal{SMP}$. 7

$correct(F)$ The set of correct processes in a run in $\mathcal{AMP}$ under the failure characteristic F. No process in $correct(F)$ ever crashes. 13

$crashed(F)$ The set of crashed processes in a run in $\mathcal{AMP}$ under the failure characteristic F, i.e., the complement of $correct(F)$. 13

$f = \frac{\partial F}{\partial r}$ The discrete derivative of $F(r)$. 9

$init_p$ The set of initial states in a given algorithm for the processor p, we set that $init_p \subseteq states_p$. 7

$states_p$ The set of states in a given algorithm for the processor p. 7

Bibliography

[ADGFT01] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Stable leader election. In *International Symposium on Distributed Computing (DISC)*, pages 108–122. Springer Berlin Heidelberg, 2001.

[ADGFT03] Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing Omega with weak reliability and synchrony assumptions. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 306–314, New York, NY, USA, 2003. ACM Press.

[ADGFT04] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 328–337. ACM Press, 2004.

[AFM$^+$04] Emmanuelle Anceaume, Antonio Fernández, Achour Mostéfaoui, Gil Neiger, and Michel Raynal. A necessary and sufficient condition for transforming limited accuracy failure detectors. *Journal of Computer and System Sciences*, 68(1):123–133, 2004.

[AG13] Yehuda Afek and Eli Gafni. Asynchrony from synchrony. In Davide Frey, Michel Raynal, Saswati Sarkar, RudrapatnaK. Shyamasundar, and Prasun Sinha, editors, *14th International Conference on Distributed Computing and Networking (ICDCN)*, volume 7730 of *LNCS*, pages 225–239. Springer Berlin Heidelberg, 2013.

[BHDPW07] Martin Biely, Martin Hutle, Lucia Draque Penso, and Josef Widder. Relating stabilizing timing assumptions to stabilizing failure detectors regarding solvability and efficiency. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 4838 of *Lecture Notes in Computer Science*, pages 4–20, Paris, November 2007. Springer Verlag.

[BRS12] Martin Biely, Peter Robinson, and Ulrich Schmid. Agreement in directed dynamic networks. In *19th International Colloquium on Structural Informa-*

tion and Communication Complexity (SIROCCO), volume 7355 of *LNCS*, pages 73–84. Springer Berlin Heidelberg, 2012.

[BRS⁺15] Martin Biely, Peter Robinson, Ulrich Schmid, Manfred Schwarz, and Kyrill Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. In Ahmed Bouajjani and Hugues Fauconnier, editors, *Third International Conference on Networked Systems (NETYS)*, pages 109–124. Springer International Publishing, 2015.

[BRS⁺18] Martin Biely, Peter Robinson, Ulrich Schmid, Manfred Schwarz, and Kyrill Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *Theoretical Computer Science*, 726:41–77, 2018.

[BSW11] Martin Biely, Ulrich Schmid, and Bettina Weiss. Synchronous consensus under hybrid process and link failures. *Theoretical Computer Science*, 412(40):5602 – 5630, 2011.

[CBFN15] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. Approximate consensus in highly dynamic networks: The role of averaging algorithms. In Magnús M. Halldòrsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 9135 of *Lecture Notes in Computer Science*, pages 528–539. Springer Berlin Heidelberg, 2015.

[CBHW10] Bernadette Charron-Bost, Martin Hutle, and Josef Widder. In search of lost time. *Information Processing Letters*, 110(21):928–933, October 2010.

[CBS09] Bernadette Charron-Bost and André Schiper. The Heard-Of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, April 2009.

[CGP15] Étienne Coulouma, Emmanuel Godard, and Joseph G. Peters. A characterization of oblivious message adversaries for which consensus is solvable. *Theoretical Compututer Science*, 584:80–90, 2015.

[CHT96] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, July 1996.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[FR10] A. Fernández and M. Raynal. From an asynchronous intermittent rotating star to an eventual leader. *IEEE Transactions on Parallel and Distributed Systems*, 21(9):1290–1303, 2010.

[HMSZ09] Martin Hutle, Dahlia Malkhi, Ulrich Schmid, and Lidong Zhou. Chasing the weakest system model for implementing omega and consensus. *IEEE Transactions on Dependable and Secure Computing*, 6(4):269–281, 2009.

[JT08] Prasad Jayanti and Sam Toueg. Every problem has a weakest failure detector. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 75–84, New York, NY, USA, 2008. ACM.

[KOM11] Fabian Kuhn, Rotem Oshman, and Yoram Moses. Coordinated consensus in dynamic networks. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing (PODC)*. ACM, 2011.

[KS06] Idit Keidar and Alex Shraer. Timeliness, failure detectors, and consensus performance. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 169–178, New York, NY, USA, 2006. ACM Press.

[MOZ05] Dahlia Malkhi, Florin Oprea, and Lidong Zhou. Ω meets paxos: Leader election and stability without eventual timely links. In *International Symposium on Distributed Computing (DISC)*, volume 3724 of *Lecture Notes in Computer Science*, pages 199–213, Cracow, Poland, 2005. Springer Berlin Heidelberg.

[MR99] Achour Mostéfaoui and Michel Raynal. Solving consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach. In P. Jayanti, editor, *International Symposium on Distributed Computing (DISC)*, volume 1693 of *Lecture Notes in Computer Science*, pages 49–63. Springer Berlin Heidelberg, September 1999.

[RRT08] Sergio Rajsbaum, Michel Raynal, and Corentin Travers. An impossibility about failure detectors in the iterated immediate snapshot model. *Information Processing Letters*, 108(3):160–164, October 2008.

[RS13] Michel Raynal and Julien Stainer. Synchrony weakened by message adversaries vs asynchrony restricted by failure detectors. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 166–175, 2013.

[SSW18] Ulrich Schmid, Manfred Schwarz, and Kyrill Winkler. On the strongest message adversary for consensus in directed dynamic networks. In Zvi Lotker and Boaz Patt-Shamir, editors, *Structural Information and Communication Complexity*, pages 102–120, Cham, 2018. Springer International Publishing.

[SW89] Nicola Santoro and Peter Widmayer. Time is not a healer. In B. Monien and R. Cori, editors, *6th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 304–313, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[SWK09] Ulrich Schmid, Bettina Weiss, and Idit Keidar. Impossibility results and lower bounds for consensus under link failures. *SIAM Journal on Computing*, 38(5):1912–1951, 2009.

[SWS16] Manfred Schwarz, Kyrill Winkler, and Ulrich Schmid. Fast consensus under eventually stabilizing message adversaries. In *Proceedings of the 17th International Conference on Distributed Computing and Networking (ICDCN)*, pages 7:1–7:10. ACM, 2016.

[TC84] R. Turpin and A. B. Coan. Extending binary Byzantine agreement to multivalued Byzantine agreement. *Information Processing Letters*, 18(2):73–6, February 1984.

[Win19] Kyrill Winkler. *Characterization of consensus solvability under message adversaries.* Wien, 2019.

[WSS16] Kyrill Winkler, Manfred Schwarz, and Ulrich Schmid. Consensus in directed dynamic networks with short-lived stability. *CoRR*, abs/1602.05852, 2016.