Abstract of "Byzantine Computability and Combinatorial Topology",
by Hammurabi das Chagas Mendes, Ph.D., Brown University, May 2016.

The tools and language of combinatorial topology have previously been very successful in characterizing distributed task solvability when processes fail only by crashing. In this work, the approach is extended to systems where processes fail arbitrarily, even maliciously, characterizing what the literature calls "Byzantine failures". Distributed tasks are formalized in terms of a pair of combinatorial structures called simplicial complexes: one models process inputs (the input complex), and another models process outputs (the output complex). A map between the input complex and output complex defines the task semantics. This thesis establishes necessary and sufficient solvability conditions for Byzantine tasks, in synchronous and asynchronous systems. For asynchronous systems, a Byzantine task is shown to be solvable if and only if a specific crash-failure task, suitably defined in terms of the Byzantine task, is solvable as well. For asynchronous colorless tasks, an important subclass of problems that includes consensus and k-set agreement, the above characterization reduces to particularly succinct and elegant forms. The frontier between possible and impossible for colorless tasks is delineated by pivotal problems such as multidimensional ϵ-approximate agreement and barycentric agreement, for which we present protocols with maximum resilience. For synchronous systems, the information dissemination throughout the rounds is modeled by simplicial complexes called protocol complexes, and certain topological properties can characterize ambiguity caused by Byzantine failures. This approach ultimately demonstrates that Byzantine processes can potentially impose one extra round of ambiguity compared to crash-failure systems in regard to solving set agreement problems – and at most that in some settings. In essence, this work shows how the language of combinatorial topology, when aligned with traditional algorithmic tools in distributed computing, provides a robust framework to study task solvability not only across different communication models, synchronous and asynchronous, but also across different failure models, crash and Byzantine.

Byzantine Computability and Combinatorial Topology

by
Hammurabi das Chagas Mendes
Sc. M., Brown University, 2012
M. S. Universidade de São Paulo, 2008
B. S., Universidade de Brasília, 2004

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2016

This dissertation by Hammurabi das Chagas Mendes is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date _____ _____

Maurice Herlihy, Director

Recommended to the Graduate Council

Date _____ _____

Anna Lysyanskaya, Reader
Brown University

Date _____ _____

Sergio Rajsbaum, Reader
Universidad Nacional Autonoma de Mexico

Approved by the Graduate Council

Date _____ _____

Peter Weber
Dean of the Graduate School

Acknowledgments

This work is dedicated to my parents, Francisco and Berenice: you encouraged me to pursue my dreams, and every accomplishment belongs to you. I am very grateful to my wife Carrie, my daughter Júlia, my sisters Alda and Ana, and God. You mean everything to me – thank you. I thank my advisor Maurice Herlihy, professors Sorin Istrail and Rodrigo Fonseca, thesis committee members Sergio Rajsbaum and Anna Lysyanskaya, brothers-in-law Hélio and Fabianno, and extended families in Brazil and in the United States. I am also thankful to friends at Brown, especially Marcelo, Eduardo, Marek, Irina, and Alexander; and friends in Brazil, especially Bruno, Marcelo, Romildo, Virgílio, Fernanda, and Diego. I am very grateful to Brown University for the immense support throughout these years, and I will always remember the *beautiful Providence*.

Contents

⋆ Parts of this thesis have been previously published by ACM and Springer, co-authored with Maurice Herlihy, Christine Tasson, Nitin Vaidya, and Vijay K. Garg.

Chapter 1

Introduction

Distributed computing is a field of Computer Science concerned with computation among autonomous, communicating computing entities. We call these entities *processes*, and they interact via a *communication system*. A *task* is a distributed coordination problem in which processes start with private inputs, taken from a finite set, communicate among themselves, and each eventually decides on a private output, also taken from a finite set. Analogous to how theoretical computability is concerned with languages and decidability, theoretical distributed computing is concerned with distributed tasks and their solvability.

Processes and communication are subject to different *failure* and *synchrony* assumptions, significant operational properties that affect our ability to solve problems. Failure and synchrony operational settings are abstracted through *models*, briefly discussed here in Sec. 1.1, but more detailed in Chap. 2.

One of the central questions in distributed computing is characterizing which tasks can be solved in which models of computation. A *protocol* is a distributed algorithm that solves a task given a model of computation. Figure 1.1 exemplifies in high-level a protocol execution for the *consensus* task [18]. Informally, in the consensus task, processes propose values from some set, say, $\{v_0, v_1, v_2\}$, and must individually finish with a single value. The value must have been proposed in the beginning of the protocol by some process.

This thesis focuses on the characterization of task solvability – that is, the question on whether protocols exist – in systems where a subset of processes can behave arbitrarily, even maliciously. This kind of failures is technically known as *Byzantine* failures, and contrast with the more benign *crash* failures, where processes only fail by halting silently, without previous notice.

The approach of this work is to use the tools and language of *combinatorial topology*

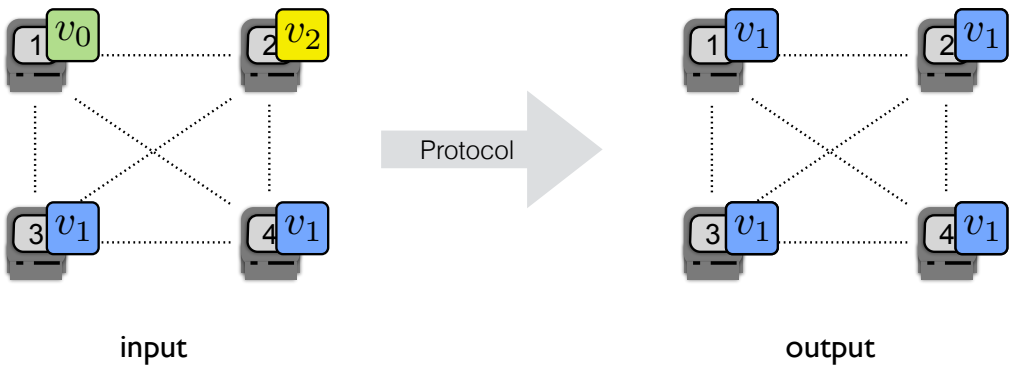

Figure 1.1: The execution of a consensus protocol: Processes input values, exchange messages via the communication channel, and finally decide on an output.

to address these questions. This technique, when well-aligned with *algorithmic techniques*, provides a robust framework to study task solvability under different failure and synchrony assumptions. This work shows that:

1. For asynchronous tasks, solvability conditions can be expressed in terms of the conditions for crash-failure systems (Chap. 5);

2. For asynchronous colorless tasks, these conditions depend on the same fundamental problems – such as barycentric agreement (Chap. 6) – as in crash-failure systems;

3. For synchronous tasks, solvability conditions can be studied using similar techniques, and have similar lower bounds, to crash-failure systems (Chap. 7).

Informally, we can say that foundations, techniques, and results in Byzantine systems "mirror", to a certain extent, the respective concepts in crash-failure systems.

1.1 Processes, Failures, and Synchrony

Processes exchange messages via pairwise channels, denoting a complete graph where processes are nodes and channels are edges. We assume that channels always deliver messages sent through them, in FIFO order. This assumption is *appropriate* and *realistic*. It is appropriate because having channels that drop messages in an arbitrary way trivially makes most interesting tasks unsolvable: consider simply dropping every single message. It is realistic because this kind of channel can be implemented on top of more unreliable channels that have at least some limited delivery guarantees – say, delivery with probability $p > 0$. For more details, please refer to Chap. 2.

In this work, we are interested on the situations where a proper subset of processes may be subject to failures. We say that *non-faulty processes* execute protocols as prescribed, while *faulty processes* malfunction in different ways.

In terms failure behavior, in the *crash-failure* model [3, 32], faulty processes may become permanently silent at any point in the execution of the protocol. These processes simply halt and never send additional messages. Alternatively, in the *Byzantine-failure* model [31, 3, 32], faulty processes display arbitrary behavior, including malicious collusion to prevent the protocol to terminate correctly.

Concerning the computation and communication synchrony, we have *synchronous* or *asynchronous* systems. In synchronous systems, communication and computation are organized in discrete rounds. In each round, any non-faulty process performs as follows, in order: (i) sends a message; (ii) receives all messages sent in the current round by the other processes; and (iii) performs internal computation. In asynchronous systems, processes have different relative speeds, and message delivery is subject to unbound, finite delays.

1.2 Thesis Overview

The tools and language borrowed from combinatorial topology have been previously successful in characterizing task solvability for synchronous and asynchronous *crash-failure* systems [28]. Tasks have been formalized in terms of a pair of combinatorial structures called *simplicial complexes* [37, 30], one for process inputs (called *input complex* $\mathcal{I}$), and one for process outputs (called *output complex* $\mathcal{O}$). A map between the input complex and output complex $\Delta : \mathcal{I} \rightarrow \mathcal{O}$ constrains the task semantics.

This thesis extends the approach of using combinatorial topology tools to characterize computability in Byzantine systems, synchronous and asynchronous. The results include new protocols for tasks such as the multidimensional ϵ-approximate agreement [34], which has both theoretical and practical interest. This thesis starts discussing models and background in Chapters 2 and 3, followed by novel computability results in Byzantine asynchronous and synchronous systems. The next subsections overview the main results presented here.

1.2.1 Multidimensional ϵ-Approximate Agreement

This thesis starts the discussion of the technical results mentioned above with the definition, analysis, and solution of a task called *multidimensional ϵ-approximate agreement*. This task seems to delineate the frontier between what is solvable and unsolvable for *colorless*

tasks [6, 24]. Colorless tasks form an important subclass of tasks that encompasses well-studied problems such as consensus [18] and k-set agreement [11],

In the multidimensional ϵ-approximate agreement task, for arbitrary $\epsilon > 0$ and $d \geq 1$, each process starts with an input value in $\mathbb{R}^d$, and all non-faulty processes must choose output values, also in $\mathbb{R}^d$, such that:

1. all outputs lie within ϵ of one another; and
2. all outputs lie in the convex hull of the inputs of the non-faulty processes.

A *t-resilient protocol* for this task is an algorithm guaranteeing that each non-faulty process decides on a correct output, despite the presence of up to t Byzantine processes.

This problem has a fascinating background. In the well-known consensus task, processes start with values from an arbitrary domain, and must choose values such that: (1) all processes decide on the same value; and (2) that value is some process' input. That task is impossible in the presence of even a single crash failure [18]. There are many ways to circumvent this impossibility – such as relying on synchrony or randomization – and one is to settle for the (unidimensional) *ϵ-approximate agreement* [15, 1]. In this case, non-faulty processes start with input values in $\mathbb{R}$, and must also finish with output values in $\mathbb{R}$, in the range of the non-faulty process inputs.

The multidimensional ϵ-agreement, as described before, is a non-trivial generalization for the ϵ-approximate agreement. We design a protocol that tolerates as many failures as possible. The protocol performance, measured in the number of messages required to be sent, depends solely on the inputs of the non-faulty processes, and is unaffected by any malicious behavior of the Byzantine processes.

While the problem above has clear applicability – consider autonomous robots that can now coordinate and converge despite the influence of possible malicious entities – we are interested in a key theoretical application that is useful to characterize task solvability for colorless tasks. In Chap. 4, we give details on the multidimensional ϵ-approximate agreement, providing a protocol, and discussing its properties and correctness. In Chap. 6, after a detour characterizing solvability for general Byzantine tasks in Chap 5, we show how the multidimensional ϵ-approximate agreement is used to solve *barycentric agreement*, which is key to a more specific solvability characterization for colorless tasks.

1.2.2 Asynchronous Byzantine Computability

Chapter 5, discusses the first theorem that gives necessary and sufficient conditions for arbitrary task solvability in Byzantine asynchronous systems. The theorem states that an

asynchronous Byzantine-failure task is solvable if and only if a suitably defined asynchronous crash-failure task is solvable as well. The crash-failure task is defined in terms of the original Byzantine-failure task. Task descriptions rely on the language of combinatorial topology mentioned before (input/output complexes and a map linking these structures). Given that solvability conditions have long been known for asynchronous crash-failure systems, the previous *equivalence theorem*, characterizes solvability for asynchronous Byzantine systems.

As mentioned before, with help from the multidimensional ϵ-approximate agreement protocol, the previous characterization reduces to a particularly elegant, compact form for colorless tasks. In Chap. 6 we show how the computability conditions for colorless tasks interrelate the number of processes, the number of failures, and the topological structure of the task's simplicial complexes.

These results suggests that the tools and language of combinatorial topology are indeed a convenient and effective way to formalize and study task solvability in a range of distinct failure models.

1.2.3 Synchronous Byzantine Computability

Chapter 7 shows that moving from crash-failure to Byzantine-failure synchronous systems may require processes one extra round in order to solve set-agreement problems. In fact, we see how under specific conditions (concerning the relationship between the number of processes and the number of failures), that extra single round, when necessary, is also sufficient in order to solve the problem.

The protocol execution is modeled by a series of simplicial complexes, each representing all possible states of the system throughout all rounds of execution. Each of those simplicial complexes is called a *protocol complex*, and our analysis relies on their *k-connectivity*, with $t \geq k \geq 1$, a general notion of graph connectivity for higher dimensions.

We see how the protocol complex can remain $(k-1)$-connected for $\lceil t/k \rceil$ rounds when processes fail arbitrarily – instead of $\lfloor t/k \rfloor$ when they can fail only by crashing, where t is an upper bound on the number of Byzantine processes and $t \geq k \geq 1$. To prove this bound, we conceived a combinatorial operator modeling the ability of Byzantine processes to *equivocate* – that is, to transmit ambiguous state information – without revealing their Byzantine nature. We compose this operator with similar crash-failure operators, extending the protocol complex connectivity for one extra round when $t \bmod k \neq 0$.

Connectivity is relevant since a $(k-1)$-connected protocol complex prevents important problems such as k-set agreement from having solutions. In fact, we show that the above

bound is tight in certain settings by (i) defining a suitable formulation of k-set agreement for Byzantine synchronous systems; and (ii) solving the problem in $\lceil t/k \rceil + 1$ rounds.

As the techniques based on combinatorial topology become applicable to synchronous systems as well, it becomes clear that these techniques are effective not only across *different failure models*, but also across *different synchrony models*.

1.3 Thesis Statement

This thesis develops the results outlined above and supports the following claim:

> The language of combinatorial topology, aligned with algorithmic and simulation techniques, provides a robust framework to study task solvability not only with crash failures, but also with Byzantine failures, in synchronous and asynchronous systems. Byzantine solvability can be expressed in terms of crash-failure solvability in asynchronous systems, relies on the same fundamental problems for asynchronous colorless tasks, and have similar adversarial bounds for set agreement in synchronous systems.

Chapter 2

Operational Model

In a distributed coordination problem, formally called a *task*, we have multiple *processes* that start with private inputs, communicate via message-passing, and finish with outputs consistent with the task semantics. In this chapter, we describe the operational details of the processes and communication that define our system.

2.1 Processes and Communication

The set of processes is defined as $\mathbb{P}$. Assuming $n+1$ processes rather than n simplifies the topological notation, but slightly complicates the computing notation, while choosing n processes has the opposite trade-off. To simplify our notation in the analyses, we settle for n ($\mathbb{P} = \{P_0, \ldots, P_{n-1}\}$) in Chap. 4, centered on computation, and $n+1$ ($\mathbb{P} = \{P_0, \ldots, P_n\}$) in Chapters 5, 6, and 7. We make the number of processes clear in the beginning of the chapters.

In this work, they always communicate via *message-passing*, via pairwise communication channels. These channels are *reliable* and *FIFO*: all transmitted messages are eventually delivered, in the order they were sent. This is technically known as *perfect authenticated channels* in the literature [9].

The previous assumptions are realistic and have practical relevance. Reliable FIFO channels are customarily implemented on top of more unreliable channels, using simple algorithmic techniques (retransmission, duplicate elimination, etc). Technically, we only require that our underlying unreliable channels deliver messages with some probability $p > 0$. The next paragraph hints how these techniques are implemented, and the details can be found in [9].

Pairwise channels model systems where each node is always reachable from every other

one in the underlying physical network, and can be implemented via routing. Reliable channels can be implemented by retransmitting messages, which works since the delivery probability is assumed to be $p > 0$. Authenticated channels model systems where the message sender is reliably identified, and can be implemented using cryptographic techniques, such as message authentication codes (MACs), coupled with shared secret keys. Senders attach a MAC to every message, accounting for the message itself, the sender, and the receiver. Receivers only accept messages that (i) have not been accepted before; and (ii) have the sender verified, matching the channel sender/receiver. FIFO channels can be implemented as follows. Senders label the i-th locally sent message with i. Receivers only inspect a message labeled with $j > 0$ after a message labeled with $j - 1$ has been inspected, keeping all received messages in an internal buffer, for each possible channel. For details on the algorithms and protocols discussed above, please refer to [9].

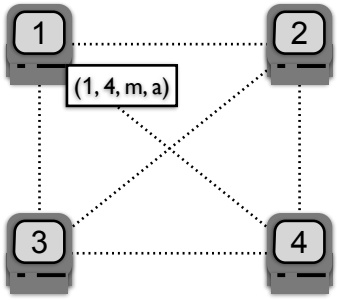

Figure 2.1: A system with 4 processes with pairwise communication channels. A message m with sender $s = 1$, receiver $r = 4$, and the message authentication $a = \mathrm{MAC}(\mathrm{s}, \mathrm{r}, \mathrm{m})$.

In light of the above observations, we assume perfect authenticated channels in this work. Note that those channels guarantee delivery, but not necessarily any upper bound on the *time* to deliver the message. This specific issue is related to synchrony, an important model parameter discussed below.

2.2 Synchrony and Failures

We now define two key characteristics of our system that deeply affect our ability to solve problems: *synchrony* among processes, and *failure* behaviors assumed in the system.

2.2.1 Synchrony

The literature usually considers two models regarding the "synchrony" of message delivery times. In *synchronous systems* [3], communication takes place in multiple discrete rounds

numbered as $0, 1, \ldots$ In each round, the following takes place in order:

1. All sending processes send their messages;
2. Each message sent in the current round is delivered to its specified receiver;
3. Each process performs an internal state change that may or may not depend on the received messages.

Note that every message sent at round i is delivered in the same round i. We can therefore understand the round structure as a global shared clock that imposes an upper bound on message delivery time. Also, informally speaking, the round structure makes the relative speeds of the processes uniform.

In *asynchronous systems* [3], we do not have such convenient structure. The message transmissions and deliveries overlap arbitrarily, with the delivery of any message happening after a finite, yet unbounded delay. The relative speed of processes is also unbound.

Different synchrony guarantees have deep impact on the design of our protocols. Perhaps the most relevant effect is the interplay of synchronization and failures, discussed below.

2.2.2 Failures

In this work, we consider *crash-failure* systems [3], where processes fail only by permanent, unannounced halting, or *Byzantine-failure* systems [31], where processes fail arbitrarily, even maliciously. *Non-faulty processes* execute protocols as prescribed, while *faulty processes* fail according to the adopted failure model.

A failure model specifies the nature of the failures (crash or Byzantine), as well as an upper bound on the number of faulty processes, denoted by t. In a crash-failure system, the faulty processes are the ones that halt at any point in the execution. In a Byzantine-failure system, we will take a different perspective. Since Byzantine processes may introduce spurious, incompatible, or illegitimate inputs, yet execute the protocol as prescribed, we consider an *adversary* that chooses the set of faulty processes, any set of size up to t. In our model, we assume that processes were initially correct, but the adversary gains control of the chosen faulty processes right before the protocol starts.

The *resilience* is defined as the maximum amount of faulty processes divided by the total number of processes. Systems with bigger resilience tolerate more failures in relation to the total number of processes.

Byzantine processes may execute the protocol correctly or incorrectly, at the discretion of the adversary. This way, Byzantine processes may execute the protocol as prescribed, yet

using an illegitimate, but valid input. It is tricky to characterize the inputs of Byzantine processes in light of the issue above, which would complicate the formalization Byzantine tasks, since they are traditionally specified in terms of the relationship between *all* process inputs and *non-faulty* process outputs.

We are certainly concerned only with the output of non-faulty processes, but in this work we focus on tasks where:

> The outputs of non-faulty processes are constrained by the inputs of non-faulty processes only.

In crash-failure systems, since process inputs are clearly defined, the literature usually constrains the outputs of non-faulty processes in terms of the inputs of all processes, faulty or non-faulty. In Byzantine-failure systems, a task definition, technically speaking, could choose to account for Byzantine inputs, associating with them an informal "apparent input". In this work, we focus on *strong formulations*: in the arguably "true" spirit of Byzantine resilience, tasks must *tolerate spurious input changes* from Byzantine processes.

Our model makes sense to many interesting, practical tasks, and has been considered for many traditional, central problems such as consensus, set agreement, and approximate agreement – where it is traditionally called *strong validity* [39, 14]. Our own multidimensional ϵ-approximate agreement of Chap. 4 has a very natural application of strong validity to multi-agent systems and robotics. Strong validity and its consequences are examined thoroughly in Chap. 5.

In any case, note that the combination of asynchrony and failures can make it impossible for a non-faulty process to distinguish between a faulty, halted process, from a non-faulty process that is being subject to slow computation or communication [3].

2.3 Full Information Protocols

We model processes as state machines that execute a *full-information* protocol [25]. These protocols abstract the task-specific behavior from communication with a decision function separated from the communication structure. Each process repeatedly:

1. Sends its state to all other processes (the first state is the input value);
2. Receives the state information from other processes;
3. Concatenates the state information to its own state.

After completing some number of iterations (which are synchronous rounds, if the system is synchronous), each process applies a problem-specific decision function to its current state in order to decide an output value.

Algorithm 1 formalizes the concept. Call the input value (respectively, output value) of a non-faulty process P_i as I_i (respectively, O_i). The initial state of a non-faulty process P_i is its input value I_i. We write $view_i^k$ to denote P_i's internal state after the k-th iteration (which are synchronous rounds, if the system is synchronous). We omit subscript or superscripts when the process or iteration are implied or obvious. The function *decided* returns *true* when the process is capable of deciding given its current state.

- In an asynchronous system, the function *sampled* returns a subset containing at least $|\mathbb{P}| - t$ processes from which P_i collects messages. These processes include the first $|\mathbb{P}| - t$ processes to deliver the current iteration's messages to P_i, plus all the others that did so before P_i checked for the current iteration's messages.

- In a synchronous system, a process samples messages from all the other processes at each synchronous round. When a message is absent, due to a crash or malevolent behavior, the associated message is simply defined as $\perp$.

Finally, the function δ is the decision function that abstract task semantics.

Algorithm 1 P_i.FullInformationProtocol(I_i)

```
1: view ← I_i
2: k ← 0
3: while not decided(view) do
4:     send view^k to all P_j ∈ ℙ
5:     S ← recv {view_j^k : P_j ∈ sampled(ℙ)}
6:     view ← view concatenated with S
7:     k ← k + 1
8: end while
9: return δ(view)
```

Any protocol can be implemented as a full-information protocol, and while this approach normally increases the number and size of transmitted messages, the scenario is identical for computability purposes.

2.4 Adversaries and Schedules

In a synchronous system, the executions of Lines 4 to 6 in Alg. 1 follow a predefined *schedule*. When executing the k-th synchronous round, Processes behave as follows in order:

1. All non-faulty processes first execute Line 4.
    - In Byzantine systems, faulty processes send arbitrary states;
    - In crash-failure systems, crashed processes send nothing.
2. All non-faulty processes execute Lines 5 and 6.
    - In Byzantine systems, faulty processes receive all transmitted messages, and perform an arbitrary state change.
    - In crash-failure systems, crashed processes receive no messages and perform no further state transitions.

If a process did not send a message for the current iteration, we associate an "empty" state with that process, denoted by $\perp$.

In an asynchronous system, we assume an *adversarial scheduler* that controls the instant when messages become available to be accepted, and, in Byzantine systems, which are the Byzantine processes, and the arbitrary behavior of the Byzantine processes. In these systems, an *action* is either:

1. The sending of a message (Line 4) by *one* process;
    - In Byzantine systems, faulty processes send arbitrary states;
    - In crash-failure systems, crashed processes send nothing.
2. The *delivery* of a message to a process, which makes it available for reception.
3. The receipt of messages (Line 5) by *one* process, which is immediately followed by a state change (Line 6);
    - In Byzantine systems, faulty processes perform an arbitrary state change.
    - In crash-failure systems, crashed processes receive no messages and perform no further state transitions.

In asynchronous systems, a *schedule* is simply a total order of actions.

Chapter 3

Topological Tools

The use of the tools and language of combinatorial topology is central to the characterization of task solvability in this thesis. This chapter presents the required concepts in a concise, yet self-contained manner. For details, please refer to standard literature such as [37, 30].

3.1 Basics

A *simplicial complex* $\mathcal{K}$ consists of a finite set V along with a collection of subsets of V closed under containment. An element of V is called a *vertex* of $\mathcal{K}$. The set of vertices of $\mathcal{K}$ is referred by $V(\mathcal{K})$. Each set in $\mathcal{K}$ is called a *simplex*, usually denoted by lower-case Greek letters: σ, τ, etc. The *dimension* $\dim(\sigma)$ of a simplex σ is $|\sigma| - 1$.

A subset of a simplex is called a *face*. A face τ of σ is called *proper* if $\dim(\tau) = \dim(\sigma) - 1$. We use "$k$-simplex" as shorthand for "k-dimensional simplex", also in "k-face". The dimension $\dim(\mathcal{K})$ of a complex is the maximal dimension of its simplexes, and a *facet* of $\mathcal{K}$ is any simplex having maximal dimension in $\mathcal{K}$. A complex is said *pure* if all facets have dimension $\dim(\mathcal{K})$. Please refer to Fig. 3.1.

The set of simplexes of $\mathcal{K}$ having dimension at most ℓ is a subcomplex of $\mathcal{K}$, which is called *ℓ-skeleton* of $\mathcal{K}$, denoted by $\text{skel}^{\ell}(\mathcal{K})$. The set of simplexes of $\mathcal{K}$ having dimension exactly ℓ is called $\text{Faces}^{\ell}(\mathcal{K})$. Please refer to Fig. 3.2. Since single simplexes are also simplicial complexes, the concepts above apply verbatim.

3.2 Maps and Geometrical Realizations

Let $\mathcal{K}$ and $\mathcal{L}$ be complexes. A *vertex map* f carries vertices of $\mathcal{K}$ to vertices of $\mathcal{L}$. If f additionally carries simplexes of $\mathcal{K}$ to simplexes of $\mathcal{L}$, it is called a *simplicial* map. A *carrier*

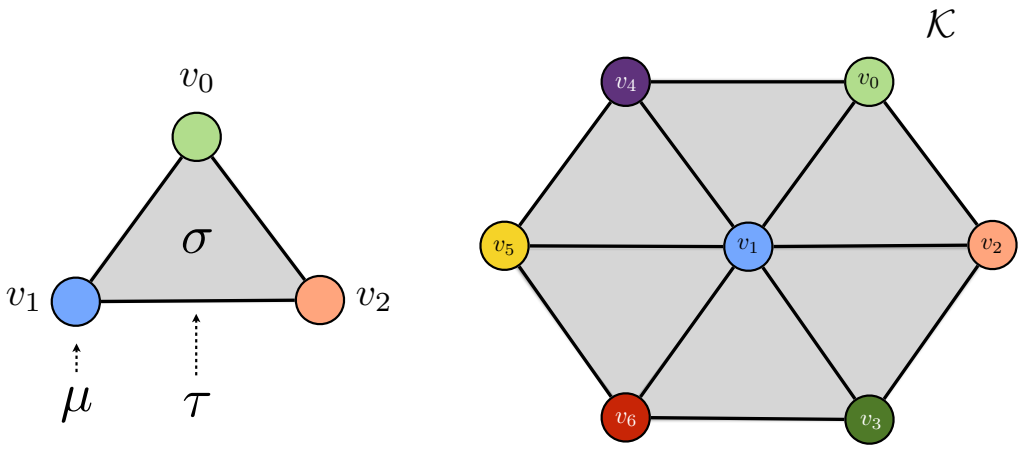

Figure 3.1: (Left) 2-simplex σ, 1-face τ of σ, and 0-face (vertex) μ of σ. (Right) A pure simplicial complex $\mathcal{K}$ containing σ. Note that $\dim(\mathcal{K}) = \dim(\sigma) = 2$.

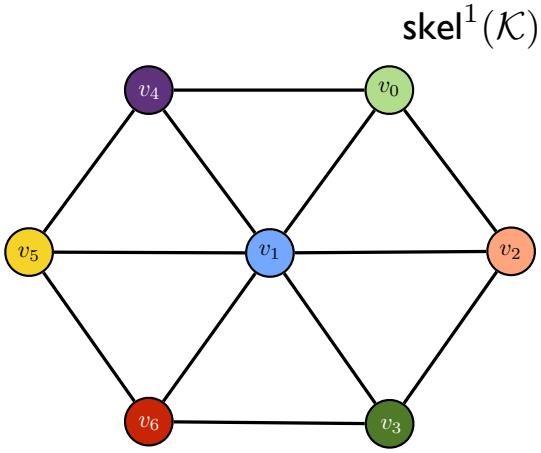

Figure 3.2: Skeleton of $\mathcal{K}$ containing simplexes of dimension at most 1.

map Φ from $\mathcal{K}$ to $\mathcal{L}$ takes each simplex $\sigma \in \mathcal{K}$ to a subcomplex $\Phi(\sigma) \subseteq \mathcal{L}$, such that for all $\sigma, \tau \in \mathcal{K}$, we have $\Phi(\sigma \cap \tau) \subseteq \Phi(\sigma) \cap \Phi(\tau)$. A simplicial map $\phi : \mathcal{K} \to \mathcal{L}$ is *carried by the carrier map* $\Phi : \mathcal{K} \to 2^{\mathcal{L}}$ if, for every simplex $\sigma \in \mathcal{K}$, we have $\phi(\sigma) \subseteq \Phi(\sigma)$. Figure 3.3 shows an example of a simplicial map being carried by a carrier map.

Although we defined simplexes and complexes in a purely combinatorial way, they can also be interpreted geometrically. An n-simplex can be identified with the convex hull of $(n+1)$ affinely-independent points in the Euclidean space of appropriate dimension. This geometric realization can be extended to complexes. The point-set that underlies such *geometric complex* $\mathcal{K}$ is called the *polyhedron* of $\mathcal{K}$, denoted by $|\mathcal{K}|$.

We can define simplicial/carrier maps between geometrical complexes. Given a simplicial

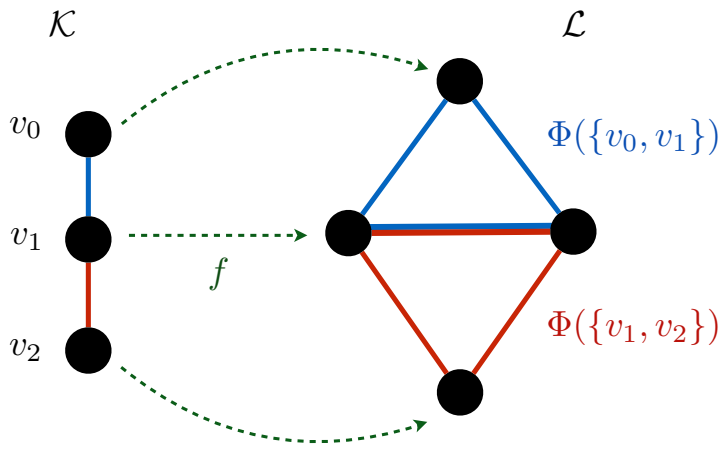

Figure 3.3: Simplicial map f carried by the carrier map Φ.

map $\phi : \mathcal{K} \to \mathcal{L}$ (resp. carrier map $\Phi : \mathcal{K} \to 2^{\mathcal{L}}$), the polyhedrons of every simplex in $\mathcal{K}$ and $\mathcal{L}$ induce a continuous simplicial map $\phi_c : |\mathcal{K}| \to |\mathcal{L}|$ (resp. continuous carrier map $\Phi_c : |\mathcal{K}| \to |2^{\mathcal{L}}|$). We say ϕ (resp. ϕ_c) is carried by Φ if, for any $\sigma \in \mathcal{K}$, we have $|\phi(\sigma)| \subseteq |\Phi(\sigma)|$ (resp. $\phi_c(|\sigma|) \subseteq \Phi_c(|\sigma|)$). We may refer to continuous realizations of simplicial maps and carrier maps simply as *continuous maps* when there is no ambiguity.

3.3 Boundary, Interior, and Open Star

Combinatorially speaking, for any simplex σ, the *boundary* of σ, denoted $\partial\,\sigma$, is the simplicial complex of $(\dim(\sigma) - 1)$-faces of σ. Geometrically speaking, the *interior* of σ is formally defined as $\operatorname{Int}\sigma = |\sigma| \setminus |\partial\,\sigma|$. The *open star* of $\sigma \in \mathcal{A}$, denoted $\operatorname{Ost}\sigma$, is the union of the interiors of all simplices in $\mathcal{A}$ containing σ, including σ itself. See Fig. 3.4.

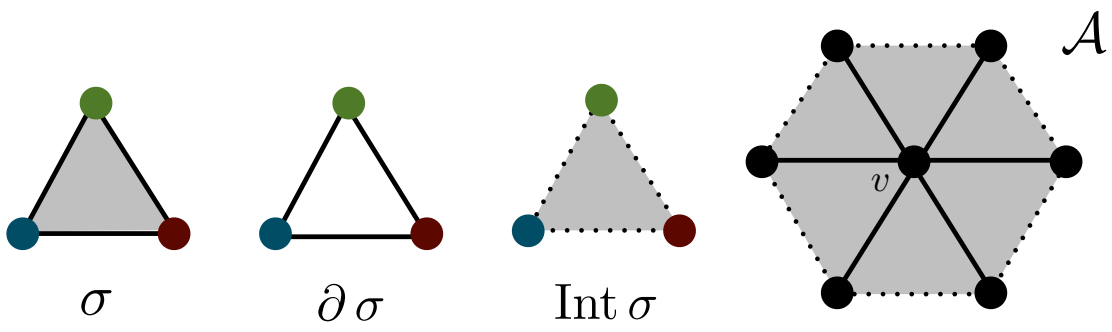

Figure 3.4: The interior and boundary of a 2-simplex σ, and the open star of a 0-simplex $\{v\} \subseteq \mathcal{A}$.

3.4 Subdivisions and Simplicial Approximation

Subdivision is an important operation over simplicial complexes, transforming individual simplexes into "finer" simplicial complexes. We formalize the concept below.

Definition 3.4.1. A subdivision of a complex $\mathcal{A}$ is a complex $\mathcal{B}$ such that:

1. for *any* $\tau \in \mathcal{B}$, $|\tau|$ is contained in the polyhedron of some $\sigma \in \mathcal{A}$;
2. for *any* $\sigma \in \mathcal{A}$, $|\sigma|$ is the union of disjoint polyhedrons of simplices belonging to $\mathcal{B}$.

We understand the subdivision as an operator Div from complexes to complexes. If we perform N consecutive applications of Div, the composite operator is denoted by Div^N.

Definition 3.4.2. A *mesh-shrinking* subdivision $\mathrm{Div}\,\mathcal{A}$ of a complex $\mathcal{A}$ is a subdivision where, for any 1-simplex $\sigma \in \mathrm{skel}^1(\mathcal{A})$, $\mathrm{Div}\,\sigma$ contains at least two distinct 1-simplices.

Intuitively, we are "shrinking" the simplices. A particularly important mesh-shrinking subdivision in this work is the *barycentric subdivision* (Fig. 3.5).

Definition 3.4.3. The barycentric subdivision of σ is a simplicial complex $\mathrm{Bary}\,\sigma$ whose vertices are faces of σ, and every k-simplex $\mu \in \mathrm{Bary}\,\sigma$ could be written as $\{\sigma_0, \ldots, \sigma_k\}$, where $\sigma_0 \subset \cdots \subset \sigma_k \subseteq \sigma$. Let

$$\mathrm{Bary}\,\mathcal{K} = \bigcup_{\sigma \in \mathcal{K}} \mathrm{Bary}\,\sigma. \tag{3.1}$$

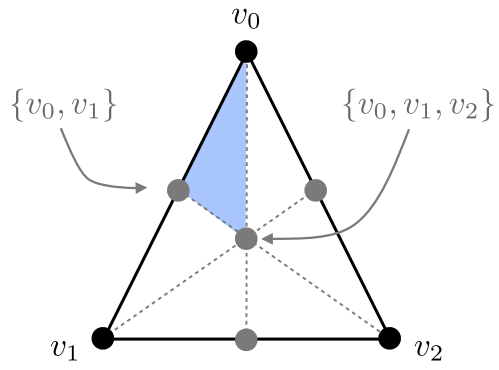

Figure 3.5: Barycentric subdivision of $\sigma = \{v_0, v_1, v_2\}$, highlighting one of its simplices $\{\{v_0\}, \{v_0, v_1\}, \{v_0, v_1, v_2\}\}$.

Given any continuous map, a simplicial approximation, defined formally below, is a "sufficiently close" combinatorial counterpart.

Definition 3.4.4. A simplicial map $\mu : \mathcal{K} \to \mathcal{L}$ is a *simplicial approximation* of a continuous map $c : |\mathcal{K}| \to |\mathcal{L}|$ if

$$c(|\operatorname{Int}\sigma|) \subseteq \bigcap_{v \in \sigma} \operatorname{Ost}\mu(v) = \operatorname{Ost}\mu(\sigma), \text{ for all } \sigma \in \mathcal{K}.$$

Not every continuous map $c : |\mathcal{K}| \to |\mathcal{L}|$ has a simplicial approximation defined over the same domain and range of c. However, we can always presume a simplicial approximation defined over the domain $\mathcal{K}$, sufficiently subdivided with a mesh-shrinking subdivision, and range $\mathcal{L}$. This important property is formalized below.

Theorem 3.4.5. (Simplicial Approximation, [37, 30]) For any continuous map $c : |\mathcal{K}| \to |\mathcal{L}|$, consider an arbitrary mesh-shrinking subdivision operator Div. Then, there exists an $N > 0$ such that c has a simplicial approximation

$$\mu : \operatorname{Div}^N \mathcal{K} \to \mathcal{L}.$$

Remark 3.4.6. The above theorem particularly applies when Div = Bary, the barycentric subdivision operator.

The concept of simplicial approximation is important to us because simplicial approximations preserve the property of being carried by carrier maps. Even when the domain is subdivided under barycentric subdivision, a similar notion of being carried, formalized below, applies. Consider a carrier map $\Phi : \mathcal{K} \to \mathcal{L}$ and a continuous map $c : |\mathcal{K}| \to |\mathcal{L}|$. Let c be carried by Φ, and $\mu : \operatorname{Bary}^N \mathcal{K} \to \mathcal{L}$ be a simplicial approximation of c. We say that μ is carried by Φ if $\mu(\operatorname{Bary}^N \sigma) \subseteq \Phi(\sigma)$ for any $\sigma \in \mathcal{K}$.

Lemma 3.4.7. If a continuous map $c : |\mathcal{K}| \to |\mathcal{L}|$ is carried by the carrier map $\Phi : \mathcal{K} \to 2^{\mathcal{L}}$, then any simplicial approximation $\mu : \operatorname{Bary}^N \mathcal{K} \to \mathcal{L}$ of c is carried by Φ.

Proof. Consider a vertex v of $\operatorname{Bary}^N \sigma$ with $\sigma \in \mathcal{K}$, and say $\mu(v) \not\subseteq \Phi(\sigma)$, implying that $\operatorname{Ost}\mu(v) \cap |\Phi(\sigma)| = \emptyset$. Since μ is a simplicial approximation of c, we have $c(|v|) \in \operatorname{Ost}\mu(v)$, which now implies that $c(|v|) \notin |\Phi(\sigma)|$ by the previous observation. This contradicts the assumption that c is carried by Φ, so we must have $\mu(v) \subseteq \Phi(\sigma)$, implying that μ is carried by Φ. □

3.5 Connectivity

Two geometrical objects A and B are *homeomorphic* if, there is a continuous map from A into B or vice-versa [38, 37]. For any $d > 0$, let the d-ball $B^d \subseteq \mathbb{R}^d$ be the set of points with

distance at most 1 from the origin in $\mathbb{R}^d$. Let the $(d-1)$-sphere $S^{d-1} \subseteq \mathbb{R}^d$ be the set of points with distance exactly equal to 1 from the origin in $\mathbb{R}^d$. Note the following obvious homeomorphism:

Fact 3.5.1. [37] For any k-simplex σ, the boundary of σ is homeomorphic to a $(k-1)$-sphere, and σ is homeomorphic to a k-ball.

Connectivity is a general notion of graph connectivity adapted for higher dimensions. The concept, which is fundamental to this work, is formalized below.

Definition 3.5.2. A simplicial complex $\mathcal{K}$ is *x-connected*, for $x \geq 0$, if any continuous map $f : |\mathcal{K}'| \to S^x$, where $\mathcal{K}' \subseteq \mathcal{K}$ is homeomorphic to an x-sphere, can be extended into a continuous map $g : |\mathcal{K}''| \to D^{x+1}$, where $\mathcal{K}'' \subseteq \mathcal{K}$ is homeomorphic to an $(x+1)$-ball. We require $\mathcal{K}' \subset \mathcal{K}''$ and $g(x) = f(x)$ if $x \in |\mathcal{K}'|$.

In analogy, think of the extremes of a pencil as a 0-sphere, and the pencil itself as a 1-ball (the extension is possible if 0-connected); the rim of a coin as a 1-sphere, and the coin itself as a 2-ball (the extension is possible if 1-connected); the outer layer of a billiard ball as a 2-sphere, and the billiard ball itself as a 3-ball (the extension is possible if 2-connected). Figure 3.6 depicts the extension from f to g, and Fig. 3.7 shows a simplicial complex that is 0-connected but not 1-connected.

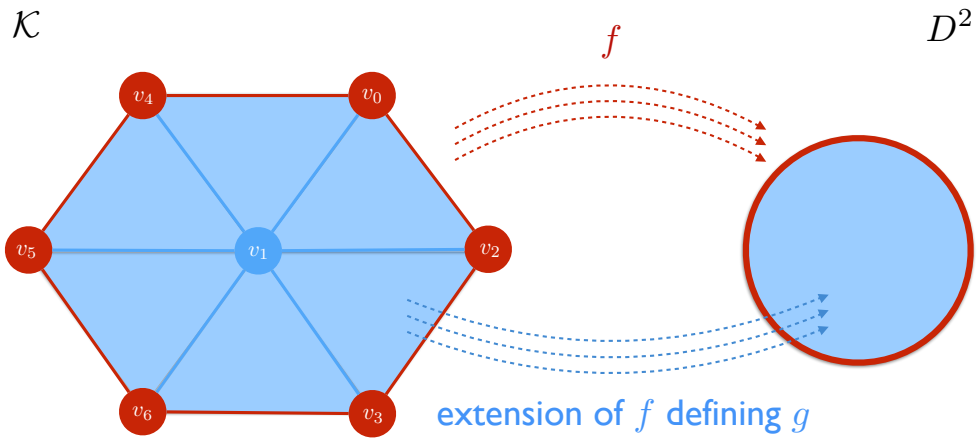

Figure 3.6: Extending a map f with range S^1 to a map g with range D^2. The extension is possible because K is 1-connected.

In this text, (-1)-connected is understood as *non-empty*, and (-2)-connected or lower imposes no restriction.

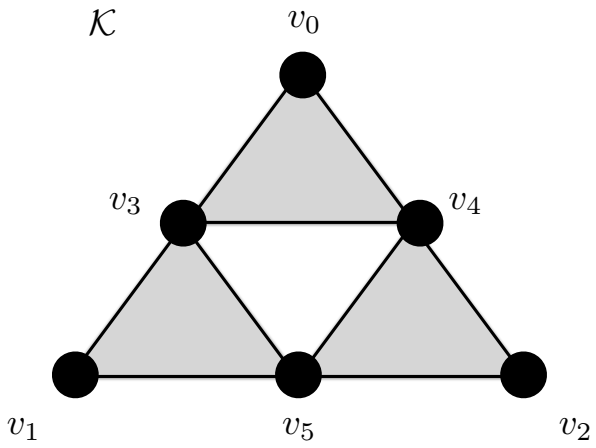

Figure 3.7: The simplicial complex $\mathcal{K}$ is 0-connected but not 1-connected.

3.6 Modeling Tasks using Simplicial Complexes

In this section, we see how crash-failure tasks can be formalized using tools of combinatorial topology, as in [22]. We delay the formalization of Byzantine tasks until Chap. 5.

Recall from Chap. 2 that the set of processes is denoted by $\mathbb{P}$, and the input (respectively, the output) of process $P_i \in \mathbb{P}$ is denoted by I_i (respectively, O_i). In our model, a vertex (P_i, V_i) represents that value V_i is attributed to P_i, and a simplex represents a global attribution of values to processes. If V_i's represent inputs, we are defining valid input configurations; if V_i's represent outputs, we are defining valid outputs configurations. Let us formalize the concept through the following definitions.

Definition 3.6.1. A *name-labeled* simplex σ is a simplex where:

1. for any vertex $v \in \sigma$ we have $v = (P_i, V_i)$ with $P_i \in \mathbb{P}$;
2. if $(P_i, V_i) \in \sigma$ and $(P_j, V_j) \in \sigma$ then $P_i \neq P_j$.

A name-labeled simplicial complex $\mathcal{K}$ is defined to contain only name-labeled simplexes.

Definition 3.6.2. For any name-labeled simplex σ,

$$\text{names}(\sigma) = \{P_i : \exists V \text{ such that } (P_i, V) \in \sigma\},$$
$$\text{views}(\sigma) = \{V_i : \exists P \text{ such that } (P, V_i) \in \sigma\}.$$

Definition 3.6.3. For any name-labeled simplicial complex $\mathcal{K}$,

$$\text{names}(\mathcal{K}) = \bigcup_{\sigma \in \mathcal{K}} \text{names}(\sigma) \text{ and } \text{views}(\mathcal{K}) = \bigcup_{\sigma \in \mathcal{K}} \text{views}(\sigma).$$

Definition 3.6.4. A *canonical* simplex σ is a simplex with $\dim(\sigma) \geq (|\mathbb{P}| - 1) - t$.

Definition 3.6.5. Two name-labeled simplices σ_1 and σ_2 *match* if

$$\text{names}(\sigma_1) = \text{names}(\sigma_2).$$

Definition 3.6.6. Let $\mathcal{K}$ and $\mathcal{L}$ be name-labeled simplicial complexes. We say that a carrier map $\Phi : \mathcal{K} \to 2^{\mathcal{L}}$ is *name-preserving* if, for any $\sigma \in \mathcal{K}$, also including vertices of $\mathcal{K}$, $\text{names}(\sigma) = \text{names}(\Phi(\sigma))$.

For any task $\mathcal{T}$, an *initial configuration* for $\mathcal{T}$ is $\sigma_I = \{(P_i, I_i) : I_i \text{ is input by } P_i\}$, a canonical name-labeled simplex describing a permitted input to $\mathcal{T}$. A *final configuration* for $\mathcal{T}$ is $\sigma_O = \{(P_i, O_i) : O_i \text{ is output by } P_i\}$, a canonical name-labeled simplex describing a permitted output of $\mathcal{T}$. Any initial or final configuration is a canonical simplex because at least $|\mathbb{P}| - t$ non-faulty processes start and finish every computation of $\mathcal{T}$.

Definition 3.6.7. A crash-failure *task specification* is formally a triple $\mathcal{T} = (\mathcal{I}, \mathcal{O}, \Delta)$ such that:

- $\mathcal{I}$ is the *input complex*. A simplex $\sigma \in \mathcal{I}$ if there is some $\sigma_I \supseteq \sigma$ that is an initial configuration, with $\dim(\sigma_I) = |\mathbb{P}| - 1$.
- $\mathcal{O}$ is the *output complex*. A simplex $\sigma \in \mathcal{O}$ if there is some $\sigma_O \supseteq \sigma$ that is a final configuration, with $\dim(\sigma_O) = |\mathbb{P}| - 1$.
- $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$ is a name-preserving carrier map. The simplex $\tau \in \Delta(\sigma)$ if the final configuration τ is valid given the initial configuration σ, with σ matching τ. Also, $\Delta(\sigma') = \emptyset$ for any non-canonical simplex σ' of $\mathcal{I}$.

If an initial configuration σ has $\Delta(\sigma) = \emptyset$, that is, has no associated final configuration, we effectively preclude any protocol. Therefore, interesting tasks are defined such that $\Delta(\sigma) \neq \emptyset$ for any initial configuration σ. We illustrate the relationship between input and output complexes, and the map Δ, in Fig. 3.8.

In this text, when name-labeling or canonicity are obvious or irrelevant, the adjective is omitted.

3.7 Pseudospheres

Informally, a *pseudosphere* is a simplicial complex formed by assigning values to processes independently, such that a process P_i receives only values from a set S_i. In many problems, input and output complexes are pseudospheres or unions of pseudospheres.

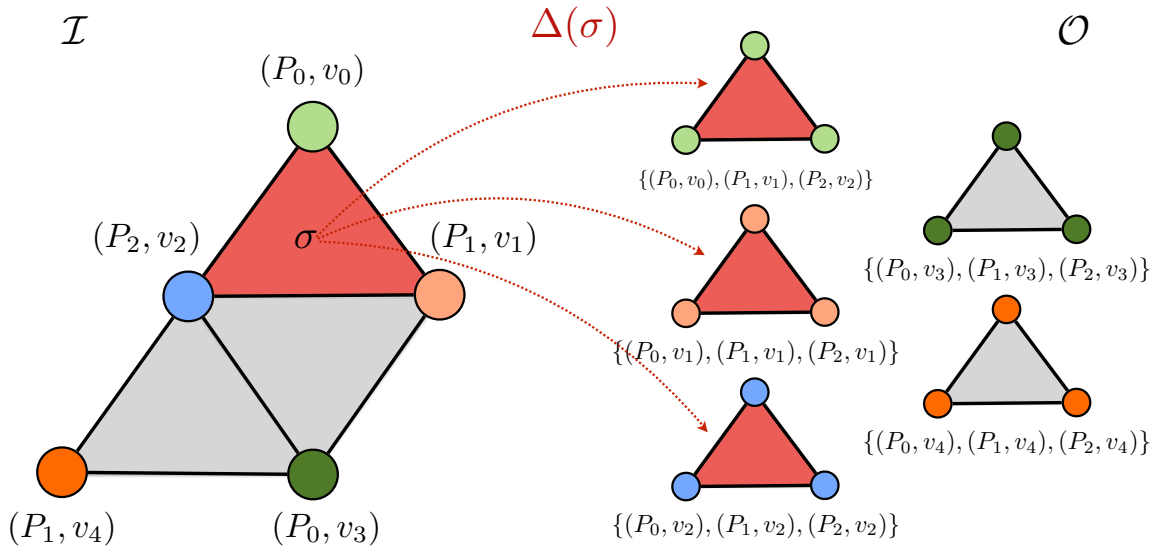

Figure 3.8: An example of a *consensus* task for the initial configurations defined in $\mathcal{I}$.

Definition 3.7.1. Let $\mathbb{S} = \{(P_i, S_i) : P_i \in \mathbb{P}'\}$, where each S_i is an arbitrary set and $\mathbb{P}' \subseteq \mathbb{P}$. A *pseudosphere* $\Psi(\mathbb{P}', \mathbb{S})$ is a simplicial complex where, if $\sigma = \{(P_i, V_i) : P_i \in \mathbb{P}', V_i \in S_i\}$, $\sigma \in \Psi(\mathbb{P}', \mathbb{S})$.

Essentially, a pseudosphere is a simplicial complex formed by independently assigning values to all the specified processes. If $S_i = S$ for all $P_i \in \mathbb{P}'$, we call the resulting pseudosphere a *simple pseudosphere*, we simply write $\Psi(\mathbb{P}', S)$. Figure 3.9 gives an example of a simple pseudosphere.

3.8 Shellability

Informally speaking, a simplicial complex is shellable if it can be built by gluing its x-simplexes along their $(x-1)$ faces only. For example, the complex in Fig. 3.1 can be built by gluing its simplexes clockwise, and every new simplex intersects the accrued construction along its border only.

Definition 3.8.1. A pure, simplicial complex $\mathcal{K}$ is *shellable* if we can arrange the facets of $\mathcal{K}$ in a linear order $\phi_0 \ldots, \phi_t$ such that $\left(\bigcup_{0 \leq i < k} \phi_i\right) \cap \phi_k$ is a pure $(\dim(\phi_k) - 1)$-dimensional simplicial complex for all $0 < k \leq t$. We call the above linear order $\phi_0, \ldots, \phi_t$ a *shelling order*.

Note that $\phi_0, \ldots, \phi_t$ is a shelling order if any $\phi_i \cap \phi_j$ $(0 \leq i < j \leq t)$ is contained in a

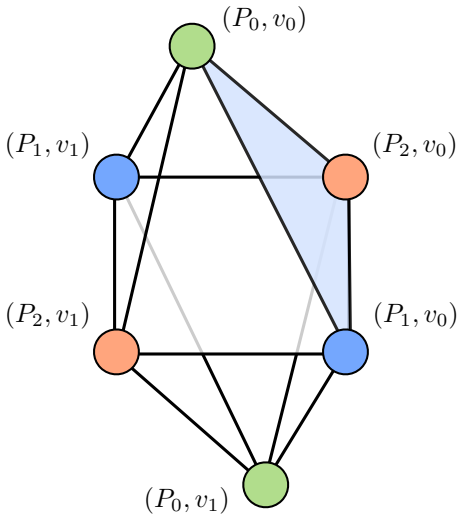

Figure 3.9: A pseudosphere $\Psi(\{P_0, P_1, P_2\}, \{v_0, v_1\})$.

$(\dim(\phi_k) - 1)$-face of ϕ_k $(0 \leq k < j)$. Hence,

$$\text{for any } i < j \text{ exists } k < j \text{ where } (\phi_i \cap \phi_j) \subseteq (\phi_k \cap \phi_j) \text{ and } |\phi_j \setminus \phi_k| = 1. \tag{3.2}$$

Shellability and pseudospheres are important tools to characterize connectivity in simplicial complexes. The following lemmas are proved in [22] and [21] (pp. 252–253).

Lemma 3.8.2. Any pseudosphere $\Psi(\mathbb{P}', \mathbb{S})$ is shellable, for any $\mathbb{S} = \{(P_i, S_i) : \forall P_i \in \mathbb{P}'\}$.

Proof. Proof in [21], pp. 252–253. □

Lemma 3.8.3. For any $k \geq 1$, if the simplicial complex $\mathcal{K}$ is shellable and $\dim(\mathcal{K}) \geq k$ then $\mathcal{K}$ is $(k-1)$-connected.

Proof. Proof in [22]. □

3.9 Nerves

Let $\mathcal{K}$ be a simplicial complex with a *cover* $\{\mathcal{K}_i : i \in I\} = \mathcal{K}$, where I is a finite index set. The *nerve* $\mathcal{N}(\{\mathcal{K}_i : i \in I\})$ is the simplicial complex with vertexes I and simplexes $J \subseteq I$ whenever $\mathcal{K}_J = \bigcap_{j \in J} \mathcal{K}_j \neq \emptyset$. We can characterize the connectivity of $\mathcal{K}$ in terms of the connectivity of the intuitively simpler nerve of $\mathcal{K}$ with the next theorem.

Theorem 3.9.1 (Nerve Theorem [30, 5]). If for any $J \subseteq I$ denoting a simplex of the nerve $\mathcal{N}(\{\mathcal{K}_i : i \in I\})$ (thus, $\mathcal{K}_J \neq \emptyset$) we have that $\mathcal{K}_J$ is $(k - |J| + 1)$-connected, then $\mathcal{K}$ is k-connected if and only if $\mathcal{N}(\{\mathcal{K}_i : i \in I\})$ is k-connected.

In the example of Fig. 3.10, subcomplexes $\mathcal{K}_0$, $\mathcal{K}_1$ and $\mathcal{K}_2$ are 1-connected, and its nerve $\mathcal{N}(\{\mathcal{K}_0, \mathcal{K}_1, \mathcal{K}_2\})$ is 0-connected, but not 1-connected. Note that every member of the cover is 0-connected, and for every pair of members of the cover, their intersection is (-1)-connected (that is, non-empty). Applying the above theorem with $k = 0$, we can conclude that the whole complex $\mathcal{K}$ is 0-connected.

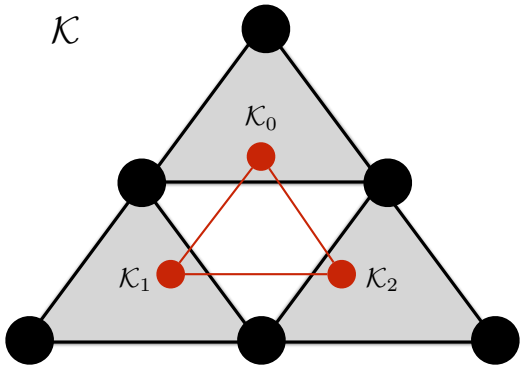

Figure 3.10: Application of the Nerve Lemma to reason about connectivity.

3.10 Protocol Complexes

In this work, we use simplicial complexes more than just to formalize tasks in terms of their inputs and outputs. We abstract protocols as continuous exchanges of internal states, in a full-information fashion [25], so we can use simplicial complexes to model executions. These complexes are called *protocol complexes*, which are formalized below for synchronous and asynchronous systems.

3.10.1 Asynchronous Systems

In an asynchronous system, we are mostly interested in representing the communication structure after the protocol finishes. For any task $(\mathcal{I}, \mathcal{O}, \Delta)$, a *protocol complex* $\mathcal{P}(\mathcal{I})$ is a simplicial complex that models the possible final states of the protocol execution for any input configuration. The possible final states for an input configuration represented by $\sigma \in \mathcal{I}$ denotes the simplicial complex $\mathcal{P}(\sigma) \subseteq \mathcal{P}(\mathcal{I})$.

The protocol complex $\mathcal{P}(\mathcal{I})$ has name-labeled n-simplexes that represent all possible global states at the end of the protocol. It is defined as follows: if $\sigma \in \mathcal{P}(\mathcal{I})$, then $\sigma = \{(P_i, \text{view}(P_i)) : \forall P_i \in \mathbb{G}\}$, where $\text{view}(P_i)$ represents P_i's view after the protocol terminates.

3.10.2 Synchronous Systems

In a synchronous system, the communication structure allows the modeling of the execution throughout the rounds. For any task $(\mathcal{I}, \mathcal{O}, \Delta)$, a *round-i protocol complex* $\mathcal{P}^i(\mathcal{I})$ is a simplicial complex that models the possible states of the protocol execution for any input configuration, at round i. The possible states for an input configuration represented by $\sigma \in \mathcal{I}$, at round i, denotes the simplicial complex $\mathcal{P}^i(\sigma) \subseteq \mathcal{P}^i(\mathcal{I})$.

Denote the processes behaving in strict accordance to the protocol for rounds 1 up to r (inclusive) as $\mathbb{G}^r$. For convenience of notation, define a round 0 as the state just before the beginning of the protocol, with $\mathbb{G}^0 = \mathbb{P}$. The round-0 protocol complex $\mathcal{P}^0(\mathcal{I})$ has name-labeled n-simplexes $\sigma_I = \{(P_i, I_i) : \forall P_i \in \mathbb{G}^0\}$, representing all the possible process inputs in the beginning of the protocol. After each round r, the round-r protocol complex $\mathcal{P}^r(\mathcal{I})$ represents all possible global states of the system at round r. For any $r \geq 0$, it is defined as follows: if $\sigma \in \mathcal{P}^r(\mathcal{I})$, then $\sigma = \{(P_i, \text{view}^r(P_i)) : \forall P_i \in \mathbb{G}^r\}$, where $\text{view}^r(P_i)$ represents P_i's view after round r.

3.11 Protocols and Decision

Protocol complexes are important to characterize the ability of processes to decide after the execution of the protocol. The formal definition of *protocol* is identical to [24], here presented for completeness:

Definition 3.11.1. A protocol for $(\mathcal{I}, \mathcal{O}, \Delta)$ is a carrier map $\mathcal{P}$ that takes $\sigma \in \mathcal{I}$ to a simplicial complex denoted by $\mathcal{P}(\sigma)$. Let $\mathcal{P}(\mathcal{I}) = \cup_{\sigma \in \mathcal{I}} \mathcal{P}(\sigma)$.

Definition 3.11.2. A protocol $\mathcal{P}$ solves $(\mathcal{I}, \mathcal{O}, \Delta)$ if there exists a simplicial map $\delta : \mathcal{P}(\mathcal{I}) \to \mathcal{O}$ carried by Δ.

Chapter 4

Asynchronous Multidimensional ϵ-Approximate Agreement

In this chapter, we discuss a generalization of the traditional consensus [18] and approximate agreement [15] problems in light of Byzantine failures. Informally, in our *multidimensional ϵ-approximate agreement task*, for arbitrary $\epsilon > 0$ and $d \geq 1$, each process starts with an input value in $\mathbb{R}^d$, and all non-faulty processes must choose output values, also in $\mathbb{R}^d$, such that: (1) all outputs lie within ϵ of one another; and (2) all outputs lie in the convex hull of the inputs of the non-faulty processes. We provide an optimal protocol for this problem, in terms of resilience, for asynchronous Byzantine systems. The protocol centers around the *safe area* concept, permitting processes to converge in Euclidean space of arbitrary dimension. We use geometric arguments, such as Helly's Theorem, to analyze and prove the correctness of the protocol.

4.1 Contributions Overview and Related Work

Our multidimensional ϵ-approximate agreement problem has a long and fascinating background. In the *consensus task* [18], processes start with values from an arbitrary domain, and must choose values satisfying:

Agreement: all processes decide on the same value; and

Validity: the decided value is some process' input.

It is well-known that asynchronous consensus is impossible in the presence of even a single crash failure [18], so the result naturally extends to Byzantine failures.

There are many ways to circumvent this impossibility: requiring synchrony, employing randomization, considering weaker agreement requirements, among others. One particularly interesting relaxation is to settle for (unidimensional) ϵ-*approximate agreement*, described in [15] back in 1986. There, non-faulty processes start with input values in $\mathbb{R}$, and must also finish with output values in $\mathbb{R}$, satisfying:

Agreement: all non-faulty processes decide on values that are within an arbitrary distance $\epsilon \geq 0$ of each other; and

Validity: all non-faulty processes decide on values in the range of the non-faulty process inputs.

In contrast to consensus, the unidimensional asynchronous ϵ-approximate agreement is possible, even with (a limited number of) Byzantine failures. We can think of the protocol geometrically: non-faulty processes start on points in $\mathbb{R}$, and then converge to arbitrarily close values in the range of their inputs, despite difficulties presented by asynchrony and malicious behavior of a subset of processes.

Let n be the number of participating processes, and t be an upper bound on the number of Byzantine processes. For asynchronous message-passing systems, early protocols required $n > 5t$ [15, 16], improved later to $n > 3t$ [1], which is optimal in terms of resilience [17]. The optimal protocol uses *reliable broadcast* [3, 44, 8] to force Byzantine processes to communicate consistently (avoiding the situation where these tell different things to different processes), as well as the *witness technique* [1] to improve data collection under failures.

4.2 Operational Model

Assuming n processes rather than $n+1$ simplifies the computing notation, but considerably complicates the topological notation. Since this chapter is solely devoted to the definition and analysis of the multidimensional ϵ-agreement task, and does not concern topological tools:

We let number of processes be n. Define $\mathbb{P} = \{P_0, \ldots, P_{n-1}\}$.

Processes exchange messages via pairwise distinct channels. Channels form a complete graph, and are *reliable*, *authenticated*, and *FIFO*: all transmitted messages are eventually delivered, in the order they were sent, and the sender is always correctly identified. Communication is *asynchronous*: the delivery of any message happens after a finite, yet unbounded

delay. The processes are asynchronous as well, with unbound relative speed. The reasonableness of this setting is discussed in Chap. 2.

Any set of up to t processes might be arbitrarily *faulty*, characterizing a *Byzantine-failure* model [31]. Faulty processes display arbitrary, even malicious behavior, which includes collusion to prevent the protocol to terminate correctly. Specifically, the faulty processes are any set of no more than t processes chosen by an *adversary*. Byzantine processes may execute the protocol correctly or incorrectly, at the discretion of the adversary. The set of all processes is denoted by $\mathbb{P}$, partitioned in non-faulty processes $\mathbb{G} \subseteq \mathbb{P}$ and faulty processes $\bar{\mathbb{G}} = \mathbb{P} \setminus \mathbb{G}$.

4.3 Formal Definition

Every process $P_i \in \mathbb{G}$ has an input $I_i \in \mathbb{R}^d$ and an output $O_i \in \mathbb{R}^d$. Let $I_G = \{I_i : P_i \in \mathbb{G}\}$ be called *non-faulty inputs*. After we run the multidimensional ϵ-approximate agreement protocol, we require the following:

Agreement: for any non-faulty processes P_i and P_j, the Euclidean distance between their outputs O_i and O_j is at most ϵ, an error tolerance fixed *a priori*.

Convexity: for any non-faulty process P_i, its output O_i is inside the convex hull of the non-faulty inputs I_G.

4.3.1 On the Generalization of Approximate Agreement

The convexity requirement is the natural generalization of the 1-dimensional validity condition, and it is essential to some applications, as discussed later. The convex hull of the non-faulty process inputs is independent of the choice of coordinate system. Convexity guarantees that if such inputs lie in a linear subspace of $\mathbb{R}^d$ (say, along a line), the non-faulty process outputs lie also in such linear subspace.

Note that our protocol performance, measured in the number of messages required to be sent, depends only on the inputs of the non-faulty processes, and is unaffected by any malicious behavior of the Byzantine processes.

While we converge values dimension-by-dimension in our protocol, we *cannot* simply reuse the 1-dimensional protocol for that goal. Fig. 4.1 shows the problem, taking $d = 2$: with the 1-dimensional algorithm used in consecutive dimensions, we may output any point in the highlighted rectangle, but the actual allowed convex hull is strictly smaller. With the safe area concept, however, we can make sure that all the individual convergence steps

maintain convexity (see Sec. 4.8). A matching lower bound for fault tolerance confirms that the safe area concept, which is fundamental for correctness, underlies an optimally-resilient protocol (see Sec. 4.7).

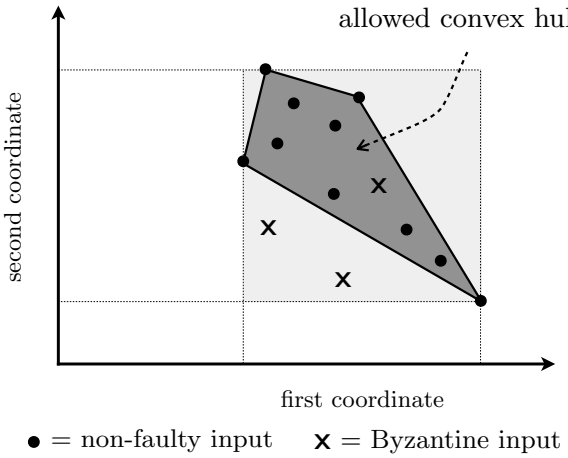

Figure 4.1: Performing traditional approximate agreement in separate dimensions breaks convexity.

In this chapter, we present a *t-resilient protocol* for the multidimensional ϵ-agreement problem, which is an algorithm guaranteeing that each non-faulty process decides on a correct output, despite the presence of up to t Byzantine processes [34]. We note that similar results were discovered independently and concurrently by Vaidya and Garg [45]. A combined paper unifies both ideas in [35].

4.4 Applications

Besides theoretical interest, our protocol has practical applicability. We now discuss some of those applications for the interested reader.

Robot convergence: Consider autonomous mobile entities, such as robots, that must converge to nearby locations in the 2 or 3-dimensional space. They must do so despite arbitrary behavior of a subset of robots (Byzantine failures), and unbounded communication delays (asynchronous communication). In other words, they cannot discern between benign and malicious robots, likewise between failed or slowly responsive ones. Finally, non-faulty robots must *respect convexity*, and only move within their original convex region – or they would wander through unsafe territory, say. Byzantine robots must not have the power to influence benign robots to move outside their original convex region.

Previous results in the robot network literature relate approximate agreement with robot convergence in the real line [7, 41]. Our protocol works if the maximum number of faulty robots, t, compared to the number of existent robots, n, is $t < n/4$ for the 2-dimensional case, and $t < n/5$ for the 3-dimensional case. In the terminology of Potop-Butucaru et al. [41], our protocols are fully asynchronous, cautious, non-oblivious *variants* of the CORDA model presented there.

Distributed voting: Say distributed voters must choose from a number of options. Each voter gives its relative preferences by assigning *weights* to options, where the weights sum up to one. For example, for three options, a voter may give 0.3 option a, 0.6 to option b, and 0.1 to option c. A preference can be viewed as the barycentric coordinates of a point in the triangle of Fig. 4.2.

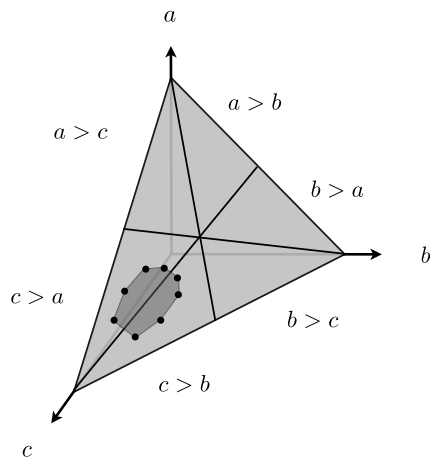

Figure 4.2: Respecting convexity means respecting unanimity in voting systems.

As the result of the voting, each process receives a new assignment of weights to options, and all assignments agree to within ϵ, and all lie within the convex hull of the original votes. The convexity requirement implicates that the voting respects unanimity: if all voters prefer c over a and b (as in Fig. 4.2), then every voter's final assignment also reflects that. Here, our protocol is applicable, and respects convexity and unanimity in Byzantine asynchronous systems. Other related interpretations for convexity exist – see Saari [42].

4.5 Asynchronous Communication Primitives

In this section, we review two existing communication primitives, namely the reliable broadcast and the witness technique. Those primitives support our main algorithmic procedure, which is basically as follows. In multiple discrete rounds, any non-faulty processes will:

1. Broadcast its current value;
2. Receive multiple process values (including its own), never waiting for more than $n - t$ process values, since t processes might have crashed;
3. Update its current value to a particular point inside a "safe area" in $\mathbb{R}^d$, guaranteed to be in the convex hull of the non-faulty inputs.

The insight of our protocol lies in step (3), which, despite seemingly simplicity, curtains elaborate combinatorial and geometric arguments for correctness and optimality. Here, we discuss the primitives corresponding to (1) and (2).

4.5.1 Reliable Broadcast

The reliable broadcast technique avoids *equivocation* – the situation where Byzantine processes convey different contents to different processes in a single round of communication. The technique works as long as $n > 3t$, and it is thoroughly discussed in [3], with original ideas due to Srikanth and Toueg [44] and Bracha [8].

The communication is organized in asynchronous rounds, and messages are decorated with the sender identification, say p, and the current round *tag*, say r. So, a message with contents c will look like $M = (p, r, c)$. The reliable broadcast technique has the following properties:

Non-faulty integrity: If a non-faulty process p never reliably broadcasts (p, r, c), no other non-faulty process will ever receive (p, r, c).

Non-faulty liveness: If a non-faulty process p does reliably broadcast (p, r, c), all other non-faulty processes eventually receive (p, r, c).

Global uniqueness: If two non-faulty processes reliably receive (p, r, c) and (p, r, c'), the messages are equal ($c = c'$), even when the sender, p, is Byzantine.

Global liveness: For two non-faulty processes p and q, if p reliably receives (b, r, c), q also reliably receives (b, r, c), even when the sender process, b, is Byzantine.

For formal proofs the above properties, see [3].

Algorithms 2 to 4 illustrate the technique for sender p, round r, and contents c, in terms of messages exchanged in the raw asynchronous channels (via **send** and **recv**). In summary, the procedure works as follows:

1. Process p broadcasts a decorated message $M = (p, r, c)$;
2. When other processes receive M, they echo it;
3. When processes receive $n - t$ echo messages for M, they send ready messages for M;
4. When processes see $t + 1$ ready messages for M, meaning that a non-faulty process necessarily advocates the existence of M, they also send ready messages;
5. Finally, when a process receives at least $n - t$ ready messages, the original message is accepted.

Algorithm 2 p.`RBSend`$((p, r, c))$

send(p, r, c) to all processes

Algorithm 3 p.`RBEcho`$()$

upon recv(q, r, c) from q **do**
  if never sent $(p, qr\{\texttt{echo}\}, \cdot)$ **then**
    send$(p, qr\{\texttt{echo}\}, c)$ to all processes
  end if
end upon
upon recv$(\cdot, qr\{\texttt{echo}\}, c)$ from $\geq n - t$ processes **do**
  if never sent $(p, qr\{\texttt{ready}\}, \cdot)$ **then**
    send$(p, qr\{\texttt{ready}\}, c)$ to all processes
  end if
end upon
upon recv$(\cdot, qr\{\texttt{ready}\}, c)$ from $\geq t + 1$ processes **do**
  if never sent $(p, qr\{\texttt{ready}\}, \cdot)$ **then**
    send$(p, qr\{\texttt{ready}\}, c)$ to all processes
  end if
end upon

Algorithm 4 p.`RBRecv`$((q, r, c))$

recv$(\cdot, qr\{\texttt{ready}\}, c)$ from $n - t$ processes
return (q, r, c)

In this text, p.RBSend$((p, r, c))$ denotes the reliable broadcast of (p, r, c) by process p, and p.RBRecv$((q, r, c))$ the reliable receipt of (q, r, c) by p. Unless otherwise noted, all messages are exchanged via reliable broadcast.

4.5.2 Witness Technique

To promote agreement, we want that the collected values of any two non-faulty processes suitably overlap in every round. However, non-faulty processes cannot wait indefinitely for more than $n - t$ messages, as t processes might be crashed. If we use reliable broadcast, which indeed never waits for more than $n - t$ processes, any two non-faulty processes will have $n - 2t$ values in common after one round of communication. With the witness technique, originally presented by Abraham et al. [1], we can make non-faulty processes have $n - t$ common values, which is essential for our correctness and optimality arguments. The witness technique will only wait for messages certain to be delivered.

Algorithm 5 overviews the technique for process p and round r. First, p reliably receives $n - t$ messages from other processes, storing them into *Val*. Then, p reliably transmits its *report*, which contains the $n - t$ messages first collected in *Val*, and reliably receives reports from other processes, storing them into *Rep*.

Algorithm 5 p.RBReceiveWitness(r)

```
    Val, Rep, Wit ← ∅
    while |Val| < n − t do
        upon RBRecv((p_x, r, c_x)) do
4:          Val ← Val ∪ {(p_x, r, c_x)}
        end upon
    end while
    RBSend((p, r, Val))
8:  while |Wit| < n − t do
        upon RBRecv((p_x, r, c_x)) do
            Val ← Val ∪ {(p_x, r, c_x)}
        end upon
12:     upon RBRecv((p_x, r, Val_x)) do
            Rep ← Rep ∪ {(p_x, r, Val_x)}
        end upon
        Wit ← {(p_x, r, Val_x) ∈ Rep : Val_x ⊆ Val}
16: end while
    return Val
```

A *witness* for p is a process whose report consists of messages received by p, as seen in Lines 4 or 10. We note that p collects reports in *Rep* until $n - t$ witnesses are identified

in *Wit*. This eventually happens since we are certain to receive $n - t$ non-faulty process reports, and, by the global liveness of reliable broadcast, we are also certain to receive the values received by each of them.

As witnesses are obtained via reliable broadcast, any two non-faulty processes have at least $n - 2t \geq t + 1$ witnesses in common, of which at least one is non-faulty. So, they receive at least $n - t$ values in common. As formally shown in [1]:

Fact 4.5.1. If $n > 3t$, any two non-faulty processes that obtain messages through the witness technique in a single round r obtain $n - t$ messages in common.

4.6 The Safe Area

In our algorithm, non-faulty processes exchange messages containing values, using reliable broadcast. Processes exchange their "current values" (initially the input) in multiple rounds. At the end of each round, they update their current values jumping to a *safe area* inside the convex hull of the non-faulty inputs.

In each round, each non-faulty process obtains a *message set*, which contains messages (P_i, r, c_i), consisting on the sending process P_i, the current round r, and the message contents c_i (normally values in $\mathbb{R}^d$). No process appears twice in a message set: for any (P_i, r, c_i) and (P_j, r, c_j) in an arbitrary message set, $P_i \neq P_j$. Also, if two processes obtain (P_i, r, c'_i) and (P_i, r, c''_i), we have that $c'_i = c''_i$. In other words, the message contents are consistent among processes, as we use reliable broadcast.

For any message set X, the *contents of* X is the multiset $\text{Cont}(X)$ such that $c \in \text{Cont}(X)$ only when $(p, r, c) \in X$, for any process p and round r. If $c \in \text{Cont}(X)$, we can also say that X *contains* c.

Definition 4.6.1. For any message set X, if $\text{Cont}(X)$ contains only values $\in \mathbb{R}^d$, X is a *valued message set*; otherwise, X is an *unrestricted message set*.

Valued message sets can alternatively be understood as multisets of values $\in \mathbb{R}^d$ (called *valued multisets*). For any valued multiset C, the *polytope of* C, written $\text{Poly}(C)$, is the convex hull of points in C. Similarly, for any valued message set X, the *polytope of* X, written $\text{Poly}(X)$, is $\text{Poly}(\text{Cont}(X))$. Furthermore, even unrestricted message sets can alternatively be understood as multisets of arbitrary contents (called *unrestricted multisets*). Our definitions and theorems are given in terms of message sets, but, considering such correspondence, they apply to multisets as well.

Consider any message set X. Note that any $X' \subseteq X$ is similarly a message set. The

non-faulty messages of X define the set $X_G = \{(P_i, r, c_i) \in X : P_i \in \mathbb{G}\}$, and the faulty messages of X define $X_B = \{(P_i, r, c_i) \in X : P_i \in \mathbb{B}\}$. We always assume $|X| > t$ and $|X_B| \leq t$. For multisets of values, the above notation is preserved. Hence, if $C = \text{Cont}(X)$, we define $C_G = \text{Cont}(X_G)$ and $C_B = \text{Cont}(X_B)$.

For any message set X, we say that any $X' \subseteq X$ containing exactly $|X| - t$ elements is a *restriction of* X. The set of all possible restrictions is $\text{Restrict}_t(X)$. The *safe area* of X, written $\text{Safe}_t(X)$, is the intersection of the polytopes of all possible restrictions of X (see Fig. 4.3, and the following definition). The previous concepts are defined verbatim for multisets of values.

Definition 4.6.2. For any valued message set X, the safe area of X is

$$\text{Safe}_t(X) = \bigcap_{X' \in \text{Restrict}_t(X)} \text{Poly}(X').$$

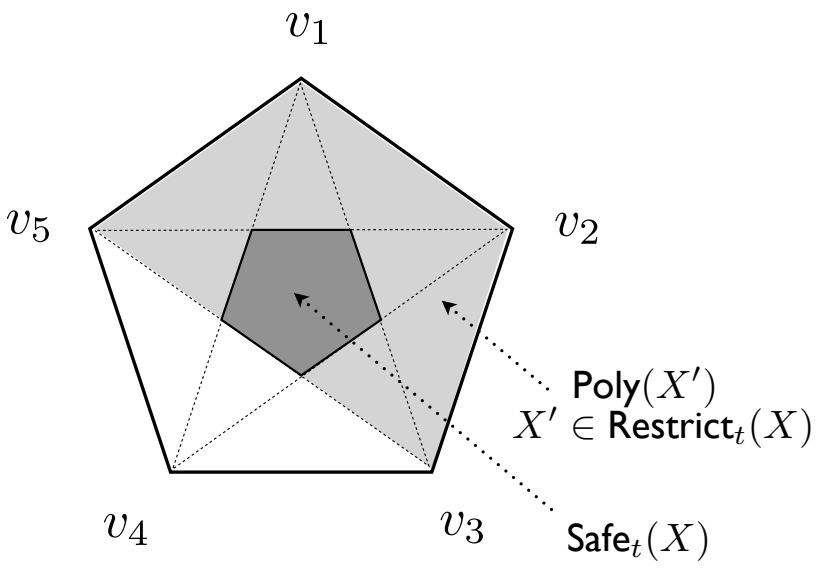

Figure 4.3: For $\text{Cont}(X) = \{v_1, \ldots, v_5\}$ and $t = 1$, we highlight $\text{Poly}(X')$, for a particular $X' \in \text{Restrict}_t(X)$, and also show $\text{Safe}_t(X)$. Values are in $\mathbb{R}^2$.

The safe area is a convex polytope, as it is an intersection of other convex polytopes.

Lemma 4.6.3. For any valued message set X, if $X' \subseteq X$, and $|X'| \geq |X| - t$, then $\text{Safe}_t(X) \subseteq \text{Poly}(X')$.

Proof. As defined before, $\text{Safe}_t(X) \subseteq \text{Poly}(X'')$, taking a particular restriction X'' of X such that $X' \supseteq X''$. Then $\text{Safe}_t(X) \subseteq \text{Poly}(X'') \subseteq \text{Poly}(X')$. □

We always consider $|X_B| \leq t$, so X always contains at least $|X| - t$ values in X_G. In other words, $|X_G| \geq |X| - t$. From the previous lemma, taking $X' = X_G$, we can conclude the following:

Corollary 4.6.4. $\text{Safe}_t(X) \subseteq \text{Poly}(X_G)$.

In the following lemmas, we relate t and $|X|$ in order to ensure the existence of the safe area. We use the following theorem from discrete geometry [13]:

Theorem 4.6.5 (Helly's Theorem). Consider any finite collection of closed convex sets $S = \{S_1, \ldots, S_x\}$ on $\mathbb{R}^d$, with $x \geq d + 1$. If every subset of $d + 1$ members of S intersect, then

$$\bigcap_{S_i \in S} S_i \neq \emptyset.$$

The relationship between t and $|X|$ as it concerns the existence of the safe area is seen in Lemmas 4.6.7 and 4.6.11. The lemma below applies to unrestricted message sets (respectively, unrestricted multisets), while the upcoming lemmas apply to valued message sets (respectively, valued multisets) only. Unless otherwise noted, we presume that message sets and multisets are valued.

Lemma 4.6.6. For any $X_1, \ldots, X_j \in \text{Restrict}_t(X)$, where X is an unrestricted message set,

$$\left| \bigcap_{1 \leq i \leq j} X_i \right| \geq |X| - jf.$$

Proof. We prove the lemma by induction on j.

Base. For $j = 1$, we have that

$$\left| \bigcap_{1 \leq i \leq 1} X_i \right| = |X_1| = |X| - t = |X| - 1t,$$

since $X_1 \in \text{Restrict}_t(X)$.

Induction Hypothesis. Assume valid

$$\left| \bigcap_{1 \leq i \leq k} X_i \right| \geq |X| - kt,$$

for $k < d + 1$.

$$\begin{aligned} \left| \bigcap_{1 \leq i \leq k+1} X_i \right| &= \left| \left(\bigcap_{1 \leq i \leq k} X_i \right) \cap X_{k+1} \right| && (\star) \\ &\geq |X| - kt - t = |X| - (k+1)t && (\star\star). \end{aligned}$$

($\star$) holds by associativity of $\cap$; ($\star\star$) happens as at most t values in $\cap_{1\leq i\leq k}\text{Cont}(X_i)$ are not in $\text{Cont}(X_{k+1})$. $\square$

Lemma 4.6.7. For any valued message set X having values in $\mathbb{R}^d$, if $|X| > (d+1)t$, then $\text{Safe}_t(X) \neq \emptyset$.

Proof. As $|X| > (d+1)t$, by definition of $\text{Restrict}_t(X)$, we know that $|\text{Restrict}_t(X)| \geq d+1$. Therefore, consider any $X_1, \ldots, X_{d+1} \in \text{Restrict}_t(X)$. By Lemma 4.6.6,

$$\left| \bigcap_{1\leq i\leq d+1} X_i \right| \geq |X| - (d+1)t > (d+1)t - (d+1)t \geq 1.$$

Any $X_1, \ldots, X_{d+1}$ from $\text{Restrict}_t(X)$ have a non-empty intersection. Therefore, we have that any $\text{Poly}(X_1), \ldots, \text{Poly}(X_{d+1})$ from $S = \{\text{Poly}(X') : X' \in \text{Restrict}_t(X)\}$ will have a non-empty intersection as well. By our first observation, we know that $|S| \geq d+1$.

Finally, as all message contents are in $\mathbb{R}^d$, applying Helly's Theorem will imply that *all* subsets of $S = \{\text{Poly}(X') : X' \in \text{Restrict}_t(X)\}$ will also have a non-empty intersection, and therefore $\text{Safe}_t(X) \neq \emptyset$. $\square$

Now, we characterize special arrangements of values in $\mathbb{R}^d$, used in further discussions and proofs. Regarding notation, we denote the m-th component of a vector v as $v[m]$.

Definition 4.6.8. Consider an arbitrary $\epsilon > 0$. A standard basic vector $e_m \in \mathbb{R}^d$ is such that $e_m[m] = 2\epsilon$, for some $1 \leq m \leq d$, and $e_m[m'] = 0$, for any $m' \neq m$. We additionally define $e_0 = 0^d$.

For example, in $\mathbb{R}^3$, we have $e_0 = [0,0,0]$, $e_1 = [2\epsilon, 0, 0]$, $e_2 = [0, 2\epsilon, 0]$, and $e_3 = [0, 0, 2\epsilon]$. Note that the distance between any different standard basic vectors exceeds our threshold ϵ in the case of multidimensional approximate agreement. These vectors might be arranged as follows:

Definition 4.6.9. A valued message set X forms a (k,t)-*simplicial state* in $\mathbb{R}^d$, for some $1 \leq k \leq d+1$, when X contains k different standard basic values (which we call $x_0, \ldots, x_{k-1}$) in $\mathbb{R}^d$ such that

1. For any $x \in \text{Cont}(X)$, we have that $x = x_m$ with some $0 \leq m \leq k-1$.
2. For any $x \in \text{Cont}(X)$, we have that x appears at most t times in $\text{Cont}(X)$.

In other words, X consists solely of k arbitrary standard basic vectors, each with at most t appearances.

In a (k,t)-simplicial state, geometrically speaking, the points form a geometrical $(k-1)$-simplex $\sigma = (x_0, \ldots, x_{k-1})$, with $x_0 \ldots x_{k-1}$ being different standard basic vectors in $\mathbb{R}^d$. For instance, Fig. 4.4 shows the simplex defined by a $(4,t)$-simplicial state, in which $4t$ values are located in 4 different positions corresponding to standard basic vectors, $e_0, \ldots, e_3$, having exactly t values in a single position.

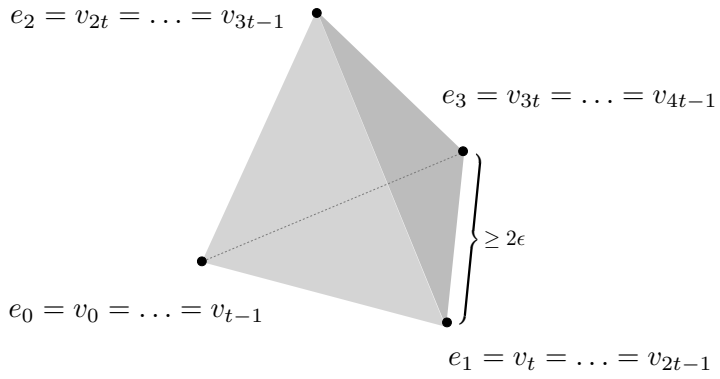

Figure 4.4: $v_0, \ldots, v_{4t-1} \in \text{Cont}(X)$, which configures a $(4,t)$-simplicial state.

Lemma 4.6.10. If the message set or multiset X configures a (k,t)-simplicial state in $\mathbb{R}^d$, for any $1 \leq k \leq d+1$, then $\text{Safe}_t(X) = \emptyset$.

Proof. Without losing generality, say that $\text{Cont}(X)$ consists solely of standard basic values $e_0 \ldots e_{k-1}$. By definition of (k,t)-simplicial state, each standard basic value appears at most t times in $\text{Cont}(X)$.

Define $X(m) = \{(p, r, e_i) \in X : i \neq m\}$, for any $0 \leq m \leq k-1$. This is the message set $X(m) \subseteq X$ excluding all messages with contents e_m. As discussed above, $\text{Poly}(X)$ denotes a geometrical simplex, say σ. Similarly, $\text{Poly}(X(m))$ denotes a proper face, say τ_m, of the geometrical simplex σ.

Since $|X(m)| \geq |X| - t$, we apply Lemma 4.6.3 for all $X(m)$, with $0 \leq m \leq k-1$. We have

$$\text{Safe}_t(X) \subseteq \bigcap_{0 \leq m \leq k-1} \text{Poly}(X(m)) = \bigcap_{0 \leq m \leq k-1} |\tau_m|,$$

which is empty, as it is the intersection of all the proper faces of the geometrical simplex σ. For standard definitions on simplicial topology, see Chap. 3. □

Lemma 4.6.11. In $\mathbb{R}^d$, there exists a valued message set X where $|X| \leq (d+1)t$ and $\text{Safe}_t(X) = \emptyset$.

Proof. Assume $\text{Cont}(X) = \{x_0, \ldots, x_{(d+1)t-1}\}$, with $x_i = e_{(i \mod d+1)}$. Hence, X configures a $(d+1, t)$-simplicial state, so, by the previous lemma, we have that $\text{Safe}_t(X) = \emptyset$. □

In this section, we formalized the concept of the safe area, and we showed that for any valued message set X (respectively, valued multiset), if $|X| > (d+1)t$ then its safe area is necessarily nonempty, and if $|X| \leq (d+1)t$ then its safe area can be empty.

4.6.1 Properties

In this section, we give some important properties of the safe area, exploited later in our algorithms. Consider a valued message set X, with $|X| > t$, and define a valued message as a message containing a value $\in \mathbb{R}^d$.

Lemma 4.6.12. $\text{Safe}_t(X) \subseteq \text{Safe}_{t-1}(X)$.

Proof.

$$\begin{aligned}
\text{Safe}_t(X) &= \bigcap_{X' \in \text{Restrict}_t(X)} \text{Poly}(X') \\
&\subseteq \bigcap_{X' \in \text{Restrict}_t(X)} \left(\bigcap_{M \in X \setminus X'} \text{Poly}(X' \cup \{M\}) \right) \\
&= \bigcap_{X' \in \text{Restrict}_{t-1}(X)} \text{Poly}(X') \quad (\star) \\
&= \text{Safe}_{t-1}(X),
\end{aligned}$$

where $(\star)$ happens since $\cap$ is associative. □

Lemma 4.6.13. For any valued message $M \notin X$, we have that $\text{Safe}_t(X) \subseteq \text{Safe}_t(X \cup \{M\})$.

Proof. First, we note that

$$\begin{aligned}
\text{Safe}_t(X) &= \bigcap_{X' \in \text{Restrict}_t(X)} \text{Poly}(X') \\
&\subseteq \bigcap_{X' \in \text{Restrict}_t(X)} \text{Poly}(X' \cup \{M\}) \\
&= \bigcap_{X' \in \text{Restrict}_t(X \cup \{M\}), M \in X'} \text{Poly}(X') = A,
\end{aligned}$$

calling A the intersection of convex polytopes of $|X \cup \{M\}| - t$ members of $X \cup \{M\}$ that *include* M.

Second, by the previous lemma, we note that

$$\begin{aligned}\text{Safe}_t(X) &\subseteq \text{Safe}_{t-1}(X) \\ &= \bigcap_{X' \in \text{Restrict}_{t-1}(X)} \text{Poly}(X') \\ &= \bigcap_{X' \in \text{Restrict}_t(X \cup \{M\}), M \notin X'} \text{Poly}(X') = B,\end{aligned}$$

calling B the intersection of convex polytopes of $|X \cup \{M\}| - t$ members of $X \cup \{M\}$ that *exclude* M. Therefore,

$$\begin{aligned}\text{Safe}_t(X) &\subseteq A \cap B \\ &= \bigcap_{X' \in \text{Restrict}_t(X \cup \{M\})} \text{Poly}(X') \\ &= \text{Safe}_t(X \cup \{M\}),\end{aligned}$$

concluding our proof. □

4.7 Necessary Condition for the Protocol

In this section, we give a necessary condition to solve multidimensional approximate agreement, providing a lower bound on the number of non-faulty processes in relation to the number of Byzantine processes. We follow the argumentation of [35].

Theorem 4.7.1. The condition $n > (d+2)t$ is necessary to solve the multidimensional approximate agreement on asynchronous systems.

Proof. We first consider $n = (d+2)t$ processes, denoted by $P_0, \ldots, P_{(d+2)t-1}$. Let the input value of process P_i, for any $0 \leq i < (d+1)t$, be the standard basic vector $e_{(i \bmod (d+1))}$. Call the processes that start with input e_j as S_j, and note that $|S_j| \leq t$ for all $0 \leq j \leq d$. Call the remaining processes $S_{d+2} = \{P_i : i \geq (d+1)t\}$.

As any correct algorithm must tolerate t failures, all non-faulty process must terminate in every execution in which processes in S_{d+2} never send any message. Suppose that all processes are non-faulty, but processes in S_{d+2} never send any message until all other processes terminate. When an arbitrary process $p \in S_x$, for any $0 \leq x < (d+1)$, terminates, it cannot distinguish among the following scenarios:

- All processes in S_{d+2} have crashed.

  In this case, to satisfy the *convexity* condition, the output value of p must be in the convex hull of the inputs of the processes in $S_1 \cup \cdots \cup S_{d+1}$. That is, the output value must be in the convex hull of

  $$D_{d+2} = \{e_i : 0 \leq i \leq d\}.$$

- All processes in S_y, with $y \neq x$ and $0 \leq y \leq d$ are faulty; all process in S_{d+2} have their message delayed.

  Recall that we are considering p at the time when it terminates. Since processes in S_{d+2} never send any message until p terminates, p cannot obtain any information about the inputs of processes in S_{d+2}. All processes in S_y have been assumed as faulty, so p cannot trust the input e_y. Thus, to satisfy the validity condition, the decision of process p must be in the convex hull defined below.

  $$D_y = \{e_i : 0 \leq i \leq d \text{ and } i \neq y\}.$$

Since $D_y \subseteq D_{d+2}$ for any $0 \leq y \leq d$, the output value of our arbitrary process p is in

$$\bigcap_{\substack{0 \leq i \leq d \\ i \neq x}} D_i,$$

which is precisely $\{e_x\}$ since we assumed $p \in S_x$.

Thus, the decision of any process in S_i is e_i for all $0 \leq i \leq d$. However, the input values at each pair of processes in $P_0, \cdots, P_d$, for instance, differ by 2ϵ in at least one element. So, the ϵ-agreement condition is not satisfied, which implies that $n = (d+2)t$ is insufficient to solve the problem. The case where $n < (d+2)t$ is similarly insufficient by an identical argument. Therefore, having $n > (d+2)t$ is a necessary condition to solve the problem in asynchronous systems. □

4.8 Protocol

We now present an algorithm for multidimensional approximate agreement over values, originally described in [34] and presented again in [35]. The process input $I \in \mathbb{R}^d$ is passed as argument. On Line 1, `CalculateRounds` takes the input and returns R, the number of rounds needed to converge along each dimension $1 \leq m \leq d$. The procedure also provides the process an updated current value $v \in \mathbb{R}^d$.

Note that although each process' output value is computed dimension-by-dimension, these computations are not really independent. First, both the number of rounds and the starting value are computed using the procedure `CalculateRounds`, which operates under a holistic approach (see Sec. 4.8.3). Most importantly, the safe area concept allows processes to choose values always within Poly(I_G).

For each dimension, indexed by m, we execute a number of *convergence rounds*, indexed by r, until we accept $> t$ halt messages, accumulated in H. The notation $m.r$ in Line 6 denotes that the r-th convergence round for dimension m. A non-faulty process sends a halt message for the current dimension after executing for R convergence rounds, as calculated in Line 1. After deciding, non-faulty processes should still continue to broadcast their decided values and to identify and broadcast witness reports, in order to ensure global progress.

Algorithm 6 p.AsyncAgree(I)

```
    (R, v) ← CalculateRounds(I)
    for m ← 1, ..., d do
        H ← ∅
        r ← 1
 5:     while |H| ≤ t do
            RBSend((p, m.r, v))
            upon V ← RBReceiveWitness(m.r) do
                S ← Safe_t(V)                    ▷ S(m) is the projection of S on coordinate m
                v ← v ∈ S such that v[m] = Midpoint(S(m))
10:             if r = R then
                    RBSend((p, m.r, {halt}))
                end if
                r ← r + 1
            end upon
15:         upon RBRecv((p', m.r', {halt})), with r' ≤ r do
                H ← H ∪ {(p', m.r', {halt})}
            end upon
        end while
    end for
20: return v
```

On Lines 6 and 7, the process transmits its current value to other processes via reliable broadcast, and receives the current values of other processes via the witness technique, which updates V. In Line 8, the safe area is calculated. In Line 9, the process computes the interval $S(m)$, the projection of the safe area on coordinate m, then chooses a point in S such that its m-th coordinate is in the midpoint of $S(m)$, which updates v (see Lemma 4.8.6). We discuss other properties of the safe area relevant to the algorithm in Sec. 4.8.1.

Note that a non-faulty process accepts only halt messages with an indexed round bigger than or equal to the current round. This is essential for convergence, as seen in Sec. 4.8.2 and Sec. 4.8.3. Regarding Line 9, in Sec. 4.8.4 we show that v is well-defined at every round, and satisfies the problem requirements at termination.

Assume we are executing our procedure for dimension m, with $1 \leq m \leq d$. On process P_i and round r, V_i^r denotes the updated message set in Line 7, S_i^r the updated safe area in Line 8, and v_i^r the updated current value in Line 9. The projection of S_i^r over coordinate m is denoted by $S(m)_i^r$. We omit subscripts or superscripts when they are irrelevant or obvious.

The following concepts capture the range of the values present in specific coordinates of current values *across* non-faulty processes:

Definition 4.8.1. For any coordinate $1 \leq m \leq d$ and round r:

1. $\min(m)^r = \min\{v[m]_x^r : P_x \in \mathbb{G}\}$;
2. $\max(m)^r = \max\{v[m]_x^r : P_x \in \mathbb{G}\}$.

Definition 4.8.2. For any coordinate $1 \leq m \leq d$ and round r, the *working range* is

$$\Delta(m)^r = \max(m)^r - \min(m)^r.$$

4.8.1 Intersecting Safe Areas

Say we are executing our procedure for dimension $1 \leq m \leq d$. For any two non-faulty processes $P_i, P_j \in \mathbb{G}$ and convergence round r, define the intersection of their received message sets as $\mathcal{V}_{i,j}^r = V_i^r \cap V_j^r$, written simply as $\mathcal{V}_{i,j} = V_i \cap V_j$ if r is irrelevant or obvious. Since processes exchange values using the witness technique, we conclude the following, from Fact 4.5.1:

Corollary 4.8.3. For any two non-faulty processes $P_i, P_j \in \mathbb{G}$, we have that $|\mathcal{V}_{i,j}| \geq n - t$.

As we show next, this implies in a non-empty intersection of safe areas between any two non-faulty processes, in every round. Formally, for any non-faulty processes $P_i, P_j \in \mathbb{G}$, we will have $S_i \cap S_j \neq \emptyset$.

The reasoning is the following. Since $|\mathcal{V}_{i,j}| \geq n - t$, the safe area taking only values in $\mathcal{V}_{i,j} = V_i \cap V_j$, is non-empty. We are interested, however, in the intersection of the safe areas of $V_i \supseteq \mathcal{V}_{i,j}$ and $V_j \supseteq \mathcal{V}_{i,j}$. In Sec. 4.6.1, we saw how adding extra values to the safe area computation at P_i and P_j maintains a non-empty intersection between $S_i = \text{Safe}_t(V_i)$ and $S_j = \text{Safe}_t(V_j)$.

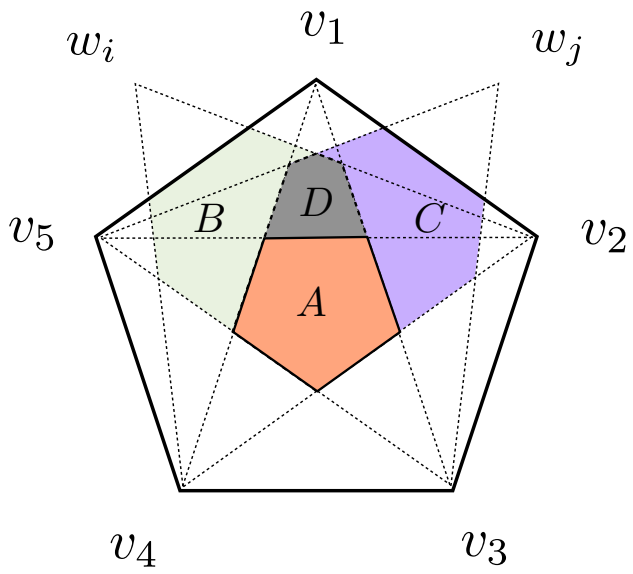

Figure 4.5: Suppose $\mathcal{V}_{i,j} = \{v_1, \ldots, v_5\}$, with $V_i = \mathcal{V}_{i,j} \cup \{w_i\}$ and $V_j = \mathcal{V}_{i,j} \cup \{w_j\}$. The safe area only grows when we consider extra points: $\text{Safe}_t(\mathcal{V}_{i,j}) = A$, $\text{Safe}_t(V_i) = A \cup B \cup D$, $\text{Safe}_t(V_j) = A \cup C \cup D$, and $\text{Safe}_t(V_i) \cap \text{Safe}_t(V_j) = A \cup D$.

Lemma 4.8.4. For any non-faulty processes $P_i, P_j \in \mathbb{G}$, and any round r, $S_i^r \cap S_j^r \neq \emptyset$

Proof. In every round, as processes exchange current values via the witness technique,

$$\begin{aligned} |\mathcal{V}_{i,j}| &\geq n - t && \text{(corollary 4.8.3)} \\ &\geq (d+2)t + 1 - t \\ &\geq (d+1)t + 1. \end{aligned}$$

By Lemma 4.6.7, we conclude that $\text{Safe}_t(\mathcal{V}_{i,j}) \neq \emptyset$. Furthermore, by an iterative application of Lemma 4.6.13, if we incorporate messages besides those of $V_i \cap V_j$, then the safe area for P_i's messages and the safe area for P_j's messages can possibly increase, but never decrease, which gives that $S_i \cap S_j \neq \emptyset$. See Fig. 4.5. □

4.8.2 Convergence

Say we are executing our procedure for dimension $1 \leq m \leq d$. If S_i^r and S_j^r intersect, they intersect when projected in any of the m coordinates.

Definition 4.8.5. Within process P_x, for coordinate m and round r, the lower limit of $S(m)_x^r$ is denoted by $\text{lo}(m)_x^r$, and the upper limit of $S(m)_x^r$ is denoted by $\text{hi}(m)_x^r$.

We effectively consider `CalculateRounds` as being round 0, so v_i^0 is v as returned by `CalculateRounds` at P_i; the working range $\Delta(m)^0$ is $\max(m)^0 - \min(m)^0$. Between consecutive dimensions, however, consider the values at the end of the previous dimension as being values of round 0.

Lemma 4.8.6. In Line 9, v is well-defined and $v \in S$.

Proof. As $|V| \geq n - t$, we know that $S = \text{Safe}_t(V) \neq \emptyset$, by Lemma 4.6.7. The safe area is convex by definition.

Get two points v' and v'' in S with $v'[m] = \text{lo}(m)$ and $v''[m] = \text{hi}(m)$. Their barycenter is in S, because S is convex, and also satisfies the requirement of the algorithm, since $(v'[m] + v''[m])/2 = \text{Midpoint}(S(m))$. □

Definition 4.8.7. The set of non-faulty values updated at round r is $V_G^r = \{v_i^r : P_i \in \mathbb{G}\}$.

Lemma 4.8.8. The working range at the current dimension is halved between consecutive rounds. So, for any $1 \leq m \leq d$ and $r \geq 1$, we have $\Delta(m)^r \leq \Delta(m)^{r-1}/2$.

Proof. Consider any two non-faulty processes P_i and P_j. Without losing generality, say that $v[m]_i^r \geq v[m]_j^r$. However, since $S(m)_i^r \cap S(m)_j^r \neq \emptyset$, there exists some real value $\ell \in S(m)_i^r \cap S(m)_j^r$. Therefore,

$$\begin{aligned} & v[m]_i^r - v[m]_j^r \\ &= \frac{\text{lo}(m)_i^r + \text{hi}(m)_i^r}{2} - \frac{\text{lo}(m)_j^r + \text{hi}(m)_j^r}{2} \\ &\leq \frac{\ell + \max(m)^{r-1}}{2} - \frac{\min(m)^{r-1} + \ell}{2} \qquad (\star) \\ &= \frac{\max(m)^{r-1} - \min(m)^{r-1}}{2}, \end{aligned}$$

so $\Delta(m)^r$ is *at least* halved between rounds. We note that $(\star)$ happens because both $S_i^r = \text{Safe}_t(V_i^r)$ and $S_j^r = \text{Safe}_t(V_j^r)$ are inside $\text{Poly}(V_G^{r-1})$, since V_i^r and V_j^r each contains at most t values outside V_G^{r-1}. The observation follows from Lemma 4.6.3 with $X' = V_G^{r-1}$. We then consider only coordinate m, and its maximum and minimum values across non-faulty processes. □

We say that $1/2$ is the *convergence factor* of the algorithm, precisely indicating that the working range is multiplied by $1/2$ (but possibly even less) between rounds. Note that the convergence factor is *constant* in relation to the number of processes n.

If we want processes values to be within ϵ of each other, it is sufficient that they be within $\epsilon/\sqrt{d}$ of each other in every coordinate $1 \leq m \leq d$.

Lemma 4.8.9. For any dimension m, after

$$\mathcal{R} \geq \log_2 \left(\frac{\sqrt{d} \cdot \max\{\Delta(m')^0 : 1 \leq m' \leq d\}}{\epsilon} \right),$$

convergence rounds, the values of the processes are within distance $\epsilon/\sqrt{d}$ of each other in that dimension.

Proof. Since

$$\mathcal{R} \geq \log_2 \left(\frac{\sqrt{d} \cdot \max\{\Delta(m')^0 : 1 \leq m' \leq d\}}{\epsilon} \right),$$

we have that

$$\begin{aligned}
2^{\mathcal{R}} &\geq \frac{\sqrt{d} \cdot \max\{\Delta(m')^0 : 1 \leq m' \leq d\}}{\epsilon} && \Rightarrow \\
\frac{\epsilon}{\sqrt{d}} &\geq (1/2^{\mathcal{R}}) \cdot \max\{\Delta(m')^0 : 1 \leq m' \leq d\} && \Rightarrow \\
\frac{\epsilon}{\sqrt{d}} &\geq \max\{(1/2^{\mathcal{R}}) \cdot \Delta(m')^0 : 1 \leq m' \leq d\} && \Rightarrow \\
\frac{\epsilon}{\sqrt{d}} &\geq \max\{\Delta(m')^{\mathcal{R}} : 1 \leq m' \leq d\} \quad (\star) && \Rightarrow \\
\frac{\epsilon}{\sqrt{d}} &\geq \Delta(m)^{\mathcal{R}},
\end{aligned}$$

which satisfies our agreement requirement. The step $(\star)$ follows from Lemma 4.8.8. □

Note that the number of rounds sufficient for convergence depends only on ϵ and d (naturally), as well on $\Delta(m)^0$, for all $1 \leq m \leq d$. The initial working range $\Delta(m)^0$ is defined to consider non-faulty process values *only*. Next, we see how we run non-faulty processes for at least $\mathcal{R}$ rounds guarantees convergence, even though Byzantine processes try (unsuccessfully) to influence how many rounds we execute. In summary, the number of rounds sufficient for convergence depends *only* on non-faulty inputs and problem parameters.

4.8.3 Initial Estimation of R

In the initial estimation of R, shown in Alg. 7, we use our notion of safe area extensively. The procedure is similar to the algorithm in [1], adapted to use the concept of the safe area. We have to make sure that R^i, the estimation of R by process $P_i \in \mathbb{G}$, depends *only* on the non-faulty inputs, or Byzantine processes could influence the communication complexity of the protocol.

In Line 2 of Alg. 7, V is the message set received by `RBReceiveWitness`, and W is a multiset containing witness reports associated with V (as seen in Alg. 5). We obtain

$n - t$ witness reports W using `RBReceiveWitness`. For each witness report (with at most t faulty values by definition), we calculate a safe area, bound to be inside I_G, and obtain its barycenter, defining the multiset U. Then, we calculate a safe area for U, and again its barycenter, defining v. In fact, instead of barycenter, any deterministic computation of a point inside the safe area is enough for the algorithm.

The *range* for any multiset of values C, considering the coordinate m, is defined as

$$\delta_C(m) = \max\{|x[m] - y[m]| : x, y \in C\}.$$

Algorithm 7 p.CalculateRounds(I)

`RBSend`$((p, 0, I))$
$(V, W) \leftarrow (Val, \text{Cont}(Wit))$ from `RBReceiveWitness`(0)
$U \leftarrow \{\text{barycenter of Safe}_t(W') : W' \in W\}$
$v \leftarrow$ barycenter of $\text{Safe}_t(U)$
$R \leftarrow \left\lceil \log_2(\sqrt{d}/\epsilon \cdot \max\{\delta_U(m) : 1 \leq m \leq d\}) \right\rceil$
return (R, v)

In the next two lemmas, we show that the initial values are well-defined and inside the convex hull of non-faulty inputs. We denote W and U within process P_x as W_x and U_x.

Lemma 4.8.10. For any $P_i \in \mathbb{G}$, U_i is well-defined and only contains values $\in \text{Poly}(I_G)$.

Proof. Consider an arbitrary $W_i' \in W_i$. By definition, any report contains exactly $n - t$ values. Therefore,

$$|W_i'| = n - t \geq (d+2)t + 1 - t = (d+1)t + 1,$$

which gives, by Lemma 4.6.7, that $\text{Safe}_t(W_i') \neq \emptyset$. We then conclude that U_i is well-defined.

Now, note that $W_i' \subseteq V$ (by definition of witness) and that V contains at most t values outside I_G. Then, W_i' also contains at most t values outside I_G. Put in other words, W_i' contains a multiset W_i'' consisting exactly of $|W_i'| - t$ values inside I_G, so $\text{Poly}(W_i'') \subseteq \text{Poly}(I_G)$.

Using Lemma 4.6.3,

$$\text{Safe}_t(W_i') \subseteq \text{Poly}(W_i'') \subseteq \text{Poly}(I_G),$$

which proves that U_i only contains values in $\text{Poly}(I_G)$. □

Lemma 4.8.11. For any non-faulty process $P_i \in \mathbb{G}$, we have that v_i^0 is well-defined and $v_i^0 \in \text{Poly}(I_G)$.

Proof. Since $|U| \geq n - t$, we use again Lemma 4.6.7 and see that $\text{Safe}_t(U) \neq \emptyset$. Therefore, v_i^0 is well-defined. Also, since all values in U are in $\text{Poly}(I_G)$, then $v_i^0 \in \text{Poly}(I_G)$. □

In the remaining lemmas, we show that the estimation of R in any non-faulty process guarantees convergence.

Definition 4.8.12. The minimum round estimation across all non-faulty processes is denoted by $\rho = \min\{R_i : P_i \in \mathbb{G}\}$.

Lemma 4.8.13. For any two non-faulty processes $P_i, P_j \in \mathbb{G}$, we have that $|U^j \setminus U^i| \leq t$.

Proof. Non-faulty processes collect $n - t$ witness reports in W. Moreover, reports are transmitted via reliable broadcast, so at most t reports in W_j are not in W_i. The result follows as P_i and P_j calculate U identically based on W. □

Lemma 4.8.14. Taking any dimension d, if all non-faulty processes run for at least ρ convergence rounds,

$$|v[m]_i^\rho - v[m]_j^\rho| \leq \frac{\epsilon}{\sqrt{d}},$$

for arbitrary non-faulty processes P_i and P_j.

Proof. By definition, for all $P_i \in \mathbb{G}$, we have that $R_i \geq \rho$. Consider arbitrary $P_i, P_j \in \mathbb{G}$. Defining $D_{j,i} = (U_j \setminus U_i)$, we know that $|D_{j,i}| \leq t$, by Lemma 4.8.13. Hence, $|U_j \setminus D_{j,i}| \geq |U_j| - t$. Using Lemma 4.6.3:

$$\begin{aligned} v_j^0 \in \text{Safe}_t(U_j) &\subseteq \text{Poly}(U_j \setminus D_{j,i}) \\ &= \text{Poly}(U_j \setminus (U_j \setminus U_i)) \\ &\subseteq \text{Poly}(U_i). \end{aligned}$$

In conclusion, for any $P_i \in \mathbb{G}$, any $P_j \in \mathbb{G}$ is such that $v_j^0 \in \text{Poly}(U_i)$. Noting the calculation of R in Line 5 of `CalculateRounds`,

$$\begin{aligned} R_i &= \left\lceil \log_2(\sqrt{d}/\epsilon \cdot \max\{\delta_{U_i}(m') : 1 \leq m' \leq d\}) \right\rceil \\ &\geq \log_2(\sqrt{d}/\epsilon \cdot \max\{\delta_{U_i}(m') : 1 \leq m' \leq d\}) \\ &\geq \log_2(\sqrt{d}/\epsilon \cdot \max\{\Delta(m')^0 : 1 \leq m' \leq d\}), \end{aligned} \tag{4.1}$$

where the last line happens since $v_j^0 \in \text{Poly}(U_i)$ for any $P_j \in \mathbb{G}$. Therefore, if all non-faulty processes run for at least ρ rounds, we precisely satisfy the condition of Lemma 4.8.9, and the result follows. □

Lemma 4.8.15. Considering a dimension d, all non-faulty processes run for at least ρ rounds.

Proof. Any non-faulty process $P_x \in \mathbb{G}$ either: (1) runs for $R_x \geq \rho$ rounds; or (2) sees $|H| \geq t+1$. If we have (2), the interesting situation, we know that P_x must have received one halt message from $P_y \in \mathbb{G}$, say $(P_y, d.r_y, \{\texttt{halt}\})$. We know that P_y executed at least $r_y \geq \rho$ rounds. However, $P_x \in \mathbb{G}$ only accepts $(P_y, d.r_y, \{\texttt{halt}\})$ if P_x ran for more than r_y rounds (Line 15), which we showed to be at least ρ. As an arbitrary non-faulty process P_x either executes for $R_x \geq \rho$ or for $r_y \geq \rho$ rounds, we are done. □

4.8.4 Satisfaction of Requirements

In this section, we put previous lemmas together and prove the correctness of our protocol. The set of non-faulty values at round r is called $V_G^r = \{v_i^r : P_i \in \mathbb{G}\}$.

Lemma 4.8.16. For any $P_i \in \mathbb{G}$, it is always the case that v_i is well-defined and $v_i \in \text{Poly}(I_G)$.

Proof. We proceed by induction on consecutive rounds. For simplicity, number all rounds, even the ones across different dimensions, with consecutive numbers.

Base. Lemma 4.8.11 shows that v_i^0 is well-defined and is in $\text{Poly}(I_G)$. As $P_i \in \mathbb{G}$ is arbitrary, $V_G^0 \subseteq \text{Poly}(I_G)$.

Induction Hypothesis. Say that $V_G^x \subseteq \text{Poly}(I_G)$. By Lemma 4.8.6, we know that v_i^{x+1} is well-defined and $v_i^{x+1} \in \text{Safe}_t(V_i^{x+1})$.

Additionally, V_i^{x+1} contains at most t values outside V_G^x, since at most t Byzantine processes are assumed. In light of Lemma 4.6.3 and our inductive hypothesis, we have that

$$\text{Safe}_t(V_i^{x+1}) \subseteq \text{Poly}(V_G^x) \subseteq \text{Poly}(I_G).$$

As $P_i \in \mathbb{G}$ is arbitrary, and $v_i^{x+1} \in \text{Safe}_t(V_i^{x+1})$, as discussed before, we know that $V_G^{x+1} \in \text{Poly}(I_G)$. □

Theorem 4.8.17. After executing the protocol, all values in V_G are within distance ϵ of one another, all inside $\text{Poly}(I_G)$.

Proof. For each dimension, non-faulty processes run for at least ρ rounds (Lemma 4.8.15), therefore, for any $v_i, v_j \in V_G$, we have that $|v[m]_i^\rho - v[m]_j^\rho| \leq \epsilon/\sqrt{d}$ (Lemma 4.8.14). Since values are always maintained within $\text{Poly}(I_G)$, by the previous lemma, the result follows. □

4.8.5 Message Complexity

In [3], it is formally shown that a single process spends $O(n^2)$ messages to reliably broadcast a message. In our protocol, non-faulty process broadcast their values in every round, and the witness algorithm (Alg 5) has a single extra reliable broadcast. Therefore, each round of communication requires $O(n^2)$ messages for each non-faulty process.

As the communication channels are FIFO, any non-faulty process executes at most $r_{\max} = \max\{R_i : P_i \in \mathbb{G}\}$ rounds for each dimension: before accepting the last halt message from round $r_{\max}$, all others are received, making $|H| \geq t + 1$.

For any non-faulty process $P_i \in \mathbb{G}$,

$$R_i = \lceil \log_2(\sqrt{d}/\epsilon \cdot \max\{\delta_U(m) : 1 \leq m \leq d\}) \rceil,$$

with $\text{Poly}(U_i) \subseteq \text{Poly}(I_G)$ (Lemma 4.8.10), which implies that

$$r_{\max} \leq \log_2(\sqrt{d}/\epsilon \cdot \max\{\delta_{I_G}(m) : 1 \leq m \leq d\}) + 1.$$

In conclusion, non-faulty processes run for

$$O(d \log(d/\epsilon \cdot \max\{\delta_{I_G}(m) : 1 \leq m \leq d\}))$$

rounds, sending

$$O(n^2 d \log(d/\epsilon \cdot \max\{\delta_{I_G}(m) : 1 \leq m \leq d\}))$$

messages in total.

4.8.6 Safe Area Calculation

We can think of d as constant, noting that $d \leq 3$ in many practical applications. We observe that $\text{Poly}(V)$ and $\text{Safe}_t(V)$ are the intersection of $O(n^d)$ halfspaces, as their facets in $\mathbb{R}^d$ may be defined through at most d vertices, out of n possible points.

Therefore, when converging on dimension d, we can interpret the halfspaces defining $\text{Safe}_t(V)$ as linear restrictions, and solve two linear programs, one maximizing $v[m]$, and one minimizing $v[m]$. These points are in the safe area, and their barycenter, also in the safe area, could be taken as the updated value v, according with Lemma 4.8.6. A detailed example of linear program involving the notion of safe area has been presented in [35], and a similar formulation would apply here: we would instead maximize and minimize a particular coordinate over the feasible solution.

4.9 Final Remarks

In this chapter, we define and solve the multidimensional ϵ-approximate agreement problem: considering arbitrary $\epsilon > 0$ and $d \geq 1$, each process starts with an input value in $\mathbb{R}^d$, and all non-faulty processes must choose output values, also in $\mathbb{R}^d$, such that (1) all outputs lie within ϵ of one another, and (2) all outputs lie in the convex hull of the inputs of the non-faulty processes. We require that $n > (d + 2)t$ to solve the problem, where n is the number of processes, t is the maximum number of faulty processes, and d is the dimension of inputs and outputs. This bound is shown to be a necessary and sufficient condition for the solution.

The multidimensional ϵ-approximate agreement is a non-trivial generalizations of its scalar counterpart, as the resilience depends on the dimension the input and output values lie in. We show that the safe area concept seems to capture very well the interdependence of dimensions, permitting a systematic convergence of values. The interdependence of dimensions is a direct cause of the resilience decrease as the dimension increases.

Besides having a clear practical applicability, the multidimensional ϵ-agreement problem is key to the solvability of colorless tasks, as it permits us to solve the barycentric agreement problem. Chapter 6 provides the details. This chapter also serves as an illustration of the kind of Byzantine tasks we are interested in: namely, those in which the outputs of the non-faulty processes are constrained in terms of the input of the non-faulty processes *only*.

Chapter 5

Asynchronous Computability Conditions

Tools adapted from combinatorial topology have been successful in characterizing task solvability in synchronous and asynchronous *crash-failure* systems, as in [28]. We extend the approach to tasks in asynchronous Byzantine systems. The results, which were originally presented in [36], suggest that the language of combinatorial topology (a generalization of the language of graphs) is a convenient and effective way to formalize a range of distinct distributed computing models.

5.1 Contributions Overview

Our principal contribution in this chapter, presented in Sec 5.4, is to give the *first theorem* with necessary and sufficient conditions to solve arbitrary tasks in asynchronous Byzantine systems. In our approach, a Byzantine-failure task is defined in terms of a pair of combinatorial structures called *simplicial complexes* [37, 30], and a map modeling task semantics. We assume an adversary that may deem a subset of processes as faulty, and constrain the output of non-faulty processes in terms of the input of non-faulty processes, according to a formal specification for the task. Our theorem says that, in asynchronous systems, a Byzantine-failure task is solvable if and only if a crash-failure *counterpart* task[1], also expressed in terms of simplicial complexes, is solvable. Given that solvability conditions have long been known for crash failures (see [28]), our equivalence theorem, presented in Sec. 5.4,

[1] In the original publication venue of this work, this task was called "dual".

provides for the first time solvability conditions for Byzantine failures in asynchronous systems.

5.1.1 Related Work

The Byzantine failure model was initially introduced by Lamport, Shostak, and Pease in [31]. Most of the literature in this area has focused on the synchronous model (survey in [19]), not on the (more demanding) asynchronous model considered here. Malkhi *et al.* [33] propose several computational models in which processes communicate via shared objects (instead of messages), and display Byzantine failures. De Prisco *et al.* [14] consider the k-set agreement problem in a variety of asynchronous settings. Their notion of the validity condition for the k-set agreement problem, however, is weaker than ours. Neiger [39] discusses a stronger validity condition similar to the constraints used here. We use again the reliable broadcast in our analysis, adapted from Bracha [8] and from Srikanth and Toueg [44].

5.2 Operational Model

Assuming $n+1$ processes rather than n simplifies the topological notation, but slightly complicates the computing notation, while choosing n processes has the opposite tradeoff. Since this chapter heavily relies on analytic tools from combinatorial topology:

We let number of processes be $n+1$. Define $\mathbb{P} = \{P_0, \ldots, P_n\}$.

Any set of up to t processes might be *faulty* or *Byzantine* [31], displaying arbitrary, even malicious behavior, at any point in the execution. The actual behavior of Byzantine processes is defined by an *adversary*. Byzantine processes may execute the protocol correctly or incorrectly, at the discretion of the adversary. The set of all processes is denoted by $\mathbb{P}$, partitioned in non-faulty processes $\mathbb{G} \subseteq \mathbb{P}$ and faulty processes $\bar{\mathbb{G}} = \mathbb{P} \setminus \mathbb{G}$.

We model processes as state machines, as seen in Sec. 2.3. The input value (respectively, output value) of a non-faulty process P_i is written I_i (respectively, O_i), and non-faulty process P_i has an internal state called *view*, which we denote by view(P_i). In the beginning of the protocol, view(P_i) is I_i.

5.3 Topological Model

The formalization of crash-failure tasks using the tools and language of combinatorial topology was illustrated in Sec. 3.6. We now illustrate how these tools model tasks in the Byzantine failure model.

In this work, as discussed in Sec. 2.2.2 in Chap. 2, we only care about the relation between inputs and outputs of the non-faulty processes. The task's outputs should be consistent, despite the participation (sometimes even correct) of the Byzantine processes, a property sometimes called *strong validity*, as seen in [39]. We consider an adversarial model in which *any set* of up to t processes may be chosen as faulty, with those processes displaying arbitrary behavior. Regardless of which processes are faulty, any final configuration of the *non-faulty processes* must be permitted in respect to the initial configuration of the *non-faulty processes*. For that goal, our task specification $\mathcal{T} = (\mathcal{I}, \mathcal{O}, \Delta)$ constrains the behavior of non-faulty processes only.

Consider a Byzantine-failure task, say $\mathcal{T}$. A *non-faulty initial configuration* for $\mathcal{T}$ is a canonical name-labeled simplex $\sigma_I = \{(P_i, I_i) \text{ with } P_i \in \mathbb{G}\}$, capturing the attribution of inputs to non-faulty processes. Additionally, a *non-faulty final configuration* for $\mathcal{T}$ is a canonical name-labeled simplex $\sigma_O = \{(P_i, O_i) \text{ with } P_i \in \mathbb{G}\}$ capturing the attribution of outputs to non-faulty processes.

In our model, we assume that processes were initially correct, but the adversary gains control of the chosen faulty processes right before the protocol starts. *Any set* of no more than t processes may be deemed as faulty, including the empty set. This perspective implies the following property:

Property 5.3.1. If σ is a non-faulty initial configuration:

1. There exists a simplex $\sigma^n \supseteq \sigma$ with $\dim(\sigma^n) = n$ where σ^n is a non-faulty initial configuration;

2. For all $\sigma' \subseteq \sigma$ with $\dim(\sigma') \geq n-t$, we have that σ' is a non-faulty initial configuration.

The formal task definition for Byzantine tasks practically mirrors the definition for crash-failure tasks, although it only constrains task semantics for non-faulty processes:

Definition 5.3.2. A Byzantine-failure *task specification* is formally a triple $\mathcal{T} = (\mathcal{I}, \mathcal{O}, \Delta)$ such that:

- $\mathcal{I}$ is the *input complex*. A simplex $\sigma \in \mathcal{I}$ if there is some $\sigma_I \supseteq \sigma$ that is a non-faulty initial configuration, with $\dim(\sigma_I) = n$.

- $\mathcal{O}$ is the *output complex*. A simplex $\sigma \in \mathcal{O}$ if there is some $\sigma_O \supseteq \sigma$ that is a non-faulty final configuration, with $\dim(\sigma_O) = n$.
- $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$ is a name-preserving carrier map. The simplex $\tau \in \Delta(\sigma)$ if the non-faulty final configuration τ is valid given the non-faulty initial configuration σ, with σ matching τ. Also $\Delta(\sigma') = \emptyset$ for any non-canonical simplex σ' of $\mathcal{I}$.

The map Δ could in principle be other than a carrier map. However, for the sake of studying task computability, it is enough to assume Δ as being a carrier map. Consider the following scenario, where $\Delta(\sigma_1 \cap \sigma_2) \not\subseteq \Delta(\sigma_1) \cap \Delta(\sigma_2)$, for $\sigma_1, \sigma_2 \in \mathcal{I}$. Consider also an asynchronous protocol, with names$(\sigma_1 \cap \sigma_2)$ representing all non-faulty processes, and names$(\sigma_1 \setminus \sigma_2)$ as well as names$(\sigma_2 \setminus \sigma_1)$ representing faulty processes. In such case, the adversary can suitably delay the faulty processes so that non-faulty processes cannot discern if the non-faulty initial configuration was σ_1, σ_2, or $(\sigma_1 \cap \sigma_2)$. Since we assumed an asynchronous protocol, a decision must be made, and it should be inside $\Delta(\sigma_1 \cap \sigma_2)$ in order to cover all possibilities.

With that in mind, take a Byzantine task $\mathcal{T}_1 = (\mathcal{I}, \mathcal{O}, \Delta)$, with Δ being an arbitrary map, and another task $\mathcal{T}_2 = (\mathcal{I}, \mathcal{O}, \Delta')$, identical to $\mathcal{T}_1$ except that $\Delta' \subseteq \Delta$ is a carrier map: $\Delta'(\sigma_1 \cap \sigma_2) \subseteq \Delta'(\sigma_1) \cap \Delta'(\sigma_2)$ for any $\sigma_1, \sigma_2 \in \mathcal{I}$. Assume that Δ' is maximal, that is, there is not another carrier map Δ'' such that $\Delta' \subseteq \Delta'' \subseteq \Delta$. As we have seen, the *unrestricted* task $\mathcal{T}_1$ is solvable only if its *constrained* task $\mathcal{T}_2$ is solvable. In addition, if the constrained task is solvable, then clearly the unconstrained task is solvable. Therefore, for the sake of studying task solvability, the map in the Byzantine task definition is simply a carrier map.

5.4 The Equivalence Theorem

In this section, we will present our main theorem. We show that, in asynchronous t-resilient systems, a task $\mathcal{T}_b = (\mathcal{I}, \mathcal{O}, \Delta)$ is solvable in the Byzantine failure model if and only if its *crash-failure counterpart* task $\mathcal{T}_c = (\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ is solvable in the crash failure model, with $\tilde{\mathcal{I}}$, $\tilde{\mathcal{O}}$, and $\tilde{\Delta}$ suitably and respectively defined in terms of $\mathcal{I}$, $\mathcal{O}$, and Δ. We call the task $\mathcal{T}_b$ the *Byzantine counterpart* of $\mathcal{T}_c$. This general strategy is illustrated on Fig. 5.1.

The use of reliable broadcast avoids the equivocation problem – the situation where Byzantine processes deliberately send conflicting information to different processes. However, Byzantine processes can still introduce a false input and execute the protocol correctly, yet selectively delaying or omitting certain messages. Each non-faulty process must choose a correct output even though a Byzantine process is indistinguishable from a non-faulty

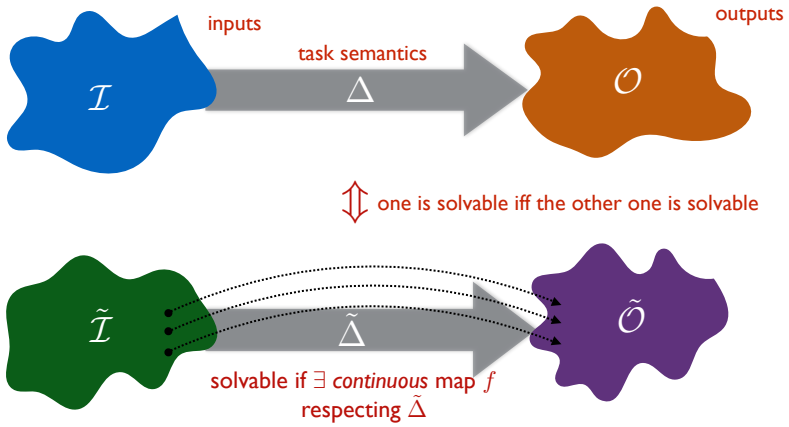

Figure 5.1: The Byzantine task $(\mathcal{I}, \mathcal{O}, \Delta)$ is solvable if and only if the crash-failure counterpart task $(\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ is solvable. The solvability of the latter is characterized by the existence of a continuous map from $\tilde{\mathcal{I}}$ to $\tilde{\mathcal{O}}$ carried by $\tilde{\Delta}$.

process having an authentic input. Moreover, if the input complex is not a simple pseudosphere, a Byzantine process can introduce a valid input that is incompatible with the other values input by non-faulty processes. Each non-faulty process must decide correctly, without necessarily detecting which, if any, of the incompatible inputs was indeed authentic. Our crash-failure counterpart task essentially addresses the two issues above, which we identify as the *main vectors* for the introduction of ambiguity by Byzantine processes.

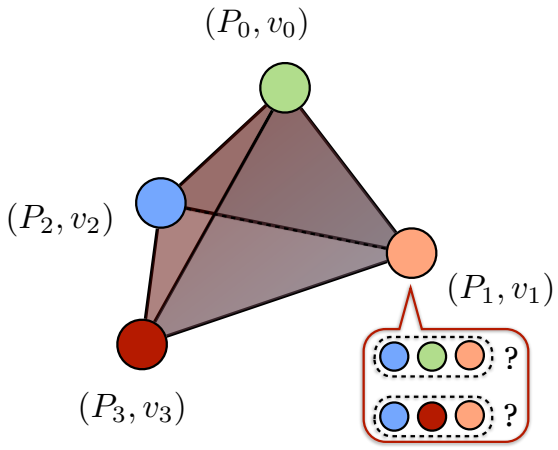

Figure 5.2: Suppose P_0, P_1, P_2 are non-faulty processes. If, say, only three value are allowed in the input, and the Byzantine process P_3 runs "correctly" with an extra input value v_3, a non-faulty process cannot be sure about which inputs come from non-faulty processes only.

5.4.1 Defining the Dual Task

We now formally define the crash-failure counterpart task $\mathcal{T}_c = (\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ in terms of the Byzantine task $\mathcal{T}_b = (\mathcal{I}, \mathcal{O}, \Delta)$.

Definition 5.4.1. An initial configuration of $\mathcal{T}_c$ is a name-labeled simplex $\sigma_I \in \tilde{\mathcal{I}}$ such that it contains a canonical name-labeled simplex $\sigma_G \in \mathcal{I}$, with views$(\sigma_I) \subseteq$ views$(\mathcal{I})$.

Definition 5.4.2. A final configuration of $\mathcal{T}_c$ is a name-labeled simplex $\tau_O \in \tilde{\mathcal{O}}$ such that it contains a canonical name-labeled simplex $\tau_G \in \mathcal{O}$, with views$(\tau_O) \subseteq$ views$(\mathcal{O})$.

Definition 5.4.3. Given a Byzantine task $\mathcal{T}_b = (\mathcal{I}, \mathcal{O}, \Delta)$, its *counterpart* crash-failure task $\mathcal{T}_c = (\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ is such that:

- $\tilde{\mathcal{I}}$ is the *input complex*. A simplex $\sigma \in \tilde{\mathcal{I}}$ if there is some $\sigma_I \supseteq \sigma$ that is a possible initial configuration of $\mathcal{T}_c$, with $\dim(\sigma_I) = n$.
- $\tilde{\mathcal{O}}$ is the *output complex*. A simplex $\sigma \in \tilde{\mathcal{O}}$ if there is some $\sigma_O \supseteq \sigma$ that is a possible final configuration of $\mathcal{T}_c$, with $\dim(\sigma_O) = n$.
- $\tilde{\Delta} : \tilde{\mathcal{I}} \to 2^{\tilde{\mathcal{O}}}$ is a name-preserving carrier map specified in Definition 5.4.4.

Particularly, if $\mathcal{I}$ (respectively $\mathcal{O}$) is a simple pseudosphere with dimension n, then $\tilde{\mathcal{I}} = \mathcal{I}$ (respectively $\mathcal{O} = \tilde{\mathcal{O}}$).

Definition 5.4.4. The map $\tilde{\Delta}$ is a name-preserving carrier map where:

1. The map $\tilde{\Delta}$ satisfies the original Byzantine specification for any possible choice of non-faulty processes:

$$\tilde{\Delta}(\sigma) = \{\tau \in \tilde{\mathcal{O}} : \forall \text{ canonical } \sigma' \subseteq \sigma \text{ with } \sigma' \in \mathcal{I}, \\ \exists \text{ matching } \tau' \subseteq \tau \text{ with } \tau' \in \mathcal{O} \text{ and } \tau' \in \Delta(\sigma')\}; \quad (5.1)$$

2. The map $\tilde{\Delta}$ is defined as a carrier map:

$$\tilde{\Delta}(\sigma_1 \cap \sigma_2) \subseteq \tilde{\Delta}(\sigma_1) \cap \tilde{\Delta}(\sigma_2) \quad (5.2)$$

for any $\sigma_1, \sigma_2 \in \tilde{\mathcal{I}}$. Also, $\tilde{\Delta}(\sigma') = \emptyset$ for any non-canonical simplex σ' of $\tilde{\mathcal{I}}$, satisfying the final constraint on $\tilde{\Delta}$ imposed by crash-failure tasks.

5.4.2 Solvability Correspondence

Given an algorithm for $\mathcal{T}_c$, we construct an algorithm for $\mathcal{T}_b$, showing that, in asynchronous, t-resilient systems, if we solve the crash-failure $\mathcal{T}_c = (\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ then we also solve its Byzantine counterpart $\mathcal{T}_b = (\mathcal{I}, \mathcal{O}, \Delta)$. In Algorithm 8, we describe the protocol solving $\mathcal{T}_b$, as run by non-faulty processes. The parameter I_i is P_i's input, and $\mathcal{P}$ is a crash-failure protocol for $\mathcal{T}_c$, which is identical across non-faulty processes. The protocol runs in "asynchronous rounds" via the reliable broadcast protocol, similarly to the multidimensional ϵ-approximate agreement in the previous chapter.

Each non-faulty process P_i will maintain a table T_i. The table has one entry for each combination of process and round: for process $p \in \mathbb{P}$ and round $r \geq 1$, the contents of the corresponding entry at T_i is denoted as $T_i(p, r)$. An unfilled entry has contents $\perp$.

In the first round, non-faulty processes exchange values in views($\mathcal{I}$). For any non-faulty processes $P_i \in \mathbb{G}$ and $P_j \in \mathbb{P}$, the entry for $T_i(P_j, 1)$ contains (P_j, v) only if $(P_j, v) \in V(\mathcal{I})$. Note that $T_i(P_i, 1) = (P_i, I_i)$, representing the process' own input (Lines 2-3). For any process set $\mathbb{S}$, let $T_i(\mathbb{S}, r)$ be the set containing $T_i(p, r)$ if and only if $p \in \mathbb{S}$.

Definition 5.4.5. A *starting process set* $\mathbb{S} \subseteq \mathbb{P}$ for $P_i \in \mathbb{G}$ is a set where the simplex $\sigma = \{e : e \in T_i(\mathbb{S}, 1)\}$ is an initial configuration of $\mathcal{T}_c$.

In subsequent rounds, non-faulty processes exchange sets of size at least $(n + 1) - t$. Such sets satisfy some requirements at the (non-faulty) sender, before being transmitted, and at the (non-faulty) receiver, before being accepted. More specifically, consider $P_i \in \mathbb{G}$. The entry for $T_i(P_i, r)$ is set to (P_i, V) as soon as the predicate $\text{Val}_i(P_i, r, V)$ becomes true. Then, the *corresponding message* for $T_i(P_i, r)$, namely (P_i, r, V), is sent (Lines 6-7). Moreover, if some $P_j \in \mathbb{P}$ sends (P_j, r, V), as soon as the message reaches P_i and the predicate $\text{Val}_i(P_j, r, V)$ becomes true, the message is accepted by P_i. Then, the *corresponding entry* for (P_j, r, V), namely $T_i(P_j, r)$, is set to (P_j, V) (Lines 9-10).

Definition 5.4.6. The predicate $\text{Val}_i(p, r, V)$ evaluates to true *only if*:

1. If $r = 1$, then $(p, V) \in V(\mathcal{I})$;
2. If $r > 1$, then
   (a) $|V| \geq (n + 1) - t$;
   (b) $T_i(p, r - 1) \in V$;
   (c) $V \subseteq T_i(\mathbb{S}, r - 1)$, for some starting process set $\mathbb{S}$ for P_i.

The loop at Line 4 runs until a particular set, an R_i-consistent process set $\mathbb{C}_i$, is found. The definition follows below.

Definition 5.4.7. An *R-consistent process set* $\mathbb{S} \subseteq \mathbb{P}$ for $P_i \in \mathbb{G}$ is one where

1. $\mathbb{S}$ is a starting process set;
2. $T_i(p, r) \neq \perp$ for all $p \in \mathbb{S}$ and $1 \leq r \leq R$;
3. $P_i \in \mathbb{S}$.

By definition, for any R-consistent process set $\mathbb{S}$, we have that $\sigma = \{e : e \in T_i(\mathbb{S}, 1)\}$ is an initial configuration of $\mathcal{T}_c$, with $|\mathbb{S}| \geq (n+1) - t$ and all messages accepted and validated up to the asynchronous round R. A *decidable* R-consistent process set $\mathbb{S}$ is one where $\mathcal{P}$, if simulated only with entries of $T_i(\mathbb{S}, 1 \ldots R) = \{T_i(\mathbb{S}, r) : 1 \leq r \leq R\}$ returns an output.

Algorithm 8 P_i.ConstrainedExecution$(I_i, \mathcal{P})$

```
1: By default T_i(p, r) ← ⊥ for all p ∈ ℙ and r ≥ 1
2: T_i(P_i, 1) ← (P_i, I_i)
3: RBSend((P_i, 1, I_i))
4: while ∄ decidable R_i-consistent process set ℂ_i do
5:     upon First V with Val_i(P_i, r, V) and r > 1 do
6:         T_i(P_i, r) ← (P_i, V)
7:         RBSend((P_i, r, V))
8:     end upon
9:     upon RBRecv((P_j, r, V)) with Val_i(P_j, r, V) do
10:        T_i(P_j, r) ← (P_j, V)
11:    end upon
12: end while
13: simulate 𝒫 using only entries of T_i(ℂ_i, 1 ... R)
14: return own decision value from the above execution
```

We also assume that non-faulty processes keep processing messages as in Lines 5 to 10 even after exiting the loop of Line 4. This can be seen as a kind of background service interleaved with the steps of the protocol, similar to [2][2]

Lemma 5.4.8. If some $P_i \in \mathbb{G}$ fills $T_i(p, r)$ with (p, V), then any other $P_j \in \mathbb{G}$ eventually fills $T_j(p, r)$ with (p, V).

Proof. By induction on r.

[2] If $\mathcal{P}$ is not compatible with messages from decided processes, it can simply ignore their messages as soon as it becomes apparent that they have reached Line 13 – possibly through a "decided" flag appended to messages.

Base: $r = 1$. If P_i fills $T_i(p, 1)$ with (p, V), then $\text{Val}_i(p, 1, V)$ is true, implying that $(p, v) \in V(\mathcal{I})$, by (1) in Definition 5.4.6. By the liveness properties of the reliable broadcast, the corresponding message $(p, 1, V)$ eventually reaches any other $P_j \in \mathbb{G}$, and $\text{Val}_j(p, 1, V)$ will be true for identical reason, filling $T_j(p, 1)$ with (p, V).

IH: Assume that for all $r' < r$, if some $P_i \in \mathbb{G}$ fills $T_i(p, r')$ with (p, V), then any other $P_j \in \mathbb{G}$ eventually fills $T_j(p, r')$ with (p, V). If P_i fills $T_i(p, r)$ with (p, V), then $\text{Val}_i(p, r, V)$ is true. Hence, P_i filled all entries $T_i(\mathbb{S}, r - 1)$, considering $\mathbb{S}$ as the set in (2)-(c) on Definition 5.4.6. By the induction hypothesis, all those entries are eventually filled in any other $P_j \in \mathbb{G}$, which eventually makes $\text{Val}_j(p, r, V)$ to be true. By the liveness properties of the reliable broadcast, the corresponding message (p, r, V) eventually reaches P_j, filling $T_j(p, r)$ with (p, V).

□

The entries in the tables on non-faulty processes represent an asynchronous, t-resilient, crash-failure execution schedule for $\mathcal{P}$. Crash failure processes execute the full-information protocol described in Sec. 2.3. For any asynchronous round $r > 0$, we interpret $T_i(P_j, r) = (P_j, V)$, as the scenario where $P_j \in \mathbb{P}$ broadcasts (P_j, r, V) and $P_i \in \mathbb{G}$ receives (P_j, r, V). By the previous lemma, if a message is received by some non-faulty process, it is eventually received by any other non-faulty process. Moreover, by the validation predicate, $T_i(P_j, r) \neq \perp$ implies in $T_i(P_j, r - 1) \neq \perp$ for all $r > 1$. An empty entry in T_i represents the inherent inability of a crash-failure processes P_i to discern between a failed process and a sender whose message is delayed.

Lemma 5.4.9. Any non-faulty process $P_i \in \mathbb{G}$ eventually reaches Line 13, and its simulation returns an output.

Proof. We show that $\mathbb{G}$ is bound to be recognized as an r-consistent process set at P_i for any $r \geq 1$ if no other process is. We proceed by induction on r.

Base: $r = 1$. Non-faulty processes execute the protocol correctly, so, by the previous lemma, $T_i(\mathbb{G}, 1)$ is eventually filled.

IH: Now assume that $T_i(\mathbb{G}, r - 1)$ is totally filled. By the previous lemma, all non-faulty process $P_j \in \mathbb{G}$ at least will have $\text{Val}_j(P_j, r, T_j(\mathbb{G}, r - 1))$ as true, sending a message (P_j, r, V) for some V, although not necessarily with $V = T_j(\mathbb{G}, r - 1)$. All those messages are eventually delivered and accepted by P_i, again by the previous lemma,

and $T_i(\mathbb{G}, r)$ is eventually filled. We conclude that $\mathbb{G}$ is bound to be recognized as an r-consistent process set at P_i for any $r \geq 1$ if no other process is.

If P_i considers solely the entries of an r-consistent process set to simulate an execution of $\mathcal{P}$, $r \geq 1$, we actually denote a valid t-resilient, asynchronous, crash-failure schedule for $\mathcal{P}$. This simulates an initial configuration of $\mathcal{T}_c$, under the perspective of P_i, up to the asynchronous round r. Hence, there exists a concrete $R_i > 0$ such that P_i reaches Line 13 with some *decidable* R_i-consistent process set $\mathbb{C}_i$, although not necessarily $\mathbb{C}_i = \mathbb{G}$, with $\mathcal{P}$ returning an output. Figure 5.3 illustrates this idea. □

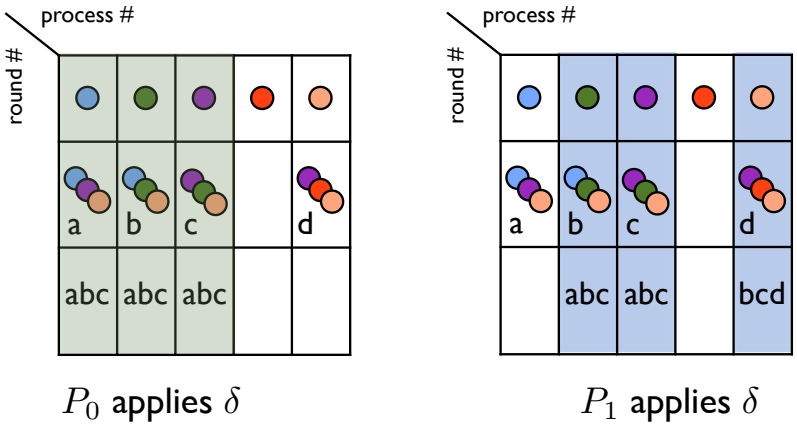

Figure 5.3: (Left) Partial view of the execution by process P_0. (Right) The same for process P_1. As we have a protocol $\mathcal{P}$ for $\mathcal{T}_c$, there exists a round in which processes can decide.

Lemma 5.4.10. If an asynchronous, t-resilient crash-failure protocol $\mathcal{P}$ solves the task $\mathcal{T}_c = (\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$, then Algorithm 8 solves its Byzantine counterpart $\mathcal{T}_b = (\mathcal{I}, \mathcal{O}, \Delta)$.

Proof. Take some $P_i \in \mathbb{G}$, calling $R_i = r$. We say that $p_1 \in \mathbb{P}$ is *seen* by P_i on its execution of $\mathcal{P}$ at Line 13 if there is a sequence $p_1 \dots p_r$ such that (p_ℓ, V_ℓ) in $T_i(p_{\ell+1}, \ell + 1)$ for all $1 \leq \ell < r$, and $p_r \in \mathbb{C}_i$. Let $\sigma_i = \{T_i(p, 1) : p \text{ is seen by} P_i\}$ be the input observed by P_i on its execution of $\mathcal{P}$ at Line 13.

A Byzantine process P_b such that $(P_b, v) \in \sigma_i$ for some $P_i \in \mathbb{G}$ is said to have *apparent input* v. As all messages are validated through the validation predicate, we have that $v \in V(\mathcal{I})$, in light of (2)-(1) in Definition 5.4.6.

The non-faulty inputs define $\sigma_G \in \mathcal{I}$, and the non-faulty plus apparent inputs define

$$\sigma_A = \bigcup_{P_p \in \mathbb{G}} \sigma_p.$$

The input observed by P_i, σ_i, is an initial configuration of $\tilde{\mathcal{I}}$ including P_i, by Definitions 5.4.7 and 5.4.5. Therefore, the simulation of $\mathcal{P}$ at Line 13 produces an output in $\tilde{\Delta}(\sigma_i) \neq \emptyset$ for all $P_i \in \mathbb{G}$. In addition, $\sigma_G \subseteq \sigma_A$, by definition of σ_A, and apparent inputs are in $V(\mathcal{I})$, as discussed before. Then, σ_A is an initial configuration of $\mathcal{T}_c$ as well, and $\tilde{\Delta}(\sigma_A) \neq \emptyset$.

By the previous lemma, recalling our assumption that $\mathcal{P}$ is a crash-failure protocol for $\mathcal{T}_c$, any non-faulty process $P_i \in \mathbb{G}$ will reach Line 13, producing an output O_i such that

$$(P_i, O_i) \in \tau_i \subseteq \tau, \tag{5.3}$$

with $\tau_i \in \tilde{\Delta}(\sigma_i)$ and $\tau \in \tilde{\Delta}(\sigma_A)$.

Because the simulations in Line 13 run with partial views of a global asynchronous, t-resilient, crash-failure schedule for $\mathcal{P}$, the decisions are consistent among non-faulty processes – or we contradict the fact that $\mathcal{P}$ solves $\mathcal{T}_c$ in asynchronous, t-resilient, crash-failure systems. More technically, we must have $(P_j, O_j) \in \tau_j \subseteq \tau$, with $\tau_j \in \tilde{\Delta}(\sigma_j)$ and with the same τ as in (5.3), above. In other words,

$$\tau_G = \{(P_i, O_i) : P_i \in \mathbb{G}\} \subseteq \tau \in \tilde{\Delta}(\sigma_A),$$

with $\tau_G \in \Delta(\sigma_G)$, by definition of $\tilde{\Delta}$. As the choice of non-faulty process is totally arbitrary, the protocol solves $\mathcal{T}_b$. □

Theorem 5.4.11. In asynchronous, t-resilient systems, the task $\mathcal{T}_b = (\mathcal{I}, \mathcal{O}, \Delta)$ is solvable in the Byzantine failure model if and only if the task $\mathcal{T}_c = (\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ is solvable in the crash failure model.

Proof. $\mathcal{T}_b$ **implies** $\mathcal{T}_c$**.** Consider an execution of $\mathcal{T}_b$ where Byzantine processes may only fail by crashing, having inputs in views($\mathcal{I}$). Non-faulty and apparent inputs, those pertaining to Byzantine processes, define $\sigma \in \tilde{\mathcal{I}}$.

At least one canonical simplex having values in $\mathcal{I}$ exists by definition of $\tilde{\mathcal{I}}$. Given a protocol for $\mathcal{T}_b$, for any canonical simplex $\sigma' \subseteq \sigma$ with $\sigma' \in \mathcal{I}$, effectively representing non-faulty processes, their outputs τ' are such that $\tau' \in \Delta(\sigma')$, in order to satisfy any adversarial definition of non-faulty processes. Of course, τ' matches σ', and the protocol is actually computing $\tilde{\Delta}$ across non-faulty processes. The implication follows because any Byzantine protocol is also a crash protocol.

$\mathcal{T}_c$ **implies** $\mathcal{T}_b$**.** Follows from Lemma 5.4.10. □

5.5 Final Remarks

In this chapter, we presented novel necessary and sufficient conditions for task solvability in asynchronous Byzantine systems. While analogous results have long existed for crash-failure systems [28], we provide solvability conditions for arbitrary Byzantine tasks for the first time. We presume a model in which any set of up to t processes could be deemed faulty, and be subject to control by an adversary just before the protocol starts. Independently of which processes are deemed faulty, any final configuration of the non-faulty processes must be permitted in respect to the initial configuration of the non-faulty processes, according to a formal task specification.

This chapter essentially demonstrates how the language and techniques of combinatorial topology can produce novel results in asynchronous systems, facilitating existential arguments while avoiding complicated, model-specific argumentation. In the next chapter, we present applications of our Equivalence Theorem to the context of colorless tasks – tasks totally defined in terms of the input and output sets of values. For that specific subclass of tasks, solvability can be expressed in terms of the relation between t, n, and the task's simplicial complexes.

Chapter 6

Asynchronous Colorless Tasks

In this section, a specific application of the equivalence theorem of the previous chapter gives novel, model-specific computability results for asynchronous colorless tasks – tasks completely specified in terms of the input and output value *sets*, and not concerning the attribution of those values to processes.

Task specifications can be simplified and specialized for asynchronous colorless tasks, and we then express computability results in a more concise and elegant language. This chapter describes this work, also originally presented in [36].

6.1 Contributions Overview and Related Work

Colorless tasks [24, 6] form an important class of problems where tasks are totally defined in terms of the input and output *sets* of values, not the particular attribution of values to processes. These tasks encompass well-studied problems such as consensus [18], k-set agreement [11], and approximate agreement [15, 34]. Set agreement problems have been studied under different validity conditions in [39, 14].

In this chapter, we provide additional characterizations for asynchronous colorless tasks capturing the relation between the number of processes, the number of failures, and the topological structure of the task's simplicial complexes. We also show that, for some colorless tasks, the resilience depends directly on the input complex dimension. Our results demonstrate an application of the equivalence theorem of Chap. 5, and are expressed in a a concise, elegant, and model-specific way for this important subclass of problems.

6.2 Operational Model

We assume the same operational constraints as in the previous chapter. Since this chapter also relies on analytic tools from combinatorial topology:

We let number of processes be $n+1$. Define $\mathbb{P} = \{P_0, \ldots, P_n\}$.

The model for Byzantine failures is identical: *any* set of up to t processes might be deemed faulty by the assumed Byzantine adversary.

For colorless tasks, input and output simplices represent *sets* of input and output values permitted in the initial/final configurations. Such sets are closed under inclusion, that is, if S represents a valid initial (resp. final) input (resp. output) set, so is any $S' \subseteq S$. The admissible *sets* of output values depend solely on the *set* of input values taken by processes. Importantly, for any set of values, any particular attribution of those values to processes is valid.

A classical example of colorless task is k-set agreement [11]. Say that processes start with input values from a finite set V. In crash-failure systems, informally speaking, a protocol solves k-set agreement if outputs satisfy:

Agreement: no more than k different outputs exist; and

Validity: any output was proposed in the input.

In Byzantine systems, a natural variation under our model is called *strong* k-set agreement: processes decide on values proposed by non-faulty processes *only*.

6.3 Topological Model

In this section, we give simpler task specification for colorless tasks. Initially, we consider crash failures, following the model in [23]. The Byzantine specification will essentially mirror the approach of Sec. 5.3.

A *colorless task* is a triple $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, where $\mathcal{I}^*$ is the *colorless input complex*, $\mathcal{O}^*$ is the *colorless output complex*, and $\Delta^* : \mathcal{I}^* \to 2^{\mathcal{O}^*}$ is the *colorless carrier map*. Each vertex in $\mathcal{I}^*$ (resp. $\mathcal{O}^*$) is a possible input (resp. output) value, and each simplex is a possible *initial input (resp. output) set*. Given an initial input set, Δ^* specifies which final output sets are legal. Colorless tasks can of course be expressed in the general model $(\mathcal{I}, \mathcal{O}, \Delta)$, as

seen in [23]:

$$\sigma^* \in \mathcal{I}^* \text{ (resp. } \mathcal{O}^*) \Leftrightarrow \Psi(\mathbb{P}, \sigma^*) \subseteq \mathcal{I} \text{ (resp. } \mathcal{O}) \tag{6.1}$$

$$\tau \in \Delta(\sigma) \Leftrightarrow \text{views}(\tau) \in \Delta^*(\text{views}(\sigma)) \tag{6.2}$$

For instance, for the $(t+1)$-set agreement, the map Δ^* is the skeleton operator skel^t as output values must be chosen among at most $t+1$ different values. Figure 6.1 exemplifies the strong 2-set agreement task.

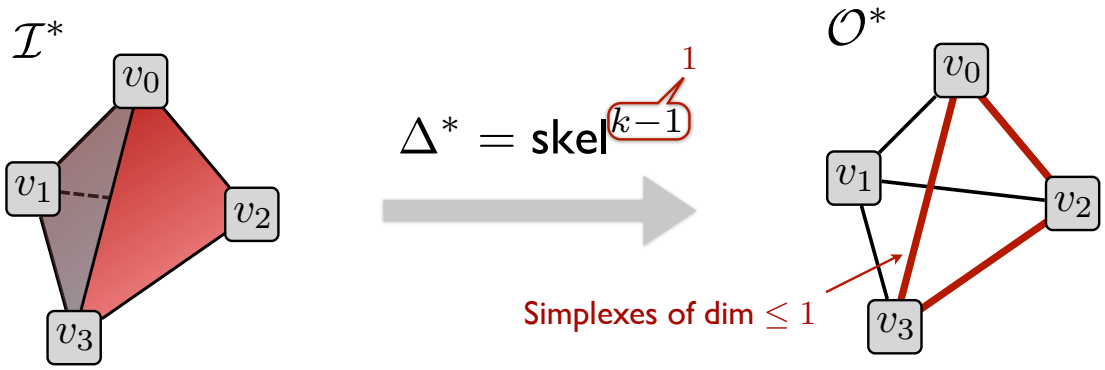

Figure 6.1: A strong 2-set agreement task formalization in the colorless model.

With Byzantine tasks, $\mathcal{I}^*$ (resp. $\mathcal{O}^*$) refers to *non-faulty* input (resp. output) sets only. Since the adversary may choose any set with up to t processes as Byzantine (including the empty set), the following relation remains valid: $\sigma^* \in \mathcal{I}^*$ (resp. $\mathcal{O}^*$) if and only if $\Psi(\mathbb{P}, \sigma^*) \subseteq \mathcal{I}$ (resp. $\mathcal{O}$). The map Δ^* is defined as in Equation 6.2, which now conditions non-faulty output sets to non-faulty input sets only.

6.4 Protocols and Complexes

Given a model for communication and failures, we can define a *protocol complex* $\mathcal{P}(\mathcal{I}^*)$ for any task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$ modeling the possible final states of the protocol execution for any input. The possible final states for an input configuration represented by σ denotes the simplicial complex $\mathcal{P}(\sigma)$. A vertex in $v \in \mathcal{P}(\mathcal{I}^*)$ is a tuple with a non-faulty process identifier and its final state. A simplex $\sigma = \{(Q_1, s_1), \ldots, (Q_x, s_x)\}$ in $\mathcal{P}(\mathcal{I}^*)$ indicates that, in some execution, non-faulty processes $Q_1, \ldots, Q_x$ finish with states $s_1, \ldots, s_x$, respectively. The same concept is valid for $\mathcal{P}(\sigma)$ with $\sigma \in \mathcal{I}^*$, but the execution is restricted to the ones where the input was σ. The formal definition of *protocol* is identical to the definition presented in Sec. 3.11.

6.5 Barycentric Agreement via Approximate Agreement

The barycentric subdivision of simplex σ is constructed, informally speaking, by subdividing σ along the barycenters of its faces. The concept is formalized in Def. 3.4.3, and illustrated in Fig. 3.5 in Chap. 3. In this section, we see how to transform a protocol for the multidimensional ϵ-approximate agreement problem, as described in Chap. 4, into a task called *barycentric agreement*.

In the barycentric agreement task, non-faulty processes start on vertices of a simplex σ and halt on vertices of a simplex in $\text{Bary}\,\sigma$. Formally:

Definition 6.5.1. The barycentric agreement is a triple $(\mathcal{I}^*, \text{Bary}(\mathcal{I}^*), \text{Bary})$. In other words, for each $\sigma \in \mathcal{I}$, we have that $\Delta(\sigma) = \text{Bary}\,\sigma$.

We now outline a non-constructive proof for a protocol solving barycentric agreement. In the next section, we provide a constructive proof solving the problem.

The point-set occupied by $\mathcal{I}$ is compact, and the open stars of the vertices of $\text{Bary}\,\mathcal{I}$ form an open cover of $\mathcal{I}$ (see Fig. 6.2).

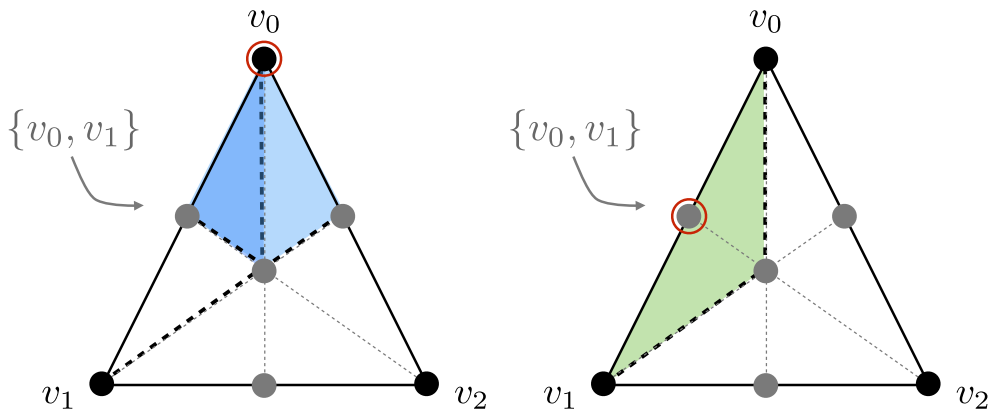

Figure 6.2: The open cover of vertexes $\{v_0\}$ and $\{v_0, v_1\}$ in $\text{Bary}\{v_0, v_1, v_2\}$

Any of those covers has a *Lebesgue* number $\lambda > 0$ [37], such that every set of diameter less than λ is contained in at least one member of the cover.

Suppose the non-faulty processes start at the vertices of an input simplex σ. Using $(\lambda/2)$-approximate agreement, each non-faulty process p_i chooses a point inside σ, the convex hull of the inputs, such that the distance between any pair of points is less than $\lambda/2$. Equivalently, each open ball of radius $\lambda/2$ around v_i contains all values chosen by the approximate agreement protocol. Because the diameter of this set is less than the Lebesgue

number λ, there is at least one vertex u_i in $\text{Bary}\,\mathcal{I}$ such that $B(v_i, \lambda/2)$ lies in the open star around u_i. Let each P_i choose any such u_i.

We must still show that the vertices u_i that were chosen by the processes P_i lie on a single simplex of $\text{Bary}\,\sigma$. Note that u_i, u_j are vertices of a common simplex if and only if the open star around u_i intersects the open star around u_j. By construction, $v_j \in B(v_i, \lambda/2)$, which is in the open star around u_i, and v_j is in the open star around u_j, hence u_i, u_j are vertices of a single simplex (see Fig. 6.3).

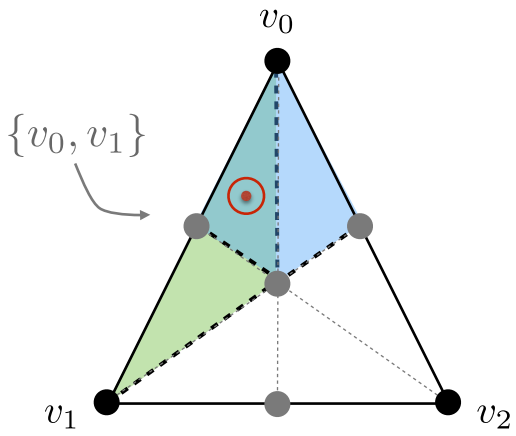

Figure 6.3: A point in the intersection of two open stars makes processes decide vertices of a single simplex.

Note that the procedure shown above requires that $n + 1 > t \cdot \max\{3, \dim(\mathcal{I}^*) + 2\}$, a requirement that follows from the multidimensional ϵ-approximate agreement.

6.6 A Constructive Proof

While the previous section shows how we can solve barycentric agreement via the multidimensional ϵ-approximate agreement, it does not denote a constructive proof since the Lebesgue number is not necessarily known *a priori*. In this section, we describe a protocol using two well-known constructions from prior work: *reliable broadcast* [44, 8], also described in Sec. 4.5.1 of Chap. 4, and *stable vectors* [2], introduced here.

6.6.1 Quorums

Let $\mathcal{M}^r$ be the set of all messages reliably broadcast by all processes during a discrete round r, and let $M_i^r \subseteq \mathcal{M}^r$ be the subset reliably received by a non-faulty process P_i. By the

global uniqueness of the reliable broadcast, if (P, r, c) and (P', r', c') are distinct messages in M_i^r then the senders are distinct: $P \neq P'$. Furthermore, by definition of M_i^r, we have that $r = r'$. If a process receives (P, r, c), we can also say that it received c from process P. Let $Good(\mathcal{M}^r)$ denote the set of distinct message contents in $\mathcal{M}^r$ that were reliably broadcast by the non-faulty processes.

Definition 6.6.1. We say that a content c has a *quorum* in M_i^r, written $c \in Quorum(M_i^r)$, if c was reliably received in M_i^r from $t+1$ or more different processes.

For any $M \subseteq \mathcal{M}^r$, we know that $Quorum(M) \subseteq Good(\mathcal{M}^r)$: if P_i obtained a quorum for c, P_i becomes aware that c was sent by a non-faulty process. Conversely, any content received from less than $t+1$ processes cannot be "trusted" – recall that non-faulty process outputs must depend only on non-faulty process inputs.

Algorithm 9 shows the procedure by which non-faulty processes obtain a set of messages containing a quorum in a communication round r. The procedure works as long as:

$$n + 1 > t \cdot \max\{3, \dim(\mathcal{I}^*) + 2\}.$$

We denote by M_i^r the set M received by process P_i at round r.

Algorithm 9 P.`RecvQuorum`(r)

```
1: M ← ∅
2: while |M| < (n + 1) − t or Quorum(M) = ∅ do
3:     upon RBRecv((Q, r, c)) do
4:         M ← M ∪ {(Q, r, c)}
5:     end upon
6: end while
7: return M
```

Informally, the procedure eventually finishes since all $(n+1) - t$ non-faulty processes eventually show up, and, since $(n+1) - t > t \cdot (\dim(\mathcal{I}^*) + 1)$, some value will appear $t+1$ or more times.

Lemma 6.6.2. If $n + 1 > t \cdot \max\{3, \dim(\mathcal{I}^*) + 2\}$, then `RecvQuorum`$(r)$ eventually returns, and, for any non-faulty process P_i,

$$|M_i^r| \geq (n+1) - t \text{ and } Quorum(M_i^r) \neq \emptyset.$$

Proof. Let $n + 1 > t \cdot \max\{3, \dim(\mathcal{I}^*) + 2\}$. Since $n + 1 > 3t$, the processes can perform reliable broadcast. Note that the $(n+1) - t$ messages sent by the non-faulty processes can be grouped by their contents:

$$(n+1) - t = \sum_{c \in Good(\mathcal{M}^r)} |\{(P, r, c) : (P, r, c) \in \mathcal{M}^r, P \text{ is non-faulty}\}|.$$

If every content c in $Good(\mathcal{M}^r)$ was reliably broadcast by at most t non-faulty processes, we would have $(n+1) - t \leq |Good(\mathcal{M}^r)| \cdot t \leq (\dim(\mathcal{I}^*) + 1) \cdot t$, which contradicts the hypothesis. Hence, at least one content in $Good(\mathcal{M}^r)$ was reliably broadcast by more than $t+1$ non-faulty processes. By the non-faulty liveness of the reliable broadcast, such content will eventually be reliably received by all non-faulty processes. □

Lemma 6.6.3. After `RecvQuorum`(r), for any two non-faulty processes P_i and P_j, we have that $|M_i^r \setminus M_j^r| \leq t$.

Proof. If $|M_i^r \setminus M_j^r| > t$, then M_j^r missed more than t messages in $\mathcal{M}^r$, the messages reliably broadcast in round r. However, this contradicts the fact that $|M_j^r| \geq (n+1) - t$ with M_j^r being obtained by reliable broadcast. □

6.6.2 Stable Vectors

Algorithm 9 ensures that any two non-faulty processes P_i and P_j collect at least $n+1-2t$ messages in common. Indeed, $|M_i^r| \geq (n+1) - t$ by Lemma 6.6.2 and $|M_i^r \setminus M_j^r| \leq t$ by Lemma 6.6.3, therefore $|M_i^r \cap M_j^r| \geq (n+1) - 2t$.

In Algorithm 10, we adapt the *stable vectors* technique presented in [2] to ensure that the two non-faulty processes P_i and P_j have $(n+1) - t$ messages in common in M_i^r and M_j^r, with a common value in $Quorum(M_i^r \cap M_j^r)$ and with sets of messages totally ordered by containment. Since we use the `RecvQuorum`() procedure, it requires that

$$n + 1 > t \cdot \max\{3, \dim(\mathcal{I}^*) + 2\}.$$

The technique works as follows.

1. P uses `RecvQuorum`() to obtain a set of messages M with $Quorum(M) \neq \emptyset$.
2. P reliably transmits its *report*, containing those messages.
3. Any further message reliably received in M causes P to reliably broadcast an updated report of M.
4. P keeps reliably receiving reports, stored in a vector R, until it identifies $(n+1) - t$ *buddies* in B, where a buddy is a process P_j whose last report R_j is M.
5. P decides, but keeps updating M and sending updated reports in the background (Algorithm 11).

Algorithm 10 P.`RecvStable`(r)

```
1:  M, B ← ∅
2:  R[x] ← ∅ for all 0 ≤ x ≤ n
3:  M ← RecvQuorum(r)
4:  RBSend((P, r{report}, M))
5:  while |B| < (n + 1) − t do
6:      upon RBRecv((P_j, r, c)) do
7:          M ← M ∪ {(P_j, r, c)}
8:          RBSend((P, r{report}, M))
9:      end upon
10:     upon RBRecv((P_j, r{report}, R_j)) do
11:         R[j] ← R_j
12:     end upon
13:     B ← {P_x : R[x] = M, 0 ≤ x ≤ n}
14: end while
15: return M                                  ▷ while activating RSEcho(M, r)
```

Algorithm 11 P.`RSEcho`(M, r)

```
1: upon RBRecv((Q, r, c)) do
2:     M ← M ∪ {(Q, r, c)}
3:     RBSend((P, r{report}, M))
4: end upon
```

The following lemmas show that the procedure terminates, and give the stable vector properties. Intuitively, any two non-faulty processes identify a common non-faulty buddy, which reliably broadcasts monotonically increasing reports, guaranteeing the properties.

Lemma 6.6.4. For any non-faulty process P_i, the sequence of transmitted reports is monotonically increasing. All other non-faulty processes receive those reports in such order.

Proof. First, note that the set M of P_i is monotonically increasing, so the sequence of transmitted reports is also monotonically increasing. As we assume FIFO channels, any other non-faulty process P_j receives those reports in the same order. □

Lemma 6.6.5. `RecvStable`(r) eventually returns.

Proof. Consider $\mathcal{S}$, the set of all messages reliably received by at least one non-faulty process. By the non-faulty liveness of the reliable broadcast, all non-faulty processes will eventually obtain $M = \mathcal{S}$, which is never expanded, or we would contradict the definition of $\mathcal{S}$. All non-faulty processes then send reports $\mathcal{R} = \mathcal{S}$, which are final, for the same reason as before.

Again, by the non-faulty liveness property of the reliable broadcast, all these processes will eventually receive $(n+1)-t$ reports $\mathcal{R}$. By the monotonicity of received reports (Lemma 6.6.4) and FIFO message delivery, those reports are not overwritten, matching the local $M = \mathcal{S}$. All non-faulty processes then return from `RecvStable`(). □

Lemma 6.6.6. After `RecvStable`(r), any two non-faulty processes P_i and P_j satisfy the following:

1. $|M_i^r \cap M_j^r| \geq (n+1)-t$;
2. $\mathit{Quorum}(M_i^r \cap M_j^r) \neq \emptyset$; and
3. $M_i^r \subseteq M_j^r$ or $M_j^r \subseteq M_i^r$.

Proof. Call B_i^r the set of buddies whose reports are stored in R, for process P_i and round r, right before it decides. Since all reports are transmitted via reliable broadcast, and every non-faulty process collects $(n+1)-t$ reports, $|B_i^r \setminus B_j^r| \leq t$ with $|B_i^r| \geq (n+1)-t$, which implies that $|B_i^r \cap B_j^r| \geq n+1-2t$. In other words, any two non-faulty processes identify $n+1-2t > t+1$ buddies in common, including a non-faulty process P_k. Therefore, $M_i^r = R_k'$ and $M_j^r = R_k''$, where R_k' and R_k'' are reports sent by P_k at possibly different occasions.

Since the set M_k^r is monotonically increasing, either $R_k' \subseteq R_k''$ or $R_k'' \subseteq R_k'$, guaranteeing property (iii). Both R_k' and R_k'' contain R_k, the first report sent by P_k, by Lemma 6.6.4. Finally, $|R_k| \geq (n+1)-t$ and $\mathit{Quorum}(R_k) \neq \emptyset$, by Lemma 6.6.2, which guarantee properties (i) and (ii). □

Algorithm 12 shows a barycentric agreement protocol, with the same resilience as the multidimensional ϵ-approximate agreement protocol, and the barycentric agreement protocol implemented using the latter: $n+1 > t \cdot \max\{3, \dim(\mathcal{I}^*)+2\}$.

Algorithm 12 P.`BaryAgree`(v)

```
1: RBSend((P, 1, {v}))
2: M ← RecvStable(1)
3: return Quorum(M)
```

Theorem 6.6.7. Algorithm 12 solves the barycentric agreement problem for Byzantine asynchronous systems.

Proof. By Lemma 6.6.6, for any two non-faulty processes P_i and P_j, we have that $M_i \subseteq M_j$ or $M_j \subseteq M_i$, and also that $\mathit{Quorum}(M_i \cap M_j) \neq \emptyset$. Therefore, $\mathit{Quorum}(M_i) \subseteq \mathit{Quorum}(M_j)$

or $Quorum(M_j) \subseteq Quorum(M_i)$, implicating that the decided values, which are faces of σ, are totally ordered by containment. □

6.7 Solvability for Colorless Tasks

In this section, we explore our colorless model and the concepts above to obtain computability conditions specific to colorless tasks. We start with an interesting consequence of having an asynchronous Byzantine protocol.

Theorem 6.7.1. If a colorless $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$ has a t-resilient protocol in asynchronous Byzantine systems, there exists a continuous map $f : |\operatorname{skel}^t(\mathcal{I}^*)| \to |\mathcal{O}^*|$ carried by Δ^*.

Proof. Assuming a protocol, we argue by reduction to the crash-failure case, and then proceed similarly to [23]. First, note that any t-resilient Byzantine protocol is also a t-resilient crash-failure protocol. From [22, 25][1], for any $\sigma \in \mathcal{I}^*$, the protocol complex $\mathcal{P}(\sigma)$ is $(t-1)$-connected in the crash-failure model, so, in light of the previous observation, it is also $(t-1)$-connected in the Byzantine-failure model, where processes fail only by crashing. Denote the appropriate protocol complex $\mathcal{P}(\mathcal{I})$, as usual. This implies that $\operatorname{skel}^x(\mathcal{P}(\sigma))$ is $(x-1)$-connected for $0 \leq x \leq t$. We will then inductively construct a sequence of continuous maps $g_x : |\operatorname{skel}^x(\mathcal{I}^*)| \to |\mathcal{P}(\operatorname{skel}^x(\mathcal{I}^*))|$, for $0 \leq x \leq t$, mapping skeletons of $\mathcal{I}^*$ to skeletons of $\mathcal{P}(\mathcal{I}^*)$ as in [23].

Base. Let g_0 map any vertex $v \in \sigma$ to any vertex $v' \in \mathcal{P}(v)$, which exists because $\operatorname{skel}^0(\mathcal{P}(v))$ is (-1)-connected by hypothesis. We just constructed

$$g_0 : |\operatorname{skel}^0(\mathcal{I}^*)| \to |\mathcal{P}(\operatorname{skel}^0(\mathcal{I}^*))|.$$

Induction Hypothesis. Assume

$$g_{x-1} : |\operatorname{skel}^{x-1}(\mathcal{I}^*)| \to |\mathcal{P}(\operatorname{skel}^{x-1}(\mathcal{I}^*))|,$$

with $x \leq t$, which sends the geometrical boundary of a x-simplex σ^x in $\operatorname{skel}^x(\mathcal{I}^*)$ to $|\mathcal{P}(\operatorname{skel}^{x-1}(\sigma^x))|$. In other words, we have $g_{x-1}(|\,\partial\,\sigma^x|) \subseteq |\mathcal{P}(\operatorname{skel}^{x-1}(\sigma^x))|$. By hypothesis, $\mathcal{P}(\sigma^x)$ is $(x-1)$-connected, so the continuous image of the $(x-1)$-sphere $|\,\partial\,\sigma^x|$ could be extended to a continuous x-ball $|\sigma^x|$, defining g_x such that $g_x(|\sigma^x|) \subseteq |\mathcal{P}(\operatorname{skel}^x(\mathcal{I}^*))|$. As all such maps agree on their intersections, we just constructed

$$g_x : |\operatorname{skel}^x(\mathcal{I}^*)| \to |\mathcal{P}(\operatorname{skel}^x(\mathcal{I}^*))| \subseteq |\mathcal{P}(\mathcal{I}^*)|.$$

[1] These papers characterize connectivity in terms of the minimum core size c, as defined by Junqueira and Marzullo [29]. For t-resilient tasks in the crash-failure model, $t = c + 1$.

As we assumed a protocol for $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, we must have $\delta^* : \mathcal{P}(\mathcal{I}^*) \to \mathcal{O}^*$, a simplicial map carried by Δ^* (given by Definition 3.11.2). Our new map f is defined as the composition $\delta_c^* \circ g_t$, where the continuous map δ_c^* is induced by the simplicial map δ^*. □

Colorless tasks have varying requirements in terms of the number of processes required for solvability. Consider strong $(t+1)$-set agreement. If $\dim(\mathcal{I}^*) \leq t$ (which includes the case when $\dim(\mathcal{I}^*) = 0$), each process can simply decide on its input, without any communication. For non-trivial cases, the protocol requires $n + 1 > t(\dim(\mathcal{I}^*) + 2)$, shown in Lemma 6.7.2. The result follows as an application of our Equivalence Theorem (Theorem 5.4.11).

Lemma 6.7.2. The strong $(t+1)$-set agreement task $\mathcal{T} = (\mathcal{I}^*, \mathcal{O}^*, \text{skel}^t)$ has a t-resilient protocol in asynchronous Byzantine systems if and only if $n+1 > t(\dim(\mathcal{I}^*)+2)$ or $\dim(\mathcal{I}^*) \leq t$.

Proof. ($\Leftarrow$) If $\dim(\mathcal{I}^*) \leq t$, a k-set agreement protocol is trivial, as non-faulty processes already start with at most $t+1$ distinct values in $\mathcal{I}^*$. Otherwise, in the situation where $(n+1) - t > t(\dim(\mathcal{I}^*) + 1)$, consider Alg. 13. Assuming for contradiction that each of the $\dim(\mathcal{I}^*) + 1$ input values is chosen by at most t different non-faulty processes, we would have that $(n+1) - t \leq t(\dim(\mathcal{I}^*) + 1)$. Therefore, at least $t+1$ non-faulty processes in fact input an identical value v, and non-faulty processes can wait for such occurrence, eventually deciding on a value inside $\mathcal{I}^*$. Also, as $(n+1) - t$ messages are received via reliable broadcast, at most t values are missed, so at most $t+1$ values are decided, solving the problem.

Algorithm 13 P_i.KSetStrictAgree(I_i)

1: **if** $\dim(\mathcal{I}^*) \leq t$ **then return** I_i
2: **end if**
3: Get $(n+1) - t$ messages with values in $\mathcal{I}^*$ via reliable broadcast, with some value appearing $t+1$ times
4: **return** O_i = the smallest value received

($\Rightarrow$) If $\mathcal{T}$ is solvable, then $\mathcal{T}' = (\sigma^*, \sigma^*, \text{skel}^t)$ is similarly solvable, taking an arbitrary d-simplex $\sigma^* = \{v_0, \ldots, v_d\}$ in $\mathcal{I}^*$ with $d = \dim(\mathcal{I}^*)$. By our equivalence theorem, and considering the relations in (6.1) and (6.2), we must be able to solve

$$\mathcal{T}'' = (\Psi(\mathbb{P}, \sigma^*), \Psi(\mathbb{P}, \sigma^*), \widetilde{\text{skel}^t}), \tag{6.3}$$

where, for any canonical name-labeled $\sigma, \tau \in \Psi(\mathbb{P}, \sigma^*)$:

$$\tau \in \widetilde{\text{skel}}^t(\sigma) \Leftrightarrow \forall \text{ canonical } \sigma' \subseteq \sigma, \exists \text{ matching } \tau' \subseteq \tau \text{ with } \text{views}(\tau') \in \text{skel}^t(\text{views}(\sigma')). \quad (6.4)$$

Assume a protocol for $\mathcal{T}''$, and for contradiction, assume that $(n+1)-t \leq t(\dim(\mathcal{I}^*)+1)$. Consider an execution where:

1. all processes behave correctly or crash;
2. all processes in $\mathbb{S} = \{P_0, \ldots, P_{n-t}\}$ terminate without receiving any message from any process in $\mathbb{T} = \{P_{n+1-t}, \ldots, P_n\}$;
3. each process $P_i \in \mathbb{S}$ starts with input $I_i = v_{i \bmod d+1}$.

In this case, define

$$\mathbb{S}_x = \{p \in \mathbb{S} : p \text{ has input } v_x\}. \quad (6.5)$$

Note that $(n+1) - t \geq d+1$, as $\mathcal{I}^*$ contains only inputs chosen by non-faulty processes and $d = \dim(\mathcal{I}^*)$ by assumption. Consequently, since $(d+1) \leq n+1-t \leq t(d+1)$, and in light of (6.5),

$$0 < |\mathbb{S}_x| \leq t \text{ for all } 0 \leq x \leq d.$$

Regarding notation, we define $\sigma_x = \{(P_i, I_i) : P_i \in \mathbb{S}_x\}$ and $\sigma_{-x} = \{(P_i, I_i) : P_i \in \mathbb{S} \setminus \mathbb{S}_x\}$, concerning the inputs; also $\tau_x = \{(P_i, O_i) : P_i \in \mathbb{S}_x\}$ and $\tau_{-x} = \{(P_i, O_i) : P_i \in \mathbb{S} \setminus \mathbb{S}_x\}$, concerning the outputs.

In order to satisfy $\widetilde{\text{skel}}^t$, given that all processes in $\mathbb{S}$ decide, and that the values of the processes in $\mathbb{T}$ are unknown, we must have:

$$\text{views}(\tau_{-x}) \in \text{skel}^t(\text{views}(\sigma_{-x})) = \text{skel}^t(\sigma^* - \{v_x\}).$$

Therefore,

$$\begin{aligned} \text{views}(\tau_x) &\subseteq \bigcap_{y \neq x} \text{skel}^t(\text{views}(\sigma_{-y})) \\ &= \bigcap_{y \neq x} \text{skel}^t(\sigma^* - \{v_y\}) \\ &\subseteq \{v_x\}, \end{aligned}$$

for any $0 \leq x \leq n-t$. In conclusion, each process decides its own input, and the decision fails to solve the problem unless $d+1 \leq t+1$, which implies $\dim(\mathcal{I}) \leq t$. In the latter situation, the protocol is trivial: each process can choose its own input without any communication. $\square$

The previous requirement on the number of processes, although not a necessary condition for the solvability of all colorless tasks, is part of an interesting sufficient condition for solvability. We shall consider the interesting cases where $\dim(\mathcal{I}^*) > 0$.

Theorem 6.7.3. For any colorless $\mathcal{T} = (\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, if

1. $n + 1 > t(\dim(\mathcal{I}^*) + 2)$; and
2. there exists a continuous map $f : |\operatorname{skel}^t(\mathcal{I}^*)| \to |\mathcal{O}^*|$ carried by Δ^*,

then we have a t-resilient protocol in asynchronous Byzantine systems for $\mathcal{T}$.

Proof. By the simplicial approximation theorem [37, 30] (also refer to Theorem 3.4.5 and Remark 3.4.6), f has a *simplicial approximation*

$$\phi : \operatorname{Bary}^N \operatorname{skel}^t(\mathcal{I}^*) \to \mathcal{O}^*,$$

for some $N > 0$, also carried by Δ^*. The Byzantine-failure protocol for non-faulty processes is shown below, presuming that $n + 1 > t(\dim(\mathcal{I}^*) + 2)$ with $\dim(\mathcal{I}^*) > 0$.

1. Execute the Byzantine strong $(t+1)$-set agreement protocol (Algorithm 13), choosing vertices on a simplex in $\operatorname{skel}^t(\mathcal{I}^*)$.
2. Execute N times the Byzantine barycentric agreement protocol, choosing vertices on a simplex in
$$\operatorname{Bary}^N \operatorname{skel}^t(\mathcal{I}^*).$$
For simplicity of presentation, we assume the approach described in Sec. 6.5, based on [34].
3. Apply $\phi : \operatorname{Bary}^N \operatorname{skel}^t(\mathcal{I}^*) \to \mathcal{O}^*$ to choose vertices on a simplex in $\mathcal{O}^*$.

As ϕ and f are carried by Δ^*, non-faulty processes starting on vertices of $\sigma_I \in \mathcal{I}^*$ finish on vertices of $\sigma_O \in \Delta^*(\sigma)$. Furthermore, since $1 \leq \dim(\sigma_I) \leq \dim(\mathcal{I}^*)$, by definition, the preconditions are satisfied for calling the protocols in steps (1) and (2). □

6.8 Strict Colorless Tasks

In fact, the Equivalence Theorem can imply even more general results, for *strict colorless tasks*, formally defined below.

Definition 6.8.1. A colorless task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$ is *strict* if

$$\Delta^*(\sigma_1 \cap \sigma_2) = \Delta^*(\sigma_1) \cap \Delta^*(\sigma_2)$$

for any $\sigma_1, \sigma_2 \in \mathcal{I}^*$.

Theorem 6.8.2. If a strict colorless task $\mathcal{T} = (\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$ with $n+1 \leq t(\dim(\mathcal{I}^*)+2)$ has an asynchronous Byzantine protocol, there is a *simplicial* map $\delta : \mathcal{I}^* \to \mathcal{O}^*$ carried by Δ^*.

Proof. If $\mathcal{T}$ is solvable, then $\mathcal{T}' = (\sigma^*, \Delta^*(\sigma^*), \Delta^*)$ is similarly solvable, taking an arbitrary d-simplex $\sigma^* = \{v_0, \ldots, v_d\}$ in $\mathcal{I}^*$ with $d = \dim(\mathcal{I}^*)$. By our equivalence theorem, and considering the relations in (6.1) and (6.2), we must be able to solve

$$\mathcal{T}'' = (\Psi(\mathbb{P}, \sigma^*), \Psi(\mathbb{P}, \Delta^*(\sigma^*)), \widetilde{\Delta^*}), \tag{6.6}$$

where, for any canonical name-labeled $\sigma, \tau \in \Psi(\mathbb{P}, \sigma^*)$:

$$\tau \in \widetilde{\Delta^*}(\sigma) \Leftrightarrow \forall \text{ canonical } \sigma' \subseteq \sigma, \exists \text{ matching } \tau' \subseteq \tau \text{ with } \text{views}(\tau') \in \Delta^*(\text{views}(\sigma')). \tag{6.7}$$

Assume a protocol for $\mathcal{T}''$, and also that $(n+1)-t \leq t(\dim(\mathcal{I}^*)+1)$. Consider an execution where: (i) all processes behave correctly or crash; (ii) all processes in $\mathbb{S} = \{P_0, \ldots, P_{n-t}\}$ terminate without receiving any message from any process in $\mathbb{T} = \{P_{n+1-t}, \ldots, P_n\}$; (iii) each process $P_i \in \mathbb{S}$ starts with input $I_i = v_{i \bmod d+1}$. In this case, define

$$\mathbb{S}_x = \{P \in \mathbb{S} : P \text{ has input } v_x\}. \tag{6.8}$$

Note that $(n+1) - t \geq d+1$, as $\mathcal{I}^*$ contains only inputs chosen by non-faulty processes and $d = \dim(\mathcal{I}^*)$ by assumption. Consequently, since $(d+1) \leq n+1-t \leq t(d+1)$, and in light of (6.8),

$$0 < |\mathbb{S}_x| \leq t \text{ for all } 0 \leq x \leq d.$$

In terms of notation, define $\sigma_x = \{(P_i, I_i) : P_i \in \mathbb{S}_x\}$ and $\sigma_{-x} = \{(P_i, I_i) : P_i \in \mathbb{S} \text{ with } P_i \notin \mathbb{S}_x\}$ regarding inputs, as well as $\tau_x = \{(P_i, O_i) : P_i \in \mathbb{S}_x\}$ and $\tau_{-x} = \{(P_i, O_i) : P_i \in \mathbb{S} \text{ with } P_i \notin \mathbb{S}_x\}$ regarding outputs.

In order to satisfy $\widetilde{\Delta^*}$, given that all processes in $\mathbb{S}$ decide, and that the values of the processes in $\mathbb{T}$ are unknown, we must have:

$$\text{views}(\tau_{-x}) \in \Delta^*(\text{views}(\sigma_{-x})) = \Delta^*(\sigma^* - \{v_x\}).$$

Therefore,

$$\begin{aligned}\text{views}(\tau_x) &\subseteq \bigcap_{y \neq x} \Delta^*(\text{views}(\sigma_{-y})) \\ &= \bigcap_{y \neq x} \Delta^*(\sigma^* - \{v_y\}) \\ &\subseteq \Delta^*(\{v_x\}),\end{aligned}$$

for any $0 \leq x \leq n - t$. Considering Def. 6.8.1, note that $\Delta^*(v_1) \neq \Delta^*(v_2)$ for $v_1 \neq v_2$ with $v_1 \in \sigma^*$ and $v_2 \in \sigma^*$. We fail to solve the problem (and incur into contradiction) unless $\tau^* = \{\Delta^*(v) : v \in \sigma^*\} \subseteq \Delta^*(\sigma^*)$, so there must exist a simplicial map from σ^* to τ^* carried by Δ^*, considering the previous observation. Because σ^* was chosen arbitrarily, we must also have a simplicial map $\delta : \mathcal{I}^* \to \mathcal{O}^*$ carried by Δ^*. □

With strict colorless tasks, a *simplicial* map δ from $\mathcal{I}^*$ to $\mathcal{O}^*$, allows processes to decide their outputs without communication: they simply apply the function δ to their input and decide. Informally speaking, a solvable, strict colorless task with $n + 1 \leq t(\dim(\mathcal{I}^*) + 2)$, is trivial in that sense.

6.9 Final Remarks

In this section, a specialized, more fitting model permits us to express slightly more specific conditions for solvability. In particular, we show that the strict k-set agreement requires a certain number of processes, except in trivial cases. Furthermore, we show that a colorless Byzantine protocol for any task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, in asynchronous systems, implies the existence of a continuous map $f : |\operatorname{skel}^t \mathcal{I}^*| \to |\mathcal{O}^*|$ carried by Δ^*. We also show that $n + 1 > t(\dim(\mathcal{I}^*) + 2)$ is enough to solve an arbitrary colorless task when such map indeed exists. If the number of processes is less than $t(\dim(\mathcal{I}^*) + 2) + 1$, and Δ^* is a *strict carrier map*, processes cannot identify a non-faulty process input besides its own. The task becomes unsolvable unless processes can decide based solely on their own input, which requires no communication. In addition to solvability conditions, we present protocols for the barycentric agreement problem that seem to delineate what is possible and impossible in terms of computability on colorless tasks.

Chapter 7

Synchronous Computability Conditions

In this chapter, we use topology notions to analyze the combinatorial structure of information dissemination in a Byzantine synchronous protocol. The structure of information dissemination is modeled by protocol complexes, and the ambiguity that arises from failures, both crashing or malicious, is modeled by protocol complex connectivity.

In this chapter, we see how Byzantine failures can impose us one extra synchronous round to deal with information ambiguity, yet, in specific settings, described in Sec. 7.4, at most that. In terms of solvability vs. number of rounds, we see how the penalty for moving from crash to Byzantine failures, modeled as $(k-1)$-connectivity in the protocol complex, can be quite limited in synchronous systems, particularly when n is relatively large compared to t.

7.1 Contributions Overview

The first contribution of this chapter comes in Sec. 7.3. We show that, in a Byzantine synchronous system, the protocol complex can remain $(k-1)$-*connected* for $\lceil t/k \rceil$ rounds, where $(k-1)$-connectivity is a general notion of graph connectivity for higher dimensions, and t is an upper bound on the number of Byzantine processes. This is potentially *one* more round than the upper bound in crash-failure systems ($\lfloor t/k \rfloor$, shown in [12]). Technically, we conceive a combinatorial operator modeling the ability of Byzantine processes to equivocate – i.e., to transmit ambiguous state information – without revealing their Byzantine nature.

We compose this operator with regular crash-failure operators, extending the protocol complex connectivity for one extra round. Connectivity is relevant since a $(k-1)$-connected protocol complex prevents important problems such as k-set agreement [11, 14] from having solutions.

The second contribution of this chapter comes in Sec. 7.4. We show that the above connectivity bound is *tight* in certain settings (described in Sec. 7.4), by (i) defining a strong validity formulation of k-set agreement for Byzantine synchronous systems; and (ii) solving the problem in $\lceil t/k \rceil + 1$ rounds. We do so with a full-information protocol that assumes n suitably large compared to t (hence, mainly of theoretical interest). Nevertheless, the protocol suits well our purpose of tightening the $\lceil t/k \rceil$ bound, and also exposes very well *the reason why* $\lceil t/k \rceil + 1$ rounds is enough.

7.1.1 Related Work

The Byzantine failure model was initially introduced by Lamport, Shostak, and Pease [31]. The use of simplicial complexes to model distributed computations was introduced by Herlihy and Shavit [27]. The asynchronous computability theorem for general tasks in [28] details the approach for wait-free computation, recently generalized by Gafni, Kuznetsov, and Manolescu [20]. Computability in Byzantine asynchronous systems, where tasks are constrained in terms of non-faulty inputs, was recently considered in [36].

The k-set agreement problem was originally defined by Chaudhuri [11]. Alternative formulations with different validity notions, or failure/communication settings, are discussed in [39, 14]. A full characterization of optimal translations between different failure settings is given in [4, 40], which requires different number of rounds depending on the relation between the number of faulty processes, and the number of participating processes.

The relationship between connectivity and the impossibility of k-set agreement is described explicitly or implicitly in [12, 28, 43]. Recent work by Castañeda, Gonczarowski, and Moses [10] considers an issue of chains of hidden values, a concept loosely explored here. The approach based on shellability and layered executions for lower bounds in connectivity has been used by Herlihy, Rajsbaum, and Tutte [26, 25, 22], assuming crash-failure systems, synchronous or asynchronous.

7.2 Operational Model

In this section, we also call the number of processes $n+1$ instead of n, for identical reasons as in our asynchronous setting: choosing $n+1$ processes simplifies the topological notation,

which we prefer for compatibility with prior work. Formally:

We let number of processes be $n+1$. Define $\mathbb{P} = \{P_0, \ldots, P_n\}$.

Similar to the asynchronous setting, *any* set of at most t processes can be *faulty* or *Byzantine* [31], and may display arbitrary, even malicious behavior, at any point in the execution. We again assume that an *adversary* defines the actual behavior of Byzantine processes. Processes behaving in strict accordance to the protocol for rounds 1 up to r (inclusive) are called *non-faulty processes up to round* r, and are denoted by $\mathbb{G}^r$. Also, *faulty processes up to round* r are denoted by $\mathbb{B}^r = \mathbb{P} \setminus \mathbb{G}^r$. A non-faulty process up to any round $r \geq 1$ is called simply *non-faulty* or *correct*, which we denote by $\mathbb{G}$.

We model processes as state machines, as seen in Sec. 2.3. The input value (respectively, output value) of a non-faulty process P_i is written I_i (respectively, O_i), and non-faulty process P_i has an internal state called *view*, which we denote by $\text{view}(P_i)$.

For simplicity of notation, we define a round 0 where processes are simply assigned their inputs. Without losing generality, all processes are assumed non-faulty up to round 0: $\mathbb{G}^0 = \mathbb{P}$ and $\mathbb{B}^0 = \emptyset$. For any round $r \geq 0$, a *global state* formally specifies: (1) the non-faulty processes up to round r; and (2) the view of all non-faulty processes up to round r.

Let $\text{view}^r(P_i)$ denote P_i's view at the end of round r. In the beginning of the protocol, $\text{view}^0(P_i)$ is I_i.

7.3 Connectivity Upper Bound

Say that non-faulty processes start the computation with inputs in $V = \{v_0, \ldots, v_d\}$, with some $d \geq k$ and $t \geq k \geq 1$. In order to prove our upper bound, we consider the following admissible execution imposed by the adversary.

Let $r = \lfloor t/k \rfloor$ and $m = t \bmod k$. We have r *crash rounds*, where each round consists of k processes failing by crashing. If $m > 0$, we have an extra *equivocation round*, where a single Byzantine process sends different views to different processes, causing extra confusion. This scenario defines a sequence of protocol complexes $\mathcal{K}^0, \ldots, \mathcal{K}^{r+1}$, and carrier maps $\mathcal{C}^i : \mathcal{K}^{i-1} \to 2^{\mathcal{K}^i}$, for $1 \leq i \leq r$, and $\mathcal{E} : \mathcal{K}^r \to 2^{\mathcal{K}^{r+1}}$.

$$\mathcal{K}^0 \xrightarrow{\mathcal{C}^1} \mathcal{K}^1 \ldots \xrightarrow{\mathcal{C}^r} \mathcal{K}^r \underbrace{\xrightarrow{\mathcal{E}} \mathcal{K}^{r+1}}_{\text{only if } m > 0}. \tag{7.1}$$

In each of the first r rounds, exactly k processes are failed by the adversary. The crash-failure operator represents each of such round schedules, and is defined as follows [22, 21]:

Definition 7.3.1. For any $1 \leq i \leq r$, the crash-failure operator $\mathcal{C}^i : \mathcal{K}^{i-1} \to 2^{\mathcal{K}^i}$ is such that

$$\mathcal{C}^i(\sigma) = \bigcup_{\tau \in \text{Faces}^{n-ik}(\sigma)} \Psi(\text{names}(\tau); [\tau : \sigma]) \tag{7.2}$$

for any $\sigma \in \mathcal{K}^{i-1}$, with $[\tau : \sigma]$ denoting any simplex μ where $\tau \subseteq \mu \subseteq \sigma$.

Definition 7.3.2. A q-connected carrier map $\Phi : \mathcal{K} \to 2^{\mathcal{L}}$ is a strict carrier map such that, for all $\sigma \in \mathcal{K}$, $\dim(\Phi(\sigma)) > q - \text{codim}_{\mathcal{K}}(\sigma)$ and $\Phi(\sigma)$ is $(q - \text{codim}_{\mathcal{K}}(\sigma))$-connected.

Definition 7.3.3. A q-shellable carrier map $\Phi : \mathcal{K} \to 2^{\mathcal{L}}$ is a strict carrier map such that, for all $\sigma \in \mathcal{K}$, $\dim(\Phi(\sigma)) > q - \text{codim}_{\mathcal{K}}(\sigma)$ and $\Phi(\sigma)$ is shellable.

After r rounds, note that $\mathcal{K}^r$ only contains simplexes with dimension exactly $n - rk$. In [22, 21], the following lemmas are proved:

Lemma 7.3.4. For $1 \leq i \leq r$, the operator $\mathcal{C}^i : \mathcal{K}^{i-1} \to 2^{\mathcal{K}^i}$ is a $(k-1)$-shellable carrier map.

Proof. Proof in [22]. □

Lemma 7.3.5. If $\mathcal{M}^1, \ldots, \mathcal{M}^x$ are all q-shellable carrier maps, and $\mathcal{M}^{x+1}$ is a q-connected carrier map, the composition $\mathcal{M}^1 \circ \ldots \mathcal{M}^x \circ \mathcal{M}^{x+1}$ is a q-connected carrier map, for any $x \geq 0$.

Proof. Proof in [22]. □

After the crash-failure rounds, if $m > 0$ the adversary picks one of the remaining processes to behave maliciously at round $r + 1$. This process, say P_b, may send different views to different processes (which is technically called *equivocation*), but, informally speaking, all views are "plausible". For example, P_i and P_j in $\mathbb{G}^{r+1}$ (i.e., two correct remaining processes) can be indecisive on whether the global state at round r is σ_1 or σ_2 in $\mathcal{K}^r$, while P_b (a Byzantine process) sends a state corresponding to σ_1 to P_i, and a state corresponding to σ_2 to P_j in round $r+1$. The process P_b *does not reveal* its Byzantine nature, yet it *promotes ambiguity* in the state information diffusion.

When a non-faulty process receives a state, it must decide assuming that one process has failed. So, if a process can receive states σ_1 and σ_2, with $\dim(\sigma_1 \cap \sigma_2) = n - rk - 1$, the *interpretation* of the message with a state σ_1 is the same as the message with a state σ_2. We capture this notion using the *equivocation* operator, described below, together with an associated *interpretation* operator Interp such that $\text{Interp}(\sigma_1) = \text{Interp}(\sigma_2)$ for processes in $\text{names}(\tau)$, where $\tau = \sigma_1 \cap \sigma_2$ with $\dim(\tau) = n - rk - 1$. Formally:

Definition 7.3.6. For any σ_1 and σ_2 in $\mathcal{K}$, with $\dim(\mathcal{K}) = n - rk$, let $(P_i, \text{Interp}(\sigma_1)) = (P_i, \text{Interp}(\sigma_2))$ if and only if

1. $\sigma_1 = \sigma_2$; **or**

2. $P_i \in \text{names}(\tau)$ where $\tau = \sigma_1 \cap \sigma_2$ and $\dim(\tau) = n - rk - 1$.

Definition 7.3.7. For any pure simplicial complexes $\mathcal{K}$ and $\mathcal{L}$ with $\dim(\mathcal{K}) \leq n - rk$ and $\mathcal{K} \supseteq \mathcal{L}$, the $\mathcal{K}$-equivocation operator $\mathcal{E}_\mathcal{K}$ is

$$\mathcal{E}_\mathcal{K}(\mathcal{L}) = \bigcup_{\tau \in \text{Faces}^{n-rk-1}(\mathcal{L})} \Psi(\text{names}(\tau); \{\text{Interp}(\sigma^*) : \sigma^* \in \mathcal{K}, \sigma^* \supset \tau\})). \tag{7.3}$$

For convenience of notation, define $\mathcal{E}_\mathcal{K}(\mathcal{K}) = \mathcal{E}(\mathcal{K})$.

Note that $\mathcal{E}_\mathcal{K}(\mathcal{L}) = \emptyset$ whenever $\dim(\mathcal{L}) < n - rk - 1$ or $\dim(\mathcal{K}) < n - rk$, and also that

$$\mathcal{E}_\mathcal{K}(\sigma) = \bigcup_{\tau \in \text{Faces}^{n-rk-1}(\sigma)} \Psi(\text{names}(\tau); \text{Interp}(\sigma)) \tag{7.4}$$

for any $\sigma \in \mathcal{K}$ with $\dim(\sigma) = n - rk$.

Lemma 7.3.8. For any pure, shellable simplicial complex with $\dim(\mathcal{K}) \leq n - rk$, the $\mathcal{K}$-equivocation operator $\mathcal{E}_\mathcal{K}$ is a carrier map.

Proof. Let $\tau \subseteq \sigma \in \mathcal{K}$. We show that $\mathcal{E}_\mathcal{K}(\tau) \subseteq \mathcal{E}_\mathcal{K}(\sigma)$. If $\dim(\tau) < n - rk - 1$ then $\mathcal{E}_\mathcal{K}(\tau) = \emptyset$ and $\mathcal{E}_\mathcal{K}(\tau) \subseteq \mathcal{E}_\mathcal{K}(\sigma)$ for any $\sigma \supseteq \tau \in \mathcal{K}$. Otherwise, if $\dim(\tau) = \dim(\sigma)$ then $\tau = \sigma$ and $\mathcal{E}_\mathcal{K}(\tau) = \mathcal{E}_\mathcal{K}(\sigma)$ as we assumed that $\sigma \supseteq \tau \in \mathcal{K}$. Therefore, $\dim(\tau) = n - rk - 1$ and $\dim(\sigma) = n - rk$, which makes $\mathcal{E}_\mathcal{K}(\tau) \subseteq \mathcal{E}_\mathcal{K}(\sigma)$ in light of Definition 7.3.7. □

Let $(\mathcal{C}^r \circ \mathcal{E})$ be the composite map such that $(\mathcal{C}^r \circ \mathcal{E})(\sigma) = \mathcal{E}_{\mathcal{C}^r(\sigma)}(\mathcal{C}^r(\sigma))$. While, for an arbitrary complex $\mathcal{K}$, $\mathcal{E}_\mathcal{K}$ is not a strict carrier map *per se*, we show in the following lemmas that $(\mathcal{C}^r \circ \mathcal{E})$ is a $(k-1)$-connected carrier map. Lemma 7.3.9 shows that $(\mathcal{C}^r \circ \mathcal{E})$ is a strict carrier map, and Lemma 7.3.10 shows that for any $\sigma \in \mathcal{K}^{r-1}$, $(\mathcal{C}^r \circ \mathcal{E})(\sigma)$ is $((k-1) - \text{codim}_{\mathcal{K}^{r-1}}(\sigma))$-connected.

Lemma 7.3.9. $(\mathcal{C}^r \circ \mathcal{E})$ is a strict carrier map.

Proof. Consider $\sigma, \tau \in \mathcal{K}^{r-1}$, with $\mathcal{L} = \mathcal{C}^r(\sigma)$ and $\mathcal{M} = \mathcal{C}^r(\tau)$. Both $\mathcal{L}$ and $\mathcal{M}$ are pure, shellable simplicial complexes with dimension $n - rk$ (Definition 7.3.1 and Lemma 7.3.4). Therefore, both the $\mathcal{L}$-equivocation and $\mathcal{M}$-equivocation operators are well-defined.

Furthermore, $\mathcal{C}^r$ is a strict carrier map, hence $\mathcal{L} \cap \mathcal{M} = \mathcal{C}^r(\sigma) \cap \mathcal{C}^r(\tau) = \mathcal{C}^r(\sigma \cap \tau)$. Note that $\mathcal{L} \cap \mathcal{M} = \mathcal{C}^r(\sigma \cap \tau)$, if not empty, is a pure, shellable simplicial complex with dimension $n - rk$. Therefore, the $(\mathcal{L} \cap \mathcal{M})$-equivocation operator is well-defined.

First, we show that $\mathcal{E}(\mathcal{L}) \cap \mathcal{E}(\mathcal{M}) \subseteq \mathcal{E}(\mathcal{L} \cap \mathcal{M})$, which implies one direction of our equality:

$$\mathcal{E}(\mathcal{C}^r(\sigma)) \cap \mathcal{E}(\mathcal{C}^r(\tau)) \subseteq \mathcal{E}(\mathcal{C}^r(\sigma) \cap \mathcal{C}^r(\tau)) = \mathcal{E}(\mathcal{C}^r(\sigma \cap \tau)).$$

For clarity, let $F(\mathcal{K}) = \text{Faces}^{n-rk-1}(\mathcal{K})$. Then,

$$\mathcal{E}(\mathcal{L}) \cap \mathcal{E}(\mathcal{M}) = \bigcup_{\mu \in F(\mathcal{L})} \mathcal{E}_{\mathcal{L}}(\mu) \cap \bigcup_{\nu \in F(\mathcal{M})} \mathcal{E}_{\mathcal{M}}(\nu) = \bigcup_{\substack{\mu \in F(\mathcal{L}) \\ \nu \in F(\mathcal{M})}} \mathcal{E}_{\mathcal{L}}(\mu) \cap \mathcal{E}_{\mathcal{M}}(\nu).$$

For arbitrary $\mu \in F(\mathcal{L})$ and $\nu \in F(\mathcal{M})$, if $\mathcal{E}_{\mathcal{L}}(\mu) \cap \mathcal{E}_{\mathcal{M}}(\nu) \neq \emptyset$, consider two cases:

1. μ and ν are in $\phi \in (\mathcal{L} \cap \mathcal{M})$. In this case,

   $$\mathcal{E}_{\mathcal{L}}(\mu) \cap \mathcal{E}_{\mathcal{M}}(\nu) = \Psi(\text{names}(\mu) \cap \text{names}(\mu); \text{Interp}(\phi)),$$

   which is inside $\mathcal{E}_{\mathcal{L}\cap\mathcal{M}}(\phi) \subseteq \mathcal{E}_{\mathcal{L}\cap\mathcal{M}}(\mathcal{L} \cap \mathcal{M})$.

2. $\mu \in \phi_1 \in \mathcal{L}$ and $\nu \in \phi_2 \in \mathcal{M}$. In this case,

   $$\mathcal{E}_{\mathcal{L}}(\mu) \cap \mathcal{E}_{\mathcal{M}}(\nu) = \Psi(\text{names}(\mu) \cap \text{names}(\nu); \text{Interp}(\phi_1) \cap \text{Interp}(\phi_2)).$$

   By Definition 7.3.6, the above is non-empty only when $\text{Interp}(\phi_1) = \text{Interp}(\alpha)$ with $\alpha \in \mathcal{L}$, $\text{Interp}(\phi_2) = \text{Interp}(\beta)$ with $\beta \in \mathcal{M}$, and there exists a non-empty set $\mathbb{P}'$ such that $\mathbb{P}' \subseteq \text{names}(\mu) \cap \text{names}(\nu) \subseteq \text{names}(\gamma)$, where $\gamma = \alpha \cap \beta$ with $\dim(\gamma) = n-rk-1$. Let $\mathbb{P}''$ be a maximal $\mathbb{P}'$ satisfying such condition. Note that $\gamma \in (\mathcal{L} \cap \mathcal{M})$.

   Since $(\mathcal{L} \cap \mathcal{M})$ is non-empty, it is pure, shellable with dimension $n - rk$, there must exist a simplex $\gamma' \supset \gamma$ with dimension $n - rk$. Moreover, $\text{Interp}(\gamma') = \text{Interp}(\phi_1) = \text{Interp}(\phi_2)$ for processes in $\text{names}(\gamma)$, given the definition of Interp. This way,

   $$\begin{aligned} \phi &= \Psi(\mathbb{P}''; \text{Interp}(\phi_1)) \\ &= \Psi(\mathbb{P}''; \text{Interp}(\phi_2)) \\ &= \Psi(\mathbb{P}''; \text{Interp}(\gamma')) \\ &\subseteq \Psi(\text{names}(\gamma); \text{Interp}(\gamma')), \end{aligned}$$

   which is inside $\mathcal{E}_{\mathcal{L}\cap\mathcal{M}}(\gamma') \subseteq \mathcal{E}_{\mathcal{L}\cap\mathcal{M}}(\mathcal{L} \cap \mathcal{M})$.

In the other direction,

$$\mathcal{E}(\mathcal{L} \cap \mathcal{M}) \stackrel{\text{def}}{=} \mathcal{E}_{\mathcal{L}\cap\mathcal{M}}(\mathcal{L} \cap \mathcal{M}) \subseteq \mathcal{E}_{\mathcal{L}}(\mathcal{L} \cap \mathcal{M}) \subseteq \mathcal{E}_{\mathcal{L}}(\mathcal{L}) \stackrel{\text{def}}{=} \mathcal{E}(\mathcal{L}).$$

This holds because (i) $\mathcal{E}_{\mathcal{L}\cap\mathcal{M}}(\mathcal{X}) \subseteq \mathcal{E}_{\mathcal{L}}(\mathcal{X})$ for any $\mathcal{X} \subseteq \mathcal{L} \cap \mathcal{M}$ (Definition 7.3.7); and (ii) $\mathcal{E}_{\mathcal{L}}$ is a carrier map (Lemma 7.3.8). The same argument proves that $\mathcal{E}(\mathcal{L} \cap \mathcal{M}) \subseteq \mathcal{E}(\mathcal{M})$, and therefore $\mathcal{E}(\mathcal{L} \cap \mathcal{M}) \subseteq \mathcal{E}(\mathcal{L}) \cap \mathcal{E}(\mathcal{M})$. □

Lemma 7.3.10. For any $\sigma \in \mathcal{K}^{r-1}$, $\mathcal{E}(\mathcal{C}^r(\sigma))$ is $((k-1) - \mathrm{codim}_{\mathcal{K}^{r-1}}(\sigma))$-connected.

Proof. Throughout the proof, consider a simplex $\sigma \in \mathcal{K}^{r-1}$ with $\mathrm{codim}_{\mathcal{K}^{r-1}}(\sigma) \leq k$. By Lemma 7.3.4, $\mathcal{M} = \mathcal{C}^r(\sigma)$ is a pure, shellable simplicial complex with $\dim(\mathcal{M}) = n - rk = d$. By Definition 7.3.7, $\mathcal{E}(\mathcal{M})$ is well-defined and $\dim(\mathcal{E}(\mathcal{M})) = n - rk - 1 = d'$. Note that $d' \geq n - t \geq 2t \geq 2k$, since $n + 1 > 3t$ and $t \geq k$.

First, we show that $\mathcal{E}(\mathcal{M})$ is "highly-connected" – that is, $(2k-1)$-connected. We proceed by induction on $\mu_0 \dots \mu_\ell$, a shelling order of facets of $\mathcal{M}$.

Base. We show that $\mathcal{E}_{\mathcal{M}}(\mu_0)$ is $(2k-1)$-connected. Considering Definition 7.3.7, we have that $\mathcal{E}_{\mathcal{M}}(\mu_0) = \mathcal{E}_{\mathcal{M}}(\tau_0) \cup \dots \cup \mathcal{E}_{\mathcal{M}}(\tau_d)$, with $\tau_0 \dots \tau_d$ being all the proper faces of μ_0.

Consider the cover $\{\mathcal{E}_{\mathcal{M}}(\tau_i) : 0 \leq i \leq d\}$ of $\mathcal{E}_{\mathcal{M}}(\mu_0)$, and its associated nerve $\mathcal{N}(\{\mathcal{E}_{\mathcal{M}}(\tau_i) : 0 \leq i \leq d\})$. For any index set $J \subseteq I = \{0 \dots d\}$, let

$$\mathcal{K}_J = \bigcap_{j \in J} \mathcal{E}_{\mathcal{M}}(\tau_j) = \Psi(\bigcap_{j \in J} \mathrm{names}(\tau_j); \mathrm{Interp}(\mu_0))$$

For any J with $|J| \leq d$, we have $\cap_{j \in J} \mathrm{names}(\tau_j) \neq \emptyset$, making $\mathcal{K}_J$ a non-empty pseudosphere with dimension $d' - |J| + 1 \geq 2k - |J| + 1$. So, $\mathcal{K}_J$ is $((2k-1) - |J| + 1)$-connected by Lemmas 3.8.2 and 3.8.3. The nerve is hence the $(d-1)$-skeleton of I, which is $(d-2) = (d'-1) \geq (2k-1)$-connected. By the Nerve Theorem, $\mathcal{E}_{\mathcal{M}}(\mu_0)$ is also $(2k-1)$-connected.

IH. Assume that $\mathcal{Y} = \cup_{0 \leq y < x} \mathcal{E}_{\mathcal{M}}(\mu_y)$ is $(2k-1)$ connected, and let $\mathcal{X} = \mathcal{E}_{\mathcal{M}}(\mu_x)$. We must show that $\mathcal{Y} \cup \mathcal{X} = \cup_{0 \leq y \leq x} \mathcal{E}(\mu_y)$ is $(2k-1)$-connected. Note that $\mathcal{X}$ is $(2k-1)$-connected by an argument identical to the one above for the base case $\mathcal{E}_{\mathcal{M}}(\mu_0)$.

Furthermore,

$$\mathcal{Y} \cap \mathcal{X} = \left(\bigcup_{0 \leq y < x} \mathcal{E}_{\mathcal{M}}(\mu_y) \right) \cap \mathcal{E}_{\mathcal{M}}(\mu_x) = \bigcup_{0 \leq y < x} (\mathcal{E}_{\mathcal{M}}(\mu_y) \cap \mathcal{E}_{\mathcal{M}}(\mu_x)) \stackrel{\star}{=} \bigcup_{i \in S} \mathcal{E}_{\mathcal{M}}(\tau_i),$$

where $i \in S$ indexes faces $\tau_i \in \mu_x$ such that for every $0 \leq y < x$ $(\mu_y \cap \mu_x) \neq \emptyset \Rightarrow (\mu_y \cap \mu_x) \subseteq \tau_i$ for $i \in S$. This set is well-defined because $\mathcal{M}$ is shellable.

The step $(\star)$ holds as $\mathcal{E}_{\mathcal{M}}(\mu_y) \cap \mathcal{E}_{\mathcal{M}}(\mu_x) \neq \emptyset$ only if $\psi = \Psi(\mathrm{names}(\mu_y \cap \mu_x); \mathrm{Interp}(\mu_x))$ exists, the latter inside $\psi' = \Psi(\mathrm{names}(\tau_j); \mathrm{Interp}(\mu_x))$, for some $j \in S$, given the observation in the previous paragraph. Using an argument identical to the one for $\mathcal{E}_{\mathcal{M}}(\mu_0)$, yet considering the cover $\{\mathcal{E}_{\mathcal{M}}(\tau_i) : i \in S\}$, its nerve is either the $(d-1)$-skeleton of S (if $S = \{0 \dots d\}$) or the whole simplex S (otherwise). In either case, the conclusion is the same, by the Nerve Theorem: $\cup_{i \in S} \mathcal{E}_{\mathcal{M}}(\tau_i)$ is $(2k-1)$-connected.

Once again, using the Nerve Theorem, since $\mathcal{Y}$ is $(2k-1)$-connected, $\mathcal{X}$ is $(2k-1)$-connected, and $\mathcal{Y} \cap \mathcal{X}$ is $(2k-1)$-connected, we have that $\mathcal{Y} \cup \mathcal{X}$ is $(2k-1)$-connected.

While the equivocation operator yields high connectivity $(2k-1)$ in the pseudosphere $\mathcal{C}^r(\sigma)$, the *composition* of $\mathcal{C}^r$ and $\mathcal{E}_{\mathcal{C}^r(\sigma)}(\mathcal{C}^r(\sigma))$ limits the connectivity to $(k-1)$, since the former map is only defined for simplexes with codimension $\leq k$. Formally, as $\mathcal{C}^r(\sigma) \neq \emptyset$ for any simplex $\sigma \in \mathcal{K}^{r-1}$ with $\text{codim}_{\mathcal{K}^{r-1}}(\sigma) \leq k$, we have that $\mathcal{E}(\mathcal{C}^r(\sigma))$ is $((k-1)-\text{codim}_{\mathcal{K}^{r-1}}(\sigma))$-connected. □

From Lemmas 7.3.9 and 7.3.10, we conclude the following.

Corollary 7.3.11. $(\mathcal{C}^r \circ \mathcal{E})$ is a $(k-1)$-connected carrier map.

Theorem 7.3.12. An adversary can keep the protocol complex of a Byzantine synchronous system $(k-1)$-connected for $\lceil t/k \rceil$ rounds.

Proof. If $m = 0$, $t \bmod k = 0$, and the adversary runs only the crash rounds failing k processes each time, for $r = \lfloor t/k \rfloor = \lceil t/k \rceil$ consecutive rounds. We have the following scenario:

$$(\mathcal{C}^1 \circ \ldots \circ \mathcal{C}^r)(\sigma).$$

Since $\mathcal{C}^i : \mathcal{K}^{i-1} \to 2^{\mathcal{K}^i}$ is a $(k-1)$-shellable carrier map for $1 \leq i \leq r$ (Lemma 7.3.4), the composition $(\mathcal{C}^1 \circ \ldots \circ \mathcal{C}^r)$ is a $(k-1)$-connected carrier map for any facet $\sigma \in \mathcal{I}$ (Lemma 7.3.5).

If $m > 0$, the adversary performs r crash rounds (failing k processes each time), followed by the extra equivocation round. We have the following scenario:

$$(\mathcal{C}^1 \circ \ldots \circ \mathcal{C}^{r-1} \circ (\mathcal{C}^r \circ \mathcal{E}))(\sigma). \tag{7.5}$$

Since $\mathcal{C}^i : \mathcal{K}^{i-1} \to \mathcal{K}^i$ is a $(k-1)$-shellable carrier map for $1 \leq i \leq r-1$ (Lemma 7.3.4), and $(\mathcal{C}^r \circ \mathcal{E})$ is a $(k-1)$-connected carrier map (Corollary 7.3.11), we have that the composition above $(\mathcal{C}^1 \circ \ldots \circ \mathcal{C}^{r-1} \circ (\mathcal{C}^r \circ \mathcal{E}))$ is a $(k-1)$-connected carrier map for any facet $\sigma \in \mathcal{I}$ (Lemma 7.3.5). □

7.4 k-Set Agreement and Lower Bound

The k-set agreement problem [11], is a fundamental task having important associations with protocol complex connectivity. In Byzantine systems, it can be difficult to characterize the input of a faulty process, since a process can ignore its "prescribed" input and behave as

having a different one. This intrinsically leads to many alternative formulations for the problem in Byzantine systems [14].

Here, we adopt a formulation that is intended both to make sense in practice, and to have implications on our connectivity arguments discussed before. Each non-faulty process P_i starts with *any* value I_i from $V = \{v_0, \ldots, v_d\}$, with $d \geq k$ as well as $t \geq k \geq 1$, and finishes with a value O_i from V, respecting:

Agreement. At most k values are decided: $|\{O_i : P_i \in \mathbb{G}\}| \leq k$.

Strong Validity. For any non-faulty process P_i, the output O_i is the input value of a non-faulty process.

Termination. The protocol finishes in a finite number of rounds.

Definition 7.4.1. A Byzantine, synchronous k-set agreement task is a triple $(\mathcal{I}, \mathcal{O}, \Delta)$ where

1. $\mathcal{I}$ is a pseudosphere $\Psi(\mathbb{P}, V)$, with $V = \{v_0, \ldots, v_d\}$, representing the input of the non-faulty processes. We have $\dim(\mathcal{I}) = n$ since, for any input assignment, we have an admissible execution where all processes are benign.

2. $\mathcal{O}$ is $\cup_{V' \subseteq V} \Psi(\mathbb{P}, V')$ where $|V'| \leq k$, representing the output of the non-faulty processes. We have $\dim(\mathcal{O}) = n$ since, for any output assignment, we have an admissible execution where all processes have been benign.

3. $\Delta : \mathcal{I} \to \mathcal{O}$ where if $\tau \in \Delta(\sigma)$, then $\dim(\sigma) \geq n - t$, $\text{views}(\tau) \subseteq \text{views}(\sigma)$, and $|\text{views}(\tau)| \leq k$.

The k-set agreement and connectivity are closely related. Lemma 7.4.2 shows that no solution is possible for k-set agreement with a $(k-1)$-connected protocol complex, which, as seen in Sec. 7.3, can occur at least until round $\lceil t/k \rceil$.

Lemma 7.4.2. If, starting $\sigma \in \mathcal{I}$, the protocol complex $\mathcal{P}(\sigma)$ is $(k-1)$-connected, then no decision function δ solves the k-set agreement problem.

Proof. Consider a k-simplex $\alpha = \{u_0, \ldots, u_k\} \subseteq \{v_0, \ldots, v_d\}$ with $k+1$ different inputs. Let $\mathcal{I}_\beta = \Psi(\mathbb{P}, \beta)$ for any $\beta \subseteq \alpha$, and $\mathcal{I}_x = \bigcup_{\beta \in \text{skel}^x(\alpha)} \Psi(\mathbb{P}, \beta)$. We construct a sequence of continuous maps $g_x : |\text{skel}^x(\alpha)| \to |\mathcal{K}_x|$ where $\mathcal{K}^x$ is homeomorphic to $\text{skel}^x(\alpha)$ in $|\text{skel}^x(\mathcal{P}(\mathcal{I}_x))|$.

Base. Let g_0 map any vertex $v \in \alpha$ to a vertex in $\mathcal{K}_v = \mathcal{P}(\mathcal{I}_{\{v\}})$. We know that $\mathcal{K}_v$ is k-connected since $\dim(\mathcal{I}_{\{v\}}) = \dim(\mathcal{I})$ and $\mathcal{P}$ is a k-connected carrier map. We just

constructed

$$g_0 : |\operatorname{skel}^0(\alpha)| \to |\mathcal{K}_0|,$$

where $\mathcal{K}^0$ is isomorphic to a $\operatorname{skel}^0(\alpha)$ in $|\operatorname{skel}^0(\mathcal{P}(\mathcal{I}_0))|$.

Induction Hypothesis. Assume $g_{x-1} : |\operatorname{skel}^{x-1}(\alpha)| \to |\mathcal{K}_{x-1}|$ for any $x \leq k$, where $\mathcal{K}_{x-1}$ is isomorphic to $\operatorname{skel}^{x-1}(\alpha)$ in $|\operatorname{skel}^{x-1}(\mathcal{P}(\mathcal{I}_{x-1}))|$. For any $\beta \in \operatorname{skel}^x(\alpha)$, we have that $\operatorname{skel}^x(\mathcal{P}(\mathcal{I}_\beta))$ is $(x-1)$-connected, hence the continuous image of the $(x-1)$-sphere in $\mathcal{P}(\mathcal{I}_\beta)$ can be extended to the continuous image of the x-ball in $\operatorname{skel}^x(\mathcal{P}(\mathcal{I}_\beta))$. We just constructed

$$g_x : |\operatorname{skel}^x(\alpha)| \to |\mathcal{K}_x|,$$

where $\mathcal{K}^x$ is isomorphic to $\operatorname{skel}^x(\alpha)$ in $|\operatorname{skel}^x(\mathcal{P}(\mathcal{I}_0))|$. In the end, we have $g_k : |\alpha| \to |\mathcal{K}_k|$ where $\mathcal{K}_k$ is isomorphic to α in $\operatorname{skel}^k(\mathcal{P}(\mathcal{I}_k))$.

Now suppose, for the sake of contradiction, that k-set agreement is solvable, so there must be a simplicial map $\delta : \mathcal{P}(\mathcal{I}) \to \mathcal{O}$ carried by Δ. Then, induce the continuous map $\delta_c : |\mathcal{K}_k| \to |\alpha|$ from δ such that $\delta_c(v) \in |\operatorname{views}(\delta(\mu))|$ if $v \in |\mu|$, for any $\mu \in \mathcal{K}_k$. Also, note that the composition of g_k with the continuous map δ_c induces another continuous map $|\alpha| \to |\partial\,\alpha|$, since by assumption δ never maps a k-simplex of $\mathcal{K}_k$ to a simplex with $k+1$ different views (so δ_c never maps a point to $|\operatorname{Int}\alpha|$). We built a *continuous retraction* of α to its own border $\partial\,\alpha$, a contradiction (please refer to [37, 30]). Since our assumption was that there existed a simplicial map $\delta : \mathcal{P}(\mathcal{I}) \to \mathcal{O}$ carried by Δ, we conclude that k-set agreement is not solvable. □

We now present a simple k-set agreement algorithm for Byzantine synchronous systems, running in $\lceil t/k \rceil + 1$ rounds. The procedure requires quite a large number of processes compared to t – we require $n + 1 \geq kt(d+2) + k$ – and was designed with the purpose of tightening the connectivity lower bound, favoring simplicity over the optimality on the number of processes,

Non-faulty processes initially execute a *gossip phase* for $\lceil t/k \rceil + 1$ rounds, followed by a *validation phase*, and a *decision phase*, where the output is chosen. Define $R = \lceil t/k \rceil$, and consider the following tree, where nodes are labeled with words over the alphabet $\mathbb{P}$. The root node is labeled as λ, which represents an empty string. Each node w such that $0 \leq |w| \leq R$ has $n+1$ child nodes labeled wp for all $p \in \mathbb{P}$. Any non-faulty process p maintains such tree, denoted T_p.

All nodes w are associated with the value $\operatorname{Cont}_p(w)$, called the *contents* of w. The meaning of the trees is the following: after the gossip phase, if node $w = p_1 \ldots p_x$ is such that $\operatorname{Cont}_p(w) = v$, then p_x told that p_{x-1} told that … p_1 had input v to p. The special

value $\perp$ represents an absent input. We omit the subscript p when the process is implied or arbitrary. We divide the processes into k disjoint groups: $\mathbb{P}(g) = \{P_x \in \mathbb{P} : x = g \bmod k\}$, for $0 \leq g < k$. For any tree T, we call $T(g)$ the subtree of T having only nodes $wp \in T$ such that $p \in \mathbb{P}(g)$.

In the validation phase, if we have a set $\mathbb{Q}$ containing $(n+1) - t$ processes that acknowledge all messages transmitted by process p (making sure that $p \in \mathbb{Q}$), at every round $1 \leq r \leq R$, we call such set the *quorum* of p, denoted *Quorum*(p). Formally, *Quorum*$(p) = \mathbb{Q} \subseteq \mathbb{P}$ such that $p \in \mathbb{Q}$, $|\mathbb{Q}| \geq (n+1) - t$, and $q \in \mathbb{Q}$ whenever $\text{Cont}(wp) = v$ implies $\text{Cont}(wpq) = v$, for any w such that $0 \leq |w| < R$. It should be clear that every non-faulty process has a quorum containing at least all other non-faulty processes. If a process p has a quorum in process $P_i \in \mathbb{G}$, we say that wp has been *validated* on P_i for any $0 \leq |w| < R$ (particularly, p has been validated on P_i). Note that in our definition either all entries for process p are validated, or none is. Lemma 7.4.3 shows that validated entries are unique across non-faulty processes.

Lemma 7.4.3. If p has been validated on non-faulty processes P_i and P_j, then $\text{Cont}_i(wp) = \text{Cont}_j(wp)$ for any $0 \leq |w| < R$.

Proof. If p has been validated on $P_i \in \mathbb{G}$, then $\text{Cont}_i(wp) = v$ implies $\text{Cont}_i(wpq) = v$ for $(n+1) - t$ different processes $q \in \mathbb{Q}_i$, and $\text{Cont}_j(wp) = v$ implies $\text{Cont}_j(wpq) = v$ for $(n+1) - t$ different processes $q \in \mathbb{Q}_j$, for any $0 \leq |w| < R$. As we have at most t non-faulty processes and $n + 1 > 3t$, $|\mathbb{Q}_i \cap \mathbb{Q}_j| \geq (n+1) - 2t > t + 1$, containing at least one non-faulty process that, by definition, broadcasts values consistently in its run. Hence, $\text{Cont}_i(wp)$ and $\text{Cont}_j(wp)$ must be identical. □

In the decision phase, if we see t processes without a quorum, we have technically identified all non-faulty processes $\mathbb{B}$. In this case, we fill R-th round values of any $b \in \mathbb{B}$ using the *completion* rule: we make $\text{Cont}(wb) = v$ if we have $(n+1) - 2t$ processes $\mathbb{G}' \subseteq \mathbb{G}$ where $\text{Cont}(wbq) = v$ for any $q \in \mathbb{G}'$ and $|w| = R - 1$. If a process p has its R-round values completed as above in process $P_i \in \mathbb{G}$, we say that wp has been *completed* on P_i for any $|w| = R - 1$. Lemma 7.4.4 shows that completed entries are identical and consistent with validated entries across non-faulty processes. (Intuitively, the completion rule was done over identical values from correct processes.)

Lemma 7.4.4. If wp has been completed or validated on a non-faulty process P_i, and wp has been completed on a non-faulty process P_j, then $\text{Cont}_i(wp) = \text{Cont}_j(wp)$.

Proof. If wp has been validated on P_i, $\text{Cont}_i(wp) = v$ implies $\text{Cont}_i(wpq) = v$ for $(n+1)-t$ different processes $q \in \mathbb{Q}$. When P_j applies the completion rule on wp, then $\text{Cont}_j(wpq) = v$ for $(n+1)-2t$ different processes $q \in \mathbb{G}$, as we have at most t faulty processes. Therefore, $\text{Cont}_i(wp) = \text{Cont}_j(wp)$.

If wp has been completed on all non-faulty processes, they all have identified t faulty processes, and the completion rule is performed over identical entries associated with non-faulty processes. Therefore, $\text{Cont}_i(wp) = \text{Cont}_j(wp)$ as well. □

We have two possible cases:

1. there is a subtree $T(g)$ with less than $\lceil t/k \rceil$ non-validated processes – call such subtree *pivotal*; or

2. no such tree exists, in which case we apply the completion rule to R-round values in $T(0)$, and define $T(0)$ as our pivotal subtree instead.

Now, any sequence of non-validated and non-completed nodes $p_1, (p_1p_2), \ldots, (p_1p_2 \ldots p_x)$, with $p_1 \neq \ldots \neq p_x$, has size $x < R = \lceil t/k \rceil$, allowing us to suitably perform consensus over consistent values (see below).

Denote the set of processes in the word w as $\text{SetProc}(w)$. For any non-validated wp with $b \in \mathbb{P}(g)$ in a pivotal subtree $T(g)$, where $1 \leq |wp| < R$, we establish consensus on $\text{Cont}(wb)$. We apply the *consensus* rule: $\text{Cont}(wb) = v$ if the majority of processes in $\mathbb{P}(g) \setminus \text{SetProc}(wb)$ is such that $wbp = v$. This rule is applied first to entries labeled wp where $|wp| = R - 1$, and then moving upwards (please refer to Alg. 14). Lemma 7.4.5 shows that the consensus rule indeed establishes consensus across non-faulty processes that identify $T(g)$ as the pivotal subtree. Essentially, we are separating the possible chains of unknown values across disjoint process groups, which either forces one of these chains to be smaller than $R = \lceil t/k \rceil$, or reveals all faulty processes, giving us the ability to perform the completion rule. This fundamental tradeoff underlies our algorithm, and ultimately explains *why* the $\lceil t/k \rceil$ connectivity bound is tight.

The decision is based on values resulting from consensus on $T(g)$, taking the minimum element appearing at least $t+1$ times. Define the multiset $C_g = \text{Multiset}(\text{Cont}(p) : p \in T(g))$ after applying the consensus rule. Let $\text{Decision}(C_g) = \min\{v : v \in C_g \text{ with cardinality} \geq t+1\}$. Lemma 7.4.6 shows that since $|\mathbb{P}(g)| > t(d+2)$, such value exists, and we can decide on a value that has been necessarily input by a non-faulty process.

Lemma 7.4.5. For any two-non-faulty processes P_i and P_j that applied the consensus rule on a pivotal subtree $T(g)$, with $0 \leq g < k$, we have that $\text{Cont}_i(p) = \text{Cont}_j(p)$ for any

$p \in \mathbb{P}(g)$.

Proof. Let $\text{SetCons}(w) = \mathbb{P}(g) \setminus \text{SetProc}(w)$ for any $w \in T(g)$ with $|w| \leq R$. If wp has been validated at P_i with $\text{Cont}_i(wp) = v$, at most t values from $S_i = \text{Multiset}(\text{Cont}_i(wpq) : q \in \text{SetCons}(wp))$ will be different than v. Since we have at most t faulty processes, at most $2t$ values from $S_j = \text{Multiset}(\text{Cont}_j(wpq) : q \in \text{SetCons}(wp))$ will be different than v. As we assume $d \geq k \geq 2$ (or we are executing the consensus algorithm), we have that $|\mathbb{P}(g)| > 4t$ because each $\mathbb{P}(g)$ contains at least $t(d+2)+1$ processes. Therefore, the majority of values in S_j is v, making $\text{Cont}_i(p) = \text{Cont}_j(p)$.

If all non-faulty processes did not validate wp, they apply the consensus rule consistently, over the same values in $\text{Multiset}(\text{Cont}(wpq) : q \in \text{SetCons}(wp))$, also making $\text{Cont}_i(p) = \text{Cont}_j(p)$. □

Lemma 7.4.6. For any two-non-faulty processes P_i and P_j that decide based on a pivotal subtree $T(g)$, with $0 \leq g < k$, the decision value (i) is well-defined; and (ii) is an input value of a non-faulty process.

Proof. By Lemma 7.4.5, every process P_i from $\mathbb{P}(g)$ will have an associated value C_i resulting from the consensus rule, and this value is consistent across non-faulty processes that decide based on $T(g)$. If $P_i \in \mathbb{G}$, C_i must be P_i's input I_i, as P_i is necessarily validated. Since we have at most t faulty processes and $|\mathbb{P}(g)| > t(d+2)$, more than $t(d+1)$ values are inputs of non-faulty processes, and since we have at most $d+1$ input values, one value must appear $t+1$ times. Therefore, our decision function is well-defined.

Since non-faulty processes that decide on $T(g)$ decide consistently on a multiset of consistent values, the decision is identical across those processes. Also, any value appearing at least $t+1$ times must have been input by a non-faulty process, as we have at most t faulty processes. □

Theorem 7.4.7. Algorithm 14 solves k-set agreement in $\lceil t/k \rceil + 1$ rounds.

Proof. Termination is trivial, as we execute exactly $R = \lceil t/k \rceil + 1$ rounds. By Lemma 7.4.5, each pivotal subtree yields a unique decision value. As we have at most k pivotal subtrees identified across non-faulty processes, up to k values are possibly decided across non-faulty processes. Finally, by Lemma 7.4.6, the decision value is an input value of some correct process. □

Algorithm 14 P_x.Agree(I)

1: **if** $k = 1$ **then**
2: **return** `Decision`(Multiset(Cont(p) : p output by consensus algorithm))
3: **end if**
4: Cont(w) $\leftarrow \perp$ for all $w \in T$
5: Cont(λ) $\leftarrow I$ ▷ Gossip
6: **for** $\ell : 1$ **to** $\lceil t/k \rceil + 1$ **do**
7: **send**$S_x^{\ell-1} = \{(w, \text{Cont}(w)) : w \in T^{\ell-1}\}$
8: **upon recv**$S_y^{\ell-1} = \{(w, v) : w \in T^{\ell-1}, v \in V \cup \{\perp\}\}$ from P_y **do**
9: Cont(wP_y) $\leftarrow v$ for all $(w, v) \in S_y^{\ell-1}$
10: **end upon**
11: **end for**
12: $\mathbb{P}' \leftarrow \{P_i : P_i$ has a quorum$\}$ ▷ Validation
13: **if** $|\mathbb{P}'| = (n + 1) - t$ **then**
14: Apply completion rule for all wb where $b \in \mathbb{P} \setminus \mathbb{P}'$ and $|wb| = \lceil t/k \rceil$
15: **end if**
16: $g \leftarrow$ any g such that $T(g)$ is pivotal ▷ Decision
17: **for** $\ell : \lceil t/k \rceil - 1$ **to** 1 **do**
18: Apply consensus rule for all non-validated wb where $b \in \mathbb{P}(g)$ and $|wb| = \ell$
19: **end for**
20: **return** `Decision`(Multiset(Cont(p) : $p \in T(g)$))

7.5 Final Remarks

In Byzantine synchronous systems, the protocol complex can remain $(k-1)$-connected for $\lceil t/k \rceil$ rounds, potentially *one* more round than the upper bound in crash-failure systems. We conceive a combinatorial operator modeling the ability of Byzantine processes to equivocate without revealing their Byzantine nature, just after $\lfloor t/k \rfloor$ rounds of crash failures. We compose this operator with the regular crash-failure operators, extending the connectivity bounds up to $\lceil t/k \rceil$. We tighten this bound, at least when n is suitably large compared to t, via a full-information protocol that solves a strong validity formulation of k-set agreement. We did not attempt to maximize the protocol resiliency – we leave this as an open question – but evidence suggests $n + 1 > t(d + 1)$ is enough to solve the problem when $k = t$ (check by using our protocol without lines 16 to 18).

Byzantine failures can impose us *one* extra synchronous round to address information ambiguity, and, in specific settings, *at most that*. In terms of solvability vs. number of rounds, the penalty for moving from crash to Byzantine failures can thus be quite limited. There is an intuitive sense in that result when we realize that

1. in synchronous systems with large enough resilience, we can detect crash failures with

a 1-round delay [4]; and

2. techniques similar to the reliable broadcast of [8, 44] deal with the problem of equivocation, also with a 1-round delay.

The final component of the puzzle seems to be separating the chains of unresolved values such that we suitably limit their size, or force the adversary to reveal the identity of all faulty processes. We did use those ideas in our protocol, yet the prospect of an algorithm that applies the same ideas, however with better resilience, is a thought-provoking perspective for future work.

Chapter 8

Conclusion

"Quem com esforço vence,
com prazer triunfa."

Heard from my father a long time ago...

In this thesis, we present task solvability conditions for Byzantine systems, synchronous and asynchronous, using the tools and language of combinatorial topology. While the approach has been successful before in characterizing solvability for crash-failure tasks, this thesis provides *novel solvability characterizations* that, in many ways, are intuitively related to the well-established crash-failure characterizations.

In Byzantine systems, it can be difficult to characterize the input of a faulty process, since a faulty process with input v_1 can behave as a correct process with input $v_2 \neq v_1$. We are however interested in tasks where the outputs of the non-faulty processes are constrained solely in terms of the input of the non-faulty processes, so the concept of "Byzantine input" is avoided – together with many trivial impossibility results. This approach makes sense as it focus on *strong problem formulations*: in the arguably "true" spirit of Byzantine resilience, tasks must *tolerate spurious input changes* from Byzantine processes.

The journey of this work started with a non-trivial generalization of ϵ-approximate agreement, called multidimensional ϵ-approximate agreement. Non-faulty processes start with inputs in $\mathbb{R}^d$, and they must choose finish with outputs in $\mathbb{R}^d$ such that (1) all outputs

are within ϵ of each other; and (2) all outputs lie in the convex hull of the non-faulty process inputs. The generalization is non-trivial, and has direct impact on resilience: we show that $n > t(d+2)$ is necessary and sufficient to solve the problem, where n is the number of processes, t is the maximum number of faulty processes, and d is the dimension in which inputs and outputs lie. We rely on *algorithmic techniques*, such as the reliable broadcast and the witness technique, as well as *geometric results*, such as the Helly's Theorem, to develop a protocol solving the problem as long as $n > (d+2)t$, matching the lower bound on the number of processes. Our protocol is centered across the concept of "safe area", a concept that captures very well the interdependence of dimensions, and permits a systematic convergence of values.

Despite obvious applicability to robotics and multi-agent computing, the initial interest on the multidimensional ϵ-approximate agreement was due to its ability to solve the barycentric agreement problem in asynchronous Byzantine systems, which has implications to the solvability of colorless tasks in those systems. This work formalizes colorless tasks as triples $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$ where $\mathcal{I}^*$ is the colorless input complex (a simplicial complex representing the possible inputs), where $\mathcal{O}^*$ is the colorless output complex (the analogous concept for outputs), and a carrier map $\Delta^* : \mathcal{I}^* \to 2^{\mathcal{O}^*}$. We then show that a colorless Byzantine protocol under asynchronous systems implies the existence of a continuous map $f : |\operatorname{skel}^t \mathcal{I}^*| \to |\mathcal{O}^*|$ carried by Δ^*, and that $n+1 > t(\dim(\mathcal{I}^*)+2)$ is enough to solve an arbitrary colorless task when such map indeed exists. Notation-wise, $n+1$ represents the number of processes, t is the maximum number of faulty processes, and $\dim(\mathcal{I}^*)$ is the input complex dimension. Note the match between the resilience bounds for the multidimensional ϵ-approximate agreement and for the sufficient conditions for colorless task solvability. This is not mere coincidence: if the number of processes is less than $t(\dim(\mathcal{I}^*)+2)+1$, a process trying to solve colorless tasks with strict carrier maps Δ^*, suffering from the inherent inability to distinguish faulty from correct processes, cannot identify a non-faulty process input besides its own, which makes the task impossible unless the process can decide based solely in its own input.

The natural next step in the journey was establishing necessary and sufficient solvability conditions for any Byzantine asynchronous task in which the non-faulty process outputs are constrained in terms of the non-faulty process inputs. This work gives the first such theorem. We formalize Byzantine asynchronous tasks as triples $(\mathcal{I}, \mathcal{O}, \Delta)$, where $\mathcal{I}$ represents the inputs of the non-faulty processes, $\mathcal{O}$ represents the outputs of the non-faulty processes, and $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$ constrains task semantics in terms of non-faulty inputs/outputs only. We assume an adversarial context where any set of no more than t processes could be

deemed faulty. Our main equivalence theorem says that a Byzantine-failure task $(\mathcal{I}, \mathcal{O}, \Delta)$ is solvable if and only if a specific crash-failure task $(\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$, constructed in terms of the Byzantine task above, is solvable. Given that solvability conditions have long been known for crash-failure tasks, our theorem establishes such conditions for Byzantine-failure tasks in asynchronous systems for the first time. Our theorem is based on a well-known *algorithmic technique* in distributed computing technically known as "simulation". We see how the language and techniques of combinatorial topology, when well-aligned with traditional simulation techniques, can produce novel results in distributed computing, across different failure models, facilitating existential arguments and avoiding more complicated, model-specific arguments.

The last step in the journey was considering Byzantine synchronous systems, establishing solvability lower bounds akin to crash-failure results. We model the structure of information dissemination as protocol complexes, and study the ambiguity that arises from failures, both crashing or malicious, by looking at protocol complex connectivity. Specifically, we see how the protocol complex in Byzantine systems can remain $(k-1)$-connected for $\lceil t/k \rceil$ synchronous rounds, where $(k-1)$-connectivity is a general notion of graph connectivity for higher dimensions, t is the maximum number of faulty processes, and $t \geq k \geq 1$. The bound was previously known for crash-failure systems ($\lfloor t/k \rfloor$ rounds), so Byzantine processes can introduce ambiguity for possibly one extra round. We prove this result by conceiving a combinatorial operator that models the ability of Byzantine processes to equivocate – that is, tell different things to different processes – without revealing their Byzantine nature, just after $\lfloor t/k \rfloor$ rounds of crash failures. We compose this operator with the regular crash-failure operators, extending the connectivity bounds up to $\lceil t/k \rceil$. The Byzantine bound is tightened, at least when n is suitably large compared to t, via a full-information protocol – once again an example of *algorithmic technique* – that solves an appropriate formulation of k-set agreement. We conclude that Byzantine failures can introduce one extra round of confusion, but, in specific settings, at most that. In a pure computability mindset, the penalty for moving from crash to Byzantine failures can be quite limited in this context.

As the techniques based on combinatorial topology become applicable to synchronous systems as well, it becomes clear that, when aligned with traditional algorithmic tools in distributed computing, including simulations, these techniques are effective not only across *different failure models* – crash and Byzantine – but also across *different synchrony models* – synchronous and asynchronous.

$\therefore$

In light of the previous results, this thesis finishes by recapitulating its statement:

> The language of combinatorial topology, aligned with algorithmic and simulation techniques, provides a robust framework to study task solvability not only with crash failures, but also with Byzantine failures, in synchronous and asynchronous systems. Byzantine solvability can be expressed in terms of crash-failure solvability in asynchronous systems, relies on the same fundamental problems for asynchronous colorless tasks, and have similar adversarial bounds for set agreement in synchronous systems.

Hammurabi das Chagas Mendes,
August of 2015.

Bibliography

[1] I. Abraham, Y. Amit, and D. Dolev. Optimal resilience asynchronous approximate agreement. In T. Higashino, editor, *Principles of Distributed Systems*, volume 3544 of *Lecture Notes in Computer Science*, pages 229–239. Springer Berlin / Heidelberg, 2005.

[2] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk. Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524–548, July 1990.

[3] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley Interscience, 2nd edition, March 2004.

[4] R. A. Bazzi and G. Neiger. Simplifying fault-tolerance: Providing the abstraction of crash failures. *Journal of the ACM*, 48(3):499–554, May 2001.

[5] A. Björner. Topological methods. In R. L. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics*, volume 2, pages 1819–1872. MIT Press, Cambridge, MA, USA, December 1995.

[6] E. Borowsky, E. Gafni, N. Lynch, and S. Rajsbaum. The BG distributed simulation algorithm. *Distributed Computing*, 14(3):127–146, 2001.

[7] Z. Bouzid, M. G. Potop-Butucaru, and S. Tixeuil. Optimal Byzantine-resilient convergence in uni-dimensional robot networks. *Theoretical Computer Science*, 411(34-36):3154–3168, 2010.

[8] G. Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, November 1987.

[9] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer, 2 edition, February 2011.

[10] A. Castañeda, Y. A. Gonczarowski, and Y. Moses. Brief announcement: Pareto optimal solutions to consensus and set consensus. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 113–115, New York, NY, USA, 2013. ACM.

[11] S. Chaudhuri. More choices allow more faults: set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, July 1993.

[12] S. Chaudhuri, M. Herlihy, N. A. Lynch, and M. R. Tuttle. Tight bounds for k-set agreement. *Journal of the ACM*, 47(5):912–943, September 2000.

[13] L. Danzer, B. Grünbaum, and V. Klee. Helly's theorem and its relatives. In V. L. Klee, editor, *Proceedings of the seventh Symposium in Pure Mathematics*, volume 7, pages 101–180. American Mathematical Society, 1963.

[14] R. de Prisco, D. Malkhi, and M. Reiter. On k-set consensus problems in asynchronous systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(1):7–21, January 2001.

[15] D. Dolev, N. Lynch, S. Pinter, E. Stark, and W. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, May 1986.

[16] A. Fekete. Asynchronous approximate agreement. In *Proceedings of the sixth annual ACM symposium on principles of distributed computing (PODC)*, pages 64–76, New York, NY, USA, 1987.

[17] M. Fischer, N. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

[18] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[19] M. J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). Technical Report YALEU/DCS/TR-273, Yale University, Department of Computer Science, 2000.

[20] E. Gafni, P. Kuznetsov, and C. Manolescu. A generalized asynchronous computability theorem. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 222–231, New York, NY, USA, 2014. ACM.

[21] M. Herlihy, D. Kozlov, and S. Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, December 2013.

[22] M. Herlihy and S. Rajsbaum. Concurrent computing and shellable complexes. In N. Lynch and A. Shvartsman, editors, *Distributed Computing*, volume 6343 of *Lecture Notes in Computer Science*, pages 109–123. Springer Berlin / Heidelberg, 2010.

[23] M. Herlihy and S. Rajsbaum. The topology of shared-memory adversaries. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on principles of distributed computing*, PODC '10, pages 105–113, New York, NY, USA, 2010.

[24] M. Herlihy and S. Rajsbaum. Simulations and reductions for colorless tasks. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing*, PODC '12, pages 253–260, New York, NY, USA, 2012.

[25] M. Herlihy, S. Rajsbaum, and M. Tuttle. An axiomatic approach to computing the connectivity of synchronous and asynchronous systems. *Electronic Notes in Theoretical Computer Science*, 230(0):79 – 102, March 2009.

[26] M. Herlihy, S. Rajsbaum, and M. R. Tuttle. Unifying synchronous and asynchronous message-passing models. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '98, pages 133–142, New York, NY, USA, 1998. ACM.

[27] M. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 111–120, New York, NY, USA, 1993. ACM.

[28] M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, November 1999.

[29] F. Junqueira and K. Marzullo. Designing algorithms for dependent process failures. In A. Schiper, A. Shvartsman, H. Weatherspoon, and B. Zhao, editors, *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 24–28. Springer Berlin Heidelberg, 2003.

[30] D. N. Kozlov. *Combinatorial Algebraic Topology*, volume 21 of *Algorithms and Computation in Mathematics*. Springer, 1st edition, October 2007.

[31] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transaction on Programming Languages and Systems*, 4(3):382–401, July 1982.

[32] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, March 1996.

[33] D. Malkhi, M. Merritt, M. K. Reiter, and G. Taubenfeld. Objects shared by Byzantine processes. *Distributed Computing*, 16(1):37–48, February 2003.

[34] H. Mendes and M. Herlihy. Multidimensional approximate agreement in Byzantine asynchronous systems. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing*, STOC'13, pages 391–400, New York, NY, USA, 2013. ACM.

[35] H. Mendes, M. Herlihy, N. Vaidya, and V. Garg. Multidimensional agreement in Byzantine systems. *Distributed Computing*, pages 1–19, 2015.

[36] H. Mendes, C. Tasson, and M. Herlihy. Distributed computability in Byzantine asynchronous systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 704–713, New York, NY, USA, 2014. ACM.

[37] J. Munkres. *Elements of Algebraic Topology*. Prentice Hall, 2nd edition, January 1984.

[38] J. Munkres. *Topology*. Pearson, 2nd edition, January 2000.

[39] G. Neiger. Distributed consensus revisited. *Information Processing Letters*, 49(4):195–201, February 1994.

[40] G. Neiger and S. Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms*, 11(3):374 – 419, 1990.

[41] M. Potop-Butucaru, M. Raynal, and S. Tixeuil. Distributed computing with mobile robots: An introductory survey. In *14th International Conference on Network-Based Information Systems*, NBiS'11, pages 318 –324, September 2011.

[42] D. Saari. *Basic Geometry of Voting*. Springer, September 1995.

[43] M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 101–110, New York, NY, USA, 1993. ACM.

[44] T. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, June 1987.

[45] N. Vaidya and V. K. Garg. Byzantine vector consensus in complete graphs. In *Proceedings of the 2013 ACM symposium on Principles of Distributed Computing*, PODC '13, New York, NY, USA, 2013. ACM.