

Fault-Tolerant Distributed Computability

by

Vikram Saraph

Sc. M., Brown University, 2015

B. S., University of Notre Dame, 2013

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 2019

© Copyright 2019 by Vikram Saraph

This dissertation by Vikram Saraph 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 Lysyanskya, Reader  
Brown University

Date \_\_\_\_\_

\_\_\_\_\_  
Sergio Rajsbaum, Reader  
National Autonomous University of Mexico

Approved by the Graduate Council

Date \_\_\_\_\_

\_\_\_\_\_  
Andrew Campbell  
Dean of the Graduate School

# Acknowledgements

First and foremost, I would like to thank my advisor, Maurice Herlihy, for his guidance during the course of the PhD program. Maurice has been supportive and understanding in his role as a mentor, but he has also always treated me as a peer and a coequal researcher. His ability to select challenging yet fruitful research questions will always be a great source of inspiration for the rest of my career.

I am thankful to Anna Lysyanskaya and Sergio Rajsbaum for serving on my dissertation committee. Anna has always been welcoming and accommodating during our meetings to discuss my research progress, while Sergio, a fellow champion of combinatorial topology, has always had unique insight into the more mathematical aspects of this sort of work. I am also thankful to my collaborator and coauthor, Eli Gafni, who was instrumental in helping me lay the groundwork for this thesis. I additionally want to thank Petr Kuznetsov and Thibault Rieutord for acting as “adversaries” against my research ideas and correcting some of my errors; I hope we may continue our collaboration in the future. Thanks to my academic siblings Thomas Dickerson, Archita Agarwal, Zhiyu Liu, Daniel Engel, and Eli Rosenthal for our brainstorming sessions on various research problems.

I have always had a soft spot for mathematics, so these acknowledgements would be incomplete without mentioning folks from the math community. I want to thank Tarik Aougab, Jeremy Kahn, and Elchanan Solomon for their generosity in listening to my attempts at describing some of the core mathematical challenges in this discipline. I also want to extend my thanks to the students I have met in the math department through the open graduate education program. The weekly excursions to Kabob and Curry were a welcome break from my research.

The computer science department’s staff has played a crucial role behind the scenes in making this come together. I want to thank Lori Agresti, Lauren Clarke, Genie DeGouveia, Jane Martin, Jane McIlmail, and Dawn Reed for their work that allowed me to worry less about deadlines and paperwork. I am also thankful for the technical staff Donald Johwa, Ben Nacar, Frank Pari, Paul

Vars, and Shaun Wallace for ensuring I have the infrastructure needed for my research.

I am grateful for the friends I have made during my time in the program, both within the Brown community itself, and also outside of it. Providence has provided innumerable opportunities and is an excellent place to build various circles of friendship. To Ashley Weber, for her company in all of our PhD adventures. Your companionship has forever made me a better person. To Altan Allawala, for being a wonderful roommate and a dedicated friend; I thoroughly enjoyed our numerous philosophical, political, and personal conversations we have had over the years. I want to thank the friends I met playing boardgames every Friday, including Nakul Gopalan, Thomas Dickerson, David Meierfrankenfeld, Justin Pombrio, John Meehan, Johannes Novotny, Amy Becker, Sasha Berkoff, and Jeroen Chua. I am also thankful for everyone that helped revived the tradition of going to the GCB on Wednesdays, including Nediya Daskalova, Ghous Amjad, Archita Agarwal, Marilyn George, Michael Markovitch, and Daniel Engel. There are many others I would like to thank for all the puzzling, trivia-solving, room-escaping, bar hopping, hiking, biking, and other adventures during my time in Providence, though you all are far too many to name.

My family has always been the foundation of my life. Thanks to Amma and Baba, and my brother Siddharth for being there for anything and everything I have ever needed. To my cousins Anshu, Ayush, Niharika, Neerad, and the rest of my extended family in India and scattered throughout the world, for your love and compassion. None of this would have been possible without you all.

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                                 | <b>1</b>  |
| 1.1      | Modeling a Distributed System . . . . .             | 1         |
| 1.2      | Thesis at a Glance . . . . .                        | 2         |
| 1.2.1    | Loop Agreement . . . . .                            | 3         |
| 1.2.2    | The Convergence Algorithm . . . . .                 | 4         |
| 1.2.3    | $t$ -Resilient Asynchronous Computability . . . . . | 4         |
| 1.2.4    | Computability against Adversaries . . . . .         | 5         |
| 1.3      | Related Work . . . . .                              | 6         |
| <b>2</b> | <b>Operational Model</b>                            | <b>8</b>  |
| 2.1      | Processes and Communication . . . . .               | 8         |
| 2.2      | Read-Write Memory and Snapshots . . . . .           | 9         |
| 2.2.1    | Immediate Snapshots . . . . .                       | 9         |
| 2.3      | Failures and Fault Tolerance . . . . .              | 10        |
| 2.3.1    | Crash Failures . . . . .                            | 10        |
| 2.3.2    | Wait-freedom . . . . .                              | 11        |
| 2.3.3    | $t$ -Resilience . . . . .                           | 12        |
| 2.3.4    | Resilience against Adversaries . . . . .            | 13        |
| <b>3</b> | <b>Mathematical Model</b>                           | <b>15</b> |
| 3.1      | Combinatorial Topology . . . . .                    | 15        |
| 3.1.1    | Basics of Simplicial Complexes . . . . .            | 15        |
| 3.1.2    | Point-set Topology . . . . .                        | 17        |

|          |  |           |
|----------|--|-----------|
| 3.1.3    | Carrier Maps and Subdivisions . . . . .                  | 18        |
| 3.1.4    | Stars, Links and Connectivity . . . . .                  | 20        |
| 3.1.5    | Shellability . . . . .                                   | 22        |
| 3.2      | Classical Topology and Homotopy . . . . .                | 23        |
| 3.2.1    | Homotopy and the Fundamental Group . . . . .             | 23        |
| 3.2.2    | Gluing . . . . .   | 25        |
| 3.2.3    | Simplicial Approximation . . . . .                       | 26        |
| 3.2.4    | The Nerve Lemma . . . . .                                | 26        |
| 3.3      | Distributed Tasks and Protocols . . . . .                | 27        |
| 3.3.1    | Tasks . . . . .  | 27        |
| 3.3.2    | Protocols . . . . .                                      | 28        |
| 3.3.3    | Immediate Snapshot Protocol . . . . .                    | 29        |
| <b>4</b> | <b>Loop Agreement</b>                                    | <b>33</b> |
| 4.1      | The Class of Loop Agreement Tasks . . . . .              | 33        |
| 4.2      | Composing Loop Agreement Tasks . . . . .                 | 35        |
| 4.2.1    | Combining Simplicial Complexes . . . . .                 | 35        |
| 4.2.2    | Implementation by Multiple Tasks . . . . .               | 37        |
| 4.2.3    | Relative Power of Multiple Task Implementation . . . . . | 39        |
| 4.2.4    | Composite Loop Agreement . . . . .                       | 42        |
| 4.3      | A Categorical Interpretation . . . . .                   | 43        |
| 4.3.1    | Category Theory . . . . .                                | 44        |
| 4.3.2    | The Category of Loop Agreement Tasks . . . . .           | 45        |
| 4.4      | The Lattice of Loop Agreement Tasks . . . . .            | 51        |
| 4.5      | Concluding Remarks . . . . .                             | 53        |
| <b>5</b> | <b>The Convergence Algorithm</b>                         | <b>54</b> |
| 5.1      | The Asynchronous Computability Theorem . . . . .         | 55        |
| 5.1.1    | Proof Approaches . . . . .                               | 56        |
| 5.2      | Non-chromatic Simplex Agreement . . . . .                | 57        |
| 5.3      | The Convergence Algorithm . . . . .                      | 59        |
| 5.3.1    | Solving Chromatic Simplex Agreement . . . . .            | 59        |

|          |   |            |
|----------|---|------------|
| 5.3.2    | Bookkeeping . . . . .                                 | 63         |
| 5.3.3    | Link-based Non-chromatic Simplex Agreement . . . . .  | 64         |
| 5.3.4    | Termination and Validity . . . . .                    | 72         |
| 5.4      | Application to General Tasks . . . . .                | 75         |
| 5.5      | Concluding Remarks . . . . .                          | 76         |
| <b>6</b> | <b><i>t</i>-Resilient Asynchronous Computability</b>  | <b>77</b>  |
| 6.1      | Delayed Snapshot Protocol and Task . . . . .          | 78         |
| 6.2      | Connectivity Properties . . . . .                     | 82         |
| 6.2.1    | Shellability of the Protocol Complex . . . . .        | 82         |
| 6.2.2    | Link-connectivity of the Protocol Complex . . . . .   | 86         |
| 6.3      | Single-Round Waiting . . . . .                        | 91         |
| 6.4      | Asynchronous Computability Theorems . . . . .         | 96         |
| 6.5      | Applications of the <i>t</i> -resilient ACT . . . . . | 97         |
| 6.6      | Concluding Remarks . . . . .                          | 99         |
| <b>7</b> | <b>Computability against Adversaries</b>              | <b>100</b> |
| 7.1      | Characterizing Adversaries . . . . .                  | 101        |
| 7.1.1    | Core Complexes . . . . .                              | 101        |
| 7.2      | Adversarial Snapshot Protocol . . . . .               | 103        |
| 7.3      | Impossibility of Single-Round Waiting . . . . .       | 105        |
| 7.3.1    | Two Rounds . . . . .                                  | 108        |
| 7.4      | Concluding Remarks . . . . .                          | 108        |
|          | <b>Bibliography</b>                                   | <b>110</b> |



# Chapter 1

## Introduction

Distributed computing is a field of computer science concerned with computation that takes place in a *distributed system* consisting of a collection of autonomous agents that communicate with one another. These agents may represent processes, threads, computers, nodes, or other computing units, though when reasoning abstractly about a distributed system, it is common to refer to these agents simply as *processes*. Processes interact with one another to solve a distributed coordination problem called a *task*. To solve a task, the processes begin with private inputs, communicate their knowledge with other processes, and decide on outputs based on their shared knowledge. Similar to how classical computability is concerned with Turing decidability, distributed computability is concerned with solvability of tasks under various constraints or assumptions on the model of our distributed system. The focus of this thesis is on computability questions arising in distributed computing.

### 1.1 Modeling a Distributed System

Assumptions on the model of a distributed system can greatly affect solvability of distributed tasks. In a distributed system, it is a fundamental requirement for processes to have the ability to communicate with one another, but this can be accomplished in different ways. Two common models of communication are *message passing*, in which processes may send private messages to one another, and *shared memory*, in which processes read and write a shared state. This thesis is concerned with the latter approach.

Processes may be subject to different *failure* models, which defines the behavior of a process if it fails. The simplest type of process failure is the *crash failure*, where crashed processes halt and fall silent. This work is concerned with crash failures; however, there is also the *Byzantine failure* model, in which failed processes may behave erratically and convey false information to other processes.

While crash failures describe behavior of failed processes, they do not describe which or how many processes may fail. In the most basic model of *fault-tolerance*, the goal is to design *wait-free* algorithms, which tolerate any number of process failures. This is a strict requirement, so weaker models such as  $t$ -resilience (which is tolerance of up to  $t$  failures) and resilience against adversaries are also discussed in this thesis.

There are various *synchrony* models of a distributed system. In a synchronous model, there is a global clock, and process execution occurs in rounds according to this clock. However, for of this work, we assume process execution is entirely *asynchronous*, meaning that there is no bound on the relative execution speed of different processes.

In summary, this thesis is concerned with distributed computability problems in which (1) processes communicate using shared memory, (2) processes fail by crashing, and (3) execution is asynchronous. With these assumptions, we wish to consider protocols under varying models of fault-tolerance. A *protocol* is a distributed algorithm that solves a task, so that a task is solvable if there exists a protocol for it. As one might expect, a more constrained model of fault-tolerance, such as wait-freedom, permits fewer tasks to be solvable in a given model. One main objective of this thesis is to characterize task solvability under models of fault-tolerance less strict than wait-freedom.

## 1.2 Thesis at a Glance

Combinatorial topology has been a useful tool for working with distributed tasks and protocols. These objects are modeled as *simplicial complexes* (or complexes, for short), which are mathematical structures that can be thought of as higher dimensional graphs. As with a graph, a complex is built from a set of vertices, but instead of containing only edges connecting pairs of vertices, there are *simplexes* that may connect sets of vertices. Each vertex represents a process state, while each simplex represents a consistent state of a *set* of processes, or a global state of the whole distributed system. Simplicial complexes allow one to discuss all possible global states of a distributed system, in one compact representation. The input and output configurations of a task are each represented

as a simplicial complex, and the task specification is a map between the two.

This combinatorial model has been applied to prove numerous results in distributed computability, one of the most foundational being the *asynchronous computability theorem*, which characterizes wait-free solvability of tasks in shared memory. Informally, the theorem states that a distributed task with inputs  $\mathcal{I}$ , outputs  $\mathcal{O}$ , and specification  $\Gamma$  is solvable if and only if there exists a map from a *subdivision* of  $\mathcal{I}$  to  $\mathcal{O}$ , in such a way that respects the task specification. The asynchronous computability theorem is a central inspiration to many results presented in this thesis.

### 1.2.1 Loop Agreement

An early motivation for formulating the asynchronous computability theorem was the fact that determining wait-free solvability of a task in shared memory is generally an undecidable problem, in the sense of classical computability. One way to prove undecidability is to exhibit the class of *loop agreement* tasks, as Herlihy and Rajsbaum did, where they characterize their relative power using associated fundamental groups, called *algebraic signatures*. Namely, they show that one loop agreement task implements another if and only if there exists a certain homomorphism between fundamental groups, which is undecidable since the word problem for groups is also undecidable.

In Chapter 4, we explore the class of loop agreement tasks by considering how to characterize their relative power when these tasks are *composed*. This study requires a novel definition of how to compose tasks in parallel. It is shown that the original characterization extends to composite loop agreement tasks in a natural way. More specifically, the relative power of a composition of two loop agreement tasks is determined by the group-theoretic product of their algebraic signatures. In this way, loop agreement tasks can be decomposed into more basic tasks and reasoned about using these “building block” tasks. Proofs of results in this chapter make use of theorems on homotopy from algebraic topology.

We further investigate the class of loop agreement tasks in Section 4.3 by looking at them from a category-theoretic perspective. Category theory is an area of math that is concerned with abstract study of mathematical objects and *morphisms*, or functions, between them. Identifying a class of objects as a category yields all the mathematical machinery that comes with it. In this case, we show that loop agreement tasks form a category, with the composition of task being the product for this category, suggesting that our definition of parallel task composition is the “correct” one. Results here suggest there is more value in exploring more general tasks in the context of category

theory. This work was published in OPODIS [38].

### 1.2.2 The Convergence Algorithm

We return to discussing the asynchronous computability theorem in Chapter 5. To prove the theorem for tasks in which process names are not relevant, one only needs a classic result from algebraic topology called the *simplicial approximation theorem*, which is a tool for turning maps between simplicial complexes into continuous functions. For general tasks where process names matter (and the complexes are *colored*), there are technical difficulties in proving the theorem. Maps between complexes must respect vertex colors, which is not guaranteed by the simplicial approximation theorem.

Herlihy and Shavit approached this issue with an approach based in point-set topology, which used an intricate proof involving technical concepts such as  $\epsilon$ -perturbations and Cauchy sequences. Gafni and Borowsky later proposed an alternative to their approach, one that is more algorithmic in flavor. They reduced the proof of the asynchronous computability theorem to constructing an algorithm to solve a certain distributed task. In particular, this *convergence algorithm* approximates any chromatic subdivision with a more *standard chromatic subdivision*, which corresponds to a well-understood protocol called the *immediate snapshot*. However, their description of the algorithm was incomplete, and no proof of correctness was provide.

In this chapter, we provide a highly detailed description of the convergence algorithm, and offer a complete proof of correctness for the algorithm. We also prove a corollary to the theorem, which offers a way of approximating a continuous function with a color-preserving one, provided that the task under consideration satisfies an important topological condition called *link-connectivity*. This work was published in the *Journal of Applied and Computational Topology* [40].

### 1.2.3 $t$ -Resilient Asynchronous Computability

A distributed algorithm is called  *$t$ -resilient* if it can tolerate up to  $t$  process failures. This requirement is weaker than wait-freedom, which states that the algorithm must tolerate any number of failures; by contrast, in the  $t$ -resilient model there is a bound on the number of failures, so it is considered a more realistic model.

The original asynchronous computability theorem characterizes *wait-free* solvability of distributed tasks in shared memory, so it is reasonable to ask what happens to the theorem’s characterization if the wait-free model of fault-tolerance is weakened to  $t$ -resilience. Indeed, in Chapter 6, we do exactly this by exploring how the combinatorial model is affected. In the original wait-free theorem, immediate snapshots played a central role as a building block for wait-free protocols. Therefore we introduce a new building block, which is a protocol called the *delayed snapshot*, for execution in  $t$ -resilient systems. This snapshot protocol serves the same purpose as the immediate snapshot did for the wait-free asynchronous computability theorem.

There is a fundamental correspondence between immediate snapshots and subdivisions, which implies that read-write protocols never change the basic topology (or shape) of the modeling spaces. However, as we see in this chapter, by weakening wait-freedom to  $t$ -resilience, and in working with the delayed snapshot instead, we see that holes can be torn into the modeling spaces. Introducing holes into a topological space allows for more flexibility of continuously mapping the space, which corresponds to more solvable tasks in a weaker model of fault-tolerance.

Results in this chapter, combined with work on the convergence algorithm, also imply an interesting complexity-theoretic result, which is that any  $t$ -resilient protocol may be implemented in such a way that uses *one* wait barrier, followed by wait-free access to shared memory. Proving this is made possible by the convergence algorithm in the previous chapter. This chapter culminates with two different formulations of asynchronous computability for  $t$ -resilient systems. This work was published in DISC [39].

#### 1.2.4 Computability against Adversaries

The  $t$ -resilient model of fault-tolerance, while more realistic than wait-freedom, still has its own drawbacks. Maximal failure sets of processes are all of the same size, which may not accurately describe real-world systems that have more heterogeneous architectures. We consider a more general model, resilience against adversaries, where an adversary can control whether certain sets of processes fail.

Adversaries have previously been characterized by their *cores* and *survivor sets*. For example, in a system with an adversary, it is always safe for a process to wait for a survivor set of processes to appear before continuing with any computation. We use this characterization to define a snapshot protocol for adversaries. Like the  $t$ -resilient model, Chapter 7 develops a snapshot protocol that

serves as a building block for protocols which are adversary-resilient. In doing so, there is a nice correspondence between the snapshot protocol, and the characterization of adversaries.

This chapter includes an impossibility result for adversaries, in which one round of adversarial snapshot is not always enough to simulate any subsequent number of rounds. Therefore one wait barrier is generally insufficient to implement an adversary-resilient protocol. This is due to the asymmetry of general adversaries, which introduce topological obstructions in model.

### 1.3 Related Work

Herlihy and Shavit [20, 21] introduced the use of algebraic and combinatorial topology to prove impossibility results. Gafni and Koutsoupias [11] were the first to use the fundamental group to show the undecidability of wait-free solvability of certain tasks. Herlihy and Rajsbaum [18, 19] extended the undecidability results to other models, introducing the family of loop agreement tasks and their algebraic signatures. Liu, Xu, and Pan [34] define *n-rendezvous tasks*, where processes begin on distinguished vertices of an embedded  $(n - 1)$ -sphere of an  $n$ -dimensional complex, and converge on a simplex of the embedded sphere. Liu, Pu, and Pan [33] explore a lower-dimensional variant of loop agreement called *degenerate loop agreement*.

The original ACT [25, 24] used combinatorial arguments to construct the color-preserving map required by the theorem, but only characterized wait-free solvability. Borowsky and Gafni [5] proposed the alternative algorithmic approach to constructing this map, but without complete definitions or a proof of correctness. Guerraoui and Kuznetsov [16] compare the two approaches. Gafni *et al.* [15] recently generalized the ACT to encompass infinite executions and other models.

The first application of the ACT was to prove the impossibility of the *k-set agreement* task [8], a result also proved, using other techniques, by Borowsky and Gafni [4], and by Saks and Zaharoglou [37]. These results generalize the classic proofs of the impossibility of consensus due to Fischer *et al.* [10] and Biran *et al.* [3].

Castaneda and Rajsbaum [7] use the ACT to show that the *renaming* task [1] for  $n + 1$  processes has no wait-free read/write protocol with  $2n$  output names when  $n + 1$  is a prime power, but that a protocol does exist when  $n + 1$  is not a prime power. Attiya *et al.* [2] and Kozlov [29] give upper bounds on the running times of such protocols.

Some prior approaches used simulation [6] to reduce certain  $t$ -resilient “colorless” protocols to

wait-free “colorless” protocols. Herlihy and Rajsbaum [23] derived task solvability conditions for *colorless tasks* which, roughly speaking, can be defined independently of process identities. The  $t$ -resilient model is a special case of adversarial shared-memory models [9].

We use the *immediate snapshot* (IS) protocol of Borowsky and Gafni [5], later extended by Raynal and Stainer [36] extended to encompass failure detectors. Kozlov [28] was the first to prove that the standard chromatic subdivision produced by immediate snapshot is, in fact, a subdivision. Gafni *et. al* [15] give a general theorem for task solvability for a class of computational models, but they do not give an explicit characterization of the protocol execution complex for  $t$ -resilient computations. Gafni *et. al.* [12] introduce the class of *affine tasks*, which generalize the two-round  $t$ -resilient task introduced in this work. Additionally, Kuznetsov *et. al.* [31] provide a combinatorial characterization of adversarial task computability by using affine tasks.

Another approach is to reduce the problem of constructing  $t$ -resilient protocols to the problem of constructing wait-free protocols. Gafni and Kuznetsov [13] [14] make progress in this direction, considering a more general failure model that permits irregular failure patterns, but a weaker notion of protocol correctness, called “weak solvability”. They provide a way to transform a task  $T$  to another task  $T'$  such that if  $T$  is weakly solvable in the general model, then  $T'$  is weakly wait-free solvable.

## Chapter 2

# Operational Model

In a distributed coordination task, or just *task* for short, there are multiple processes coordinating with one another via a shared communication medium in order reach outputs compliant with a given task specification. Here, we outline more details of the operational model used for the remainder of the work. In the subsequent chapter we describe how combinatorial topology can be used to reason about the operational model.

### 2.1 Processes and Communication

A distributed system is a set of  $n + 1$  *processes* denoted  $\{p_0, \dots, p_n\}$ , each with a unique identifier or *name*. Recall that these processes are an abstraction, and can represent any sort of computational unit, from cores on a process to individual computers in a cluster. Since we are concerned with computability questions, the implementation details of each unit does not affect the problems we consider in this work. We assume  $n + 1$  processes (rather than  $n$ ) since doing so simplifies notation used in topological reasoning. Unless otherwise specified, it is assumed here that there are always  $n + 1$  processes in the system.

Recall that processes execute and communicate asynchronously, so that there is no bound on the execution time between any processes. Any process may be delayed arbitrarily long before it takes another computational step. Generally speaking, this disallows one process from waiting on any other specific process, though as we will see, some form of waiting is permitted under weak enough failure models.



There are many different models of communications, though in this work, processes communicate with one another using shared memory. In the most basic kind of shared memory model, processes have access to a shared set of memory registers, which store values from some domain. Each process can either *read* a register, which returns the register's contents, or *write* a new value to the register, overwriting the old contents. Multiple processes can concurrently access a single register; however each register is assumed to be *atomic*, so that concurrent read and write operations appear as if they are executed in a sequential order. Atomic registers are used as building blocks for more complex communication protocols.

## 2.2 Read-Write Memory and Snapshots

There are many variations of the asynchronous read-write model, though they are all computationally equivalent and thus are interchangeable when discussing computability. This work makes use of two distinct alternatives. In the *atomic snapshot* model, the  $n+1$  processes have access to an array of  $n+1$  atomic registers. Each process has exactly one designated register to which it can write atomically. Additionally, each process can atomically read the entire array of registers. This operation is called a *snapshot*. Note that this model can be implemented from atomic registers.

### 2.2.1 Immediate Snapshots

It is sometimes convenient to use a more structured variant of the atomic snapshot, known as the *immediate snapshot* model [4, 37]. As with the atomic snapshot model, processes share an array of registers. However in this model, there is only one operation, the *immediate snapshot*, which takes place in two contiguous steps. In the first step, a process writes its view to memory, possibly concurrently with other processes. In the step *immediately* following, it takes a snapshot of memory, also possibly concurrent with other processes. More formally, letting  $S_i$  denote the snapshot observed by process  $p_i$ , the immediate snapshot satisfies the following three conditions:

1. *Self-containment*: For all  $i$ ,  $p_i \in S_i$
2. *Atomicity*: For all  $i, j$ , either  $S_i \subseteq S_j$  or  $S_j \subseteq S_i$
3. *Immediacy*: For all  $i, j$ , if  $p_i \in S_j$ , then  $S_i \subseteq S_j$ .

The first condition states that a process must appear in its own snapshot. The second condition requires the set of snapshots to be linearly ordered, which corresponds to snapshots appearing to take place in a sequential. These first two conditions define the atomic snapshot, and adding the third condition defines the immediate snapshot. The third condition states that if one process appears in another's snapshot, then the entire snapshot of the first process must be contained in the second process's snapshot.

Note that immediate snapshots can be implemented from atomic snapshots. Modern processes implement these models directly, though either model can be simulated by more conventional models, and vice-versa [22, Ch. 14]. The atomic snapshot model is convenient for expressing certain algorithms. The immediate snapshot model has an elegant topological interpretation, making it useful for topological proofs of impossibility or correctness.

The immediate snapshot can be iterated on a sequence of memory arrays  $M_1, \dots, M_N$ . Here, each process executes an immediate snapshot in which it writes its view, and updates its view to be the retrieved snapshot. The process iteratively proceeds to the next immediate snapshot array using its new view. This is called the *iterated immediate snapshot*.

## 2.3 Failures and Fault Tolerance

Imagine a distributed system in which any number of processes may fail, as they may fail in any modern, real-world system. If these processes are running a distributed algorithm, then they may have to account for failed processes during the course of their computation. In other words, the distributed algorithm must be able to *tolerate* process failures.

### 2.3.1 Crash Failures

The most general way in which a process can fail is by exhibiting erratic behavior, where it communicates incorrect information to other processes. This kind of failure is called a *Byzantine* failure, though it is not considered in this work. Instead, we consider a simpler notion of failures, called *crash failures*. When a failed process crashes, it stops its execution and falls silent to other processes, no longer participating in the distributed algorithm. Note that since our model is asynchronous, a crashed process is indistinguishable to other processes from a slow one, since a slow process can take arbitrarily long to respond. A process is *non-faulty* if it executes a given protocol without crashing,

while a process that crashes is called *faulty*. Henceforth, all processes that fail do so by crashing.

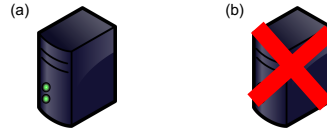


Figure 2.1: In the above, (a) is a non-faulty process, and (b) is a crashed (failed) process.

We have defined the behavior of failed processes, but we have not yet considered *which* or *how many* may processes may fail. This work considers three different models of failure and fault tolerance: wait-freedom,  $t$ -resilience, and resilience against an adversary.

### 2.3.2 Wait-freedom

A distributed algorithm that tolerates an arbitrary number of failures is called *wait-free*. The fault-tolerance guarantee ensuring correctness of a distributed algorithm, even in the presence of arbitrary failures, is called *wait-freedom*. In a correctly implemented wait-free algorithm, no process can wait on any set of other processes, because if one process did wait on another, the second one may crash and the first one would never make progress. This cannot happen in a wait-free algorithm.

In a wait-free system, any combination of processes may fail. For example, if we consider a three-process system, then Figure 2.2 shows all the ways in which processes may fail. The case where all processes fail is excluded, since in this scenario, there is nothing to say since no processes make progress.

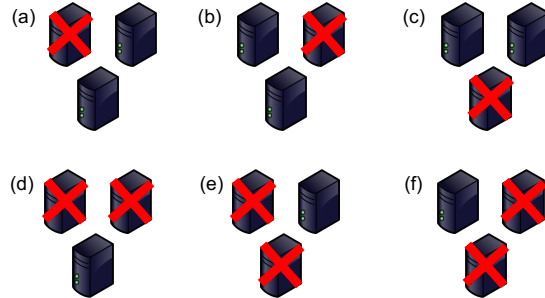


Figure 2.2: All combinations of failed processes in a wait-free system.

The original *asynchronous computability theorem* characterized wait-free read-write solvability of distributed tasks using combinatorial topology. Later in this thesis, we aim to generalize this theorem to weaker guarantees of fault tolerance.

### 2.3.3 $t$ -Resilience

In practice, wait-freedom is a strict property to guarantee of an algorithm. Wait-free algorithms typically require participants to communicate redundant information to one another, in the event that some participants crash and cannot respond. Such a strong guarantee may be necessary in certain real-time systems, though often it may not be required. In many instances wait-freedom can be weakened.

In a real-world system, perhaps it is reasonable to assume that at least a certain percentage of processes are non-faulty. It may be unlikely, say, for 90% of all processes to fail simultaneously.

So there is value in considering weaker conditions than the guarantee of wait-freedom. In a  $t$ -resilient distributed system, no more than  $t$  processes ever fail. Therefore in such a system, it is safe for a process to wait for some set of  $n + 1 - t$  processes (inclusive of itself), since at least this many processes must be non-faulty. A  $t$ -resilient distributed algorithm is one that can tolerate up to  $t$  failures, while a distributed task is solvable  $t$ -resiliently if there is a  $t$ -resilient protocol for it. Using this notation, the wait-free model is the same as the  $n$ -resilient model for a system with  $n + 1$  processes, since as noted before, a wait-free system technically allows all but one process to fail.

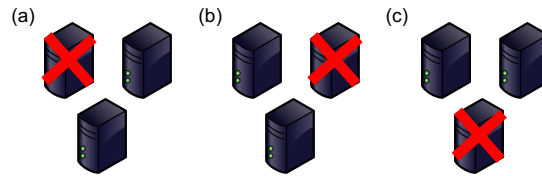


Figure 2.3: All combinations of failed processes in 1-resilient system.

For example, consider a system in which up to  $1/3$  of processes may fail. Then Figure 2.3 shows all ways that processes may fail in such a system with three processes. There are fewer possible ways in which processes may fail, compared to wait-freedom.

### 2.3.4 Resilience against Adversaries

$t$ -resilience is a more realistic model of failure and fault tolerance when compared to wait-freedom, but it is not without its own limitations. Failures in this model are in some sense uniform, since all maximal failure sets having the same size. Furthermore, it does not capture the possibility of certain processes failing together as a group, as may happen if processes share the same resources or network infrastructure.

To address these limitations, one considers a yet more general failure model called *resilience against an adversary*. An *adversary* in a distributed system is an entity capable of failing certain sets of processes. It is modeled by the sets of processes it is capable of failing. For example, if we think of a  $t$ -resilient system as one with an adversary, then this adversary can fail any set of size up to  $t$ . But in general, adversaries may fail maximal sets of different sizes. In this work, the only limitation placed on an adversary is that if can fail some set of processes, then it can fail any subset. This type of adversary is sometimes called *superset-closed*, since the non-faulty sets of such an adversary is closed under taking supersets.

The sets of processes that can be failed by an adversary are called its *failure set*, and completely determine the adversary. Maximal failure sets are inconvenient to work with in practice; instead, it is more common to work with an adversary's *cores* and *survivor sets*. A core is a minimal set of processes that cannot all fail simultaneously, while a survivor set is a minimal set of processes that intersects every core. Cores and survivor sets each completely determine the failure sets of an

adversary.

As an example, consider an adversary in a system with three processes,  $p$ ,  $q$ , and  $r$ , and suppose the adversary can fail any subset of  $\{p, q\}$ , or it can fail  $r$ , but not both. Then its failure sets are shown in Figure 2.4.

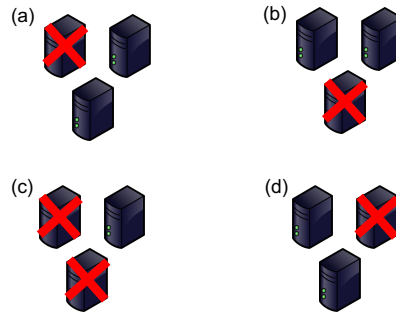


Figure 2.4: Failure sets of an adversary.

Its failure sets are  $\{p\}$ ,  $\{q\}$ ,  $\{p, q\}$ , and  $\{r\}$ . From these, one calculates the cores as  $\{p, r\}$  and  $\{q, r\}$ , and from the cores, the survivor sets  $\{p, q\}$  and  $\{r\}$ .

The maximal failure sets of a  $t$ -resilient adversary are sets of size  $t$ , its cores are sets of size  $t + 1$ , and its survivor sets are those of size  $n + 1 - t$ . Similar to waiting in  $t$ -resilience, it is always safe to wait on a survivor set of an adversary. If we name our adversary  $\mathcal{A}$ , then a distributed algorithm is called  $\mathcal{A}$ -resilient if it can tolerate any failures caused by  $\mathcal{A}$ .

## Chapter 3

# Mathematical Model

The mathematical model for distributed tasks and protocols draws from the basics of combinatorial topology, which is a kind of generalization of graph theory to higher dimensions. In this area of mathematics, the primary object of study is the simplicial complex, which, informally, is a set of vertices with higher dimensional adjacencies beyond (1-dimensional) edges. In this section, we present a primer on combinatorial topology, and describe how these concepts are employed in the mathematical model and how it relates to the operational models in the previous chapter. We also provide the necessary background from classical topology. A complete formal description of the model appears in Herlihy *et al.* [22]. In the chapters to follow, we explain how the model is applied to tackle various computability problems in distributed computing.

### 3.1 Combinatorial Topology

The first subsection establishes definitions from both combinatorial as well as point-set topology. We build higher level abstractions from these definitions.

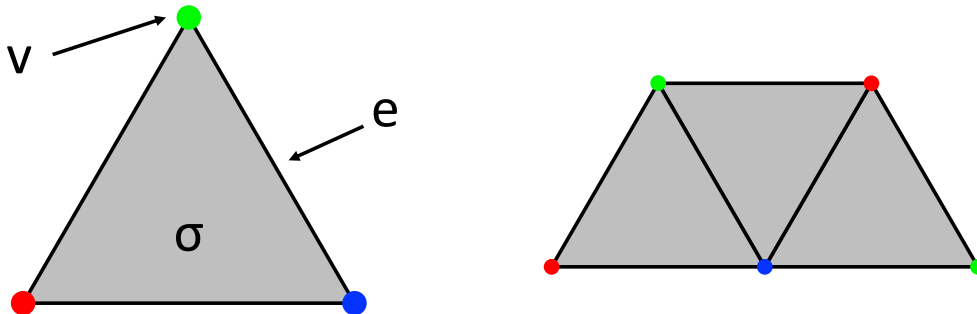
#### 3.1.1 Basics of Simplicial Complexes

A *simplicial complex* (or just *complex*)  $\mathcal{K}$  consists of a finite set  $V$  together with a collection of subsets of  $V$  closed under containment. An element of  $V$  is called a *vertex*, and each set in  $\mathcal{K}$  is called a *simplex*. The *dimension* of  $\sigma$  is defined to be  $\dim(\sigma) = |\sigma| - 1$ . A subset of a simplex is called a *face*. We use “ $k$ -simplex” as shorthand for “ $k$ -dimensional simplex” and similarly for

“ $k$ -face”. If  $\tau \subseteq \sigma$  are simplexes in some complex, then  $\tau$  is a *subsimplex* of  $\sigma$ .

The dimension  $\dim(\mathcal{K})$  of a complex is the maximum dimension of its simplexes. A maximal simplex of  $\mathcal{K}$  (with respect to containment) is called a *facet* of  $\mathcal{K}$ . A complex is *pure* if all its facets have the same dimension. The set of simplexes of  $\mathcal{K}$  having dimension at most  $m$  is called the  *$m$ -skeleton* of  $\mathcal{K}$ , denoted  $\text{skel}^m(\mathcal{K})$ . The *boundary* of a simplex, denoted  $\partial\Delta^k$ , is a subcomplex of  $\Delta^n$  consisting of all its  $(k - 1)$ -faces.

If  $\mathcal{L} \subset \mathcal{K}$  are complex, then the *deletion* of  $\mathcal{L}$  from  $\mathcal{K}$  is the subcomplex of  $\mathcal{K}$  consisting of all simplexes that do not intersect  $\mathcal{L}$ .



(a) A two-dimensional simplex colored by three colors. The vertex  $v$  is a 0-simplex and the edge  $e$  is a 1-simplex. The set  $\sigma$  is the entire 2-simplex.

(b) A two-dimensional simplicial complex with three facets. It is pure and chromatic, colored by three colors.

Figure 3.1: An example simplex and simplicial complex.

A *coloring* of complex  $\mathcal{K}$  is a function  $\ell : V \rightarrow D$ , where  $D$  is some finite set of *colors*. A *properly-colored* simplex is one whose vertices have distinct colors under  $\ell$ , and a *chromatic* complex is one whose simplexes are all properly colored. For complexes  $\mathcal{K}$  and  $\mathcal{L}$ , a *vertex map*  $\phi : \mathcal{K} \rightarrow \mathcal{L}$  carries vertices of  $\mathcal{K}$  to vertices of  $\mathcal{L}$ . If in addition  $\phi$  carries simplexes of  $\mathcal{K}$  to simplexes of  $\mathcal{L}$  then it is called a *simplicial map*. If  $\mathcal{K}$  and  $\mathcal{L}$  are chromatic, then  $\phi$  is *chromatic* if for all vertices  $v \in \mathcal{I}$ ,  $v$  and  $\phi(v)$  are labeled with the same color.

To further discuss important concepts from combinatorial topology, we introduce some basics of point-set topology.



### 3.1.2 Point-set Topology

To discuss subdivisions and connectivity, it is useful to begin with some basic definitions from point-set topology. A topological space is a set  $X$  together with a collection of subsets  $\mathcal{T}$ , whose elements are called *open sets*, satisfying the following conditions:

1.  $\emptyset$  and  $X$  are open,
2. Arbitrary unions of open sets are open,
3. Finite intersections of open sets are open.

The complement of an open set is a *closed set*. The Euclidean space  $\mathbb{R}^n$  is the prototypical example of a topological space, with open sets generated by  $\epsilon$ -balls. Virtually all spaces considered in this thesis are subspaces of some  $\mathbb{R}^n$ . A *continuous function*  $f : X \rightarrow Y$  is a set function such that  $f^{-1}(\mathcal{O})$  is open for any open  $\mathcal{O}$ . With this formal definition in mind, a *homeomorphism* is a continuous function with a continuous inverse. If the domain of a continuous function is closed, then it is a homeomorphism if and only if it is bijective.

Intuitively, homeomorphisms do not fundamentally change the shape of the underlying space. For example, any square and circle are homeomorphic, informally because both are two-dimensional objects and neither have holes. A more topological example is the  $n$ -ball and  $n$ -simplex, or their boundaries: the  $(n - 1)$ -sphere and the complex  $\partial\Delta^n$ .

Roughly speaking, a homeomorphism will never change the dimension of a space it acts on; however, as we will see in the next subsection, there are transformations of spaces that may grow or shrink the dimension, but still preserves important connectivity properties of the space on which it acts.

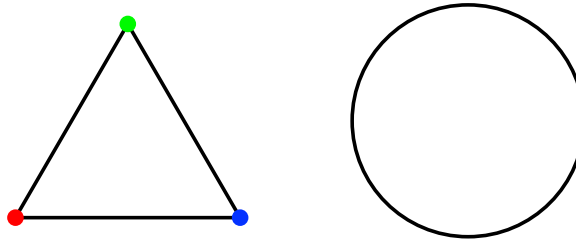


Figure 3.2: The triangle (or  $\partial\Delta^2$ ) and the circle are homeomorphic. The circle can be “straightened out” to make a triangle.

We return to discussing simplicial complexes, but in a more geometric context. Although complexes have been defined in a purely combinatorial way, they can also be realized as topological spaces. Following Munkres [35], a *geometric  $n$ -simplex* is the convex hull of a set of  $n + 1$  affinely independent points in a Euclidean space of appropriate dimension. A *geometric complex* is a collection of geometric simplexes closed under containment such that every pair of distinct simplexes has disjoint interiors. The point-set occupied by a simplex  $\sigma$  or complex  $\mathcal{K}$  is denoted  $|\sigma|$  or  $|\mathcal{K}|$ , and is called its *polyhedron*.

It is common to transition between a simplicial complex and its geometric realization, and in some contexts where appropriate, even identify the two.

### 3.1.3 Carrier Maps and Subdivisions

A *carrier map*  $\Phi : \mathcal{K} \rightarrow 2^{\mathcal{L}}$  takes each simplex  $\sigma \in \mathcal{K}$  to a subcomplex  $\Phi(\sigma) \subseteq \mathcal{L}$  such that for  $\sigma, \tau \in \mathcal{K}$ ,  $\Phi(\sigma \cap \tau) \subseteq \Phi(\sigma) \cap \Phi(\tau)$ . A carrier map is *chromatic* if for every  $n$ -simplex  $\sigma \in \mathcal{K}$ ,  $\Phi(\sigma)$  is chromatic and pure of dimension  $n$ . A simplicial map  $\phi : \mathcal{K} \rightarrow \mathcal{L}$  is *carried by* a carrier map  $\Phi : \mathcal{K} \rightarrow 2^{\mathcal{L}}$  if for every simplex  $\sigma \in \mathcal{K}$ ,  $\phi(\sigma) \subseteq \Phi(\sigma)$ . Let  $\Delta^n$  be a simplex whose vertices are labeled with  $(n + 1)$  distinct colors. If  $\Phi$  is a carrier map, and  $\sigma$  a simplex of  $\Phi(\Delta^n)$ , then the *carrier* of  $\sigma$  in  $\Delta^n$ , denoted  $\text{Car}(\sigma, \Delta^n)$ , is the smallest face  $\tau$  of  $\Delta^n$  such that  $\sigma \in \Phi(\tau)$ .

A *subdivision* of a simplex  $\sigma$  is a complex  $\text{Div}(\sigma)$  such that  $|\text{Div}(\sigma)| = |\sigma|$ . Figure 3.3 illustrates two useful subdivisions: the *barycentric subdivision*  $\text{Bary}(\mathcal{K})$  and the *standard chromatic subdivision*

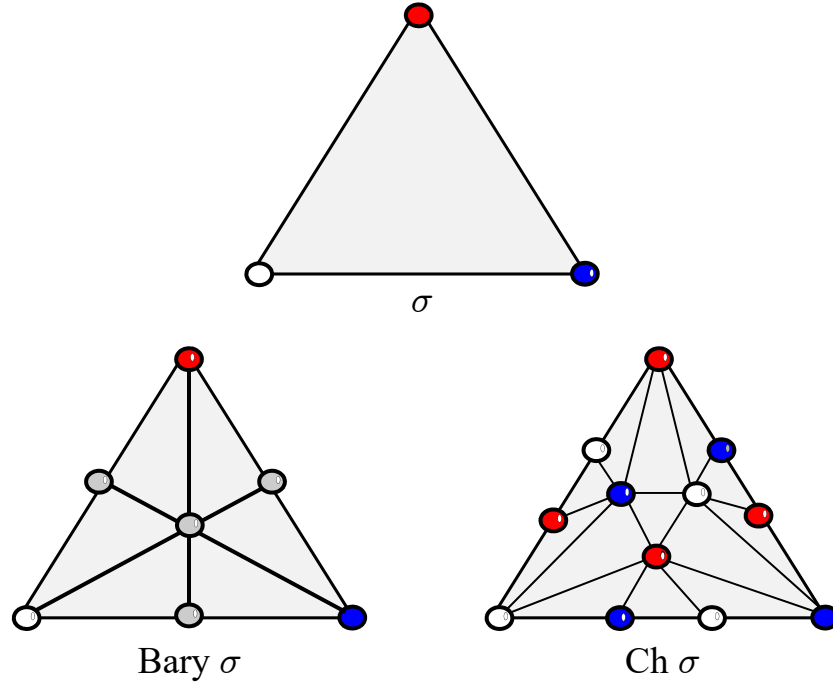


Figure 3.3: Barycentric and standard chromatic subdivisions. The barycentric subdivision is not chromatic, but the standard chromatic subdivision is.

$\text{Ch}(\mathcal{K})$ . A subdivision is a special case of a carrier map; the subdivision operator maps each simplex to its subdivision. A simplicial map  $\phi$  from one subdivision of  $\Delta^n$  to another is *carrier-preserving* if for every simplex  $\sigma$  in the first subdivision,  $\sigma$  and  $\phi(\sigma)$  have the same carriers.

Consider a family of carrier maps  $\Gamma_n$ , with input complex  $\Delta^n$ , and outputs landing in either  $\text{Bary}(\Delta^n)$  or  $\text{Ch}(\Delta^n)$  (depending on chromaticity). Then such a family is called *boundary consistent* if for any  $k$ -simplex  $\sigma$  of  $\Delta^n$ , we have the following

$$\Gamma_n(\Delta^n) \cap \text{Ch}(\sigma) = \Gamma_k(\text{Ch}(\sigma))$$

Respectively for the barycentric subdivision. Note that this is not the standard definition of boundary consistency, but in this thesis it has been modified to generalize to carrier maps.

### 3.1.4 Stars, Links and Connectivity

Stars and links are useful concepts for describing and working with discrete neighborhoods on simplices. Stars play a similar role to the open neighborhoods of topological spaces. We begin with the definition of a star.

**Definition 3.1.1.** *The star of a simplex  $\sigma$  in complex  $\mathcal{K}$ , denoted  $St(\sigma, \mathcal{K})$ , is the subcomplex of  $\mathcal{K}$  consisting of all simplexes  $\tau$  that contain  $\sigma$ , along with all subsimplexes of  $\sigma$ .*

In some circumstances, the *open star* is used in place of the star. The open star of a point is simply the interior of the geometric realization of its star; that is, its boundary is excluded.

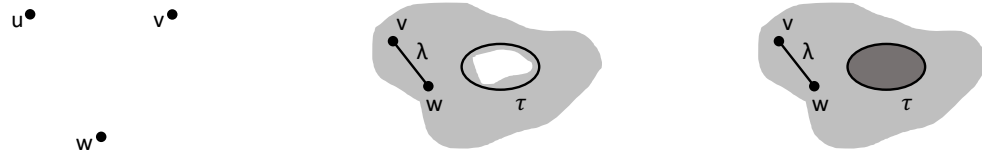
**Definition 3.1.2.** *The link of a simplex  $\sigma$  in complex  $\mathcal{K}$ , denoted  $Lk(\sigma, \mathcal{K})$ , is the subcomplex of  $\mathcal{K}$  consisting of all simplexes  $\tau$  disjoint from  $\sigma$  such that  $\tau \cup \sigma \in \mathcal{K}$ .*

The link is a subcomplex of the star, but one dimension lower. One can think of the link of  $\sigma$  as a simplicial neighborhood that encompasses  $\sigma$  but does not intersect with it. Or alternatively, can be thought of as the boundary of the star. See Figure 3.5 for examples of links in simplicial complexes.

The link is used to define a combinatorial notion of connectivity, called link-connectivity. However before we introduce link-connectivity, we require the standard definition of connectivity from algebraic topology.

We write  $S^k$  to denote the  $k$ -dimensional sphere. Then a topological space  $\mathcal{K}$  is  *$k$ -connected* if, for all  $0 \leq m \leq k$ , any continuous map  $f : S^m \rightarrow X$  can be extended to a continuous  $F : D^{m+1} \rightarrow X$ , where the sphere  $S^m$  is the boundary of the disk  $D^{m+1}$ . This definition also applies to simplicial complexes, where one considers connectivity of the geometric realization.

One way to think about  $k$ -connectivity is that any map  $f$  of the  $k$ -sphere that cannot be “filled in” represents a  $k$ -dimensional “hole” in the complex. Though not standard mathematical terminology, we refer to  $k$ -connectivity as topological connectivity (when  $k$  is understood), in order to disambiguate from the link-connectivity property defined next.



- (a) A  $(-1)$ -connected space. It is non-empty, but paths do not exist between points.
- (b) A  $0$ -connected space. Paths exist between any two points, but the oval cannot be filled in.
- (c) A  $1$ -connected space. Paths exist between points, and any ovals can be filled in.

Figure 3.4: Examples of  $(-1)$ -connected,  $0$ -connected, and  $1$ -connected spaces

See Figure 3.4 for examples illustrating topological connectivity. In the first diagram, there is no path between any of the three vertices, so the set is disconnected (also called  $(-1)$ -connected by convention, which only requires non-emptiness). The second diagram is path-connected, or  $0$ -connected, since one can find a path between any two points. However it has a “hole” where a closed path  $S^1$  encircling it cannot be extended to  $D^2$ . In the third diagram, we have an example of a simply-connected space, or  $1$ -connected, where also embeddings of the sphere of  $S^1$  can also be extended.

We can now define link-connectivity for pure simplicial complexes, in terms of topological connectivity.

**Definition 3.1.3.** *A pure  $n$ -dimensional complex  $\mathcal{K}$  is link-connected if for all  $\sigma \in \mathcal{K}$ ,  $\text{Lk}(\sigma, \mathcal{K})$  is  $(n - \dim(\sigma) - 2)$ -connected.*

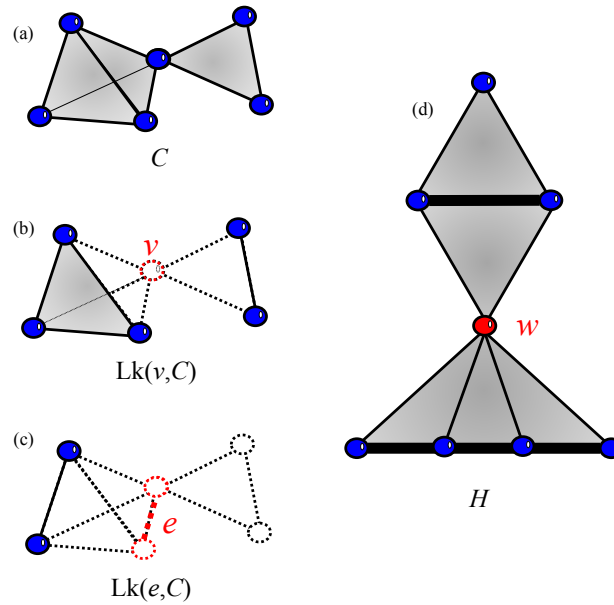


Figure 3.5: Links in a simplicial complex. The complex on the righthand side is not link-connected, due to the problematic red vertex and its insufficiently connected link.

Link-connectivity imposes a topological connectivity requirement on the link of each simplex of a given complex, dependent on the simplex's dimension. Informally, link-connectivity ensures that a complex cannot be “pinched” too thinly at any given simplex. It is known that all subdivided simplexes are link-connected. Figure 3.5 (d) shows an example of a simplicial complex that is not link-connected.

### 3.1.5 Shellability

It is a common theme in mathematics to understand an object by decomposing it into its parts. Here we consider one particular way of decomposing simplicial complexes. A pure simplicial complex is called shellable if it can be assembled, in a nice way, from its facets in a one-at-a-time order.

**Definition 3.1.4.** *Let  $C$  be a pure simplicial complex of dimension  $n$ , and let  $\psi_0, \dots, \psi_s$  be an enumeration of its facets. Then  $\{\psi_i\}$  is called a shelling order if for all  $j$ , the complex  $\bigcup_{i=0}^j \psi_i \cap \psi_{j+1}$  is pure of dimension  $n - 1$ . A complex with a shelling order is said to be shellable. The complex obtained by assembling a prefix of the shelling is called an intermediate complex.*

See Figure 3.6 for a simple example of a shelling order on a simplicial complex. In later chapters,

we demonstrate how shellability can be used to not only assemble a simplicial complex from its facets, but also from suitably defined subcomplexes, in a manner that yields desirable connectivity properties.

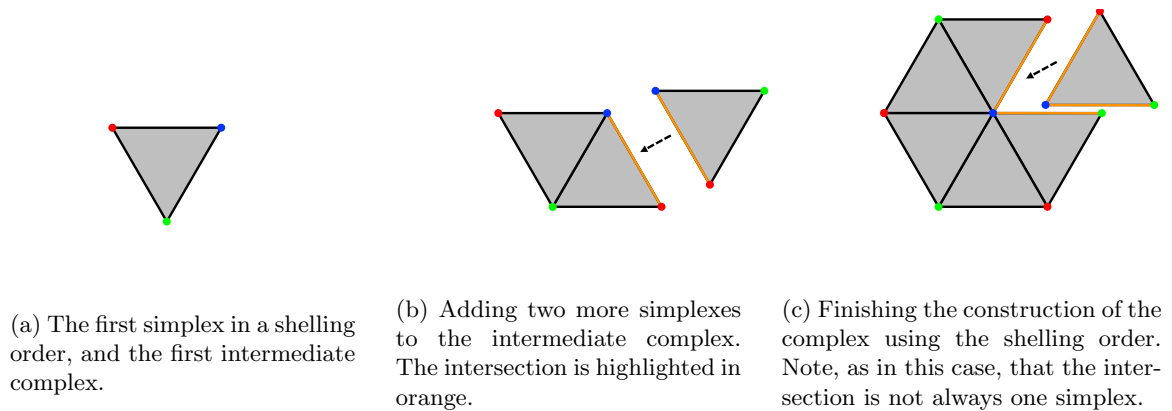


Figure 3.6: Shelling of a hexagonal simplicial complex.

## 3.2 Classical Topology and Homotopy

### 3.2.1 Homotopy and the Fundamental Group

Given a topological space  $X$  and a base point  $x_0 \in X$ , a *loop* in  $X$  based at  $x_0$  is a continuous function  $\lambda : [0, 1] \rightarrow X$  such that  $\lambda(0) = \lambda(1) = x_0$ . Two loops  $\lambda_1$  and  $\lambda_2$  based at  $x_0$  are (loop) *homotopic* if one loop can be continuously deformed to the other. More precisely,  $\lambda_1$  and  $\lambda_2$  are homotopic if there is a continuous function  $H : [0, 1] \times [0, 1] \rightarrow X$  such that  $H(0, -) = \lambda_1$ ,  $H(1, -) = \lambda_2$ , and  $H(-, 0) = H(-, 1) = x_0$ . Homotopy is an equivalence relation. We write  $[\lambda]$  to denote the equivalence class of all loops homotopic to  $\lambda$ .

Let  $\alpha : [0, 1] \rightarrow X$  and  $\beta : [0, 1] \rightarrow X$  be two loops based at  $x_0$ . Then we can *concatenate*  $\alpha$  and  $\beta$  to get another loop,  $\alpha \cdot \beta$ , defined by traversing  $\alpha$ , returning to  $x_0$ , and then traversing  $\beta$ . The loop  $\alpha \cdot \beta : [0, 1] \rightarrow X$ , also based at  $x_0$ , is defined as

$$(\alpha \cdot \beta)(t) = \begin{cases} \alpha(2t) & \text{for } 0 \leq t \leq \frac{1}{2} \\ \beta(2t - 1) & \text{for } \frac{1}{2} \leq t \leq 1 \end{cases}$$

Concatenation behaves well with homotopy. If  $\alpha$  and  $\beta$  are homotopic to  $\alpha'$  and  $\beta'$ , respectively,

then  $[\alpha \cdot \beta] = [\alpha' \cdot \beta']$ . From this it follows that concatenation is associative on classes of loops based at  $x_0$ . In fact, concatenation is a group operation on classes of loops based at  $x_0$ , with the inverse computed by traversing a loop in the opposite direction, and the identity element being the class of all loops homotopic to the constant loop at  $x_0$ . Formally, the inverse of  $[\alpha]$  is the class of the loop  $\alpha^{-1}(t) = \alpha(1 - t)$ , and the class  $[e]$  of loop  $e(t) = x_0$  serves as the identity.

**Definition 3.2.1.** *Let  $X$  be a topological space, and let  $x_0 \in X$  be a base point. Then the fundamental group of  $X$  at  $x_0$ , denoted  $\pi_1(X, x_0)$ , is the set of all loop homotopy classes with concatenation as its group operation. If  $X$  is path-connected, then  $\pi_1(X, x_0)$  is independent of  $x_0$ , and we simply write  $\pi_1(X)$ .*

*If  $f : (X, x_0) \rightarrow (Y, y_0)$  is a base point-preserving continuous function, then  $\pi_1$  also induces a group homomorphism  $f_* : \pi_1(X, x_0) \rightarrow \pi_1(Y, y_0)$  called the induced homomorphism, defined by  $f_*([\lambda]) = [f \circ \lambda]$ .*

An important property of the fundamental group is how it behaves with the product of topological spaces.

**Fact 3.2.2.** *Let  $X$  and  $Y$  be topological spaces. Then  $\pi_1(X \times Y) \cong \pi_1(X) \times \pi_1(Y)$ .*

It turns out that the fundamental group of a space is trivial if and only if that space is simply-connected, or 1-connected. In fact, the fundamental group can be generalized to higher kinds of connectivity; these generalizations are called the *higher homotopy groups*, and are denoted  $\pi_n(X)$  for each  $n$ . Exploration of these groups is beyond the scope of this thesis, but there is one particular fact about homotopy groups that proves useful. Namely, that the  $n$ -th higher homotopy group of a space is trivial if and only if the space is  $n$ -connected.

Homotopy is defined for loops with based points, but it has a more general definition. We define this, along with deformation retractions, a specific kind of homotopy.

**Definition 3.2.3.** *Two continuous functions  $f, g : X \rightarrow Y$  are homotopic if there is a continuous  $H : X \times [0, 1] \rightarrow Y$  such that  $H(-, 0) = f$  and  $H(-, 1) = g$ . In this case, we write  $f \simeq g$  if this is the case. If in addition  $X \subseteq Y$  and  $H$  fixes  $X$ , then  $H$  is called a deformation retraction and we say  $Y$  deformation retracts onto  $X$ .*

If  $\delta$  is a simplicial approximation of a continuous function  $h$ , then it is known that  $|\delta| \simeq h$ . Using homotopy, we can define an equivalence between topological spaces called homotopy equivalence.



This is an equivalence weaker than homeomorphism, though homotopy equivalence still preserves useful properties.

**Definition 3.2.4.** *Let  $X$  and  $Y$  be topological spaces. Then  $X$  and  $Y$  are homotopy equivalent, or  $X \simeq Y$ , if there are continuous functions  $f : X \rightarrow Y$  and  $g : Y \rightarrow X$  such that  $g \circ f \simeq id_X$  and  $f \circ g \simeq id_Y$ . The maps  $f$  and  $g$  are called homotopy equivalences and are homotopy inverses of one another.*

Homeomorphic spaces are clearly homotopy equivalent. In addition, the higher homotopy groups are invariant under homotopy equivalences. This has the implication that one may freely apply homotopy equivalences to a space, without changes whether or not the space is  $n$ -connected. We state this as a fact, which can be found in [17].

**Fact 3.2.5.** *Let  $X$  and  $Y$  be topological spaces. If  $X \simeq Y$ , then  $\pi_n(X) \cong \pi_n(Y)$ .*

The next two facts are specifically about the interaction between simplicial complexes and homotopy. We call a continuous function  $g : |\mathcal{A}| \rightarrow |\mathcal{B}|$  *cellular* if  $g$  maps skeletons to skeletons, or more precisely, if  $g(|\text{skel}^n(\mathcal{A})|) \subseteq |\text{skel}^n(\mathcal{B})|$  for every  $n$ . Then every continuous  $f : |\mathcal{A}| \rightarrow |\mathcal{B}|$  is homotopic to such a map  $g$ , as seen below.

**Fact 3.2.6** (Cellular Approximation). *Let  $f : |\mathcal{A}| \rightarrow |\mathcal{B}|$  be a continuous function between simplicial complexes  $\mathcal{A}$  and  $\mathcal{B}$ . Then  $f$  is homotopic to a cellular function  $g : |\mathcal{A}| \rightarrow |\mathcal{B}|$ . Furthermore, if  $\mathcal{C} \subseteq \mathcal{A}$  is a subcomplex such that  $f$  is already cellular on  $|\mathcal{C}|$ , then we may require the homotopy between  $f$  and  $g$  to fix  $|\mathcal{C}|$ .*

Now suppose we have a homotopy on a subcomplex and we want to extend it to the entire simplicial complex. The next fact, also found in Hatcher [17], allows us to do this.

**Fact 3.2.7** (Homotopy Extension). *Let  $\mathcal{C} \subseteq \mathcal{A}$  and  $\mathcal{B}$  be simplicial complexes, and let  $F : |\mathcal{A}| \rightarrow |\mathcal{B}|$  be a continuous function. Suppose we have a homotopy  $H : |\mathcal{C}| \times [0, 1] \rightarrow |\mathcal{B}|$  such that  $H(-, 0) = F|_{|\mathcal{C}|}$ . Then there is a homotopy extending  $H$  to all of  $|\mathcal{A}|$ , respecting  $F$ . That is, we can find homotopy  $H' : |\mathcal{A}| \times [0, 1] \rightarrow |\mathcal{B}|$  such that  $H'|_{|\mathcal{C}| \times [0, 1]} = H$  and  $H'(-, 0) = F$ .*

### 3.2.2 Gluing

The pasting lemma, a classic result taken from point-set topology, is a for constructing piecewise maps. It generally applies to open and closed sets in topological spaces, though in this work, we

often apply it to construct a continuous map by defining a map on all facets of a complex, then gluing all maps together.

**Lemma 3.2.8.** *Let  $X_1$  and  $X_2$  be open or closed sets of a common topological space, with continuous  $f_i : X_i \rightarrow Y$  that coincide on the intersection  $X_1 \cap X_2$ . Then the function  $f : X_1 \cup X_2 \rightarrow Y$ , defined in terms of the  $f_i$ , is a continuous function.*

### 3.2.3 Simplicial Approximation

We will also need the following version of the *simplicial approximation theorem*, which is originally a result from classical algebraic topology. It allows one to take a continuous function between simplicial complexes, and turn it into a simplicial one with similar properties. We define a simplicial approximation.

**Definition 3.2.9.** *Let  $\mathcal{B}$  and  $\mathcal{C}$  be abstract simplicial complexes, let  $f : |\mathcal{B}| \rightarrow |\mathcal{C}|$  be a continuous map, and let  $\varphi : \mathcal{B} \rightarrow \mathcal{C}$  be a simplicial map. The map  $\varphi$  is called a simplicial approximation to  $f$ , if for every simplex  $\alpha$  in  $\mathcal{B}$  we have*

$$f(\text{Int } |\alpha|) \subseteq \bigcap_{a \in \alpha} \text{St}^\circ(\varphi(a)) = \text{St}^\circ(\varphi(\alpha)), \quad (3.2.1)$$

where  $\text{Int } |\alpha|$  denotes the interior of  $|\alpha|$ , and  $\text{St}^\circ(\varphi(a))$  is the open star.

**Theorem 3.2.10.** *Let  $\mathcal{B}$  and  $\mathcal{C}$  be simplicial complexes. Given a continuous map  $f : |\mathcal{B}| \rightarrow |\mathcal{C}|$ , there is an  $N > 0$  such that  $f$  has a simplicial approximation  $\varphi : \text{Ch}^N(\mathcal{B}) \rightarrow \mathcal{C}$ .*

This theorem remains true if we replace Ch with Bary.

### 3.2.4 The Nerve Lemma

We end this section with yet another classical result from algebraic topology, called the nerve lemma. This topic is covered both by Hatcher as well as Kozlov [27] [17].

**Definition 3.2.11.** *Let  $\mathcal{K}$  be a simplicial complex and let  $\{K_i\}_{i \in I}$  be a collection of subcomplexes covering  $\mathcal{K}$ , which is to say that  $\bigcup_{i \in I} K_i = \mathcal{K}$ . Then the nerve complex  $\mathbb{N}(\{K_i\}_{i \in I})$  is a simplicial complex whose vertices are the  $K_i$  and whose simplexes are collections  $\{K_j\}_{j \in J}$  such that  $\bigcap_{j \in J} K_j$  is nonempty.*

The nerve complex of a simplicial cover encodes how the components of the cover intersect with one another. The complex obtained in this way sometimes shares connectivity properties with the original complex, as described by the nerve lemma below. We can therefore use the nerve complex to reason about connectivity properties of the original complex.

**Lemma 3.2.12** (Nerve Lemma). *Let  $\{K_i\}_{i \in I}$  be a simplicial cover of  $\mathcal{K}$ , and let  $k$  be some fixed integer. For any nonempty  $J \subseteq I$ , define  $\mathcal{K}_J = \bigcap_{j \in J} \mathcal{K}_j$ , and suppose that  $\mathcal{K}_J$  is  $(k - |J| + 1)$ -connected or empty for all such  $J$ . Then  $\mathcal{K}$  is  $k$ -connected if and only if  $\mathcal{N}(\{K_i\}_{i \in I})$  is  $k$ -connected.*

### 3.3 Distributed Tasks and Protocols

Simplicial complexes are used to model two main objects of concern: tasks and protocols. Tasks represent distributed coordination problems to be solved by a distributed system, while protocols are the distributed algorithms that solve these tasks.

#### 3.3.1 Tasks

Formally, a *task* is a triple  $(\mathcal{I}, \mathcal{O}, \Gamma)$ , where  $\mathcal{I}$  and  $\mathcal{O}$  are the task's *input* and *output* complexes, and  $\Gamma : \mathcal{I} \rightarrow 2^{\mathcal{O}}$  is a carrier map. An initial configuration where each process  $p_i$  is assigned input value  $v_i$  is represented as an  $n$ -simplex  $\sigma = (s_0, \dots, s_n) \in \mathcal{I}$ , where each vertex  $s_i$  is labeled with  $v_i$ . Similarly, a legal final configuration where each process  $p_i$  halts with output value  $w_i$  is represented as  $n$ -simplex  $\tau = (t_0, \dots, t_n) \in \mathcal{O}$ , where each vertex  $t_i$  is labeled with  $w_i$ . For each  $\sigma \in \mathcal{I}$ ,  $\Gamma(\sigma) \subseteq \mathcal{O}$  is the set of legal final configurations when the processes that appear in  $\sigma$  participate in the task. This carrier map is called the task's *specification*.

Vertices of task's complexes are labeled process states, but they are also colored by process names. Since each process is assumed to have a unique name  $p_i$ , such a labeling would result in chromatic input and output complexes. A task is called *colored* if we label processes with their names, with the additional requirement that  $\Gamma$  is chromatic. Otherwise, the task is called *colorless*. Intuitively, in a colorless task, any process is allowed to behave like any other process by adopting the other's input value, since process names are irrelevant.

An example of a colored task is *consensus*, where participating processes each begin with some input and must decide on a common output. The task also requires the processes to agree on some process's input. That is, their decision must satisfy two conditions:

1. *Agreement*: all processes decide on the same value
2. *Validity*: this common decision value was some process's input.

A simplified version of this is 2-process binary consensus, where there are only two processes, and each process may start with either 0 or 1. Then in this case, it is relatively straightforward to illustrate the input and output complexes, as well as the task specification, as in Figure 3.7.

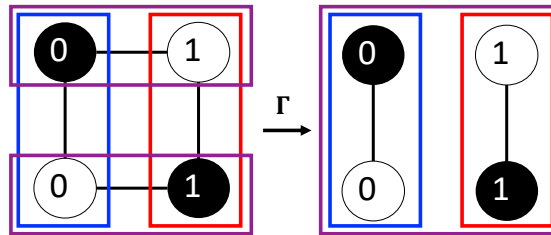


Figure 3.7: The input complex, output complex, and task specification of 2-process binary consensus.

In the figure, vertices are colored by process name (in this case, white and black), and labeled by input 0 or 1. The colored boxes depict how the task specification  $\Gamma$  maps inputs to outputs. When the process inputs are the same, as in the red and blue boxes, then the processes must decide on that value itself, so in the case the image of the task specification is only one simplex. However, when there are mixed inputs, as in the purple boxes, then the processes can either both decide 0 or both decide 1. In this case, the specification maps each mixed-input edge to the entire output complex (both edge), since either choice of decision value is valid.

### 3.3.2 Protocols

A *protocol* is also a triple  $(\mathcal{I}, \mathcal{P}, \Xi)$ , where  $\mathcal{I}$  and  $\mathcal{P}$  are the protocol's *input* and *protocol* complexes, and  $\Xi : \mathcal{I} \rightarrow 2^{\mathcal{P}}$  is a carrier map called the *execution* map. The input complex is constructed in the same way as for tasks. Vertices of the protocol complex represent uninterpreted process states. Similar to a task's specification, the execution map defines which set of process states can result from a given input configuration. As before, if the input complex is chromatic, then all other complexes and maps are required to be chromatic. A protocol is *colored* if its input complex is taken to be chromatic, and *colorless* otherwise.

A protocol  $(\mathcal{I}, \mathcal{P}, \Xi)$  *wait-free solves* a task  $(\mathcal{I}, \mathcal{O}, \Gamma)$  if there exists a color-preserving simplicial

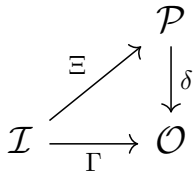
map  $\delta : \mathcal{P} \rightarrow \mathcal{O}$ , called a *decision map*, such that

$$\delta(\Xi(\sigma)) \subseteq \Gamma(\sigma) \quad (3.3.1)$$

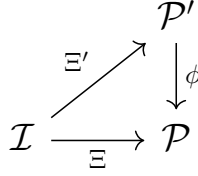
for each  $\sigma \in \mathcal{I}$ . For weaker models of fault tolerance, 3.3.1 is only required to hold for  $\sigma$  corresponding to valid executions. For instance, a task is solvable *t-resiliently* if equation 3.3.1 holds for all  $\sigma$  such that  $|\sigma| \geq n + 1 - t$ . Similarly, for adversaries, 3.3.1 need only hold for  $\sigma$  containing a survivor set. This definition of solvability clear implies that a task solvable in a stricter model of fault-tolerance is also solvable in a weaker one.

In Section 3.3.3, we formally model the immediate snapshot as a kind of protocol defined here.

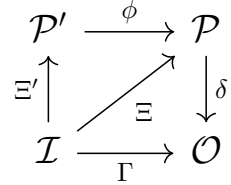
Protocols can simulate other protocols. A protocol  $(\mathcal{I}, \mathcal{P}, \Xi)$  *simulates* another  $(\mathcal{I}, \mathcal{P}', \Xi')$  if there exists a simplicial map  $\phi : \mathcal{P}' \rightarrow \mathcal{P}$ , called a *simulation*, such that  $(\phi \circ \Xi)(\sigma) \subseteq \Xi'(\sigma)$ . The operational intuition is that  $\phi$  sends simulated process states to real states.



(a) A protocol solving a task.



(b) A protocol simulating another protocol.



(c) A simulated protocol solving a task.

Figure 3.8: Protocols solve tasks, and protocols can simulate other protocols. The power set notation for carrier maps' range (e.g.  $2^{\mathcal{O}}$ ) is omitted above to more succinctly convey the intuition.

Figure 3.8c shows how a simulated protocol can be used to solve a task, by combining the notations of simulation and solvability.

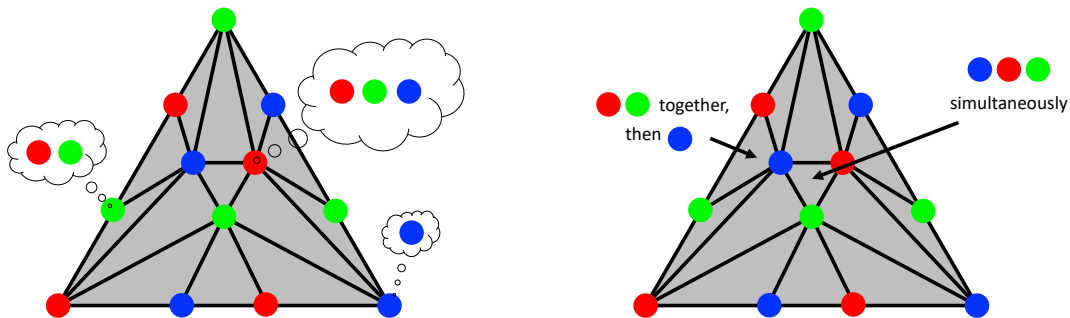
### 3.3.3 Immediate Snapshot Protocol

Here, we give a concrete example of a wait-free protocol which is used throughout the rest of this thesis. This is the immediate snapshot protocol introduced in the previous chapter.

There are multiple equivalent ways of modeling read-write memory, with the most primitive building block being the atomic register. Atomic registers can simulate atomic snapshots, where any single process is capable of reading a shared array in one atomic step. Atomic snapshots allow one to reason about protocols in terms of the number of rounds each process must run, though the

corresponding topological structure of this protocol can be quite complex [30].

The immediate snapshot, described in the previous chapter, further simplifies the model used to reason about read-write memory. In fact, it may be modeled formally as a protocol, in which processes begin with their names as inputs, write their names to the shared memory, and take a snapshot. The input complex for  $n + 1$  processes is then just the standard simplex  $\Delta^n$ , colored by process names. The output complex turns out to be the standard chromatic subdivision of  $\Delta^n$ , which is denoted  $\text{Ch}(\Delta^n)$ .



(a) Each vertex represents a process state, which in this case is the snapshot read by the process. (b) Each simplex in the complex represents one possible interleaving of the three processes.

Figure 3.9: The immediate snapshot protocol complex for three processes, with example states and configurations.

The execution map of this protocol is given by the subdivision operator  $\text{Ch}$  itself, since each simplex in  $\Delta^n$  is indeed mapped to its subdivision, which is a subcomplex of  $\text{Ch}(\Delta^n)$ . Hence the protocol is given by the triple  $(\Delta^n, \text{Ch}(\Delta^n), \text{Ch})$ .

See Figure 3.9 illustrating processes states and global configurations in the immediate snapshot protocol with three processes. Figure 3.9a shows examples of process state for three different processes during three different execution.

- The highlighted blue vertex is the state of the blue process during an execution where it runs solo, or first.
- The highlighted green vertex is a state where the green process observes both itself and the red process in its immediate snapshot, so either the red process executed solo (the bottom-right

red vertex), or the red process ran concurrently with the green one.

- The highlighted red vertex is a state where the red process sees all three processes in its snapshot. In this case, as far as the red process knows, the either two may have run concurrently with itself, or may have preceded the red process.

Figure 3.9b shows two examples of final process configurations after running the immediate snapshot protocol, which are simplexes. In the central simplex, all three processes execute concurrently. In the other simplex highlighted, the red and green processes execute concurrently, then the blue process executes afterwards.

For colorless tasks, the *colorless* immediate snapshot protocol is used, which is a minor variation of the immediate snapshot. As with the ordinary snapshot, the colorless immediate snapshot operates on an array of clean memory. But since in a colorless task, process names are not relevant and only the inputs are, the process only stores its input before taking a snapshot. So a snapshot only consists of a set of states, without reference to names. The colorless snapshot's protocol complex is the barycentric subdivision.

Recall that the immediate snapshot protocol can be iterated on a sequence of clean arrays of memory. Doing so corresponds to repeatedly applying the standard chromatic subdivision operator. For instance, the second standard chromatic subdivision of  $\Delta^2$  is depicted in Figure 3.10 (colors are omitted).

Due to the correspondence between immediate snapshot protocols and the iterated standard chromatic subdivision, these two are useful tools in studying computability results of distributed tasks. Both are used extensively throughout this thesis.

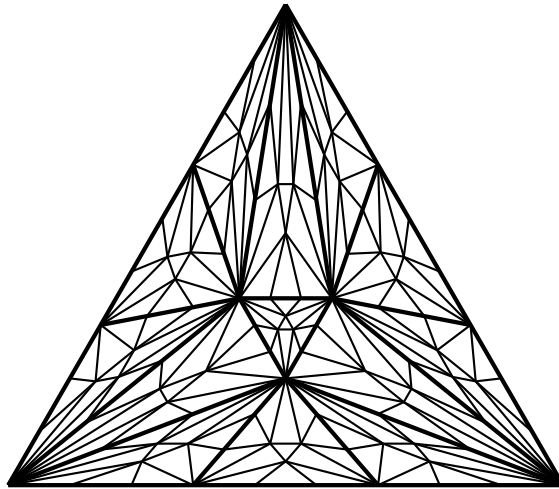


Figure 3.10: The second standard chromatic subdivision of  $\Delta^2$ , which is the protocol complex for two rounds of immediate snapshots.



## Chapter 4

# Loop Agreement

In this chapter, we discuss a simple class of colorless tasks called loop agreement, originally introduced to prove undecidability of solvability of tasks in read-write memory. Each loop agreement task models convergence of processes on some given simplicial surface. Processes begin at points in a designated *loop*, and they must meet at (decide on) a simplex of the surface. Herlihy and Rajsbaum characterized these tasks by their *algebraic signatures*, each consisting of a group and a distinguished element.

We extend their work by defining the *composition* of multiple loop agreement tasks to create a new one with the same combined power, and generalize their algebraic signature construct to these composite tasks [38]. In this way, one can think of loop agreement tasks in terms of their basic building blocks. We also explored a category-theoretic perspective of loop agreement by defining a category of tasks, showing that the algebraic signature is a functor, and proving that task composition is the categorical product. This section uses more advanced techniques and definitions from algebraic topology, which are outlined in the first subsection of the appendix.

### 4.1 The Class of Loop Agreement Tasks

We begin with the definitions of edge paths and triangle loops, which are required to formally define the designated loop on a loop agreement task's surface. An edge path is essentially a sequence of adjacent edges, while a triangle loop is a set of three mutually adjacent edge paths.

**Definition 4.1.1.** *An edge path in a complex  $\mathcal{C}$  is an alternating sequence of vertices and edges,*

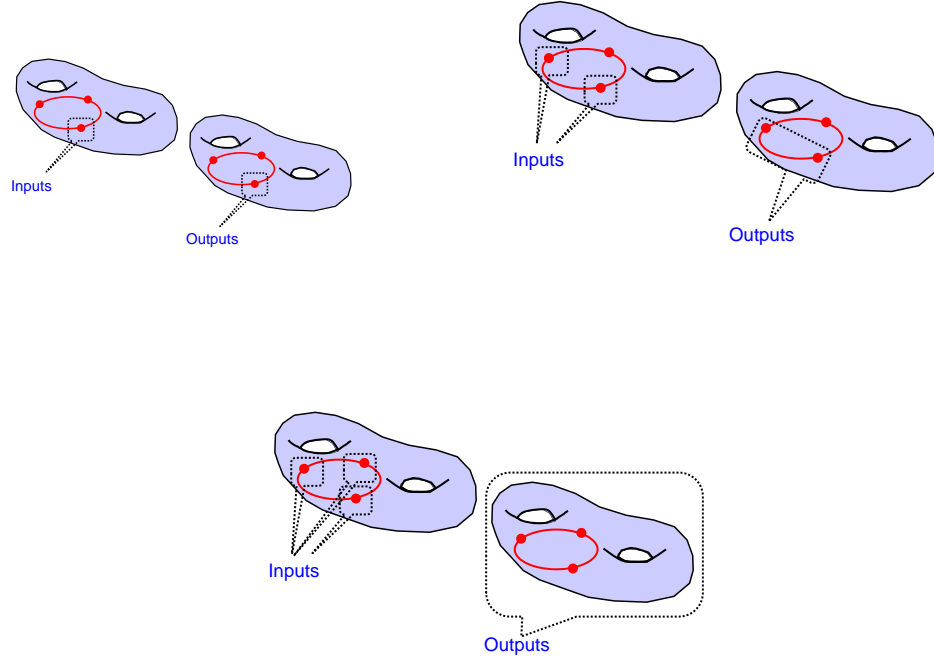


Figure 4.1: Loop agreement. If there is one input vertex, the processes stay put. If there are two, processes must converge on the path between these two vertices. If all three possible inputs appear, processes are free to meet at any simplex.

$v_1, e_1, v_2, e_2, \dots, v_{k-1}, e_{k-1}, v_k$ , where  $e_i = \{v_i, v_{i+1}\}$ . An edge loop is an edge path with  $v_0 = v_k$ . A triangle loop in  $\mathcal{C}$  is a six-tuple  $\lambda = (v_0, v_1, v_2, p_{01}, p_{12}, p_{20})$  such that each  $v_i$  is a vertex in  $\mathcal{C}$  and  $p_{ij}$  is an edge path between  $v_i$  and  $v_j$ .

Triangle loops are indeed loops in the topological sense (that is, an embedding of  $S^1$ ), but they are also subcomplexes with designated vertices and edge paths. We may now define the class of loop agreement tasks.

**Definition 4.1.2.** A loop agreement task is a task  $(\mathcal{I}, \mathcal{O}, \Gamma)$  for which  $\mathcal{I} = \Delta^2$ ,  $\mathcal{O}$  is a connected 2-dimensional complex with triangle loop  $\lambda = (v_0, v_1, v_2, p_{01}, p_{12}, p_{20})$ , and  $\Gamma$  is defined as:

$$\Gamma(\sigma) = \begin{cases} \{v_i\} & : \sigma = \{i\} \\ p_{ij} & : \sigma = \{i, j\} \\ \mathcal{O} & : \sigma = \{0, 1, 2\} \end{cases}$$

We write  $\text{Loop}(\mathcal{O}, \lambda)$  to denote this task. Input vertices are carried to the designated vertices of  $\lambda$ , the input edges are carried to paths between designated vertices, and the input triangle is carried to the whole output complex. See Figure 4.1 for an illustration. The *algebraic signature* of  $\text{Loop}(\mathcal{O}, \lambda)$  is  $(\pi_1(\mathcal{O}), \lambda)$ . This signature characterizes the relative power of such tasks.

We state the original theorem of Herlihy and Rajsbaum, characterizing the relative power of loop agreement tasks in terms of algebraic signatures.

**Theorem 4.1.3** (Herlihy and Rajsbaum). *Task  $\text{Loop}(\mathcal{K}_1, \lambda_1)$  implements  $\text{Loop}(\mathcal{K}_2, \lambda_2)$  if and only if there exists a group homomorphism  $h : \pi_1(\mathcal{K}_1) \rightarrow \pi_1(\mathcal{K}_2)$  such that  $h([\lambda_1]) = [\lambda_2]$ .*

As an aside, their theorem implies that the problem of wait-free task solvability in read-write memory is undecidable; see Herlihy *et al.* [22] for details. This theorem is generalized to allow multiple tasks to implement another.

## 4.2 Composing Loop Agreement Tasks

Now that we have defined the class of loop agreement theorems, and stated the original results, we discuss how loop agreement tasks can be combined into a single task.

### 4.2.1 Combining Simplicial Complexes

Before composing loop agreement tasks, we must consider how to combine their underlying simplicial complexes. Here, define the *categorical product* of simplicial complexes. This is later used to define implementation by multiple tasks, and task composition.

**Definition 4.2.1.** *Let  $\mathcal{C}_1$  and  $\mathcal{C}_2$  be simplicial complexes, and let  $V(\mathcal{C}_1)$  and  $V(\mathcal{C}_2)$  be their vertex sets, respectively. Then the (categorical) product of simplicial complexes is a complex  $\mathcal{C}_1 \times \mathcal{C}_2$  with vertex set  $V(\mathcal{C}_1) \times V(\mathcal{C}_2)$ . A subset  $\sigma$  of  $V(\mathcal{C}_1) \times V(\mathcal{C}_2)$  is a simplex in  $\mathcal{C}_1 \times \mathcal{C}_2$  if and only if  $\rho_1(\sigma)$  and  $\rho_2(\sigma)$  are simplexes in  $\mathcal{C}_1$  and  $\mathcal{C}_2$ , where  $\rho_1$  and  $\rho_2$  are projections onto the first and second coordinates, respectively.*



(a) The usual topological product of two topological spaces, which is a two-dimensional object.

(b) The simplicial complex obtained from the categorical product of two edges, which is a three-dimensional object.

Figure 4.2: Topological product (left) versus categorical product (right) of two 1-simplices.

Intuitively, the product of complexes is a way of combining two complexes in the “best possible way,” and operationally, the product captures all possible combinations of process views if two tasks are solved in parallel. Note, however, that this differs from the topological product of their geometric realizations.

See Figure 5.1 showing the difference between the topological product of two 1-simplices versus their categorical product. In general, given simplicial complexes  $\mathcal{A}$  and  $\mathcal{B}$  of dimensions  $m$  and  $n$ , respectively, the dimension of their topological product  $|\mathcal{A}| \times |\mathcal{B}|$  is  $m + n$ . However, the dimension of their categorical product  $|\mathcal{A} \times \mathcal{B}|$  is  $(m + 1)(n + 1) - 1$ , which is multiplicative in its two factors.

The topological spaces  $|\mathcal{A}| \times |\mathcal{B}|$  and  $|\mathcal{A} \times \mathcal{B}|$  are not homeomorphic, particularly because the categorical product is of higher dimension. However, the two spaces are homotopy equivalent. In the example given, this can be seen visually in which the tetrahedron is “flattened” against the square. This homotopy equivalence is stated as a theorem below.

**Fact 4.2.2.** *Let  $\mathcal{A}$  and  $\mathcal{B}$  be simplicial complexes. Then  $|\mathcal{A}| \times |\mathcal{B}| \simeq |\mathcal{A} \times \mathcal{B}|$ .*

See Kozlov’s book on combinatorial algebraic topology for a detailed proof of this result [27]. It follows that  $|\mathcal{A}| \times |\mathcal{B}|$  and  $|\mathcal{A} \times \mathcal{B}|$  have the same fundamental group. This will allow us to pass between the categorical product of  $\mathcal{A}$  and  $\mathcal{B}$  and the topological product of  $|\mathcal{A}|$  and  $|\mathcal{B}|$ .

### 4.2.2 Implementation by Multiple Tasks

To implement one task from a collection of others, we run protocols for each task from the collection, and use the combined output as the return value. Given two loop agreement tasks, the composite task's output complex is the 2-skeleton of the product of their output complexes, and the composite task's loop is the "diagonal" of the product of the two original loops. If  $\lambda_1$  and  $\lambda_2$  are two loops, we denote the corresponding diagonal loop by  $\lambda_1 \star \lambda_2$ . A formal definition of this follows.

**Definition 4.2.3.** *Let  $\lambda_1 = (v_0, v_1, v_2, p_{01}, p_{12}, p_{20})$  and  $\lambda_2 = (w_0, w_1, w_2, q_{01}, q_{12}, q_{20})$  be triangle loops in complexes  $\mathcal{A}$  and  $\mathcal{B}$ , respectively. Then the diagonal product of  $\lambda_1$  and  $\lambda_2$ , denoted  $\lambda_1 \star \lambda_2$ , is the triangle loop  $(u_0, u_1, u_2, r_{01}, r_{12}, r_{20})$  in  $\mathcal{A} \times \mathcal{B}$ , where  $u_i = (v_i, w_i)$ . The path  $r_{ij}$  is defined by traversing  $p_{ij}$  while  $w_i$  is fixed, followed by traversing  $q_{ij}$  while  $v_j$  is fixed. Note that we will use  $p_{ij} \star q_{ij}$  to denote the path defined by  $r_{ij}$  as above, though strictly speaking, the  $\star$  operator denotes two different operations in  $\lambda_1 \star \lambda_2$  and  $p_{ij} \star q_{ij}$ .*

Figure 4.3 below illustrates the diagonal product of two triangle loops  $\lambda_1$  and  $\lambda_2$  below, as a subcomplex of their product  $\lambda_1 \times \lambda_2$ . For ease of illustration, the loops are drawn as paths; the top and bottom vertices are identified. While each edge path in this example consists of just one edge, the diagram can be used to understand paths of any length. In this case, instead of  $\lambda_1 \times \lambda_2$  being a union of just three 3-simplexes, it would be a union of several more.

While the diagonal product  $\lambda_1 \star \lambda_2$  does not appear directly in the definition of implementation by multiple tasks, it is used in the relevant proofs characterizing relative power. The diagonal product is also needed to define composite loop agreement tasks.

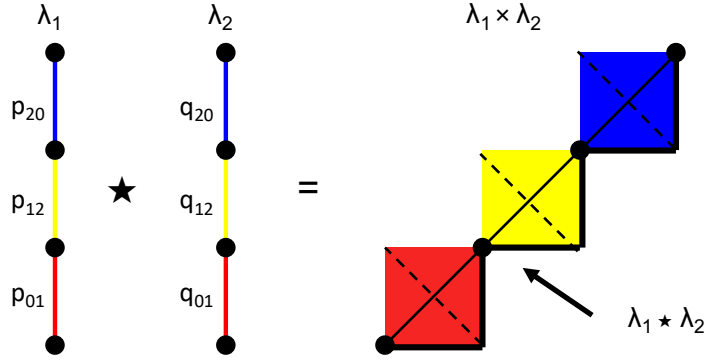


Figure 4.3: The diagonal product of two triangle loops  $\lambda_1 = (p_{01}, p_{12}, p_{20})$  and  $\lambda_2 = (q_{01}, q_{12}, q_{20})$ . The tetrahedra illustrate their categorical product when regarded as simplicial complexes.

**Definition 4.2.4.** Let  $T_1 = \text{Loop}(\mathcal{K}_1, \lambda_1)$ ,  $T_2 = \text{Loop}(\mathcal{K}_2, \lambda_2)$ , and  $T = \text{Loop}(\mathcal{K}, \lambda)$  be loop agreement tasks. Let  $\Gamma_1, \Gamma_2$ , and  $\Gamma$  be their respective specification maps. We say  $T_1$  and  $T_2$  implement  $T$  if there is an  $N \in \mathbb{N}$  and a simplicial map

$$\phi : \text{Bary}^N(\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)) \rightarrow \mathcal{K}$$

such that

$$(\phi \circ \text{Bary}^N)(\text{skel}^2(\Gamma_1(\sigma) \times \Gamma_2(\sigma))) \subseteq \Gamma(\sigma)$$

Here is the intuition behind this definition. To solve task  $T$  using protocols for  $T_1$  and  $T_2$ , the participating processes first execute protocols for  $T_1$  and  $T_2$ , resulting in a simplex on the combined output complex  $\mathcal{K}_1 \times \mathcal{K}_2$ . More specifically, since there are at most three participants, they end up on a simplex of  $\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$ . The processes then exchange results via  $N$  rounds of reading and writing to auxiliary read-write memory, ending up on a simplex of  $\text{Bary}^N(\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2))$ . Finally, each process calls a decision map  $\phi$  to choose a vertex in  $\mathcal{K}$ .

### 4.2.3 Relative Power of Multiple Task Implementation

The relative power of multiple task implementation can also be characterized using algebraic signatures. As with the original characterization, two loop agreement tasks implement another if and only if there is a satisfactory homomorphism between their respective fundamental groups. This is stated as a theorem.

**Theorem 4.2.5.** *Let  $T_1 = \text{Loop}(\mathcal{K}_1, \lambda_1)$ ,  $T_2 = \text{Loop}(\mathcal{K}_2, \lambda_2)$ , and  $T = \text{Loop}(\mathcal{K}, \lambda)$ . Then  $T_1$  and  $T_2$  implement  $T$  if and only if there exists a group homomorphism*

$$h : \pi_1(\mathcal{K}_1) \times \pi_1(\mathcal{K}_2) \rightarrow \pi_1(\mathcal{K})$$

*such that  $h([\lambda_1], [\lambda_2]) = [\lambda]$ .*

Theorem 4.2.5 describes only two loop agreement tasks implementing a third, but by finite induction, one can easily generalize this to  $n$  tasks. Its proof is broken down into two other theorems, which jointly prove Theorem 4.2.5. The first theorem is a topological characterization of two tasks implementing a third, while the second theorem is on the correspondence between continuous functions and group homomorphisms.

**Theorem 4.2.6.** *Tasks  $T_1$  and  $T_2$  implement  $T$  if and only if there exists a continuous function  $f : (\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2), \lambda_1 \star \lambda_2) \rightarrow (\mathcal{K}, \lambda)$ .*

We prove Theorem 4.2.6 by proving each direction individually via the following lemmas.

**Lemma 4.2.7.** *If there is a continuous function  $f : (\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2), \lambda_1 \star \lambda_2) \rightarrow (\mathcal{K}, \lambda)$ , then  $T_1$  and  $T_2$  implement  $T$ .*

*Proof.* Suppose such a function  $f$  exists, and let  $\Gamma_1$ ,  $\Gamma_2$ , and  $\Gamma$  be the specification maps for  $T_1$ ,  $T_2$ , and  $T$ , respectively. To prove  $T_1$  and  $T_2$  implement  $T$ , we require an  $N \in \mathbb{N}$  and a simplicial map  $\phi : \text{Bary}^N(\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)) \rightarrow \mathcal{K}$  such that for each  $\sigma \in \mathcal{I}$ , we have  $(\phi \circ \text{Bary}^N)(\text{skel}^2(\Gamma_1(\sigma) \times \Gamma_2(\sigma))) \subseteq \Gamma(\sigma)$ . We will construct such a  $\phi$  by taking a simplicial approximation of a suitably defined continuous function.

Let  $p_{01}$ ,  $p_{12}$ , and  $p_{20}$ , and  $q_{01}$ ,  $q_{12}$ , and  $q_{20}$  be the designated edge paths of  $\lambda_1$  and  $\lambda_2$ , respectively. Consider

$$X = |(p_{01} \times q_{01})| \cup |(p_{12} \times q_{12})| \cup |(p_{20} \times q_{20})| \subseteq |\mathcal{K}_1 \times \mathcal{K}_2|$$

as a topological subspace. Clearly, each  $|p_{ij} \times q_{ij}|$  deformation retracts to the corresponding path  $|p_{ij} \star q_{ij}|$  in  $|\lambda_1 \star \lambda_2|$ . In other words, we have a continuous function

$$H : X \times [0, 1] \rightarrow |\mathcal{K}_1 \times \mathcal{K}_2|$$

such that  $H(x, 0) = x$ ,  $H(X, 1) = |\lambda_1 \star \lambda_2|$ , and  $H(a, t) = a$  for each  $a \in |\lambda_1 \star \lambda_2|$ ,  $x \in X$ , and  $t \in [0, 1]$ . Now using Fact 3.2.7, we can extend  $H$  to a continuous function  $H' : |\mathcal{K}_1 \times \mathcal{K}_2| \times [0, 1] \rightarrow |\mathcal{K}_1 \times \mathcal{K}_2|$ . In particular, define  $r : |\mathcal{K}_1 \times \mathcal{K}_2| \rightarrow |\mathcal{K}_1 \times \mathcal{K}_2|$  as  $r(x) = H(x, 1)$ . This is a continuous function from  $|\mathcal{K}_1 \times \mathcal{K}_2|$  to itself that fixes  $|\lambda_1 \star \lambda_2|$  while collapsing  $X$  to  $|\lambda_1 \star \lambda_2|$ . We restrict  $r$  to  $|\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)|$  and invoke Fact 3.2.6 to get a function

$$g : |\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)| \rightarrow |\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)|$$

that fixes  $|\lambda_1 \star \lambda_2|$  while collapsing  $\text{skel}^2(X)$  to  $|\lambda_1 \star \lambda_2|$ . Now let  $F = f \circ g$ . This is a continuous function  $F : |\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)| \rightarrow |\mathcal{K}|$  which maps  $\lambda_1 \star \lambda_2$  to  $\lambda$ .

To show  $F$  is carried by  $\Gamma$ , first consider the case where  $|\sigma| = 1$ , where  $\sigma$  is just a point. Then the point  $|\Gamma_1(\sigma) \times \Gamma_2(\sigma)|$  is contained in  $|\lambda_1 \star \lambda_2|$ , so is fixed under  $g$ , and hence mapped to the appropriate point in  $\lambda$  by the given function  $f$ . The case  $|\sigma| = 2$  is similar. We have  $|\Gamma_1(\sigma) \times \Gamma_2(\sigma)| \subseteq X$ , which collapses to  $|\lambda_1 \star \lambda_2|$  under  $g$ . The function  $f$  maps this to  $\lambda$ , as desired. The final case is when  $|\sigma| = 3$ , which does not require any part of the proof above, since  $\Gamma(\sigma) = \mathcal{K}$ . In all cases, we see that  $F$  is carried by  $\Gamma$ . Letting  $\phi : \text{Bary}^N(\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)) \rightarrow \mathcal{K}$  be a simplicial approximation of  $F$ ,  $\phi$  is also carried by  $\Gamma$ , so we have the required decision map.

□

**Lemma 4.2.8.** *If tasks  $T_1$  and  $T_2$  implement  $T$ , then there is a continuous function*

$$f : (\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2), \lambda_1 \star \lambda_2) \rightarrow (\mathcal{K}, \lambda)$$

*Proof.* Assuming  $T_1$  and  $T_2$  implement  $T$ , we have a simplicial map  $\phi : \text{Bary}^N(\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)) \rightarrow \mathcal{K}$  that is carried by  $\Gamma$ . In particular,  $\phi$  maps  $\lambda_1 \star \lambda_2$  to  $\lambda$ . Let  $f : (\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2), \lambda_1 \star \lambda_2) \rightarrow (\mathcal{K}, \lambda)$ , defined by  $f(x) = |\phi|(x)$ . Then  $f$  maps  $|\lambda_1 \star \lambda_2|$  to  $|\lambda|$  since  $\phi$  does this as well. □

Lemmas 4.2.7 and 4.2.8 together prove Theorem 4.2.6. Next, we prove the correspondence



between continuous functions and group homomorphisms. In order to do this, we refer to the following result shown in Herlihy and Rajsbaum [19]. See their paper for a proof.

**Lemma 4.2.9.** *Let  $\mathcal{K}$  and  $\mathcal{L}$  be finite, connected, 2-dimensional simplicial complexes, and let  $h : \pi_1(\mathcal{K}) \rightarrow \pi_1(\mathcal{L})$  be a homomorphism with  $h([\sigma]) = [\tau]$ . Then there exists a continuous  $f : |\mathcal{K}| \rightarrow |\mathcal{L}|$  such that  $f_* = h$  and  $f \circ \sigma = \tau$ .*

The above lemma is applied to prove the a correspondence between maps. However before we continue with the next theorem, we state a fact from algebraic topology that will aid us in the proof.

**Fact 4.2.10.** *Let  $\mathcal{C}$  be a complex. Then the inclusion  $\iota : \text{skel}^2(\mathcal{C}) \rightarrow \mathcal{C}$  induces an isomorphism on fundamental groups.*

This fact essentially states that the fundamental group of a simplicial complex is completely characterized by its 2-skeleton, so we may apply the 2-skeleton operator without changing the group.

**Theorem 4.2.11.** *There exists a continuous function  $f : (\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2), \lambda_1 \star \lambda_2) \rightarrow (\mathcal{K}, \lambda)$  if and only if there exists a group homomorphism  $h : \pi_1(\mathcal{K}_1) \times \pi_1(\mathcal{K}_2) \rightarrow \pi_1(\mathcal{K})$  such that  $h([\lambda_1], [\lambda_2]) = [\lambda]$ .*

*Proof.* First suppose we have a continuous function  $f : (\text{skel}^2(|\mathcal{K}_1 \times \mathcal{K}_2|), \lambda_1 \star \lambda_2) \rightarrow (\mathcal{K}, \lambda)$ . We begin by constructing a homomorphism

$$h' : \pi_1(|\mathcal{K}_1 \times \mathcal{K}_2|) \rightarrow \pi_1(\mathcal{K})$$

with  $h'([\lambda_1 \star \lambda_2]) = [\lambda]$ . Let  $\iota : \text{skel}^2(|\mathcal{K}_1 \times \mathcal{K}_2|) \rightarrow |\mathcal{K}_1 \times \mathcal{K}_2|$  be the inclusion map, whose induced homomorphism is actually an isomorphism, by Fact 4.2.10. Then we let

$$h' = f_* \circ \iota_*^{-1}$$

In order to show  $h'([\lambda_1 \star \lambda_2]) = [\lambda]$ , it suffices to show that  $\iota_*^{-1}([\lambda_1 \star \lambda_2]) = [\lambda_1 \star \lambda_2]$ . However, notice that  $[\lambda_1 \star \lambda_2] = \iota_*([\lambda_1 \star \lambda_2])$  since  $\lambda_1 \star \lambda_2$  is already in  $\text{skel}^2(|\mathcal{K}_1 \times \mathcal{K}_2|)$ , so  $\iota_*^{-1}([\lambda_1 \star \lambda_2]) = [\lambda_1 \star \lambda_2]$  as required.

Now, we define the desired homomorphism

$$h : \pi_1(\mathcal{K}_1) \times \pi_1(\mathcal{K}_2) \rightarrow \pi_1(\mathcal{K})$$

using  $h'$ . Let  $\alpha_1$  and  $\alpha_2$  be loops in  $\mathcal{K}_1$  and  $\mathcal{K}_2$  respectively. By Fact 3.2.6,  $\alpha_1$  and  $\alpha_2$  are homotopic to edge loops  $\beta_1$  and  $\beta_2$ . Now define  $h$  as  $h([\alpha_1], [\alpha_2]) = h'([\beta_1 \star \beta_2])$ . Then it follows that  $h([\lambda_1], [\lambda_2]) = [\lambda]$ . To show  $h$  is well-defined, we need to show that  $|\beta_1 \star \beta_2| \simeq |\beta'_1 \star \beta'_2|$  for other edge-loop representatives  $\beta'_1$  and  $\beta'_2$  of  $\alpha_1$  and  $\alpha_2$ . We can find edge homotopies  $H_1$  and  $H_2$  taking  $\beta_1$  and  $\beta_2$  to  $\beta'_1$  and  $\beta'_2$ , respectively, so  $H_1 \star H_2$  is an edge homotopy from  $|\beta_1 \star \beta_2| \simeq |\beta'_1 \star \beta'_2|$ , proving that  $h$  is well-defined. We have thus found the required  $h$ , which proves the forward direction of the theorem.

Now suppose we start with a homomorphism  $h$  as described above. We reverse the above argument. We begin by constructing a homomorphism  $h' : \pi_1(|\mathcal{K}_1 \times \mathcal{K}_2|) \rightarrow \pi_1(\mathcal{K})$ . Let  $\alpha$  be a loop in  $|\mathcal{K}_1 \times \mathcal{K}_2|$ . As before,  $\alpha$  is homotopic to some edge loop  $\beta$  of  $\mathcal{K}_1 \times \mathcal{K}_2$ . We define  $h'([\alpha]) = h([\rho_1 \circ \beta], [\rho_2 \circ \beta])$ , where the  $\rho_i$  are the projection maps. This map is clearly well-defined and a homomorphism since it is the composition of  $h$  and the induced maps of the  $\rho_i$ .

Now we define a homomorphism

$$h'' : \pi_1(\text{skel}^2(|\mathcal{K}_1 \times \mathcal{K}_2|)) \rightarrow \pi_1(\mathcal{K})$$

with  $h''([\lambda_1 \star \lambda_2]) = [\lambda]$ , using  $h'$ . Let  $\iota$  be the inclusion map, as before. Then we define  $h'' = h' \circ \iota_*$ . Since  $\iota_*([\lambda_1 \star \lambda_2]) = [\lambda_1 \star \lambda_2]$ , we see that  $h''([\lambda_1 \star \lambda_2]) = [\lambda]$ . Finally, we invoke Lemma 4.2.9 on  $h''$  to obtain the required  $f$ . This proves the backward direction of the theorem, and completes the proof. □

Theorems 4.2.6 and 4.2.11 together prove Theorem 4.2.5.

#### 4.2.4 Composite Loop Agreement

In defining multiple implementation, we said that tasks  $T_1$  and  $T_2$  implement  $T$  if we can use the combined output complex  $\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$  of  $T_1$  and  $T_2$  to solve  $T$ . We can think of parallel execution of protocols for  $T_1$  and  $T_2$  as solving a task with input complex  $\Delta^2$ , output complex  $\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$ , and specification  $\Gamma_1 \times \Gamma_2$ . We get a task  $T' = (\Delta^2, \text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2), \Gamma_1 \times \Gamma_2)$ , and from the definitions it is clear that  $T_1$  and  $T_2$  implement  $T$  if and only if  $T'$  implements  $T$ . Unfortunately,  $T'$  is not a loop agreement task, since processes starting on an edge in  $\Delta^2$  can land on any edge in  $\lambda_1 \times \lambda_2$  and still obey the task specification. However, the subcomplex  $\lambda_1 \times \lambda_2$  is not a loop. We address this

by defining a loop agreement task  $T_1 \times T_2$  with output complex  $\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$  with triangle loop  $\lambda_1 \star \lambda_2$ . We then show that  $T'$  and  $T_1 \times T_2$  implement one another, so that they are equivalent.

Refer to Figure 4.3 for a simple illustration of  $\lambda_1 \star \lambda_2$ .

**Definition 4.2.12.** *Let  $T_1 = \text{Loop}(\mathcal{K}_1, \lambda_1)$  and  $T_2 = \text{Loop}(\mathcal{K}_2, \lambda_2)$  be loop agreement tasks. Then the composition of  $T_1$  and  $T_2$ , denoted  $T_1 \times T_2$ , is the loop agreement task  $\text{Loop}(\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2), \lambda_1 \star \lambda_2)$ .*

From this definition, together with what was proved in the previous section, we get the following propositions.

**Proposition 4.2.13.** *Tasks  $T_1$  and  $T_2$  implement  $T_1 \times T_2$ .*

*Proof.* In the proof of Theorem 4.2.5, we show the following lemma:  $T_1$  and  $T_2$  implement  $T$  if there exists a continuous function  $f : (\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2), \lambda_1 \star \lambda_2) \rightarrow (\mathcal{K}, \lambda)$ . Letting  $T = T_1 \times T_2$ , the proposition is an immediate consequence of this lemma.  $\square$

So two tasks  $T_1$  and  $T_2$  are capable of implementing their composite task  $T_1 \times T_2$ . Likewise, we are able to show the reverse direction. That is, the composite tasks is just as powerful as each component loop agreement task.

**Proposition 4.2.14.** *Task  $T_1 \times T_2$  implements  $T_1$  (respectively  $T_2$ ).*

*Proof.* Lemma 6.2 from Herlihy and Rajsbaum [19] states that it suffices to show there exists a continuous function  $f : \text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2) \rightarrow \mathcal{K}_1$  mapping  $\lambda_1 \star \lambda_2$  to  $\lambda_1$ . It is easy to see that the projection map  $\rho_1 : \text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2) \rightarrow \mathcal{K}_1$  satisfies this condition. The proof that  $T_1 \times T_2$  implements  $T_2$  is identical.  $\square$

### 4.3 A Categorical Interpretation

The above two propositions together imply that  $T_1$  and  $T_2$ , and  $T_1 \times T_2$  have the same computational power, in the sense that they can implement the same loop agreement tasks. This claim can be made more rigorous using the language of category theory. In the next subsection, we offer a brief introduction to elements of category theory; see MacLane [32] for a complete and formal treatment. Then we describe how the class of loop agreement tasks can be interpreted as a category.

### 4.3.1 Category Theory

A *category*  $C$  is a collection of *objects*, denoted  $\text{Ob}(C)$ , together with a collection of *morphisms* between objects, denoted  $\text{Hom}(C)$ . Each morphism has a *domain* and *codomain*, which are both objects in  $\text{Ob}(C)$ . If  $f$  is a morphism with domain  $X$  and codomain  $Y$ , we write  $f : X \rightarrow Y$ . This notation is intentionally suggestive of ordinary set functions.

As with set functions, morphisms can be composed. Formally,  $\text{Hom}(C)$  is equipped with a binary operation called *composition*. If  $f$  and  $g$  are morphisms, then their composition is denoted  $f \circ g$ . Note that function composition is only defined when the codomain of the first morphism is equal to the domain of the second. The composition operation is required to be associative; that is, given  $f : W \rightarrow X$ ,  $g : X \rightarrow Y$ , and  $h : Y \rightarrow Z$ , composition must obey the rule that  $h \circ (g \circ f) = (h \circ g) \circ f$ . Composition also requires the existence of an identity morphism for each object  $X$ , denoted  $\text{id}_X$ , such that for each  $f : X \rightarrow Y$ , we have  $f \circ \text{id}_X = f = \text{id}_Y \circ f$ .

Sets and set functions comprise the category of sets, which is typically denoted **Set**. The category of topological spaces, denoted **Top**, has spaces as its objects and continuous functions as its morphisms. There is also the category of groups, **Grp**, consisting of groups and groups homomorphisms. Algebraic signatures belong to a similar category called the category of *pointed* groups, **pGrp**, whose objects are groups with distinguished elements, and whose morphisms are group homomorphisms that map distinguished elements to distinguished elements.

Objects and morphisms of one category can be transformed into to objects and morphisms of another. Given categories  $C$  and  $D$ , a *functor*  $F : C \rightarrow D$  assigns to each object  $X \in \text{Ob}(C)$  an object  $F(X) \in \text{Ob}(D)$ , and to each morphism  $f : X \rightarrow Y$  a morphism  $F(f) : F(X) \rightarrow F(Y)$ . Functors must respect composition, so that given two compatible morphisms  $f, g \in \text{Hom}(C)$ , we must have  $F(f \circ g) = F(f) \circ F(g)$ . Functors should also respect identity morphisms:  $F(\text{id}_X) = \text{id}_{F(X)}$ . A common example of a functor is the fundamental group functor  $\pi_1 : \mathbf{pTop} \rightarrow \mathbf{Grp}$ , which maps pointed topological spaces to their respective fundamental groups, and maps continuous functions to their induced homomorphisms. The geometric realization  $|\cdot| : \mathbf{SimC} \rightarrow \mathbf{Top}$  is a functor from the category of simplicial complexes with simplicial maps to **Top**, which maps complexes and simplicial maps to their respective geometric realizations. In the next section we define **Loop**, the category of loop agreement tasks.

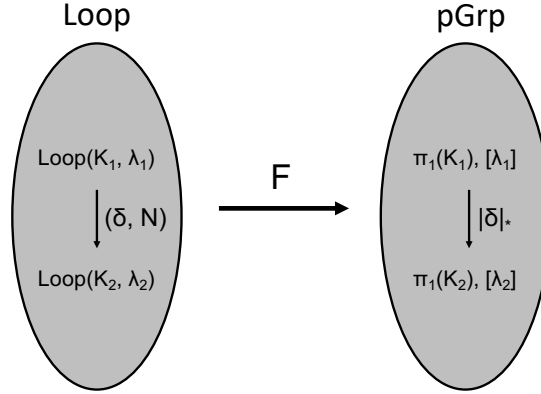


Figure 4.4: An example of a functor of categories, which here is the algebraic signature between **Loop** and **pGrp**. Note that functor not only map objects to objects, but also morphisms to morphisms.

The categorical product of simplicial complexes was introduced in the previous section, but it the concept has a broader definition within the context of category theory. The *categorical product* is a kind of operation that combines two objects from the same category, to form a new object that is representative of components in some standard way. Informally, the categorical product of two objects is the most generic object that maps onto the original two.

**Definition 4.3.1.** *Let  $C$  be a category, and let  $X_1$  and  $X_2$  be objects in this category. The categorical product of  $X_1$  and  $X_2$  is the unique object  $X_1 \times X_2$  satisfying the following condition: there exist morphisms (called projections)  $\rho_1 : X_1 \times X_2 \rightarrow X_1$  and  $\rho_2 : X_1 \times X_2 \rightarrow X_2$  such that for any object  $X$  with morphisms  $f_1 : X \rightarrow X_1$  and  $f_2 : X \rightarrow X_2$ , there exists a unique morphism  $f : X \rightarrow X_1 \times X_2$  such that  $f_1 = \rho_1 \circ f$  and  $f_2 = \rho_2 \circ f$ . That is,  $f_1$  and  $f_2$  factor through  $X_1 \times X_2$  in a unique way, via  $f$ . The morphism  $f$  is called the product morphism of  $f_1$  and  $f_2$ .*

We will see that loop agreement task composition is an example of a categorical product. Next, we show how the class of loop agreement tasks forms a category.

### 4.3.2 The Category of Loop Agreement Tasks

We define **Loop**, the category of loop agreement tasks. Let  $\text{Ob}(\mathbf{Loop})$  be the collection of all loop agreement tasks  $\text{Loop}(\mathcal{K}, \lambda)$ , where  $\mathcal{K}$  ranges over all finite connected 2-dimensional complexes, and  $\lambda$  ranges over all edge loops. Morphisms in **Loop** are valid decision maps between tasks. That is,

given tasks  $T_1 = \text{Loop}(\mathcal{K}_1, \lambda_1)$  and  $T_2 = \text{Loop}(\mathcal{K}_2, \lambda_2)$ , a morphism  $f : T_1 \rightarrow T_2$  is a pair  $(\delta, N)$  where  $N \in \mathbb{N}$  and  $\delta : \text{Bary}^N(\mathcal{K}_1) \rightarrow \mathcal{K}_2$  is a decision map such that  $T_1$  solves  $T_2$  via  $\delta$ . Composition of morphisms is defined as follows. Given objects  $T_1 = \text{Loop}(\mathcal{K}_1, \lambda_1)$ ,  $T_2 = \text{Loop}(\mathcal{K}_2, \lambda_2)$ ,  $T_3 = \text{Loop}(\mathcal{K}_3, \lambda_3)$ , and morphisms  $f_1 : T_1 \rightarrow T_2$ ,  $f_2 : T_2 \rightarrow T_3$  where  $f_1 = (\delta_1, N_1)$  and  $f_2 = (\delta_2, N_2)$ , the composition  $f_2 \circ f_1$  is defined as  $(\delta_2 \circ \text{Bary}^{N_2}(\delta_1), N_1 + N_2)$ .

Two morphisms are considered equivalent if their simplicial maps are homotopic. Note that by identifying morphisms (in this case homotopic ones), we are constructing a *quotient category* of the original one. In order to construct a quotient category, the equivalence must be compatible with composition. However, it is known that homotopy is compatible with compositions of continuous functions. It remains to be shown that **Loop** is indeed a category.

**Theorem 4.3.2.** *Loop is a category.*

*Proof.* Let  $T_i$  and  $f_i$  be defined as above, and let  $\Gamma_i$  be the tasks' respective specification maps. To show **Loop** is a category, we need to show that  $\text{Hom}(\mathbf{Loop})$  is closed under composition, composition is associative, and identity morphisms exist. Showing that  $\text{Hom}(\mathbf{Loop})$  is closed under composition amounts to showing that  $T_1$  solves  $T_3$  via

$$\delta_2 \circ \text{Bary}^{N_2}(\delta_1) : \text{Bary}^{N_1+N_2}(\mathcal{K}_1) \rightarrow \mathcal{K}_3$$

For brevity we define  $\delta = \delta_2 \circ \text{Bary}^{N_2}(\delta_1)$ .

From the definition of task implementation, we know that

$$\delta_1 \circ \text{Bary}^{N_1} \circ \Gamma_1 \subseteq \Gamma_2$$

and

$$\delta_2 \circ \text{Bary}^{N_2} \circ \Gamma_2 \subseteq \Gamma_3$$

and we want to show

$$\delta \circ \text{Bary}^{N_1+N_2} \circ \Gamma_1 \subseteq \Gamma_3$$

We have the sequence of containments

$$\delta_2 \circ \text{Bary}^{N_2} \circ \delta_1 \circ \text{Bary}^{N_1} \circ \Gamma_1 \subseteq \delta_2 \circ \text{Bary}^{N_2} \circ \Gamma_2 \subseteq \Gamma_3$$

We also know that

$$\text{Bary}^{N_2} \circ \delta_1 = \text{Bary}^{N_2}(\delta_1) \circ \text{Bary}^{N_2}$$

so

$$\begin{aligned} & \delta_2 \circ \text{Bary}^{N_2} \circ \delta_1 \circ \text{Bary}^{N_1} \circ \Gamma_1 \\ &= \delta_2 \circ \text{Bary}^{N_2}(\delta_1) \circ \text{Bary}^{N_2} \circ \text{Bary}^{N_1} \circ \Gamma_1 \\ &= \delta \circ \text{Bary}^{N_1+N_2} \circ \Gamma_1 \subseteq \Gamma_3 \end{aligned}$$

Therefore  $T_1$  solves  $T_3$  via  $\delta$ , so  $\text{Hom}(\mathbf{Loop})$  is closed under our definition of composition.

Verifying associativity follows a similar argument. Again, let  $T_i$  and  $f_i$  be defined as above, and in addition let  $T_4 = \text{Loop}(\mathcal{K}_4, \lambda_4)$  and let  $f_3 : T_3 \rightarrow T_4$  with  $f_3 = (\delta_3, N_3)$ . We must show that

$$(f_3 \circ f_2) \circ f_1 = f_3 \circ (f_2 \circ f_1)$$

But

$$\begin{aligned} & (f_3 \circ f_2) \circ f_1 \\ &= (\delta_3 \circ \text{Bary}^{N_3}(\delta_2), N_2 + N_3) \circ (\delta_1, N_1) \\ &= (\delta_3 \circ \text{Bary}^{N_3}(\delta_2) \circ \text{Bary}^{N_2+N_3}(\delta_1), N_1 + N_2 + N_3) \end{aligned}$$

and

$$\begin{aligned} & f_3 \circ (f_2 \circ f_1) \\ &= (\delta_3, N_3) \circ (\delta_2 \circ \text{Bary}^{N_2}(\delta_1), N_1 + N_2) \\ &= (\delta_3 \circ \text{Bary}^{N_3}(\delta_2 \circ \text{Bary}^{N_2}(\delta_1)), N_1 + N_2 + N_3) \\ &= (\delta_3 \circ \text{Bary}^{N_3}(\delta_2) \circ \text{Bary}^{N_2+N_3}(\delta_1), N_1 + N_2 + N_3) \end{aligned}$$

so  $(f_3 \circ f_2) \circ f_1 = f_3 \circ (f_2 \circ f_1)$ . Therefore composition is associative.

The last requirement, existence of identity morphisms, is trivial to show. Task  $T_1$  solves itself via the decision map  $(\text{id}_{\mathcal{K}_1}, 0)$ . This finishes the proof that  $\mathbf{Loop}$  is a category.

□

Next, we show that the algebraic signature of Herlihy and Rajsbaum can be formulated as a functor between **Loop** and **pGrp**. We first define the mapping between objects and morphism, and then prove it satisfies the properties of a functor.

**Definition 4.3.3.** *Let  $T_1, T_2 \in Ob(\mathbf{Loop})$  with  $T_1 = Loop(\mathcal{K}_1, \lambda_1)$  and  $T_2 = Loop(\mathcal{K}_2, \lambda_2)$ , and let  $f_1 : T_1 \rightarrow T_2$  with  $f_1 = (\delta_1, N_1)$  be a morphism between the two. Then the algebraic signature functor is a functor  $S : \mathbf{Loop} \rightarrow \mathbf{pGrp}$  defined as follows. Object  $T_1$  is mapped to  $(\pi_1(\mathcal{K}_1), [\lambda_1])$ , while morphism  $f_1 : T_1 \rightarrow T_2$  is mapped to  $|\delta_1|_* : (\pi_1(\mathcal{K}_1), [\lambda_1]) \rightarrow (\pi_2(\mathcal{K}_2), [\lambda_2])$ .*

**Theorem 4.3.4.**  *$S : \mathbf{Loop} \rightarrow \mathbf{pGrp}$  is a functor.*

*Proof.* We use the fact that  $\pi_1$  and  $|\cdot|$  are both functors. We need to show that  $S$  preserves identity morphisms and respects composition of morphisms. Let  $T_1, T_2$ , and  $f$  be defined as above, and let  $T_3 = Loop(\mathcal{K}_3, \lambda_3)$  and let  $f_2 : T_2 \rightarrow T_3$  with  $f_2 = (\delta_2, N_2)$ . Then, using the functoriality of  $\pi_1$  and  $|\cdot|$ , we have

$$\begin{aligned}
& S(f_2 \circ f_1) \\
&= S((\delta_2 \circ \text{Bary}^{N_1}(\delta_1), N_1 + N_2)) \\
&= |\delta_2 \circ \text{Bary}^{N_1}(\delta_1)|_* \\
&= (|\delta_2| \circ |\text{Bary}^{N_1}(\delta_1)|)_* \\
&= |\delta_2|_* \circ |\delta_1|_* \\
&= S(f_2) \circ S(f_1)
\end{aligned}$$

so  $S$  respects composition. Now let  $\text{id}_{T_1}$  be the identity morphism of  $T_1$ . Then  $S(\text{id}_{T_1}) = S((\text{id}_{\mathcal{K}_1}, 0)) = |\text{id}_{\mathcal{K}_1}|_* = \text{id}_{\pi_1(\mathcal{K}_1)}$ , so  $S$  also preserves identity morphisms.  $S$  is well-defined since  $\pi_1$  cannot distinguish between homotopic functions. We conclude that  $S$  is a functor. □

We are almost ready to prove that composition of loop agreement tasks is in fact the categorical product in **Loop**, but first we need a lemma describing the categorical product in **SimC**<sub>2</sub>, which is slightly different than the one in **SimC**. Namely, we must take 2-skeletons after taking the product of complexes.



**Lemma 4.3.5.** *If  $\mathcal{K}_1$  and  $\mathcal{K}_2$  are objects in  $\mathbf{SimC}_2$ , then  $\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$  is their categorical product in  $\mathbf{SimC}_2$ .*

*Proof.* In order to define the product, we must first define projection maps. These projection maps are  $\rho_1 : \text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2) \rightarrow \mathcal{K}_1$  and  $\rho_2 : \text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2) \rightarrow \mathcal{K}_2$ , defined as  $\rho_1(v_1, v_2) = v_1$  and  $\rho_2(v_1, v_2) = v_2$ . That is, the  $\rho_i$  are the restrictions to the 2-skeleton of the projection maps found in Definition 4.2.1, so they are clearly simplicial.

Now suppose we have a 2-dimensional complex  $\mathcal{K}$  with simplicial maps  $\delta_1 : \mathcal{K} \rightarrow \mathcal{K}_1$  and  $\delta_2 : \mathcal{K} \rightarrow \mathcal{K}_2$ . Then we define  $\delta : \mathcal{K} \rightarrow \text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$  as  $\delta(v) = (\delta_1(v), \delta_2(v))$ . This is the only possible set function  $\delta$  that makes the diagram commute; that is,  $\delta$  is the only set function such that  $\delta_1 = \rho_1 \circ \delta$  and  $\delta_2 = \rho_2 \circ \delta$ . This proves uniqueness, but we must also show that  $\delta$  is simplicial.

Let  $\sigma$  be a simplex in  $\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$ . Then  $\delta_1(\sigma)$  and  $\delta_2(\sigma)$  are simplexes in  $\mathcal{K}_1$  and  $\mathcal{K}_2$ , respectively. But as we have shown,  $\delta_1(\sigma) = \rho_1(\delta(\sigma))$  and  $\delta_2(\sigma) = \rho_2(\delta(\sigma))$ , so in particular, we see that  $\rho_1(\delta(\sigma))$  and  $\rho_2(\delta(\sigma))$  are simplexes. Hence by Definition 4.2.1,  $\delta(\sigma)$  is a simplex in  $\mathcal{K}_1 \times \mathcal{K}_2$ , and furthermore it is a simplex in  $\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$  since the dimension of  $\sigma$  is at most 2. So  $\delta$  is a simplicial map, which proves that  $\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$  is the categorical product of  $\mathcal{K}_1$  and  $\mathcal{K}_2$  in  $\mathbf{SimC}_2$ .  $\square$

Note that Lemma 4.3.5 easily generalizes to  $\mathbf{SimC}_n$  and the  $n$ -skeleton. We use the above construction of the categorical product in  $\mathbf{SimC}_2$  to define a product in the category of loop agreement tasks.

**Theorem 4.3.6.** *Composition of loop agreement tasks is the categorical product in  $\mathbf{Loop}$ .*

*Proof.* Let  $T_1 = \text{Loop}(\mathcal{K}_1, \lambda_1)$  and  $T_2 = \text{Loop}(\mathcal{K}_2, \lambda_2)$  be tasks as defined before, and let  $\Gamma_1$  and  $\Gamma_2$  be their specification maps, respectively. Let  $\Gamma_\times$  be the specification map of  $T_1 \times T_2$ . We must first define decision maps from  $T_1 \times T_2$  to  $T_1$  and  $T_2$  that make  $T_1 \times T_2$  the categorical product. We know that  $\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$  is the categorical product of  $\mathcal{K}_1$  and  $\mathcal{K}_2$  in the category  $\mathbf{SimC}_2$ , and that the product comes with projection maps  $\rho_1 : \text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2) \rightarrow \mathcal{K}_1$  and  $\rho_2 : \text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2) \rightarrow \mathcal{K}_2$ . Using these, we define maps  $g_1 : T_1 \times T_2 \rightarrow T_1$  and  $g_2 : T_1 \times T_2 \rightarrow T_2$  with  $g_1 = (\rho_1, 0)$  and  $g_2 = (\rho_2, 0)$ , and we show that these maps make  $T_1 \times T_2$  the categorical product of  $T_1$  and  $T_2$ .

$$\begin{array}{ccccc}
& & T & & \\
& \swarrow f_1 & \vdots f & \searrow f_2 & \\
T_1 & \xleftarrow{g_1} & T_1 \times T_2 & \xrightarrow{g_2} & T_2
\end{array}$$

Figure 4.5: The universal property defining the product informally states that if  $T$  solves tasks  $T_1$  and  $T_2$ , then there is a unique decision map,  $f$ , for which  $T$  solves  $T_1 \times T_2$ .

We showed in Proposition 4.2.14 that  $g_1$  and  $g_2$  solve  $T_1$  and  $T_2$ , respectively. To prove that  $g_1$  and  $g_2$  are the projection maps satisfying Definition 4.3.1, we consider a task  $T$  that implements both  $T_1$  and  $T_2$ , say via maps  $f_1 = (\delta_1, N_1)$  and  $f_2 = (\delta_2, N_2)$ , respectively. Let  $T = \text{Loop}(\mathcal{K}, \lambda)$  and let  $\Gamma$  be its specification map. We must find a decision map that solves  $T_1 \times T_2$  from  $T$ . Without loss of generality, assume  $N_1 \geq N_2$ , so let  $\delta'_2 : \text{Bary}^{N_1}(\mathcal{K}) \rightarrow \mathcal{K}_2$  be a simplicial approximation of  $\delta_2$ . Then  $\delta = (\delta_1, \delta'_2)$  is a map from  $\text{Bary}^{N_1}(\mathcal{K})$  to  $\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$ , though it does not necessarily carry  $\lambda$  to  $\lambda_1 \star \lambda_2$ . Instead,  $g = (\delta, N_1)$  is a morphism from  $\text{Loop}(\mathcal{K}, \lambda)$  to  $\text{Loop}(\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2), \delta(\lambda))$ . However, it is easy to see that  $\delta(\lambda)$  is homotopic to  $\lambda_1 \star \lambda_2$ . Using Fact 3.2.7, we can extend this to a homotopy on all of  $\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$ , so we obtain a continuous function  $h : |\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)| \rightarrow |\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)|$ . Let  $\gamma : \text{Bary}^M(\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)) \rightarrow \text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)$  be a simplicial approximation of  $h$ . Then notice that  $g' = (\gamma, M)$  is a morphism from  $\text{Loop}(\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2), \delta(\lambda))$  to  $\text{Loop}(\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2), \lambda_1 \star \lambda_2)$ . So  $f = g' \circ g$  is a morphism  $f : T \rightarrow T_1 \times T_2$ . We must also show that  $f = (\gamma \circ \text{Bary}^M(\delta), N_1 + M)$  makes the diagram commute. Let  $\delta' = \gamma \circ \text{Bary}^M(\delta)$ . We know that  $\rho_i \circ \delta \simeq \delta_i$  by construction of  $\delta$ , and it is also clear that  $\delta' \simeq \delta$ , by construction of  $\delta'$  and  $\gamma$ . It follows that  $\rho_i \circ \delta' \simeq \delta_i$ , proving that  $f$  makes the diagram commute. Thus we have the required product morphism.

Finally, it remains to show that  $f$  is unique. Let  $f'$  be any such morphism making the diagram commute, and let  $\delta'$  be its simplicial map. Then, as set maps, we know that  $\delta' = (\rho_1 \circ \delta', \rho_2 \circ \delta')$ . However, we are assuming that  $|\rho_1 \circ \delta'| \simeq |\delta_1|$  and  $|\rho_2 \circ \delta'| \simeq |\delta_2|$ , so this allows us to conclude that  $|\delta'| = (|\rho_1 \circ \delta'|, |\rho_2 \circ \delta'|) \simeq (|\delta_1|, |\delta_2|)$ . Therefore  $|\delta'| \simeq (|\delta_1|, |\delta_2|)$ , which is homotopic to the map constructed in the existence proof above. So  $\delta$  is unique up to homotopy, meaning that  $f$  is unique. This proves that  $g_1$  and  $g_2$  are satisfactory projection maps, proving that  $T_1 \times T_2$  is in fact the categorical product of  $T_1$  and  $T_2$ .

□

The category  $\mathbf{pGrp}$  also has products. We define this product below, and state without proof that it is indeed the categorical product. This follows immediately from the fact that the direct product of groups is the categorical product in  $\mathbf{Grp}$  [32].

**Fact 4.3.7.** *Let  $(G_1, g_1)$  and  $(G_2, g_2)$  be objects in  $\mathbf{pGrp}$ . Then  $(G_1 \times G_2, (g_1, g_2))$  is their categorical product.*

With this in mind, the following corollary is a simple consequence of Theorem 4.2.5. It states that the algebraic signature functor preserves categorical products of objects.

**Corollary 4.3.8.** *The functor  $S : \mathbf{Loop} \rightarrow \mathbf{pGrp}$  preserves products.*

*Proof.* Let  $T_1 = \text{Loop}(\mathcal{K}_1, \lambda_1)$  and  $T_2 = \text{Loop}(\mathcal{K}_2, \lambda_2)$  be objects in  $\mathbf{Loop}$ . Then  $S(T_1) = (\pi_1(\mathcal{K}_1), [\lambda_1])$  and  $S(T_2) = (\pi_1(\mathcal{K}_2), [\lambda_2])$ , so  $S(T_1) \times S(T_2) = (\pi_1(\mathcal{K}_1) \times \pi_1(\mathcal{K}_2), ([\lambda_1], [\lambda_2]))$ . However, from the proof of Theorem 4.2.11, we see that  $(\pi_1(\mathcal{K}_1) \times \pi_1(\mathcal{K}_2), ([\lambda_1], [\lambda_2])) \cong (\pi_1(\text{skel}^2(\mathcal{K}_1 \times \mathcal{K}_2)), [\lambda_1 \star \lambda_2]) = S(T_1 \times T_2)$ , so in fact  $S(T_1 \times T_2) \cong S(T_1) \times S(T_2)$ . Therefore  $S$  preserves products.  $\square$

## 4.4 The Lattice of Loop Agreement Tasks

In this subsection, we present some simple conclusions derived from the algebraic signature functor applied to certain loop agreement tasks. The first result is on  $(3, 2)$ -set agreement, or the most powerful task among the class of loop agreement.

**Proposition 4.4.1.** *Let  $T$  be  $(3, 2)$ -set agreement, and let  $T'$  be any other loop agreement task. Then  $T \times T'$  and  $T$  are equivalent.*

*Proof.* Recall that  $(3, 2)$ -set agreement is the task  $\text{Loop}(\text{skel}^1(\Delta^2), \zeta)$ , where  $\zeta$  is the triangle loop  $(0, 1, 2, ((0, 1), ((1, 2), ((2, 0))))$ . This triangle loop generates  $\pi_1(\text{skel}^1(\Delta^2))$ , so  $S(T) = (\pi_1(\text{skel}^1(\Delta^2)), [\zeta]) \cong (\mathbb{Z}, 1)$ . Let  $S(T') = (G, g)$ . Then by Corollary 4.3.8,  $S(T \times T') = S(T) \times S(T') = (\mathbb{Z} \times G, (1, g))$ . The homomorphism  $\phi : \mathbb{Z} \times G \rightarrow \mathbb{Z}$  defined by projection onto the first coordinate sends  $(1, g)$  to 1, and the homomorphism  $\psi : \mathbb{Z} \rightarrow \mathbb{Z} \times G$  defined by  $\psi(n) = (n, g)$  sends 1 to  $(1, g)$ . So  $T \times T'$  and  $T$  implement one another, so they are equivalent.  $\square$

Since  $(3, 2)$ -set agreement was shown to be universal for loop agreement by Herlihy and Rajsbaum [19], it is operationally intuitive that composing it with any other loop agreement task should not change the relative power of  $(3, 2)$ -set agreement, since it is already the most powerful task. The next result is about the least powerful loop agreement task, or simplex agreement.

**Proposition 4.4.2.** *Let  $T$  be any simplex agreement task, and let  $T'$  be any other loop agreement task. Then  $T \times T'$  and  $T$  are equivalent.*

*Proof.* Since the output complex of  $T$  is a subdivided simplex, it has trivial fundamental group, so  $S(T) = (1, e)$ . As before, let  $S(T') = (G, g)$ . By Corollary 4.3.8,  $S(T \times T') = S(T) \times S(T') = (1 \times G, (g, e))$ , which is clearly isomorphic to  $(G, g)$ . So  $T \times T'$  and  $T$  implement one another, so are equivalent. □

Herlihy and Rajsbaum also showed that simplex agreement is implemented from any loop agreement task [19], so it is also intuitively clear that composing a task with simplex agreement should not change the relative power of the original task. Adding simplex agreement to a task will not increase its power.

The next result is about self-composition of tasks.

**Proposition 4.4.3.** *Let  $T$  be any loop agreement task. Then  $T \times T$  and  $T$  are equivalent.*

*Proof.* Let  $S(T) = (G, g)$ . Then by Corollary 4.3.8,  $S(T \times T) = S(T) \times S(T) = (G \times G, (g, g))$ . Letting  $\phi : G \rightarrow G \times G$  be the diagonal map  $\phi(x) = (x, x)$ ,  $\phi$  maps  $g$  to  $(g, g)$ , and letting  $\psi : G \times G \rightarrow G$  be projection onto a coordinate,  $\psi$  maps  $(g, g)$  to  $g$ . So  $T \times T$  and  $T$  are equivalent. □

The above result states that composing a loop agreement task with copies of itself will not change its relative power. This is also reasonable since nothing new is gained by adding a task of the same relative power.

Using this result, together with other properties of the categorical product such as commutativity and associativity, one can show that the class of loop agreement tasks forms a *join semi-lattice*, with the categorical product as the join operation. The  $(3, 2)$ -set agreement and simplex agreement tasks are the top and bottom of this lattice, respectively.

## 4.5 Concluding Remarks

It is a common technique to study a class of objects by mapping these objects into a class of simpler ones in such a way that preserves enough information about the original class of objects. This is the motivation behind the fundamental group from algebraic topology, and is also the motivation of the algebraic signature of Herlihy and Rajsbaum in their work on loop agreement. In this work we formalized and further extended the algebraic signature characterization by defining the composition of tasks and relating compositions of tasks to products of groups, and in doing so we partially answered the questions raised in the original paper. It remains to be answered how much further this characterization can be extended, or whether there is anything more to be learned from the algebraic signature functor between loop agreement tasks and groups with distinguished elements. One may further in the direction of category theory, such as asking whether the algebraic signature has an adjoint (roughly, inverse).

The categorical techniques in this chapter can be applied to general tasks. For example, tasks with decision maps form a category **Task**, with loop agreement as a subcategory. In the case of loop agreement, we are able to extract valuable information about tasks by mapping them into groups. Another question to be explored is what kind of functors may be applied to general tasks. Also in the case of loop agreement, we were able to identify parallel composition with the category product. It may be possible to define parallel composition for more general tasks, for instance using the complex  $\text{skel}^n(\mathcal{O}_1 \times \mathcal{O}_2)$ .

In this work we only considered colorless tasks, but there could also be value in investigating whether any of the ideas presented here can be generalized or applied to colored tasks, since colored tasks tend to be more useful in practice.

## Chapter 5

# The Convergence Algorithm

The class of loop agreement tasks is a witness showing that task solvability is generally an undecidable problem. However, there are other means of characterizing the solvability of tasks. Herlihy and Shavit's [25, 24] *asynchronous computability theorem* use combinatorial topology to characterize which tasks are solvable in asynchronous read-write memory. The theorem states that a protocol for a task exists if and only if there is a chromatic simplicial map from the input complex to output complex, compatible with the task's specification.

The proof of the asynchronous computability theorem is straightforward for colorless tasks, since it essentially reduces to the *simplicial approximation theorem simplicial approximation* [35, p.89] from classical topology. However when colors (or process names) are introduced, then proving the corresponding asynchronous computability theorem requires an additional difficult step. This is because the simplicial approximation does not make any guarantee about preserving colors.

To construct a color-preserving map required a long construction involving point-set topology, such as  $\epsilon$ -balls and Cauchy sequences. Borowsky and Gafni [5] later proposed an alternative proof strategy for the theorem in which the essential color-preserving property was guaranteed by an algorithm (referred to as the *convergence algorithm*), rather than by a combinatorial construction. The description of this algorithm was sketchy, however, and no proof was provided.

In this chapter, we give a complete description of the convergence algorithm, along with its first proof of correctness. We begin by providing further background on existing results concerning the asynchronous computability theorem, then describe a colored task called *chromatic simplex agreement*, explain how it is solved, and apply it to prove the theorem. The convergence algorithm,

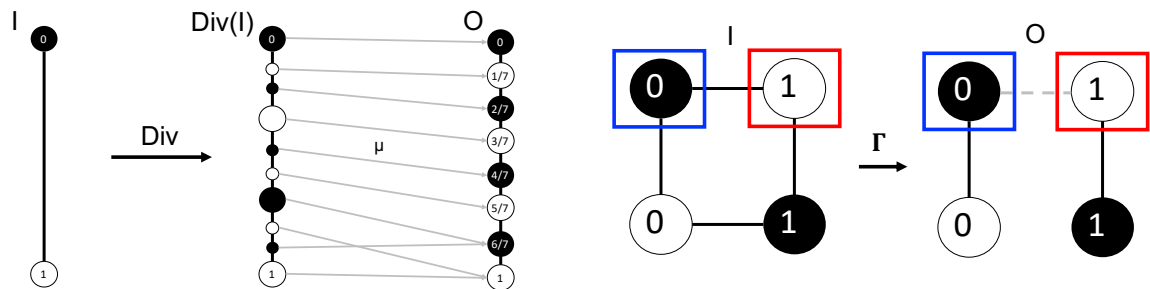
which solves chromatic simplex agreement, is also applicable to results in subsequent chapters in the form of a more broadly applicable corollary.

## 5.1 The Asynchronous Computability Theorem

The asynchronous computability theorem states that a task  $T = (\mathcal{I}, \mathcal{O}, \Gamma)$  has a wait-free protocol in read-write memory if and only if there is a color-preserving, simplicial map from a chromatic subdivision of  $\mathcal{I}$  to  $\mathcal{O}$  that complies with the task specification map  $\Gamma$ . Below we state this theorem more formally.

**Theorem 5.1.1.** *A task  $(\mathcal{I}, \mathcal{O}, \Gamma)$  has a wait-free read-write protocol if and only if there is a chromatic subdivision  $\text{Div}(\mathcal{I})$  and a color-preserving simplicial map  $\mu : \text{Div}(\mathcal{I}) \rightarrow \mathcal{O}$  carried by  $\Gamma$ .*

We give examples showing when the asynchronous theorem applies and when it fails. Consider two different two-process tasks: approximate agreement, in which processes start on opposite ends of an edge and must converge to an edge somewhere in the middle, and binary consensus, as described in Chapter 3.



(a) Approximate agreement is solvable.

(b) Binary consensus is not solvable.

Figure 5.1: Two tasks for which the asynchronous computability theorem applies.

Approximate agreement is a task that is solvable in read-write memory. Accordingly, as we see in Figure 5.1a, there is a subdivision  $\text{Div}$  (in this case given by  $\text{Ch}^2$ ) and a color-preserving map  $\mu$  such that  $\mu : \text{Div}(\mathcal{I}) \rightarrow \mathcal{O}$  complies with the task specification, which only requires that the two

processes meet at an edge. However, in the case of binary consensus, the task requires the black vertex labeled 0 in the input complex to map the similarly labeled vertex in the output, and likewise for the white vertex. But since there is no path connecting the black and white vertexes in the output complex, it is impossible to find a simplicial map that preserves the edge between the white and black vertices in the input complex, no matter how many times the input is subdivided. So binary consensus is not solvable in read-write memory, as is expected.

### 5.1.1 Proof Approaches

In one direction, the claim is relatively easy. If there exists a read-write protocol, then there is also an immediate snapshot protocol for solving the task, due to the equivalence between read-write protocols and immediate snapshot protocols. Now since there is an immediate snapshot protocol for the task  $(\mathcal{I}, \mathcal{O}, \Gamma)$ , this means exactly that there is a color-preserving simplicial decision map  $\delta : \text{Ch}^N(\mathcal{I}) \rightarrow \mathcal{O}$ , compliant with  $\Gamma$ . Therefore  $\text{Ch}^N(\mathcal{I})$  is the desired subdivision and  $\delta$  is the desired simplicial map.

The other direction is more difficult: we must show that given a chromatic subdivision  $\text{Div}(\mathcal{I})$  and a color-preserving map  $\mu : \text{Div}(\mathcal{I}) \rightarrow \mathcal{O}$  carried by  $\text{Div}$ , we can find a read-write protocol solving the task. The most straightforward strategy is to show the existence of a color-preserving simplicial map

$$\phi : \text{Ch}^N(\mathcal{I}) \rightarrow \text{Div}(\mathcal{I}),$$

for some  $N > 0$ , such that for all  $\sigma \in \mathcal{I}$ ,  $\phi(\text{Ch}^N \sigma) \subseteq \text{Div} \sigma$ . If we have such a map  $\phi$ , then we can compose  $\phi$  with the simplicial map  $\mu$  already given to us, as follows:

$$\text{Ch}^N(\mathcal{I}) \xrightarrow{\phi} \text{Div}(\mathcal{I}) \xrightarrow{\mu} \mathcal{O}.$$

These maps can be used to construct a protocol. From an input simplex  $\sigma$ , each process performs the following three steps. First, a process executes an  $N$ -layer immediate snapshot protocol, halting on a vertex  $x$  of the complex  $\text{Ch}^N \sigma$ . Then, it computes another vertex  $y = \phi(x)$ , which is in  $\text{Div} \sigma$ . Finally, it computes  $z = \mu(y)$ , which is the output vertex. This protocol solves the task since both maps  $\phi$  and  $\mu$  are carrier-preserving.

In previous work, a simplicial map  $\phi : \text{Ch}^N(\mathcal{I}) \rightarrow \text{Div}(\mathcal{I})$  was constructed by taking a



simplicial approximation  $\psi : \text{Ch}^N(\mathcal{I}) \rightarrow \text{Div}(\mathcal{I})$ , carried by  $\Delta$ , and then modifying this map to make it preserve colors. The bulk of the original proof is concerned with “perturbing”  $\psi$  to make it color-preserving, a somewhat delicate construction. Borowsky and Gafni suggested a more algorithmic approach: treat this problem as a task,  $(\mathcal{I}, \text{Div} \mathcal{I}, \text{Div})$ , in which processes start on vertices of matching color on a simplex  $\sigma$  of  $\mathcal{I}$ , and halt on vertices of matching color on a single simplex of  $\text{Div}(\mathcal{I})$ . They give a protocol (called the convergence algorithm) that solves this *chromatic simplex agreement* (CSA) task, which induces the desired map. As noted, the original paper lacked a complete description of the algorithm and a proof, both of which are presented here.

## 5.2 Non-chromatic Simplex Agreement

We first consider a more simple task, called non-chromatic simplex agreement (NCSA), in which processes start on the same input complex  $\mathcal{I}$  and must converge to any simplex of  $\text{Div}(\mathcal{I})$ . Processes do not have to land on vertices of any specified color, though they must remain where they are if they run solo. As we will see in section 5.3.3, we use a variation of NCSA, called *link-based* non-chromatic simplex agreement (LNCSA), as a subroutine for defining the convergence algorithm.

Informally, in the NCSA task, the  $n + 1$  processes start on any set of vertices from a given complex, and converge to any  $n$ -simplex, subject to the constraint that any process that executes in isolation must halt on the vertex where it began.

**Definition 5.2.1.** *Let  $\mathcal{K}$  be a complex with vertex set  $V$ , and let  $\Delta(V)$  denote the complex whose simplexes are all the subsets of  $V$ . The  $(n + 1)$ -process non-chromatic simplex agreement task over  $\mathcal{I}$  is the  $(n + 1)$ -process colorless task  $(\text{skel}^n(\Delta(V)), \mathcal{K}, \Gamma)$ , where, for each  $\sigma \in \Delta(V)$ ,  $\Gamma(\sigma) = \{v\}$  if  $\sigma = \{v\}$ , and  $\Gamma(\sigma) = \text{skel}^k(\mathcal{K})$  if  $|\sigma| = k > 1$ .*

NCSA can be solved in read-write memory provided that  $\mathcal{I}$  exhibits sufficient topological connectivity. We present an inductively constructed protocol on skeletons of the input complex, and use the complex’s connectivity to extend the protocol to higher dimensions.

**Theorem 5.2.2.** *If  $\mathcal{K}$  is an  $n$ -dimensional  $(n - 1)$ -connected complex, then there is a wait-free read-write protocol for solving NCSA over  $\mathcal{K}$ .*

*Proof.* Let  $\mathcal{I} = \text{skel}^n(\Delta(V))$ . We inductively define a continuous function

$$f : |\mathcal{I}| \rightarrow |\mathcal{K}|$$

carried by  $\Gamma$ , and then take its simplicial approximation. We induct over skeletons of  $\mathcal{I}$  by defining, for each  $m$ , a continuous map

$$f^m : |\text{skel}^m(\mathcal{I})| \rightarrow |\text{skel}^m(\mathcal{K})|$$

carried by  $\Gamma$ . We begin with the base case by setting  $f^0(v) = v$ , the inclusion of vertices into  $\mathcal{K}$ . As a constant map,  $f^0$  is clearly continuous. Furthermore, it is carried by  $\Gamma$  since the task requires a solo process to remain at the vertex at which it starts. Therefore we have a map satisfying the base case.

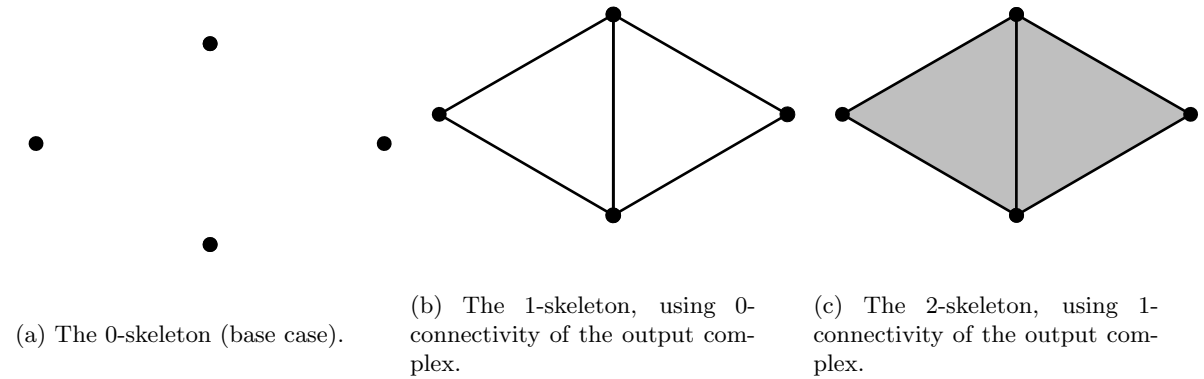


Figure 5.2: Building up the skeletons of the input complex. Connectivity of the output complex is used to extend the continuous map with each step.

Now inductively assume we have constructed a continuous map

$$f^m : |\text{skel}^m \mathcal{I}| \rightarrow |\mathcal{K}|.$$

Let  $\{\sigma_i\}$  be the set of facets of  $\text{skel}^{m+1}(\mathcal{I})$ , and let  $\partial\sigma_i$  denote the boundary of  $\sigma_i$ , which is equivalently its set of facets. Since  $\mathcal{K}$  is  $m$ -connected, we can extend each  $f^m|_{\partial\sigma_i}$  to a function

$$f_i^{m+1} : |\sigma_i| \rightarrow |\text{skel}^{m+1}(\mathcal{K})|.$$

Since adjacent  $f_i^{m+1}$  agree on their intersection by construction, we can use the pasting lemma (see 3.2.8), we can glue all these maps together. This yields a combined map  $f^{m+1}$  on all of  $\text{skel}^m(\mathcal{I})$ . Moreover, it extends the map  $f^m$  and is carried by  $\Gamma$ . Therefore by induction on skeletons, we obtain a map

$$f^n = f : |\mathcal{I}| \rightarrow |\mathcal{K}|,$$

carried by  $\Gamma$ . Finally, we take a simplicial approximation of  $f$  to get a simplicial map

$$\phi : \text{Ch}^k(\mathcal{I}) \rightarrow \mathcal{K}$$

By the simplicial approximation theorem,  $\phi$  is also carried by  $\Gamma$ , so it is a decision map that solves the NCSA task.  $\square$

## 5.3 The Convergence Algorithm

Suppose we want to solve chromatic simplex agreement on a subdivided complex  $\text{Div}(\mathcal{I})$ . In this section, we describe the convergence algorithm for solving chromatic simplex agreement. The algorithm proceeds in rounds, with participating processes beginning on a simplex of  $\mathcal{I}$ . The processes gradually converge to one simplex of the subdivision, with at least one process make its decision with each round.

### 5.3.1 Solving Chromatic Simplex Agreement

Algorithm 1 shows pseudocode for the convergence algorithm, which is informally explained below. Then in the section to follow, we more formally described all auxiliary information that each process computes and tracks during execution of the convergence algorithm. Finally, prove correctness of the algorithm.

Participating processes begin on a simplex of  $\mathcal{I}$ . In the first round, the processes to run a simplex agreement protocol on  $\text{Bary}(\text{Div}(\mathcal{I}))$ . Recalling the construction of the barycentric subdivision, this corresponds to the processes collectively choose a nested sequence of simplexes in  $\text{Div}(\mathcal{I})$ . Each process then writes its simplex obtained this way to shared memory array, and takes an immediate snapshot of the array, in order to observe simplexes written by other participating processes. Within the obtained snapshot of simplexes, if a process sees a vertex of its color in all such simplexes, then

the process returns that vertex and finishes the convergence algorithm. Otherwise, it computes its *view*<sup>1</sup> as the union of all simplexes it saw in the snapshot, minus the vertex of its own color, if one exists. The process proceeds to the next round.

Inductively, a process entering round  $r > 1$  has already computed a view, as in the first round described above. Each process begins round  $r$  by writing this view to a shared memory array. Then it takes an immediate snapshots of the array from this round, simultaneously with taking a snapshot from the *first* round. By retaking a snapshot of the first round, a process updates the *participating set* of processes it observes running the protocol. Each process maintains what it believes to be the participating set of processes through its entire execution. Next, after taking a snapshot of the views from the *current* round, the process computes its *core*<sup>2</sup>, defined to be the intersection of the views from the current round's snapshot.

Using these two snapshots, the process computes its *convergence complex*, a simplicial neighborhood of the core (defined in the next section). Note that processes typically construct different convergence complexes, though all such complexes will be ordered by inclusion. Each process then chooses a *starting vertex* of its own color in its convergence complex, and runs the LNCSA protocol on the barycentric subdivision of its convergence complex. Processes collectively obtain a nested sequence simplexes, similar to the first round. Each process then replaces its core with the smallest core observed when running LNCSA. Each process writes its simplex and its new core to shared memory and takes a snapshot. If a process sees a vertex of its color in each simplex it saw, it decides on that vertex and finishes. Otherwise, it updates its view and proceeds to the next round.

---

<sup>1</sup> Borowsky and Gafni called the view the “core”.

<sup>2</sup> Borowsky and Gafni called this the “intersection of cores”.



```

shared participating[ $n+1$ ];
shared views[ $n+1$ ];
shared simplexes[ $n+1, n+1$ ];
protocol chromaticSimplexAgree( $p, v, \mathcal{I}, \text{Div}$ ):
  participating[ $p$ ] :=  $p$ ;
   $s := \text{simplexAgree}(v, \mathcal{I}, \text{Div})$ ;
  immediate
    simplexes[ $1, p$ ] := ( $s, \emptyset$ );
     $\text{snap} := \text{snapshot}(\text{simplexes}[1])$ ;
    if  $\exists u : u \in \bigcap_{t \in \text{snap}} t$  and  $\chi(u) = p$  then
      | return  $u$ ;
    else
      |  $\text{toss} := \{u : u \in \bigcup_{t \in \text{snap}} t \text{ and } \chi(u) = p\}$ ;
      |  $w := \bigcup_{t \in \text{snap}} t - \text{toss}$ ;
   $r := 2$ ;
  while True do
    immediate
      | views[ $r, p$ ] :=  $w$ ;
      |  $\text{snap} := \text{snapshot}(\text{views}[r])$ ;
      |  $P := \text{snapshot}(\text{participating})$ ;
     $c := \bigcap_{x \in \text{snap}} x$ ;
     $s, \bar{c} := \text{linkNonchromaticSimplexAgree}(v, c, P, \mathcal{I}, \text{Div})$ ;
    immediate
      | simplexes[ $r, p$ ] := ( $s, \bar{c}$ );
      |  $\text{snap} := \text{snapshot}(\text{simplexes}[r])$ ;
    if  $\exists u : u \in \bigcap_{t \in \text{snap}} t$  and  $\chi(u) = p$  then
      | return  $u$ ;
    else
      |  $\text{toss} := \{u : u \in \bigcup_{t \in \text{snap}} t \text{ and } \chi(u) = p\}$ ;
      |  $w := \bigcup_{t \in \text{snap}[0]} t \cup \bigcap_{d \in \text{snap}[1]} d - \text{toss}$ ;
      |  $r := r + 1$ ;

```

**Algorithm 1:** The convergence algorithm

### 5.3.2 Bookkeeping

If each process executes Algorithm 1, it will decide on a vertex of its color in a simplex on  $\text{Div}(\mathcal{I})$ , thus solving CSA over  $\mathcal{I}$ . In this section we establish terminology used in proving correctness of the convergence algorithm.

Here is some notation to track all information that each individual process computes during its run. Let  $p_0, \dots, p_n$  be the processes participating in the convergence algorithm. Let  $p$  be any such process. For each round  $r > 1$ , the state of process  $p$  consists of the following: its *participating set*  $P_p^r$ , its *core*  $c_p^r$ , its *convergence complex*  $\mathcal{C}_p^r$ , its *starting vertex*  $v_p^r$ , its *simplex*  $s_p^r$ , and its *view*  $w_p^r$ . Here is how each of these items is computed during round  $r$  of the convergence algorithm:

1. The *participating set*  $P_p^r$  is the set of processes (and corresponding input vertices) that process  $p$  sees when it takes a snapshot of the first round's shared memory. This is the set of processes which  $p$  believes (by round  $r$ ) to be executing the algorithm, and only increase with each subsequent round.
2. The *core*  $c_p^r$  is the set of vertices that may be decision values of processes (other than  $p$ ) that have finished the executing the convergence algorithm. It is defined to be

$$c_p^r = \bigcap_{j \in J} w_{p_j}^r,$$

where  $J$  is the index set of processes, seen by  $p$ , in its snapshot of views from the current round  $r$ . During the execution of the LNCSA protocol, the process  $p$  may observe smaller cores, in which case  $p$  recomputes its own core as

$$\bar{c}_p^r = \bigcap_{k \in K} c_{p_k}^r$$

where  $I$  is the index set of processes seen by  $p$  during the LNCSA protocol.

3. The *convergence complex*  $\mathcal{C}_p^r$  is the complex on which  $p$  runs the LNCSA protocol, in order to choose a new simplex consistent with decision values already determined by other processes. It is computed as

$$\mathcal{C}_p^r = \text{Lk}\left(\bigcap_{k \in K} c_{p_k}^r, \bigcup_{k \in K} P_{p_k}^r\right).$$

This is the link of the smallest observed core, contained in the complex determined by the largest observed participating set.

4. The *starting vertex*  $v_p^r$  is the vertex on which  $p$  begins running the LNCSA protocol. Any vertex from  $C_p^r$  with the same color as  $p$  may be chosen. Starting vertices, in addition to process names, are contained in participating sets.
5. The *simplex*  $s_p^r$  is the simplex in  $C_p^r$  which  $p$  chooses as a result of running the LNCSA protocol. They collectively represent the domain of values from which processes may choose decision values in round  $r$ .
6. The *view*  $w_p^r$  is the set of vertices that  $p$  sees in round  $r$ . It is computed as

$$w_p^r = \bigcup_{i \in I} s_{p_i}^r \cup \bigcap_{i \in I} \bar{c}_{p_i}^r - \{u_p^r\}$$

where  $I$  is the index set of processes seen by  $p$  in its snapshot of simplexes and cores, and  $u_p^r$  is the vertex with the same color as  $p$ , if it exists. It is a union of observed simplexes, together with the smallest core, minus a vertex of your own color.

### 5.3.3 Link-based Non-chromatic Simplex Agreement

We have mostly described the convergence algorithm, with the exception of how to solve the LNCSA task. In this section we provide more details of LNCSA, how it is solved, and how it fits into the bigger picture of the convergence algorithm.

Recall that the convergence algorithm should solve the task  $(\mathcal{I}, \text{Div}(\mathcal{I}), \text{Div})$ , known as chromatic simplex agreement. Consider an execution of the convergence algorithm, specifically at the point in time where processes have just computed their convergence complexes, in some given round. We want these processes to collectively converge to a simplex that is consistent with the smallest core among them, since this core represent decision values of processes that have finished the algorithm. They should do this by selecting vertices on the largest convergence complex, since the largest one is the link of the smallest core. This observation is at the heart of LNCSA task.

Before we can properly define LNCSA, there are a few technical lemmas required to proceed. The first lemma notes that all views and cores, defined only as sets of vertices until now, are indeed



simplexes of the subdivision  $\text{Div}(\mathcal{I})$ . This lemma allows us to, for example, compute links of cores and define the convergence complex.

**Lemma 5.3.1.** *All views and cores are simplexes.*

*Proof.* This is a proof by induction on the numbers of rounds in the convergence algorithm. It is clear that views from the first round are simplexes, since each one is a subset of the largest simplex chosen during simplex agreement. By downward closure of complexes, subsets of simplexes are also simplexes.

The cores computed in the second round are also simplexes, since they are intersections of the views from the first round. Intersections of simplexes are simplexes.

Inductively assume that all views from round  $r$  are simplexes. Clearly all cores  $c_p^{r+1}$  in round  $r+1$  are simplexes as well, since they are intersections of views from round  $r$ . Views in round  $r+1$  are computed as

$$w_p^{r+1} = \bigcup_{i \in I} s_{p_i}^{r+1} \cup \bigcap_{i \in I} \bar{c}_{p_i}^{r+1} - \{u_p^{r+1}\}$$

Now consider a process  $p$ , and let  $I$  denote the index set of processes contained in snapshot of simplexes taken by  $p$ . Then we know that there is a largest simplex among processes in the snapshot, or that

$$\bigcup_{i \in I} s_{p_i}^{r+1} = s_{p_\ell}^{r+1}$$

for some  $\ell \in I$ , since the simplexes are ordered by inclusion. Then  $s_{p_\ell}^{r+1}$  is the largest simplex seen by  $p$  in its snapshot. Furthermore, we know that  $s_{p_\ell}^{r+1} \cup \bar{c}_{p_\ell}^{r+1}$  is a simplex, by definition of the link of a simplex, since  $s_{p_\ell}^{r+1}$  is chosen from the link of the core. But the intersection of all cores is contained in the core of  $p$ , or

$$\bigcap_{i \in I} \bar{c}_{p_i}^{r+1} \subseteq \bar{c}_{p_\ell}^{r+1}$$

so from the definition of how view  $r+1$  is computed, we get the containment

$$w_p^{r+1} = \bigcup_{i \in I} s_{p_i}^{r+1} \cup \bigcap_{i \in I} \bar{c}_{p_i}^{r+1} - \{u_p^{r+1}\} \subseteq s_{p_\ell}^{r+1} \cup \bar{c}_{p_\ell}^{r+1}.$$

We just showed that the expression on the righthand side is a simplex, so by downward closure, we conclude that  $w_p^{r+1}$  is a simplex.

We have shown that the views and cores in round  $r+1$  are also simplexes. Therefore by induction,

all views and cores are simplexes.  $\square$

We are now able to appropriately compute convergence complexes, since we can take links of cores due to cores being simplexes. The next lemma states that the set of all convergence complexes of the processes is ordered by inclusion. In some sense, this ensures that the LNCSA convergence subtasks solved by different processes are coherent, even though each process may have computed a different convergence complex.

**Lemma 5.3.2.** *The convergence complexes of participating processes for any given round are ordered by inclusion.*

*Proof.* Consider any round  $r$ . If exactly one or fewer processes have not decided, then the claim is trivial. So let  $p_1$  and  $p_2$  be distinct processes that have not decided, and let  $\mathcal{C}_i$  denote the convergence complex for  $p_i$ . We must show a containment relationship between  $\mathcal{C}_1$  and  $\mathcal{C}_2$ . Without loss of generality, suppose  $p_1$  takes a snapshot of the view arrays for rounds 1 and  $r$  before  $p_2$ . Then  $p_2$  sees more views than  $p_1$ , so it computes a larger intersection for its core. In other words, this meaning that  $c_2 \subseteq c_1$ , where  $c_i$  is the core of  $p_i$  in the current round  $r$ . Furthermore, if  $Q_i$  is the set of participating processes seen by  $p_i$  (or those in the snapshot from the first round), then  $Q_1 \subseteq Q_2$ .

Since the link operator is order reversing in the first argument, and order preserving in the second, we get the following inclusions:

$$\mathcal{C}_1 = \text{Lk}(c_1, Q_1) \subseteq \text{Lk}(c_2, Q_2) \subseteq \text{Lk}(c_2, Q_2) = \mathcal{C}_2$$

We conclude that  $\mathcal{C}_1 \subseteq \mathcal{C}_2$ . Therefore the convergence complexes of any two processes participating in the LNCSA protocol for round  $r$  are comparable, so the set of all convergence complexes is ordered by inclusion.  $\square$

We now have a complex (or set of complexes) on which processes converge and run an LNCSA protocol. The next lemma allows each process to choose a valid starting vertex before beginning the protocol. In particular, for any process, there is a vertex of its own color in its convergence complex.

The next lemma allows each process to pick a vertex of its own color in its convergence complex when it begins solving LNCSA. This is not immediate since convergence complexes generally have a smaller dimension than that of the full complex.

**Lemma 5.3.3.** *Let  $p$  be a process has not finished executing the convergence algorithm by round  $r$ . Then there is at least one vertex of its color in its convergence complex  $\mathcal{C}_p^r$ .*

*Proof.* Recall that  $c_p^r$  is the core of  $p$  in round  $r$ . Since the convergence complex  $\mathcal{C}_p^r$  is a link of  $c_p^r$ , it is enough to show that no vertex in  $c_p^r$  has the same color as  $p$ . This is because  $\mathcal{C}_p^r$  and  $c_p^r$  are colored by complementary sets of colors.

The core  $c_p^r$  is computed as the intersection of views that  $p$  sees in its snapshot of the shared array of views in round  $r$ . This snapshot must include  $w_p^{r-1}$ , which is the view of  $p$  itself. But  $w_p^{r-1}$  cannot contain a vertex of color  $p$ , since any such vertex is explicitly removed in the computation of  $w_p^{r-1}$ . Since  $w_p^{r-1}$  does not contain a vertex of color  $p$ , neither does  $c_p^r$  since it is the intersection of all views from this round. Hence the core does not contain a vertex of color  $p$ . Since  $\mathcal{C}_p^r$  is its link in a chromatic complex, it must contain at least one vertex of color  $p$ .  $\square$

We now have a complex for LNCSA and vertices on which each process may take as inputs. The next lemma is a more technical one, and constrains how decision values obtained in previous rounds appear in views and cores of other processes in later rounds.

**Lemma 5.3.4.** *Suppose a process decides on a vertex. Then that vertex is contained in the views and cores of all processes in all subsequent rounds.*

*Proof.* Suppose process  $p$  decides on vertex  $u$  in round  $r_p$ . We proceed by induction on the number of rounds greater than  $r_p$ , first showing that all views contain vertex  $u$ . The fact that cores contain  $u$  as well follows as a simple corollary.

As the base case, we first show that every view computed in round  $r_p$  itself contains vertex  $u$ . Let  $p'$  be any other process that does not decide this round. Then there are two cases for process  $p'$  when it computes its new view:

1. Process  $p'$  sees the simplex that  $p$  wrote to shared memory,
2. Process  $p'$  does not see this simplex.

In the first case,  $p'$  sees the simplex that  $p$  wrote. Let this simplex be  $\sigma$ . Since the convergence algorithm stipulates that a process only decides when it sees a vertex of its color in every simplex, and  $p$  decides in this round, then simplex  $\sigma$  must contain vertex  $u$ . Otherwise  $p$  would not decide in this round. So  $p'$  includes  $\sigma$  in the computation of its view, which includes the union of all simplexes observed in the snapshot. Therefore the view of  $p'$  in round  $r_p$  contains  $u$ .

Now we consider the second case, where  $p'$  does not see the simplex that  $p$  wrote. But for  $p$  to decide on  $u$ ,  $u$  must have been contained in all simplexes it observed in its snapshot, since this is again the requirement for a process to decide. In particular, since  $p'$  did not see the simplex written by  $p$ , it must be that  $p'$  preceded  $p$  in writing its simplex to the array. Therefore  $p$  must have observed the simplex written by  $p'$ . Therefore the simplex written by  $p'$  must have contained vertex  $u$ . including the simplex of  $p'$ . Since this simplex contains  $u$ , the view of  $p'$  will also contain  $u$ .

In either case, the view of  $p'$  contains  $u$ , and since  $p'$  was taken to be arbitrary, the same holds for any other process that does not decide by round  $r_p$ .

Next, we inductively assume that all views from some round  $r \geq r_p$  contain vertex  $u$ . We want to show that all views from the next round  $r + 1$  also contain  $u$ . First, consider the cores computed in round  $r + 1$ . Since cores are intersections of views from the previous round  $r$ , and all previous views contain  $u$ , it follows that all cores computed in round  $r + 1$  also contain  $u$ . Furthermore, recall how any process  $p'$  computes its view in round  $r + 1$ . We have:

$$w_{p'}^{r+1} = \bigcup_{i \in I} s_{p_i}^{r+1} \cup \bigcap_{i \in I} \bar{c}_{p_i}^{r+1} - \{u_{p'}^r\}.$$

This includes the intersection of all cores. But since each core contains  $u$ , we get the following containment:

$$u \in \bigcap_{i \in I} \bar{c}_{p_i}^{r+1} \subseteq w_{p'}^{r+1}$$

so the view of  $p'$  computed in round  $r + 1$  also contains  $u$ . Therefore all views in round  $r + 1$  also contain  $u$ .

We have shown that assuming all views in round  $r \geq r_p$  contain vertex  $u$ , this implies that all views in round  $r + 1$  also contain  $u$ . Hence by induction, all views in all rounds after  $r_p$  contain  $u$ .

Finally, since cores are computed as intersections of views in each round, all cores in rounds  $r > r_p$  must also contain  $u$ . This finishes the proof.  $\square$

Intuitively, the above lemma states that decision values are stable as the convergence algorithm proceeds.

### Defining LNCSA

We now have all the parts required to define the LNCSA task. Recall that  $(\mathcal{I}, \text{Div}(\mathcal{I}), \text{Div})$  is the chromatic simplex agreement task we want to solve via the convergence algorithm. Take any round  $r$ , and suppose the processes have just computed their links. We describe the LNCSA they will solve as the next step of the algorithm.

Informally, in repeatedly solving the LNCSA task, the processes converge to a simplex that is consistent with the smallest core among them, or vertices of the decided processes. Stated in a different way, these processes will converge on the largest link among the processes.

We proceed with a formal definition of the LNCSA task, beginning with process inputs to the task. As usual, process inputs are represented with a simplicial complex, denote  $\mathcal{A}$ , with vertices representing state. Each process's input state consists of the following data:

1. Core  $c$ ,
2. Participating set  $P$ ,
3. Starting vertex  $v$  chosen from the convergence complex.

Note that by Lemma 5.3.3, each process is able to choose a valid starting vertex on its convergence complex. This set of input data is expressed compactly as a triple  $(v, c, P)$ , with the implicit requirements that that  $v \in P$  and  $\text{Car}(c, \mathcal{I}) \subseteq P$ .

We have a vertex set for the input complex, so now we define its simplexes. A set of triples (or vertices)  $\{(v_i, c_i, P_i)\}$  is a simplex in  $\mathcal{A}$  if the  $v_i$  all have distinct colors, and  $\{c_i\}$  and  $\{P_i\}$  are ordered by inclusion. We require the  $v_i$  to have different colors, since each process must choose a vertex of its own color. Furthermore, since cores and participating sets are computed using immediate snapshots, the sets  $\{c_i\}$  and  $\{P_i\}$  are ordered by inclusion. So the LNCSA task will only receive inputs that already satisfy these constraints. So  $\mathcal{A}$  is a subcomplex of

$$\Delta(V) \times \text{Bary}(\text{Div}(\mathcal{I})) \times \text{Bary}(\mathcal{I}),$$

where  $V$  is the vertex set of  $\mathcal{I}$ . The barycentric subdivision in the above expression correspond to sets of simplexes ordered by inclusion. The second component's domain consists of simplexes from  $\text{Div}(\mathcal{I})$  since cores are simplexes in this subdivision. The third component's domain consists of simplexes from  $\mathcal{I}$  since participating sets are just sets of processes.

We have described the input complex, so we next describe the output complex of LNCSA. This complex is given by  $\mathcal{B} = \text{Bary}(\text{Div}(\mathcal{I}))$ . This is because processes should collectively choose a set of simplexes that are ordered by inclusion.

Having defined the input and output complexes, we now define the task specification, which is a carrier map between the two complexes. Processes that execute in isolation stay where they are. Otherwise, processes converge to a simplex in the largest link they see. So if  $\sigma = \{(v_i, c_i, P_i)\}$  is a simplex in  $\mathcal{A}$ , then we have

$$\Gamma(\sigma) = \begin{cases} \{v\} & \text{if } \sigma = \{(v, c, P)\} \\ \text{Lk}(\bigcap c_i, \bigcup P_i) & \text{otherwise} \end{cases}$$

But Lemma 5.3.2 states that convergence complexes are ordered by inclusion, which implies that  $\Gamma$  is monotonic. This makes  $\Gamma$  a carrier map.

We have defined the input complex  $\mathcal{A}$ , output complex  $\mathcal{B}$ , and task specification  $\Gamma$  of the LNCSA task. We state this as a definition below.

**Definition 5.3.5.** *Let  $\mathcal{A}$ ,  $\mathcal{B}$ , and  $\Gamma$  be complexes and carrier map as described above. Then LNCSA is the colorless task given by the triple  $(\mathcal{A}, \mathcal{B}, \Gamma)$ .*

We have a formal definition of the LNCSA task. Note the similarity of this task with the NCSA task from the previous section, where processes converge on a subdivision but are not required to choose vertices of their own color.

We provide an immediate snapshot protocol for solving the LNCSA task.

**Lemma 5.3.6.** *Let  $r$  be any round in the convergence algorithm, after which the processes have computed their cores. Then there is a wait-free immediate snapshot protocol for converging to a simplex on  $\text{Lk}(\bigcap_{k \in K} c_{pk}^r, \bigcup_{k \in K} P_{pk}^r)$ . In other words, the task  $(\mathcal{A}, \mathcal{B}, \Gamma)$  is solvable.*

*Proof.* We show that this task is solvable by constructing a continuous map

$$f : |\mathcal{A}| \rightarrow |\mathcal{B}|$$

carried by  $\Gamma$ . Once we have such a map, we can apply the simplicial approximation theorem to produce a carrier-preserving map, thereby showing the task is solving since we will have produced a

decision map. We induct on skeletons of the input complex in order to construct such a continuous  $f$ . More specifically, for each  $m$ , we build up continuous maps

$$f^m : |\text{skel}^m(\mathcal{A})| \rightarrow |\mathcal{B}|$$

defined on successive skeletons of the input complex. As the base case, we define  $f^0$  as  $f^0((v, c, P)) = v$  for each vertex  $(v, c, P)$  of  $\mathcal{A}$ . The function  $f^0$  is clearly continuous and carried by  $\Gamma$ , since  $f^0$  is a constant map and  $\Gamma$  requires solo processes to stay where they are. Now inductively assume we have defined  $f^m$ . We want to define an extension of the map  $f^m$  to the next skeleton, or

$$f^{m+1} : |\text{skel}^{m+1}(\mathcal{A})| \rightarrow |\mathcal{B}|.$$

Let  $\{\sigma_i\}$  be some enumeration of the facets of  $\text{skel}^{m+1}(\mathcal{A})$ . For each  $i$ , we index the vertices of each simplex as  $\sigma_i = \{(v_{ij}, c_{ij}, P_{ij})\}_{j \leq m+2}$ .

Given this indexing, we calculate the smallest core and largest participating set among the processes, in order to determine the largest convergence complex among them. These cores and participating sets are computed as

$$\bar{c}_i = \bigcap_j c_{ij} \quad \text{and} \quad \bar{P}_i = \bigcup_j P_{ij}.$$

This is again due to the fact that cores and participating sets are ordered by inclusion.

By the inductive hypothesis on the map  $f^m$ , we know that  $f^m$  is carried by  $\Gamma$ . Looking at the definition of  $\Gamma$ ,  $f^m$  being carried by  $\Gamma$  exactly means that its image is contained in the convergence complex

$$\mathcal{L}_i = \text{Lk}(\bar{c}_i, \bar{P}_i).$$

Furthermore, we also know that  $\dim(\mathcal{L}_i) \geq \dim(\sigma_i) = m$ , since each  $v_{ij} \in \mathcal{L}_i$  as the processes' starting vertices, each  $v_{ij}$  has different color. and the fact that  $\mathcal{L}_i$  is chromatic. Suppose  $\dim(\bar{P}_i) = n$ . We know that  $\mathcal{L}_i$  a link of a simplex in the subdivided simplex  $\text{Bary}(\text{Div}(\bar{P}_i))$ . As a subdivided simplex, the latter complex is link-connected. Therefore, using link-connectivity of  $\text{Bary}(\text{Div}(\bar{P}_i))$

and the fact that  $\dim(\mathcal{L}_i) \geq m$ , we conclude that  $\mathcal{L}_i$  has connectivity of degree at least

$$n - 2 - (n - m - 1) = m - 1$$

Using this observation, along with the fact that  $\text{Im}(f^m) \subseteq \mathcal{L}_i$ , we can extend  $f^m|_{\partial\sigma_i}$  to a function  $f_i^{m+1} : |\sigma_i| \rightarrow |\mathcal{L}_i|$ , which is possible since  $\partial\sigma_i$  is homeomorphic to a sphere. So we have extended the map  $f^m$  to every individual simplex of the  $(m + 1)$ -skeleton, however these maps need to be pieced together to form one map.

Using the pasting lemma, we can glue together the  $f_i^{m+1}$  defined on simplex  $\sigma_i$  to obtain a continuous function

$$f^{m+1} : |\text{skel}^{m+1}(\mathcal{A})| \rightarrow \mathcal{B}$$

which extends the original  $f^m$  on its entire domain. By construction,  $f^{m+1}$  is carried by  $\Gamma$ , since its image lands in the  $(m + 1)$ -skeleton of the output complex. So by induction, we have obtained a continuous function  $f : |\mathcal{A}| \rightarrow |\mathcal{B}|$  defined on the whole input complex. Finally, by the simplicial approximation theorem, we get a simplicial map  $\delta$  that is also carried by  $\Gamma$ . We conclude that the task  $(\mathcal{A}, \mathcal{B}, \Gamma)$  is wait-free solvable using immediate snapshots.  $\square$

We now have a protocol for the LNCSA task, which completes the description of the convergence algorithm. We must finally show that the convergence algorithm itself is a correct protocol for solving the chromatic simplex agreement task.

### 5.3.4 Termination and Validity

We now have all the necessary results to prove correctness of the convergence algorithm. In proving correctness of the algorithm, the two main properties to prove are its *termination* and *validity*. Showing that all non-faulty processes terminate is essential to wait-freedom of the algorithm, while validity ensures that each process decides on a vertex compliant with the task specification.

We begin by showing termination, or that all processes participating in the algorithm eventually decide. The intuition here is that with each round, at least one process is able to decide on a vertex, since there is at least one process that sees a valid decision vertex contained in the view of all other processes.

**Theorem 5.3.7.** *All processes participating in the convergence algorithm eventually decide.*



*Proof.* We show that at least one process decides each round. Consider any round in the convergence algorithm, and let  $\{p_0, \dots, p_\ell\}$  be the set of processes participating in the algorithm. The processes begin this round by each of them running the LNCSA protocol over the barycentric subdivisions of their convergence complexes. By Lemma 5.3.6, the processes collectively converge to a simplex  $\tau$  on the largest subcomplex, say  $\text{Bary}(\mathcal{C})$ , where each process computes some subsimplex of  $\tau$ . That is, by definition of the barycentric subdivision, the simplex  $\tau \in \text{Bary}(\mathcal{A}_{i_k})$  corresponds to a chain of simplexes, given by

$$\sigma_{i_0} \subseteq \dots \subseteq \sigma_{i_\ell}$$

Their intersection is  $\sigma_{i_0}$ , which is clearly nonempty, so choose a vertex  $\hat{v} \in \sigma_{i_0}$ . Then the color of  $\hat{v}$ , which we denote as  $\chi(\hat{v})$ , cannot be the color of any non-participating process. To see this, note that all convergence complexes  $\text{Bary}(\mathcal{C})$  are all subcomplexes of the processes' carrier in  $\text{Div}(\mathcal{T})$ , and this subcomplex is colored by processes of only those that are participating in the algorithm.

Neither can  $\chi(\hat{v})$  be the color of a process that decided in a previous round. Towards a contradiction, suppose not. Then this means the largest convergence complex contains vertices of color  $\chi(\hat{v})$ , meaning that the corresponding core could not contain a vertex of color  $\chi(\hat{v})$ , since the link contains vertices of color exactly those not in the corresponding core. This contradicts Lemma 5.3.4, since decided vertices must be contained in all cores.  $\times$

Let  $\hat{p} \in P$  be the process whose color is  $\chi(\hat{v})$ , and let  $K \subseteq \{0, \dots, \ell\}$  be the index set of processes that  $\hat{p}$  saw during its snapshot of the simplex array. Then

$$\bigcap_{k \leq \ell} \sigma_{i_k} \subseteq \bigcap_{k \in K} \sigma_{i_k},$$

so  $\hat{v} \in \bigcap_{k \in K} \sigma_{i_k}$ . Since  $\hat{p}$  sees  $\hat{v}$  in all simplexes in its snapshot, then according to the algorithm,  $\hat{p}$  decides on  $\hat{v}$ . We have therefore show that at least one process  $\hat{p}$  decides in this round, for any round taken arbitrary. From this it follows that every process participating in the algorithm must decide in at most  $n + 1$  rounds. □

This proves that the convergence algorithm terminates, hence is wait-free. Finally, we show that the set of vertices chosen by the processes comply with the task specification of chromatic simplex

agreement on  $\text{Div}(\mathcal{I})$ , which we state in the following way.

**Theorem 5.3.8.** *Participating processes converge to a simplex in their carrier, and each process chooses a vertex of its own color.*

*Proof.* We show that for each round  $r$ , the set of vertices that processes have decided form a simplex. We argue by induction, beginning with the first round. In the first round, it is clear that decision values form a simplex, since processes can only decide on vertices from the largest simplex chosen by solving the first LNCSA. Any subset of this largest simplex is clearly also a simplex, so the set of decision values from the first round (which must be of size at least one) is a simplex. This shows the base case holds.

Inductively, we consider a round  $r$ , and we inductively assume that the vertices which processes decided during the rounds up to  $r$  form a simplex, which we call  $\tau_r$ . Consider round  $r + 1$ . Processes in this round choose vertices on a simplex of the barycentric subdivision of the largest convergence complex, say  $\text{Bary}(\mathcal{C})$ , which is determined by the smallest core. So vertices on which processes decide this round are contained in an ascending chain of simplexes in  $\mathcal{C}$ . Call the largest such simplex  $\Sigma_{r+1}$ , and let  $c_{r+1}$  be the smallest core, corresponding to the largest convergence complex. Then by definition of the link,  $c_{r+1} \cup \Sigma_{r+1}$  must be a simplex in  $\mathcal{C}$ , since the largest link is determined by the smallest core  $c_r$  in round  $r$ . From Lemma 5.3.4, we must have  $\tau_r \subseteq c_{r+1}$ , since all decision values are contained in subsequent cores and views. So  $\tau_r \cup \Sigma_{r+1}$  is also a simplex in  $\mathcal{C}$ , by downward closure of simplicial complexes. Let  $\sigma_{r+1} \subseteq \Sigma_{r+1}$  be the set of all vertices that processes on decide during round  $r + 1$ . Again by downward closure,

$$\tau_r \cup \sigma_{r+1} \subseteq \tau_r \cup \Sigma_{r+1},$$

so  $\tau_r \cup \sigma_{r+1}$  is a simplex. But by definition of  $\tau_r$ , we have that  $\tau_{r+1} = \tau_r \cup \sigma_{r+1}$ , where  $\tau_{r+1}$  is exactly the set of vertices on which processes have decided up to round  $r + 1$ . So  $\tau_{r+1}$  is also a simplex. By induction on the number of rounds, it follows that the processes' decision values form a simplex.

Processes can only choose decision values in their carrier, since all links are computed relative to any observed participating sets, and all decision values are chosen from their links (of cores). That is, at no point does a process ever jump outside of the participating processes' carrier. Furthermore, each process only ever decides on a process of its own color, as stipulated by the convergence

algorithm.

Therefore the processes collectively converge to a simplex in  $\text{Div}(\mathcal{I})$ , and each chooses a vertex of its own color. This completes the proof of validity,  $\square$

Having proven termination and validity of the convergence algorithm, the correctness of the algorithm follows. So the convergence algorithm solves chromatic simplex agreement. Since the algorithm is implemented entirely in read-write memory, then from the correspondence between solvability of immediate snapshots and existence of a decision map, we get a color-preserving simplicial map  $\phi : \text{Ch}^N(\mathcal{I}) \rightarrow \text{Div}(\mathcal{I})$ . From this fact, the asynchronous computability theorem follows immediately. Recall from Section 5.1 the more difficult direction of theorem, in which we are given a color-preserving simplicial map  $\mu : \text{Div}(\mathcal{I}) \rightarrow \mathcal{O}$ . Then composing  $\mu$  and  $\phi$  yield a decision map for the general task  $(\mathcal{I}, \mathcal{O}, \Gamma)$ .

## 5.4 Application to General Tasks

The chromatic simplex agreement task over a chromatic subdivision  $\text{Div}(\mathcal{I})$  is defined as the task  $(\mathcal{I}, \text{Div}(\mathcal{I}), \text{Div})$ . The proof of the convergence algorithm shows that this task has a wait-free read-write protocol. To enable the convergence algorithm to work, we required that  $\text{Div}(\mathcal{I})$  be link-connected. This was an important connectivity property for links within the subdivided simplex, since it allowed us to solve LNCSA over iteratively smaller convergence complexes. Phrased in more topological and combinatorial terms, the convergence algorithm allows us to find a color-preserving simplicial map

$$\phi : \text{Ch}^N(\mathcal{I}) \rightarrow \text{Div}(\mathcal{I})$$

carried by  $\text{Div}$ , one that approximates the continuous (identity) map  $\text{id} : |\mathcal{I}| \rightarrow |\text{Div}(\mathcal{I})|$  also carried by  $\text{Div}$ . In fact, link-connectivity is the essential property of the output complex for this argument to go through. In particular, the convergence algorithm may be applied to more general continuous functions  $f : |\mathcal{I}| \rightarrow |\mathcal{O}|$  carried by certain well-behaved  $\Gamma$ . If one is given the assumption that  $\Gamma(\sigma)$  is link-connected for all  $\sigma \in \mathcal{I}$ , we can use the convergence algorithm to obtain a chromatic simplicial map  $\phi : \text{Ch}^N(\mathcal{I}) \rightarrow \mathcal{O}$  also carried by  $\Gamma$ . We state this observation as a theorem.

**Theorem 5.4.1.** *Let  $f : |\mathcal{I}| \rightarrow |\mathcal{O}|$  be a continuous map between chromatic complexes and let  $\Gamma : \mathcal{I} \rightarrow \mathcal{O}$  be a carrier map such that  $\Gamma(\sigma)$  is link-connected for each  $\sigma \in \mathcal{I}$ . Suppose  $f$  is carried*

by  $\Gamma$ . Then there exists a chromatic, carrier-preserving simplicial map  $\phi : \text{Ch}^N(\mathcal{I}) \rightarrow \mathcal{O}$ .

A task whose specification map satisfies the above link-connectivity condition is referred to as a *link-connected task*. The content of Theorem 5.4.1 is that the convergence algorithm may be applied to any link-connected task.

## 5.5 Concluding Remarks

The heart of the asynchronous computability theorem is in the construction of a color-preserving map from a chromatic subdivision of the input complex to the output complex. While it is easy to construct a simplicial map using the well-known *simplicial approximation theorem* [35, p.89], the only prior construction [22, 24] lacked any algorithmic insight into the asynchronous computability theorem. The work in this section not only provides an alternative proof of this theorem, but it highlights the importance of link-connectivity of tasks.

While general tasks may not necessarily be link-connected, many complexes considered in the literature do exhibit this property. In the next chapters, we use Theorem 5.4.1 as a means for translating between the combinatorics and the topology of a given task. In certain settings, this theorem provides us a way of reducing combinatorial questions to topological ones, which can allow for the application of more classical results from classical and geometric topology.

## Chapter 6

# $t$ -Resilient Asynchronous Computability

In the previous chapter, we introduced the asynchronous computability theorem, which characterizes the wait-free solvability of tasks in read-write memory. We presented an algorithmic proof of this theorem, and provided an explicit description of the convergence algorithm. In this chapter, we show that the asynchronous computability theorem can be generalized to a more general model of process failure, namely the  $t$ -resilient model [39]. In a system that exhibits  $t$ -resilience, only up to  $t$  processes may fail at any time (as opposed to arbitrary numbers of process failures in the wait-free model). A protocol is  $t$ -resilient if it is correct even in the presence of up to  $t$  failures, while a task is solvable  *$t$ -resiliently* if it has a  $t$ -resilient protocol.

The  $t$ -resilient asynchronous computability theorem, presented in this chapter, is a characterization for tasks that are solvable  $t$ -resiliently. Recall that the wait-free theorem states that a task  $(\mathcal{I}, \mathcal{O}, \Gamma)$  has a wait-free read-write protocol if and only if there is a subdivision  $\text{Div}(\mathcal{I})$  of  $\mathcal{I}$  and a simplicial map  $\phi : \text{Div}(\mathcal{I}) \rightarrow \mathcal{O}$  that complies with the task's specification  $\Gamma : \mathcal{I} \rightarrow 2^{\mathcal{O}}$ . To prove a corresponding theorem for  $t$ -resilient solvability, we replace subdivision with a specific carrier map, denoted  $\text{Ch}_t^N$ , to be defined in this chapter. Then given this new carrier map, we show that a  $(\mathcal{I}, \mathcal{O}, \Gamma)$  has a  $t$ -resilient protocol if and only if there is a simplicial map  $\phi : \text{Ch}_t^N(\mathcal{I}) \rightarrow \mathcal{O}$  carried by  $\Gamma$ . We also give a continuous version of this theorem, which requires that the task be link-connected. Given such a task, we show that only one application of the new carrier map is

required to characterize  $t$ -resilient solvability.

The immediate snapshot protocol gave rise to the standard chromatic subdivision  $\text{Ch}$ . Accordingly, immediate snapshots may be thought of as a building block for solving tasks wait-free, since iterating the protocol enough times implies the existence of a suitable decision map. Here, we give a protocol that is analogous to the immediate snapshot, which serves a similar role for  $t$ -resiliently solvable tasks.

It is not immediately straightforward how to generalize the immediate snapshot protocol to be  $t$ -resilient. We propose the *delayed snapshot* protocol, a three-phase protocol that runs one round of wait-free immediate snapshot, followed by a phase where each process waits for  $n + 1 - t$  processes to catch up, and then some processes run a second round of wait-free immediate snapshot. For a  $t$ -resilient asynchronous computability theorem, we see that this protocol (and its associated complex) produces the desired carrier map, suggesting that  $t$ -resilient delayed snapshot is the correct generalization of the wait-free immediate snapshot.

We also provide insight into the power of waiting. In a wait-free protocol, no process can wait for others to take a step, because those others may have undetectably crashed. In a  $t$ -resilient protocol, by contrast, it is safe to wait for all but  $t$  processes to take steps. We provide a proof that any task that has a  $t$ -resilient protocol has a protocol with only a single waiting step: all steps before and after can be executed wait-free.

## 6.1 Delayed Snapshot Protocol and Task

We begin by describing more precisely the topological structure of the candidate carrier map for the  $t$ -resilient asynchronous computability theorem. As with the standard chromatic subdivision, which itself is interpretable as a carrier map, this new map is defined by a family of boundary-consistent simplicial complexes. These complexes are called the *delayed snapshot* complexes.

We denote the  $t$ -resilient,  $(n + 1)$ -process delayed snapshot complex by  $\text{Ch}_t(\Delta^n)$ , which is intentionally suggestive of the standard chromatic subdivision. Informally, the complex  $\text{Ch}_t(\Delta^n)$  is the subcomplex of the two-round immediate snapshot complex  $\text{Ch}^2(\Delta^n)$  obtained by removing parts of the boundary of  $\Delta^n$ . More specifically, any simplexes that meet a low-dimensional skeleton of  $\Delta^n$  are discarded. See Figure 6.1 illustrating the 1-resilient, 3-process delayed snapshot complex.

Let  $\mathcal{C}$  be a simplicial complex with subcomplex  $\mathcal{B} \subseteq \mathcal{C}$ . Then recall that the *deletion* of  $\mathcal{B}$  in  $\mathcal{C}$ ,

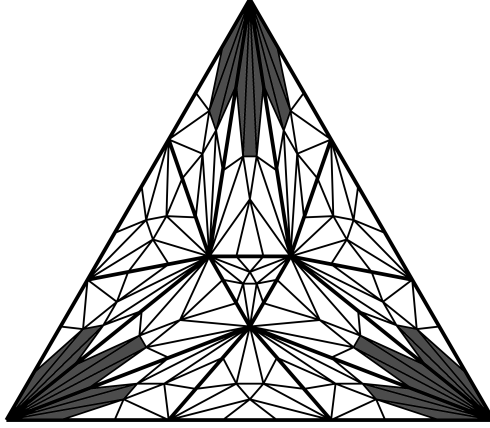


Figure 6.1:  $\text{Ch}_1(\Delta^2)$  as a subcomplex of  $\text{Ch}^2(\Delta^2)$ . The grayed-out simplexes are discarded from  $\text{Ch}^2(\Delta^2)$  to obtain  $\text{Ch}_1(\Delta^2)$ . This corresponds to a 3-process, 1-resilient protocol.

written  $\text{dl}(\mathcal{B}, \mathcal{C})$ , is the subcomplex of  $\mathcal{C}$  consisting of all simplexes that do not intersect  $\mathcal{B}$ . We use the deletion operation to formally define the family of delayed snapshot complexes.

**Definition 6.1.1.** *The complex  $\text{dl}(\text{Ch}^2(\text{skel}^{n-t-1}(\Delta^n)), \text{Ch}^2(\Delta^n))$ , denoted  $\text{Ch}_t(\Delta^n)$ , is called the delayed snapshot complex. It is the subcomplex of  $\text{Ch}^2(\Delta^n)$  obtained by stripping away all simplexes of  $\text{Ch}^2(\Delta^n)$  that intersect  $\text{skel}^{n-t-1}(\Delta^n)$ .*

Similar to how the standard chromatic subdivision can be iterated, the delayed snapshot carrier map can also be iterated, since it is boundary-consistent. The carrier map is iteratively applied to simplexes in the complex. The  $N$ th iterate is denoted  $\text{Ch}_t^N$ .

The delayed snapshot complex can also be interpreted as a task. In the *delayed snapshot task*  $(\Delta^n, \text{Ch}_t(\Delta^n), \text{Ch}_t)$ , each process starts on the vertex of  $\Delta^n$  labeled with its name, halts on a vertex of  $\text{Ch}_t(\Delta^n)$  labeled with its name, and all processes converge on a single simplex of  $\text{Ch}_t(\Delta^n)$ .

We provide a  $t$ -resilient protocol for solving this task, called the *delayed snapshot* protocol, shown in Protocol 3. Processes share two  $(n+1)$ -element arrays, `mem0` and `mem1`, and a shared variable, `done`. Each process calls the wait-free immediate snapshot protocol to store its name in `mem0` and take a snapshot of that array (Lines 8-8). Here, the **immediate** block's first line assigns to that process's element in `mem0` and the second line immediately assigns an atomic snapshot of `mem0` to a local variable, `snap0`. If the process does not see at least  $n+1-t$  processes in `snap0`, it waits until `done` is set to true. Otherwise, it stores `snap0` in `mem1`, takes an immediate snapshot of `mem1`, sets `done` to true, and then returns (Lines 14-16).

```

1 shared mem0[ $n+1$ ];
2 shared mem1[ $n+1$ ];
3 shared done;
4 done := false;
5 protocol DelayedSnapshot(id):
6   immediate
7   |   mem0[id] := id;
8   |   snap0 := snapshot(mem0);
9   if |snap0| ≤  $n - t$  then
10  |   while not done
11  |   | skip
12  immediate
13  |   mem1[id] := snap0;
14  |   snap1 := snapshot(mem1);
15  done := true;
16  return snap1;

```

**Algorithm 2:** Delayed snapshot protocol

We now show that the delayed snapshot protocol solves the delayed snapshot task.

**Theorem 6.1.2.** *Protocol 3 is a  $t$ -resilient delayed snapshot protocol.*

*Proof.* Since the protocol consists of two successive immediate snapshots on clean memory, the processes converge to a simplex of  $\text{Ch}^2(\Delta^n)$ , the second standard chromatic subdivision. We show termination and validity of the protocol.

To show that any process executing the delayed snapshot protocol, in a  $t$ -resilient system, eventually finishes its execution, consider the set of all non-faulty processes. Because we are only considering  $t$ -resilient executions as valid executions, there must be at least  $n + 1 - t$  processes that are non-faulty, otherwise too many processes would be faulty. Let  $p$  be the last non-faulty process to execute the first immediate snapshot of the delayed snapshot protocol (Lines 8-8). Since  $p$  shows up last among non-faulty processes, this process must observe the effects of at least  $n + 1 - t$  processes, including itself, in the snapshot it obtains. Therefore  $p$  does not wait at Line 11 at the while loop, since it has seen enough other participants to proceed. This process may then freely execute the second immediate snapshot wait-free. Before process  $p$  finishes the protocol, it signals other waiting processes by setting *done* to *true*. By setting this variable, any other process can move forward past the intermediate barrier and continue wait-free, executing both immediate snapshots. Therefore once process  $p$  finishes its execution, every other participating process can finish the protocol. This shows that the protocol terminates.

Finally, we check that the processes collectively choose a valid simplex within the output complex



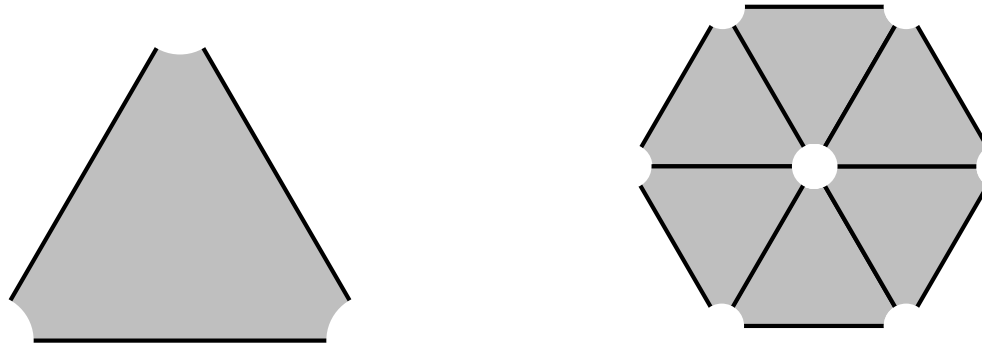
$\text{Ch}_t(\Delta^n)$ . Define process  $p$  as above, or the participant that is last in executing the first immediate snapshot. Before setting the done variable,  $p$  must have written its view to mem1. Thus any process blocked at Line 11 will see the view of  $p$ . But this view includes at least  $n + 1 - t$  processes, so any subsequent process taking a snapshot of mem1 will see at least  $n + 1 - t$  processes as well. But the vertices of  $\text{Ch}_t(\Delta^n)$  correspond exactly to the local views in which processes see at least  $n + 1 - t$  processes, so each process will choose a vertex in  $\text{Ch}_t(\Delta^n)$ .

It is clear that processes collectively choose some simplex in  $\text{Ch}^2(\Delta)$ , and we have shown that they all choose vertices in  $\text{Ch}_t(\Delta^n)$ . So the processes converge on a simplex in  $\text{Ch}_t(\Delta^n)$ , hence validity of the snapshot protocol is shown. This proves correctness of the protocol, and shows that the delayed snapshot task is solvable  $t$ -resiliently.

□

We have now defined the delayed snapshot task and protocol. As previously suggested, this protocol can be iterated on a sequence of clean memory arrays, which has the effect of iterating the operator on each simplex obtained from previous iterators. The iterated complex is denoted  $\text{Ch}_t^N(\Delta^n)$ , and is called the  $N$ -round delayed snapshot complex.

As we see later in the chapter, the delayed snapshot protocol is used, in a manner similar to immediate snapshots, so characterize  $t$ -resilient solvability of tasks. However before continuing with this result, we first need to discuss and investigate certain connectivity and combinatorial properties of the delayed snapshot complex. This is primarily where the topological analysis and characterization of wait-free computability versus  $t$ -resilient computability begin to diverge. As previously noted, running an immediate snapshot protocol corresponds to subdividing the given input complex, and more importantly, subdivisions do not fundamentally change the connectivity simplicial complexes. However, executing the delayed snapshot protocol does not result in a subdivision of the input complex; it may also remove parts of the boundary of each simplex. While removing parts of the boundary leaves a single simplex fully connected, as depicted in Figure 6.2a, in general, it tears holes in the input complex.



(a)  $\text{Ch}_1(\Delta^2)$  is (carrier-preserving) homotopy equivalent with this topological space.

(b) By applying the 1-resilient delayed snapshot protocol to a hexagonal input complex, we get a hole in the middle.

Figure 6.2: As illustrated with the 3-process, 1-resilient delayed snapshot,  $\text{Ch}_t$  is capable of tearing holes in simplicial complexes.

For example, see Figure 6.2b above, where each simplex in a hexagonal complex has been replaced with a copy of  $\text{Ch}_1(\Delta^2)$ . Since the corners of each 2-simplex have been removed, we have effectively torn a hole in the middle of the complex. As we will see, connectivity of the protocol complex and corresponding asynchronous computability theorem characterization are fundamentally related.

## 6.2 Connectivity Properties

The goal of this section is to show that  $\text{Ch}_t^N(\Delta^n)$  is link-connected; as suggested in the previous chapter, link-connectivity of this complex is used to apply the corollary of the convergence algorithm for link-connected tasks. Therefore, once we have link-connectivity, we can use the convergence algorithm to instead reason about the topology of our simplicial complexes. Throughout this section we use the 3-process, 1-resilient delayed snapshot complex,  $\text{Ch}_1(\Delta^2)$ , as a running example.

We begin with arguing for the shellability of the first round complex.

### 6.2.1 Shellability of the Protocol Complex

While higher-round complexes are generally not fully connected, and therefore not shellable, the first round complex is indeed shellable. Roughly speaking, a pure  $n$ -dimensional simplicial complex is called *shellable* if it can be constructed by “gluing” facets together along their  $(n-1)$ -faces. Shellable

complexes are better understood by reasoning about how they are pieced together; for example, they are amenable to arguments by induction on shelling order.

We begin with shellability of the standard chromatic subdivision, and use the shelling order in subsequent arguments.

**Theorem 6.2.1.**  $\text{Ch}(\Delta^n)$  is shellable.

*Proof.* Following Kozlov [?], the first immediate snapshot execution by processes  $p_0, \dots, p_n$  is described by an *ordered partition*  $S_0, \dots, S_m$ , where the  $S_i$  are disjoint sets of processes. Operationally, the processes in  $S_0$  all write and perform the immediate snapshot concurrently, followed by the processes in  $S_1$ , and so on.

Note that any simplex  $\sigma$  in  $\text{Ch}(\Delta^n)$  represents one interleaving of processes executing the immediate snapshot protocol. Let  $S_0, \dots, S_m$  be the ordered partition whose simplex is given by  $\sigma$ . Then we define  $\text{Flip}_i(\sigma)$ :

$$\text{Flip}_i(\sigma) = \begin{cases} \{p_i\} \cup S_1, \dots, S_m & \text{if } S_0 = \{P_i\} \\ \{p_i\}, S_0 \setminus \{p_i\}, S_1, \dots, S_m & \text{if } |S_0| > 1 \text{ and } p_i \in S_0 \end{cases}$$

Flipping an execution changes exactly one process's view, so  $\sigma$  and  $\text{Flip}_i(\sigma)$  share an  $(n - 1)$  face.

Let  $\Delta^n = (v_0, \dots, v_n)$  be the ordered vertex set of the standard simplex. Then define the *extended star* of  $v_i$  to be the union of  $\text{St}(v_i, \text{Ch}(\Delta^n))$  with  $\text{Flip}_i(\sigma)$  for each facet  $\sigma$  of  $\text{St}(v_i, \text{Ch}(\Delta^n))$ . Figure 6.3 shows an example of an extended star.

We inductively define a shelling order on  $\text{Ch}(\Delta^n)$ , by inducting on the dimension. The base case, where  $n = 0$ , is trivial since there is just one simplex. So we inductively assume we have constructed a shelling order on  $\text{Ch}(\Delta^n)$ . Observe that  $\text{Lk}(v_i, \text{Ch}(\Delta^n))$  is isomorphic to  $\text{Ch}(\Delta^{n-1})$ , which is shellable by the induction hypothesis. Then this link's shelling order in fact induces a shelling order on the star  $\text{St}(v_i, \text{Ch}(\Delta^n))$ . A shelling order on the extended star is then constructed by appending, in any order, the flipped simplexes of the star to the star's shelling order. The shelling order for  $\text{Ch}(\Delta^n)$  is constructed by concatenating the shelling orders for the extended stars, and eliminating duplicates that may result since extended stars have nontrivial intersection.

Every facet of  $\text{Ch}(\Delta^n)$  is included in this order because every execution  $S_0, \dots, S_m$  is either a facet of some  $\text{St}(v_i, \text{Ch}(\Delta^n))$  (if  $S_0$  is a singleton) or one flip away from some star. Therefore we have a shelling order on  $\text{Ch}(\Delta^n)$ , so the complex is shellable.  $\square$

The delayed snapshot complex is a subcomplex of the second standard chromatic subdivision. We use the above shelling order to construct a shelling order for the delayed snapshot complex, in parts. We introduce some additional notation to facilitate this component-wise construction.

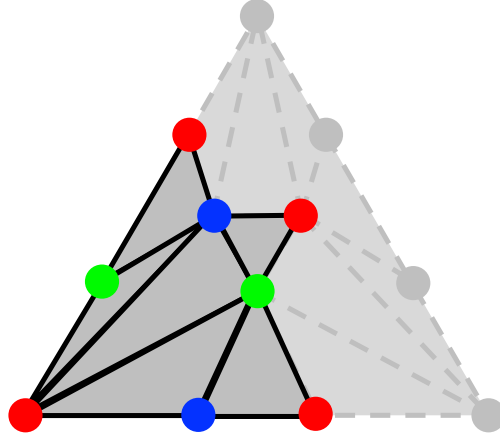


Figure 6.3: The extended star of a vertex in  $\text{Ch}(\Delta^2)$ .

For every  $\sigma \in \text{Ch}(\Delta^n)$ , define  $\text{Ch}_\sigma(\sigma) = \text{Ch}^2(\sigma) \cap \text{Ch}_t(\sigma)$ , the restriction of the delayed snapshot complex  $\text{Ch}_t(\Delta^n)$  to  $\sigma$ . Intuitively, the complex  $\text{Ch}_\sigma(\sigma)$  is a subcomplex of the standard chromatic subdivision of just one simplex  $\sigma$ . Note that  $\text{Ch}_t(\Delta^n)$  is then the union of all subcomplexes  $\text{Ch}_\sigma(\sigma)$ , for each  $\sigma \in \text{Ch}(\Delta^n)$ . We provide a shelling order on each of the  $\text{Ch}_\sigma(\sigma)$ .

**Theorem 6.2.2.**  $\text{Ch}_\sigma(\sigma)$  is shellable.

*Proof.* Observe that  $\text{Ch}_\sigma(\sigma)$  is obtained by deleting a subset of vertices from the full subdivision  $\text{Ch}(\sigma)$ . So let the deleted vertices be the set  $\{v_i : i \in I\}$ , for some index set  $I$ . Then notice that  $\text{Ch}_\sigma(\sigma)$  is actually a union of the extended stars of the vertices that remain in the complex. This complex can be constructed as a prefix of the shelling order on the full subdivision  $\text{Ch}(\sigma)$ . In other words, the shelling order is the concatenation of the shelling orders for the extended stars of all the vertices *not* in  $I$ .  $\square$

We have a shelling order on each individual subcomplex  $\text{Ch}_\sigma(\sigma)$  of  $\text{Ch}_t(\Delta^n)$ , which collectively exhaust all simplexes of the latter. So we use these shelling orders to define one combined shelling order on the full complex  $\text{Ch}_t(\Delta^n)$ .

**Theorem 6.2.3.**  $\text{Ch}_t(\Delta^n)$  is shellable.

*Proof.* A shelling order on  $\text{Ch}_t(\Delta^n)$  is constructed by concatenating the shelling orders for each  $\text{Ch}_\sigma(\sigma)$ . We induct on the components  $\text{Ch}_\sigma(\sigma)$  of  $\text{Ch}_t(\Delta^n)$ .

Let  $\ell$  be the shelling order on  $\text{Ch}(\Delta^n)$  constructed in Theorem 6.2.1. We begin by enumerating the facets of  $\text{Ch}(\Delta^n)$ , so that  $\sigma_i = \ell(i)$ . The complex  $\text{Ch}_t(\Delta^n)$  is built up from the  $\text{Ch}_\sigma(\sigma)$ , in the order of the shelling order on  $\text{Ch}(\Delta^n)$ . The base case here is trivial, since each  $\text{Ch}_\sigma(\sigma)$  is shellable and we start with only one component. Now inductively assume  $\text{Ch}_{\sigma_i}(\sigma_i)$  has been attached to the intermediate complex, and let  $\mathcal{C}_i$  denote this intermediate complex, with shelling order  $\ell_i$ . Then  $\sigma_{i+1}$  is the next simplex in the shelling order on  $\text{Ch}(\Delta^n)$ . Let  $\alpha$  be any ordering on the vertices of  $\sigma_{i+1}$ , such that every vertex  $v$  already contained in  $\mathcal{C}_i$  precedes every vertex  $w$  not contained in  $\mathcal{C}_i$ . Let  $\ell_{\alpha, \sigma_{i+1}}$  be the shelling order constructed earlier for  $\text{Ch}_{\sigma_{i+1}}(\sigma_{i+1})$ , but with vertices ordered by  $\alpha$  instead of the ordering on  $\Delta^n$ . This ensures that simplexes may still be properly attached to the complex while still ensuring the shellability condition. Then we append  $\ell_{\alpha, \sigma_{i+1}}$  to  $\ell_i$  to obtain a shelling order  $\ell_{i+1}$  on  $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \text{Ch}_{\sigma_{i+1}}(\sigma_{i+1})$ .

Since the  $\text{Ch}_\sigma(\sigma)$  exhaust the delayed snapshot complex, then by induction, this defines a shelling order on the entire complex  $\text{Ch}_t(\Delta^n)$ .  $\square$

Figure 6.4 shows the shelling order obtained on the facets of  $\text{Ch}_0(\Delta^2)$  by following the construction from the theorem.

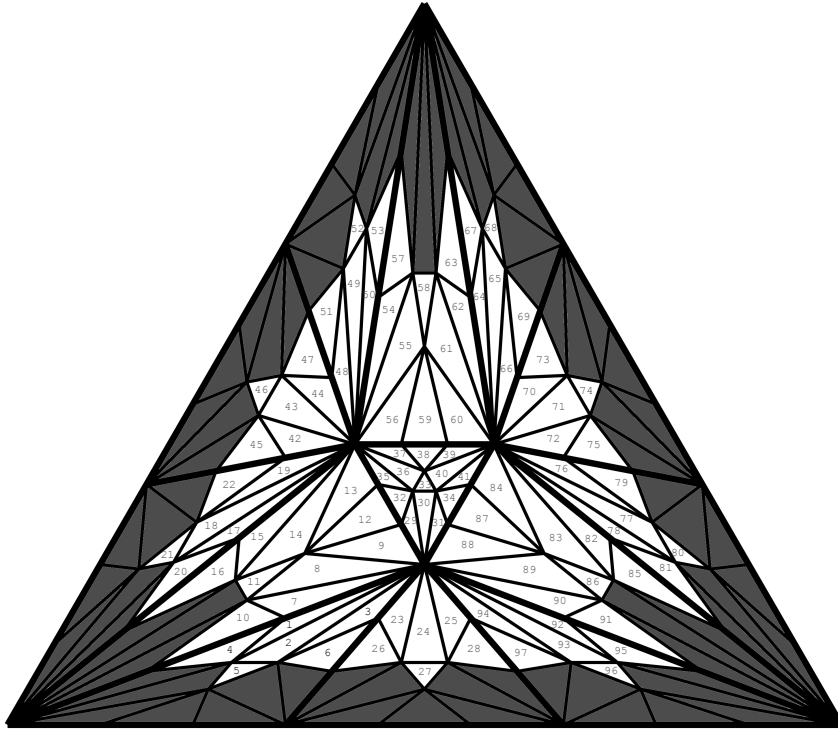


Figure 6.4: A numbered shelling order on  $\text{Ch}_0(\Delta^2)$  (as before, darkened simplexes are absent).

Shellability gives us a coherent way of breaking up a complex into its simplexes, but as hinted at in some of the proofs of this section, it can also be used as a tool for breaking up complex into component *subcomplexes*. In the next section, we do exactly this, by exploiting the shelling order on  $\text{Ch}_t(\Delta^n)$  to show that  $\text{Ch}_t^N(\Delta^n)$  is always link-connected.

## 6.2.2 Link-connectivity of the Protocol Complex

We can use shellability to show that any iteration of the delayed snapshot protocol complex,  $\text{Ch}_t^N(\Delta^n)$ , is also link connected. This follows from a more general result in combinatorial topology, which states that all combinatorial manifolds are link-connected.

**Definition 6.2.4.** A combinatorial manifold is a pure simplicial complex in which the link of any simplex is either a topological disk or a sphere of the correct dimension.

Vertices satisfying the above condition are called regular. We state this as a definition below.

**Definition 6.2.5.** A vertex of a simplicial complex are regular if its link is a sphere (if the vertex is interior) or a disk (if the vertex is on the boundary).

As proved in [], to show a complex is a combinatorial manifold, it suffices to show that all of its vertices are regular. The link condition on higher-dimensional simplexes follows from the requirement.

Due to the regularity condition on links of a combinatorial manifold, it is clear that any combinatorial manifold is also link-connected. Therefore to prove that  $\text{Ch}_t^N(\Delta^n)$  is link-connected, it is enough to show that each vertex is regular. This is accomplished by gluing together  $\text{Ch}_t^N(\Delta^n)$  from copies of  $\text{Ch}_t^{N-1}(\Delta^n)$ , according to the shelling order on  $\text{Ch}_t(\Delta^n)$ . At each step, we ensure that every vertex remains regular.

**Theorem 6.2.6.**  $\text{Ch}_t^N(\Delta^n)$  is link-connected.

*Proof.* We prove the stronger statement that  $\text{Lk}(v, \text{Ch}_t^N(\Delta^n))$  is a combinatorial manifold. We exhibit the shelling order on  $\text{Ch}_t(\Delta^n)$  to derive link-connectivity of the multi-round protocol complex. Let  $\ell$  be the shelling order on  $\text{Ch}_t(\Delta^n)$ . We build up  $\text{Ch}_t^N(\Delta^n)$  by attaching copies of  $\text{Ch}_t^{N-1}(\Delta^n)$  as prescribed by the shelling order  $\ell$ .

We first induct on the dimension  $n$ . The base case where  $n = 0$  is trivial, since  $\Delta^0$  is just a point. So we inductively assume that  $\text{Ch}_t^N(\Delta^k)$  is a combinatorial manifold for some dimension  $k$ , for all  $t$  and all  $N$ . Next, we continue with a nested induction on the number of rounds  $N$  to prove that  $\text{Ch}_t^M(\Delta^{k+1})$  is a combinatorial manifold. The base case here is  $M = 0$ , which is also trivial since  $\Delta^{k+1}$  is the standard simplex and clear a combinatorial manifold itself. Inductively assume that  $\text{Ch}_t^M(\Delta^{k+1})$  is a combinatorial manifold, for some number of iterations  $M$ . We use the shelling order  $\ell$  to put together  $\text{Ch}_t^{M+1}(\Delta^{k+1})$  using copies of  $\text{Ch}_t^M(\Delta^{k+1})$ .

Using a third level of induction, we induct on the shelling order  $\ell$  to prove that any intermediate step in the construction via the shelling is a combinatorial manifold. Let  $\mathcal{C}_i$  denote the complex obtained after attaching the  $i$ th copy of  $\text{Ch}_t^M(\Delta^{k+1})$ . The base case is trivial since we assumed  $\text{Ch}_t^M(\Delta^{k+1})$  to be a combinatorial manifold. Now, inductively assume that  $\mathcal{C}_i$  is a combinatorial manifold. To show that  $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \ell(i+1)$  is also a combinatorial manifold, it is enough to show that the link of any vertex on the boundary of  $\ell(i+1)$  is regular, since no other link in  $\mathcal{C}_i$  will change upon attaching  $\ell(i+1)$ .

Take any boundary vertex  $v$  in  $\ell(i+1)$ , in which case  $v$  is also a boundary vertex of  $\mathcal{C}_i$ . From the inductive hypothesis on  $i$ , we know that  $\text{Lk}(v, \mathcal{C}_i)$  is a  $(k-1)$ -disk, and from the inductive hypothesis on  $M$ , we know that  $\text{Lk}(v, \ell(i+1))$  is also a  $(k-1)$ -disk. The union of these two links is the link of

$v$  in  $\mathcal{C}_{i+1}$ . Formally,

$$\text{Lk}(v, \mathcal{C}_{i+1}) = \text{Lk}(v, \mathcal{C}_i) \cup \text{Lk}(v, \ell(i+1))$$

We calculate their intersection. There are two cases to consider: either  $v$  remains a boundary vertex after attaching  $\ell(i+1)$ , or  $v$  is an interior vertex of  $\mathcal{C}_{i+1}$ . First suppose  $v$  is an interior vertex of  $\mathcal{C}_{i+1}$ . Then  $\text{Lk}(v, \mathcal{C}_i)$  and  $\text{Lk}(v, \ell(i+1))$  intersect on the boundary of  $\text{Lk}(v, \ell(i+1))$ , which is a  $(k-2)$ -sphere. Since  $\text{Lk}(v, \mathcal{C}_i)$  and  $\text{Lk}(v, \ell(i+1))$  are  $(k-1)$ -disks, this must mean that  $\text{Lk}(v, \mathcal{C}_i) \cup \text{Lk}(v, \ell(i+1))$  is a  $(k-1)$ -sphere, as required. In the second case, suppose  $v$  is a boundary vertex of  $\mathcal{C}_{i+1}$ . Then  $\text{Lk}(v, \mathcal{C}_i)$  and  $\text{Lk}(v, \ell(i+1))$  intersect at a  $(k-1)$ -disk. Since  $\text{Lk}(v, \mathcal{C}_i)$  and  $\text{Lk}(v, \ell(i+1))$  are  $(k-1)$ -disks, this must mean that  $\text{Lk}(v, \mathcal{C}_i) \cup \text{Lk}(v, \ell(i+1))$  is again a  $(k-1)$ -disk, as required. So  $\mathcal{C}_{i+1}$  is also a combinatorial manifold.

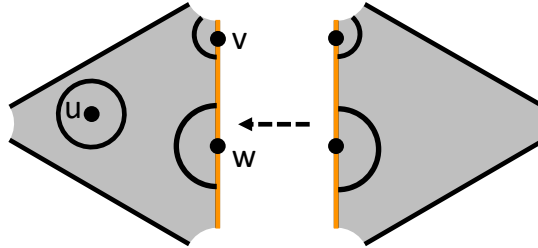


Figure 6.5: Gluing two copies of  $\text{Ch}_1(\Delta^2)$  together, and the possible ways each vertex's link may change as a result. As an interior vertex,  $u$  has the same link, but the links of  $v$  and  $w$  change with the new copy of  $\text{Ch}_1(\Delta^2)$ .

We now wrap up all the inductive steps. By induction on  $i$  (the shelling order), we conclude that  $\text{Ch}_t^{M+1}(\Delta^{k+1})$  is a combinatorial manifold for any  $t$ . By induction on  $M$  (the number of iterations), we conclude that  $\text{Ch}_t^N(\Delta^{k+1})$  is a combinatorial manifold for any  $t$  and  $N$ . Finally, by induction on  $k$  (the dimension), we conclude that  $\text{Ch}_t^N(\Delta^n)$  is a combinatorial manifold for any  $t$ ,  $N$ , and  $n$ . Since combinatorial manifolds are link-connected,  $\text{Ch}_t^N(\Delta^n)$  is link-connected.  $\square$

We can also exhibit the shellability of  $\text{Ch}_t(\Delta^n)$  to show that  $\text{Ch}_t^N(\Delta^n)$  is topologically connected



up to dimension  $t-1$ . We use this property to construct continuous maps into the  $N$ -round complex, in effect yielding a wait-free simulation of the  $N$ -round task from the single-round task.

Topological connectivity of the iterated delayed snapshot complex is a consequence of the following corollary of the nerve lemma, which another celebrated theorem from combinatorial topology. Statements of this corollary and the nerve lemma are found in Herlihy *et al.* [22], and in the introduction of this thesis. The proof of the corollary can be found in the book.

**Corollary 6.2.7.** *If  $\mathcal{K}$  and  $\mathcal{L}$  are  $k$ -connected complexes such that  $\mathcal{K} \cap \mathcal{L}$  is  $(k-1)$ -connected, then  $\mathcal{K} \cup \mathcal{L}$  is also  $k$ -connected.*

Similar to the approach for link-connectivity, we can piece together  $\text{Ch}_t^N(\Delta^n)$  from copies of  $\text{Ch}_t^{N-1}(\Delta^n)$  and iteratively apply the above corollary to obtain that the  $N$ -round complex is sufficiently connected. This is stated and proved below.

**Theorem 6.2.8.**  *$\text{Ch}_t^N(\Delta^n)$  is  $(t-1)$ -connected.*

*Proof.* We induct on  $N$ , or the number of rounds. The base case we begin with in this proof  $N = 1$ . We know that  $\text{Ch}_t(\Delta^n)$  is  $(t-1)$ -connected because it is a known result that any pure, shellable  $n$ -complexes are  $(n-1)$ -connected and therefore  $(t-1)$ -connected for  $t \leq n$ . Now inductively assume, on the number of rounds, that  $\text{Ch}_t^M(\Delta^n)$  is  $(t-1)$ -connected, for some  $M$ . Let  $\psi_0, \dots, \psi_s$  be a shelling order on the facets of  $\text{Ch}_t(\Delta^n)$ , the one-round complex. Let  $\mathcal{L}_j = \bigcup_{i=0}^j \psi_i$ , in which case  $\text{Ch}_t^{M+1}(\Delta^n) = \text{Ch}_t^M(\mathcal{L}_s)$ . To prove that  $\text{Ch}_t^{M+1}(\Delta^n)$  is  $(t-1)$ -connected, we induct on the shelling order of  $\text{Ch}_t(\Delta^n)$ .

For the base case, we know from the above inductive hypothesis that  $\text{Ch}_t^M(\mathcal{L}_0) = \text{Ch}_t^M(\psi_0) = \text{Ch}_t^M(\Delta^n)$  is  $(t-1)$ -connected, so inductively assume that  $\text{Ch}_t^M(\mathcal{L}_k)$  is  $(t-1)$ -connected. Since  $\{\psi_i\}$  is a shelling of  $\text{Ch}_t(\Delta^n)$ , we know that  $\mathcal{L}_k \cap \psi_{k+1}$  is a pure  $(n-1)$ -dimensional subcomplex of  $\Delta^n$ . We show that

$$\text{Ch}_t^M(\mathcal{L}_{k+1}) = \text{Ch}_t^M(\mathcal{L}_k \cup \psi_{k+1}) = \text{Ch}_t^M(\mathcal{L}_k) \cup \text{Ch}_t^M(\psi_{k+1})$$

is  $(t-1)$ -connected via Corollary 6.2.7, which requires that  $\text{Ch}_t^M(\mathcal{L}_k) \cap \text{Ch}_t^M(\psi_{k+1})$  is  $(t-2)$ -connected. This tells us that  $\text{Ch}_t^M(\mathcal{L}_{k+1})$  is  $(t-1)$ -connected, which by induction proves that  $\text{Ch}_t^{M+1}(\Delta^n)$  is  $(t-1)$ -connected, which again by induction proves that  $\text{Ch}_t^N(\Delta^n)$  is  $(t-1)$ -connected, finishing the proof.

Note the following equality:

$$\mathrm{Ch}_t^M(\mathcal{L}_k) \cap \mathrm{Ch}_t^M(\psi_{k+1}) = \mathrm{Ch}_{t-1}^M(\mathcal{L}_k \cap \psi_{k+1})$$

This fact follows from the boundary consistency of the  $\mathrm{Ch}_t$  operator. We prove using the nerve lemma that this complex is  $(t-2)$ -connected. Let  $\mathcal{C} = \mathcal{L}_k \cap \psi_{k+1} \subseteq \Delta^n$  be the intersection on which the new facet is attached, and let  $\{\mathcal{K}_i\}_{i \in I}$  be the cover of  $\mathcal{C}$  consisting of all its facets. Then this implies that  $\{\mathrm{Ch}_{t-1}^M(\mathcal{K}_i)\}_{i \in I}$  is a cover of  $\mathrm{Ch}_{t-1}^M(\mathcal{C})$ . But for any nonempty indexing set  $J \subseteq I$ , the set

$$\sigma = \bigcap_{j \in J} \mathcal{K}_j \subseteq \mathcal{C}$$

is a simplex of dimension  $n - |J|$ , since  $\mathcal{C}$  is a pure  $(n-1)$ -dimensional subcomplex of  $\Delta^n$ . Therefore, from the boundary consistency of  $\mathrm{Ch}_t$ , we get the following equality:

$$\bigcap_{j \in J} \mathrm{Ch}_{t-1}^M(\mathcal{K}_j) = \mathrm{Ch}_{t-|J|}^M(\sigma).$$

If  $t - |J| < 0$ , then  $\mathrm{Ch}_{t-|J|}^M(\sigma)$  is empty, so there is nothing to prove, otherwise from the first inductive hypothesis,  $\mathrm{Ch}_{t-|J|}^M(\sigma)$  is  $(t - |J| - 1)$ -connected. So we can apply the nerve lemma to find that the nerve  $\mathcal{N}(\{\mathrm{Ch}_{t-1}^M(\mathcal{K}_i)\})$  is  $(t-2)$ -connected if and only if  $\{\mathrm{Ch}_{t-1}^M(\mathcal{C})\}$  is  $(t-2)$ -connected. However, it is clear that  $\mathcal{N}(\{\mathrm{Ch}_{t-1}^M(\mathcal{K}_i)\}) \cong \mathrm{skel}^{t-1}(\mathcal{N}(\{\mathcal{K}_i\}))$ , since we discard all points in the  $(t-1)$ -skeleton of  $\mathcal{C}$  when we apply  $\mathrm{Ch}_{t-1}$  to it. Furthermore,  $\mathcal{N}(\{\mathcal{K}_i\})$  is  $(n-2)$ -connected since  $\mathcal{C}$  is homeomorphic to either an  $(n-1)$ -disc or an  $(n-1)$ -sphere. So  $\mathcal{N}(\{\mathcal{K}_i\})$  is  $(t-2)$ -connected, so  $\mathrm{Ch}_{t-1}^M(\mathcal{C}) = \mathrm{Ch}_t^M(\mathcal{L}_k) \cap \mathrm{Ch}_t^M(\psi_{k+1})$  is  $(t-2)$ -connected.

This proves that  $\mathrm{Ch}_t^M(\mathcal{L}_{k+1})$  is  $(t-1)$ -connected. By induction on the shelling order, the full complex  $\mathrm{Ch}_t^M(\mathcal{L}_s) = \mathrm{Ch}_t^{M+1}(\Delta^n)$  is  $(t-1)$ -connected. Finally, by induction on dimension, we arrive at the conclusion that  $\mathrm{Ch}_t^N(\Delta^n)$ , for all  $N$ ,  $t$ , and  $n$ .  $\square$

We have shown that the  $N$ -round delayed snapshot complex is both link-connected and  $(t-1)$ -connected. The first property is used to translate our reasoning into purely topological terminology, while the second lets us simulate the  $N$ -round protocol from the 1-round protocol. We go into more detail in the next section.

### 6.3 Single-Round Waiting

In this section we prove the existence of a simulation  $\text{Ch}_t(\Delta^n) \rightarrow \text{Ch}_t^N(\Delta^n)$ . Recall that one protocol  $(\mathcal{I}, \mathcal{P}, \Gamma_1)$  simulates another protocol  $(\mathcal{I}, \mathcal{P}_2, \Gamma_2)$  if there is a *simulation*  $\phi : \mathcal{P}_1 \rightarrow \mathcal{P}_2$  which is a color-preserving, simplicial map, such that  $\Gamma_1(\phi(\sigma)) \subseteq \Gamma_2(\sigma)$  holds for all inputs  $\sigma$ .

We begin by constructing a continuous map from the one-round delayed snapshot complex to the  $N$ -round (iterated) delayed snapshot complex. This is turned into a simplicial map using the fact that the  $N$ -round complex is link-connected together with invoking the convergence algorithm.

Operationally, finding such a simplicial map means that any task solvable  $t$ -resiliently can be solved using only one delayed snapshot round, followed by some number of wait-free immediate snapshot rounds, implying that only one waiting statement is necessary in any  $t$ -resilient protocol.

To find a continuous map  $|\text{Ch}_t(\Delta^n)| \rightarrow |\text{Ch}_t^N(\Delta^n)|$ , we first retract the domain onto a topological subspace. If  $Y \subseteq X$  are topological spaces, then a *retraction* [17] from  $X$  to  $Y$  is a continuous map  $f : X \rightarrow Y$  such that  $f$  restricted to  $Y$  is the identity.

Define the complex

$$\text{Bary}_t(\Delta^n) = \{\sigma \in \text{Bary}(\Delta^n) : \forall v \in \sigma, \dim(\text{Car}(v, \text{Bary})) \geq n - t\}.$$

That is,  $\text{Bary}_t(\Delta^n)$  is the induced subcomplex of  $\text{Bary}(\Delta^n)$  defined by vertices whose carriers have dimension at least  $n - t$ . One can show that by retracting the holes in  $\text{Ch}_t(\mathcal{I})$ , we can retract the whole complex to  $\text{Bary}_t(\mathcal{I})$ .

**Lemma 6.3.1.** *The space  $\text{Bary}_t^\circ(\mathcal{I}) = |\text{Bary}_t(\mathcal{I})| - |\text{skel}^{n-t}(\mathcal{I})|$  retracts to  $|\text{Bary}_{t-1}(\mathcal{I})|$ .*

*Proof.* This problem is reduced to one on facets. We retract each facet of  $\text{Bary}_t^\circ(\mathcal{I})$  to the corresponding facet in  $|\text{Bary}_{t-1}(\mathcal{I})|$ . Let  $\{\sigma_i\}$  be an enumeration of the facets of  $\text{Bary}_t(\mathcal{I})$ . Then each  $\sigma_i$  contains exactly one vertex  $v_i$  of minimum carrier dimension. Define  $\sigma_i^\circ = |\sigma_i| - \{v_i\}$ , which is a subspace, and define  $\sigma_i' = \sigma_i - \{v_i\}$ , which is a subsimplex. Then we define a retraction from  $\sigma_i^\circ$  to  $\sigma_i'$  via projection along the line defined by  $v_i$  onto  $\sigma_i'$ . More specifically, take any  $x \in |\sigma_i^\circ|$ , and let  $L_i \subseteq |\sigma_i^\circ|$  be the unique line segment defined by the points  $x$  and  $v_i$ . Note that  $L_i$  is well-defined since  $x \neq v_i$ . Then define  $r_i(x)$  to be the unique point contained in  $L_i \cap |\sigma_i'|$ . Such  $r_i$  is continuous, and it is also clear that  $r_i$  fixes  $|\sigma_i'|$ . To see this, if we take  $x \in |\sigma_i'|$ , then  $x \in L_i \cap |\sigma_i'| = \{r_i(x)\}$ , and hence  $r_i(x) = x$ . We have defined a retraction  $r_i$  for each  $\sigma_i^\circ$  of  $\text{Bary}_t^\circ(\mathcal{I})$ .

We now use the classical pasting lemma to glue all the  $r_i$  together. To apply the lemma, we must argue that if  $\sigma_i^\circ$  and  $\sigma_j^\circ$  have a nonempty intersection in  $\text{Bary}_t^\circ(\mathcal{I})$ , then their retraction maps  $r_i$  and  $r_j$  agree on their intersection. So choose any such  $\sigma_i$  and  $\sigma_j$  with  $\sigma_i^\circ \cap \sigma_j^\circ \neq \emptyset$ . We consider the two cases

1.  $v_i \neq v_j$ ,
2.  $v_i = v_j$ ,

where  $v_i$  and  $v_j$  are defined as above. Suppose the first case holds. Then take any point  $x \in \sigma_i^\circ \cap \sigma_j^\circ = |\sigma_i \cap \sigma_j|$ . We know that  $\sigma_i \cap \sigma_j$  is a simplex. Towards a contradiction, suppose that  $x \notin |\sigma'_i|$ . Then the carrier of  $x$  must be  $\sigma_i$ , since  $\sigma'_i = \sigma_i - \{v_i\}$  and  $x \neq v_i$ . But the carrier of  $x$  is the unique simplex of least dimension that contains  $x$ , and  $x \in |\sigma_i \cap \sigma_j|$ , so it must be that  $\sigma_i = \sigma_i \cap \sigma_j$ . But this is a contradiction since  $v_j \in \sigma_j - \sigma_i$ . Therefore  $x \in |\sigma'_i|$ , and by a symmetric argument,  $x \in |\sigma'_j|$ , so in fact  $x \in |\sigma'_i \cap \sigma'_j|$ . However  $r_i$  and  $r_j$  are both the identity on  $|\sigma'_i \cap \sigma'_j|$ , so  $r_i(x) = r_j(x) = x$ .

The case where  $v_i = v_j$  is straightforward, since  $L_i = L_j$ . In both cases we have  $r_i(x) = r_j(x)$ , so we apply the pasting lemma to get a continuous function  $r : \text{Bary}_t^\circ(\mathcal{I}) \rightarrow |\text{Bary}_{t-1}(\mathcal{I})|$ . This is a retraction since all of its components are retractions. So  $\text{Bary}_t^\circ(\mathcal{I})$  retracts to  $|\text{Bary}_{t-1}(\mathcal{I})|$ .  $\square$

The above lemma can be applied iteratively to reduce the dimension of the retract by one with each step. This gives us a retraction from  $|\text{Ch}_t(\mathcal{I})|$  to  $|\text{Bary}_t(\mathcal{I})|$  that is carrier-preserving. Figure 6.6 shows an example of such a retraction.

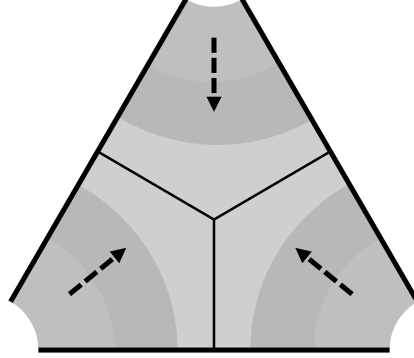


Figure 6.6: Retracting the two-dimensional space  $|\text{Ch}_1(\Delta^2)|$  to a one-dimensional subspace, in a carrier-preserving way. This is possible since its corners have been deleted.

**Theorem 6.3.2.** *There is a map  $f : |\text{Ch}_t(\mathcal{I})| \rightarrow |\text{Bary}_t(\mathcal{I})|$  that is continuous and carrier-preserving.*

*Proof.* We prove this theorem for  $\mathcal{I} = \Delta^n$ , then use the pasting lemma to extend the result to arbitrary  $\mathcal{I}$ .

Let  $\mathcal{B}_t^k(\Delta^n) = |\text{Bary}_k(\Delta^n)| - |\text{skel}^{n-t}(\Delta^n)|$  for any given  $k$ . Then  $\mathcal{B}_t^k(\Delta^n) \subseteq \text{Bary}_k^\circ(\Delta^n)$  and  $\mathcal{B}_t^t(\Delta^n) = \text{Bary}_t^\circ(\Delta^n)$ . Starting with  $\mathcal{B}_t^n$ , we can iteratively apply Lemma 6.3.1 to retract  $\mathcal{B}_t^k(\Delta^n)$  to  $\mathcal{B}_t^{k-1}(\Delta^n)$  and ultimately construct a retraction  $r : \mathcal{B}_t^n \rightarrow |\text{Bary}_t(\Delta^n)|$ . The map  $r$  itself is carrier-preserving since it is the identity on  $\text{Bary}_t(\Delta^n)$ . We then restrict  $r$  to the subspace  $|\text{Ch}_t(\Delta^n)| \subseteq |\Delta^n|$  to obtain  $f : |\text{Ch}_t(\Delta^n)| \rightarrow |\text{Bary}_t(\Delta^n)|$ , which is also continuous and carrier-preserving since  $r$  is continuous and carrier-preserving. So we obtain the desired retraction.

Since this map implicitly defines a carrier-preserving retraction for each facet of a given input complex, the collection of retractions are glued together using the pasting lemma to get a continuous  $f : |\text{Ch}_t(\mathcal{I})| \rightarrow |\text{Bary}_t(\mathcal{I})|$ .  $\square$

We use topological connectivity of  $\text{Ch}_t^N(\mathcal{I})$  to map  $\text{Bary}_t(\mathcal{I})$  into the iterated complex. Therefore the mapping  $|\text{Ch}_t(\mathcal{I})| \rightarrow \text{Ch}_t^N(\mathcal{I})$  define factors through the retracted subspace  $\text{Bary}_t(\mathcal{I})$ .

**Theorem 6.3.3.** *There is a continuous, carrier-preserving map  $f : |\text{Bary}_t(\mathcal{I})| \rightarrow |\text{Ch}_t^N(\mathcal{I})|$ .*

*Proof.* We define a function  $f : |\text{Bary}_t(\mathcal{I})| \rightarrow |\text{Ch}_t^N(\mathcal{I})|$ , for any given  $n$ , by induction on the skeletons of  $\text{Bary}_t(\mathcal{I})$ . Starting with the base case, we define a map  $f^0 : |\text{skel}^0(\text{Bary}_t(\mathcal{I}))| \rightarrow |\text{Ch}_t^N(\mathcal{I})|$ .

Take any  $v \in \text{skel}^0(\text{Bary}_t(\mathcal{I}))$ , and let  $\tau$  be its carrier in  $\mathcal{I}$ , with  $d = \dim(\tau)$ . Then  $d \geq n - t$ . Define the constant

$$t' = t - n + d$$

Note it follows that  $t' \geq 0$ . From Theorem 6.2.8, we know that  $\text{Ch}_{t'}^N(\tau)$  is  $(t' - 1)$ -connected, so the complex is also at least  $(-1)$ -connected, or nonempty. This allows us to pick a point  $w \in \text{Ch}_{t'}^N(\tau)$ , so that we may define  $f^0(v) = w$ . Doing this for all such  $v$  gives us a function

$$f^0 : |\text{skel}^0(\text{Bary}_t(\mathcal{I}))| \rightarrow |\text{Ch}_t^N(\mathcal{I})|$$

which is clearly continuous since its domain consists of disconnected points. The constructed map is also carrier-preserving since we chose  $w$  to be in the image of the carrier of  $v$ .

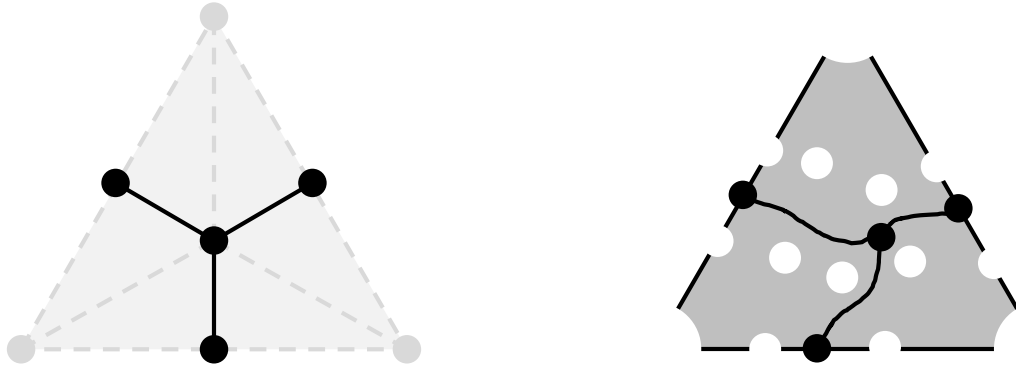
Inductively assume we have a continuous, carrier-preserving map

$$f^k : |\text{skel}^k(\text{Bary}_t(\mathcal{I}))| \rightarrow |\text{Ch}_t^N(\mathcal{I})|$$

and let  $\{\sigma_i\}$  be the facets of  $\text{skel}^{k+1}(\text{Bary}_t(\mathcal{I}))$ . Take any  $\sigma_i$  and let  $\tau_i$  be its carrier in  $\mathcal{I}$ , with  $d = \dim(\tau_i)$ . As before, define  $t' = t - n + d$ . Then  $t' \geq k + 1$ . By the inductive hypothesis, we have defined the map  $f^k$  on  $\partial\sigma_i$ . Restrict  $f^k$  to each  $\sigma_i$ , so that  $f_i^k = f^k|_{\sigma_i}$ . We extend each one individually and then glue them all together.

Since  $f^k$  is carrier-preserving, the image of  $\partial\sigma_i$  under  $f_i^k$  is contained in  $|\text{Ch}_{t'}^N(\tau_i)|$ . By Theorem 6.2.8, the complex  $\text{Ch}_{t'}^N(\tau_i)$  is  $(t' - 1)$ -connected, so it is also  $k$ -connected from the inequality in the previous paragraph. So we can extend  $f_i^k$  to a continuous  $f_i^{k+1} : |\sigma_i| \rightarrow |\text{Ch}_{t'}^N(\tau_i)|$ . Each  $f_i^{k+1}$  is carrier-preserving by the way it is constructed. We apply the pasting lemma to obtain a continuous map  $f^{k+1}$  defined on all of  $|\text{skel}^{k+1}(\text{Bary}_t(\mathcal{I}))|$ . Then  $f^{k+1}$  is carrier-preserving since each of its components are, so the result follows.

By induction on  $k$ , we get a continuous, carrier-preserving map  $f : |\text{Bary}_t(\mathcal{I})| \rightarrow |\text{Ch}_t^N(\mathcal{I})|$ .  $\square$



(a)  $\text{Bary}_1(\Delta^2)$  is a retraction of  $|\text{Ch}_1(\Delta^2)|$ , of strictly lower dimension.

(b)  $\text{Bary}_1(\Delta^2)$  maps into any  $|\text{Ch}_1^N(\Delta^2)|$ , in a carrier-preserving way.

Figure 6.7: A mapping of  $\text{Bary}_1(\Delta^2)$  into  $|\text{Ch}_1^N(\Delta^2)|$ .

We can compose the maps in the two theorems above to obtain a continuous map from the one-round complex to the iterated complex. We can then apply the simplicial approximation theorem to obtain a simplicial map. However, we still need to make this map color-preserving.

Recall Theorem 5.4.1, a corollary of the convergence algorithm described in the previous section. This allows us to turn a carrier-preserving, simplicial map into a color-preserving, provided that the image of any simplex under the carrier map is link-connected. But link-connectivity of  $\text{Ch}_t^N(\Delta^n)$  guarantees this condition, so we may apply Theorem 5.4.1. We obtain the following theorem:

**Theorem 6.3.4** (Wait Reduction). *For any  $N > 0$ , there is a chromatic, carrier-preserving, simplicial map  $\phi : \text{Ch}^M(\text{Ch}_t(\mathcal{I})) \rightarrow \text{Ch}_t^N(\mathcal{I})$  for some sufficiently large  $M$ .*

*Proof.* From Theorems 6.3.3 and 6.3.2, we get the composition of maps:

$$|\text{Ch}_t(\mathcal{I})| \rightarrow |\text{Bary}_t(\mathcal{I})| \rightarrow |\text{Ch}_t^N(\mathcal{I})|$$

Let this composition be  $f : |\text{Ch}_t(\mathcal{I})| \rightarrow |\text{Ch}_t^N(\mathcal{I})|$ . Then applying the Corollary 5.4.1, we get a color-preserving, simplicial map  $\phi : \text{Ch}^M(\text{Ch}_t(\mathcal{I})) \rightarrow \text{Ch}_t^N(\mathcal{I})$  respecting the task specification.  $\square$

So there is a  $t$ -resilient read-write protocol that simulates  $N$  rounds of delayed snapshots with only a single delayed snapshot, followed by some number of wait-free immediate snapshots. The one-round protocol simulates the  $N$ -round protocol. This simulation is used to present a more topological version of an asynchronous computability theorem for  $t$ -resilient systems.

## 6.4 Asynchronous Computability Theorems

There are two versions of the  $t$ -resilient asynchronous computability theorem: a discrete version, which characterizes solvability in terms of the existence of a color-preserving simplicial map, and a similar continuous version, which provides a characterization for link-connected tasks in terms of the existence of a continuous function. We begin with the discrete version.

**Theorem 6.4.1** (Discrete  $t$ -resilient ACT). *A task  $(\mathcal{I}, \mathcal{O}, \Gamma)$  has a  $t$ -resilient read-write protocol if and only if there exists a color-preserving, carrier-preserving, simplicial map  $\phi : \text{Ch}_t^N(\mathcal{I}) \rightarrow \mathcal{O}$  for some natural number  $N$ .*

*Proof.* Given such a map, consider the following protocol. Each process with input vertex  $v$  runs  $N$  delayed snapshot rounds, choosing a vertex  $w$  in  $\text{Ch}_t^N(\mathcal{I})$ . The process decides on  $x = \phi(w)$ . This protocol satisfies the task specification because  $\phi$  is carried by  $\Gamma$ .

In the other direction, we argue that any  $t$ -resilient protocol can be put into normal form as an equivalent sequence of  $t$ -resilient delayed snapshots. It is known that any wait-free read-write protocol can be expressed in normal form as a *layered immediate snapshot* protocol [22, Ch.14]: Each layer has its own array, where each process writes its current state, takes an immediate snapshot, and the value returned by that snapshot becomes the process's new state.

In a  $t$ -resilient model, we can guarantee that each layer's immediate snapshot returns states written by at least  $n + 1 - t$  distinct processes by waiting until enough processes have finished the prior layer (as in Line 11 of Algorithm 3). We call such a layer a *barrier* layer. Given a  $t$ -resilient protocol expressed as a sequence of barrier layers, we can add a wait-free layer between each pair of barrier layers without reducing the protocol's computational power. A sequence of  $2L$  layers that alternates wait-free and barrier layers is a sequence of  $L$  delayed snapshot layers.

□

We could choose to replace  $\text{Ch}_t^N(\mathcal{I})$  in the discrete  $t$ -resilient ACT with the complex  $\text{Ch}^M(\text{Ch}_t(\mathcal{I}))$ , using the wait reduction theorem. Doing this would mean that the task is solvable by one round of the delayed snapshot protocol, followed by some number of wait-free immediate snapshots. Though we do not do this for the discrete  $t$ -resilient ACT, we will do so for continuous  $t$ -resilient ACT.

While the discrete  $t$ -resilient ACT provides a clean characterization of  $t$ -resilient solvability, there



is an even more succinct statement replacing simplicial maps with continuous functions. The main difference is that we lose any notion of process names, since there is no clear continuous analog of *chromatic* simplicial maps. To address this, we give an alternative to the discrete  $t$ -resilient ACT, subject to a link-connectivity condition on the output complex.

**Theorem 6.4.2** (Continuous  $t$ -resilient ACT). *Let  $T = (\mathcal{I}, \mathcal{O}, \Gamma)$  be an  $(n + 1)$ -process task such that  $\Gamma(\sigma)$  is link-connected for all  $\sigma \in \mathcal{I}$ . Then there is a  $t$ -resilient read-write protocol for  $T$  if and only if there is a continuous, carrier-preserving  $f : |\text{Ch}_t(\mathcal{I})| \rightarrow |\mathcal{O}|$ .*

*Proof.* First suppose we have a  $t$ -resilient protocol for task  $T$ . The discrete  $t$ -resilient ACT ensures there is a chromatic, carrier-preserving, simplicial map  $\phi : \text{Ch}_t^N(\mathcal{I}) \rightarrow \mathcal{O}$ . We apply Theorem 6.3.4 to turn  $\phi$  into a color-preserving, carrier-preserving, simplicial map  $\phi' : \text{Ch}^M(\text{Ch}_t(\mathcal{I})) \rightarrow \mathcal{O}$ . Then let  $f = |\phi'|$ , the geometric realization of  $\phi'$ . Recalling that  $\text{Ch}^M$  does not change the topology of simplicial complexes, we get a continuous, carrier preserving  $f : |\text{Ch}_t(\mathcal{I})| \rightarrow |\mathcal{O}|$ .

In the other direction, given a continuous, carrier-preserving  $f : |\text{Ch}_t(\mathcal{I})| \rightarrow |\mathcal{O}|$ , there is a color-preserving, carrier-preserving simplicial map  $\phi : \text{Ch}^M(\text{Ch}_t(\mathcal{I})) \rightarrow \mathcal{O}$ . Operationally, this provides us a  $t$ -resilient protocol for solving  $T$ , in which each process executes one round of delayed snapshot, followed by  $M$  rounds of immediate snapshots, halting on a vertex  $v$  in  $\text{Ch}^M(\text{Ch}_t(\mathcal{I}))$ . The process decides  $\phi(v)$ , which is correct because  $\phi$  is carrier-preserving.  $\square$

## 6.5 Applications of the $t$ -resilient ACT

In this section we provide a simple application of the  $t$ -resilient ACT to the test-and-set task, which models the common *test-and-set* synchronization hardware primitive.

**Lemma 6.5.1.** *If  $\mathcal{K}$  and  $\mathcal{L}$  are chromatic complexes,  $\mathcal{K}$  is link-connected, and  $\phi : \mathcal{K} \rightarrow \mathcal{L}$  is color-preserving, then the subcomplex  $\phi(\mathcal{K}) \subseteq \mathcal{L}$  is also link-connected.*

*Proof.* Color-preserving maps between chromatic complexes must be injective, hence such a map is isomorphic onto its image. So in particular, it preserves links. Therefore the complex  $\phi(\mathcal{K})$  is link-connected.  $\square$

In the following two tasks, processes have only their names as inputs. In the *test-and-set* task, exactly one participating process decides 0, and the rest decide 1. Figure 6.8 shows this task's

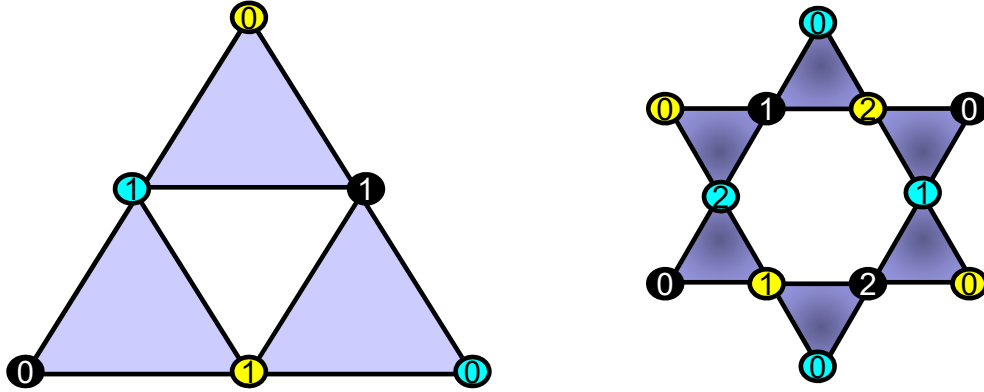


Figure 6.8: Test-and-set and fetch-and-increment tasks

output complex  $\mathcal{O}$ . If there were a  $t$ -resilient test-and-set protocol, there would be a color-preserving simplicial map  $\phi : \text{Ch}_t^n(\Delta^n) \rightarrow \mathcal{O}$ . It is not hard to see that the image of  $\text{Ch}_t^n(\Delta^n)$  must be all of  $\mathcal{O}$ . But  $\text{Ch}_t^n(\Delta^n)$  is link-connected, while  $\mathcal{O}$  is not, contradicting Lemma 6.5.1.

In the *fetch-and-increment* task, if  $k$  processes participate, they decide distinct integers between 0 and  $k - 1$ . Figure 6.8 shows this task's output complex  $\mathcal{O}$ . By the same argument, there can be no a color-preserving simplicial map  $\phi : \text{Ch}_t^n(\Delta^n) \rightarrow \mathcal{O}$  because  $\mathcal{O}$  is not link-connected.

In the  $k$ -set agreement task, each process has a private input, each process decides some process's input, and no more than  $k$  distinct inputs can be decided. We can use the  $t$ -resilient ACT to see that there is a  $t$ -resilient  $(t + 1)$ -set agreement protocol. Without loss of generality, assume each process's input is its own name, so the task's input complex is the simplex  $\Delta^n$ . Simply assign each vertex  $v$  in  $\text{Ch}_t(\Delta)$  the least process name in  $v$ 's carrier.

We can also use the  $t$ -resilient ACT to see that there is no  $t$ -resilient  $t$ -set agreement protocol. Omitting details, start by “coloring” each vertex of  $\text{Ch}_t(\Delta)$  with its decision value. We can extend this coloring from  $\text{Ch}_t(\Delta^n)$  to all of  $\text{Ch}^2(\Delta^n)$  simply by assigning each additional vertex a decision value from its carrier. The result is called a *Sperner coloring*, and the classical Sperner's Lemma [22, Ch.4] states that at least one  $n$ -simplex in  $\text{Ch}^2(\Delta^n)$  has all  $n + 1$  colors. Going back from  $\text{Ch}^2(\Delta^n)$  to  $\text{Ch}_t(\Delta^n)$  requires discarding at most  $n - t$  vertices from any simplex, leaving at least  $t + 1$  colors on some simplex of  $\text{Ch}_t(\Delta^n)$ . This simplex corresponds to an execution where  $(t + 1)$  distinct values are chosen, violating the  $t$ -set agreement condition. It is straightforward to extend this construction to any  $\text{Ch}_t^N(\Delta)$  by considering the  $N$ -fold *relative subdivision* [35] of  $\text{Ch}_t(\Delta^n)$  in  $\text{Ch}^2(\Delta^n)$ .

## 6.6 Concluding Remarks

In this chapter we generalized the asynchronous computability theorem to the  $t$ -resilient model of fault-tolerance, given topological and combinatorial conditions analogous to the original theorem characterizing wait-free solvability. This work highlights the power of the convergence algorithm in reducing a combinatorial or algorithmic problem into a topological one, which allows the application of more mathematical machinery from classical algebraic topology. It further evidences the thesis that there is much potential from classical methods in topology to be discovered in the context of distributed computability.

## Chapter 7

# Computability against Adversaries

In the previous chapter we demonstrated how the asynchronous computability theorem can be generalized to the  $t$ -resilient model of fault-tolerance. As mentioned in the introduction though, the  $t$ -resilient model assumes that failures are, in some sense, *uniform*, with maximal failure sets all being the same size. In real-world distributed systems, failures may be less symmetric, and more correlated, if certain sets of processes depend on common infrastructure.

The introduction discuss a more general model of fault-tolerance, called *resilience against an adversary*, that captures non-uniform failures such as these. In a distributed system, recall that an *adversary* is an entity capable of failing certain subsets of processes [9], but not others. Any adversary considered in this chapter is *superset-closed*, or one satisfying the following condition: if it capable of failing a set  $F$  of processes, then it can also fail any proper subset  $F' \subset F$ . We focus on these kinds of adversaries because they capture many realistic settings respecting the principle that fault-tolerant algorithms should continue to be correct if run in a system that displays *fewer failures* than anticipated. We also show that superset-closed adversaries exhibit nicer mathematical properties.

As before, it is natural to ask whether or how the asynchronous computability theorem generalizes to characterizing solvability of a task in a system with an adversary. In this chapter, we discuss how one would approach generalizing this theorem for computability against adversaries. As with previous work on wait-freedom and  $t$ -resilience, we define a snapshot protocol, called the *adversarial snapshot*, as a potential building block for protocols resilient against a given adversary. Furthermore, we show that it is not possible to approach this problem simply by generalizing the  $t$ -resilient

approach; that is, the one-round adversarial snapshot does not simulate multiple rounds. We explore ways in which to characterize resilience against an adversary, and present evidence supporting the hypothesis that *two* rounds of adversarial snapshot can simulate any higher number of rounds.

## 7.1 Characterizing Adversaries

For this chapter, we fix a set of  $n + 1$  processes called  $\Pi$ , and consider these processes running in a system with an *adversary*  $\mathcal{A}$ , which has the ability to control whether certain subsets of processes fail. There are a handful of ways to characterize such an adversary. The most straightforward is by enumerating its *faulty sets*, or sets of processes that may *fail* together in some execution. We assume that faulty sets are *downward closed*, since this is essentially what is meant for an adversary to be superset-closed (closed under taking supersets of non-faulty processes).

For this work, it is more convenient to characterize adversaries in terms of their *cores* and *survivor sets* [26]. Recall that a core is a minimal sets of processes  $C \subseteq \Pi$  that the adversary cannot simultaneously fail. Dual to the notion of cores are survivor sets (sometimes called *hitting sets*), which are minimal sets of processes that intersect every core. Just like with maximal failure sets, cores and survivor sets each completely determine an adversary.

In any execution, the set of non-faulty processes within that execution always contains a survivor set, so it is safe for any process to wait for all members of *some* survivor set to show up.

Adversaries generalize the  $t$ -resilient model by capturing non-uniform failures, so that maximal faulty sets may have different sizes. The cores of the  $t$ -resilient adversary, denoted  $\mathcal{A}_t$ , are the sets of size  $t + 1$ , and while its survivors sets are those of size  $n + 1 - t$ . The wait-free adversary has one core given by the entire set of processes  $\Pi$ ; its survivor sets are singleton sets of processes.

### 7.1.1 Core Complexes

Next, we define the *core complex*, which is a more topological way of characterizing and discussing adversaries. The core complex is a useful construction in defining the adversarial snapshot task and protocol, and also completely determines an adversary.

**Definition 7.1.1** (core complex). *Let  $\mathcal{A}$  be an adversary in a system of  $n + 1$  processes  $\Pi$ . Then the core complex of  $\mathcal{A}$ , denoted  $\mathcal{C}(\mathcal{A})$ , is the subcomplex of  $\Delta^n$  whose facets are given by simplexes  $\Pi - C$ , for each  $C$  a core of  $\Pi$ .*

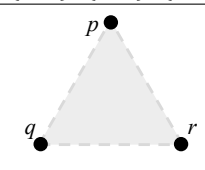
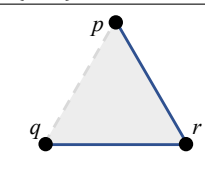
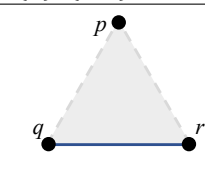
|                      |   |  |   |
|----------------------|---|--|---|
| <b>Adversary</b>     | 1-resilient   | $p$ and $q$ non-faulty   | $p$ non-faulty, $q$ or $r$ non-faulty ( $\mathcal{B}$ )                             |
| <b>Cores</b>         | $\{p, q\}, \{p, r\}, \{q, r\}$  | $\{p\}, \{q\}$   | $\{p\}, \{q, r\}$   |
| <b>Survivor sets</b> | $\{p, q\}, \{p, r\}, \{q, r\}$  | $\{p, q\}$   | $\{p\}, \{q, r\}$   |
| <b>Core complex</b>  |  |  |  |

Figure 7.1: Examples of three-process adversaries and their cores, survivor sets, and core complexes.

That is, the core complex of an adversary  $\mathcal{A}$  is constructed by taking each the complement of each core  $C$ , and letting  $\Pi - C$  be a maximal simplex in the core complex. Since the set of cores and set of core complements determine one another, it is clear that the core complex is another way to characterize adversaries.

For example, if  $\mathcal{A}_t$  is the  $t$ -resilient adversary, then its core complex is given by  $\text{skel}^{n-t-1}(\Delta^n)$ . This is because the cores of  $\mathcal{A}_t$  are the subsets of  $\Pi$  of size  $t + 1$ , so the cores' complements are all sets of size  $n - t - 1$ . An adversary that is not a  $t$ -resilient adversary is called *irregular*. See the table in Figure 7.1, illustrating the three-process, 1-resilient adversary, and two other, irregular adversaries. We name the adversary in the third column as adversary  $\mathcal{B}$ , and use it as a running example for the rest of this chapter.

When only a subset of processes are active in a system with an adversary, it is useful to define a new adversary that captures the power of the original against this subset of processes. We define the restriction of an adversary  $\mathcal{A}$ , where  $\mathcal{A}$  operates only against processes that are non-faulty. This is technically useful in later sections, since adversaries have an inductive structure given by restricting to subsets of processes.

**Definition 7.1.2** (restricted adversary). *Let  $\mathcal{A}$  be an adversary in a system with processes  $\Pi$ , and suppose  $\Pi' \subseteq \Pi$  have been failed by the adversary. Then the restricted adversary  $\mathcal{A}|_{\Pi'}$  is one that operates against the non-faulty processes  $\Pi - \Pi'$ , and has cores  $C - \Pi'$ , for each core  $C$  of  $\mathcal{A}$  (redundant sets obtained this way are discarded). A restricted adversary with an empty core is degenerate.*

Using the concept of a core complex, we define the adversarial snapshot protocol. We define it

as a task to be solved, and give an adversary-resilient protocol to solve it.

## 7.2 Adversarial Snapshot Protocol

The adversarial snapshot is defined as a subtask of the two-round immediate snapshot. Recall the definition of deletion within a simplicial complex: if  $\mathcal{K} \subseteq \mathcal{L}$  are simplicial complexes, then the *deletion* of  $\mathcal{K}$  in  $\mathcal{L}$ , denoted  $dl(\mathcal{K}, \mathcal{L})$ , is the subcomplex of  $\mathcal{L}$  containing all simplexes that do not intersect  $\mathcal{K}$ . Then the adversarial snapshot complex is defined as a deletion of the second standard chromatic subdivision.

**Definition 7.2.1** (Adversarial snapshot task). *Let  $\mathcal{A}$  be an adversary. Then the adversarial protocol complex is defined as  $Ch_{\mathcal{A}}(\Delta^n) = dl(Ch^2(\mathcal{C}(\mathcal{A})), Ch^2(\Delta^n))$ , or the complex obtained by removing all simplexes of  $Ch^2(\Delta^n)$  that intersect  $\mathcal{C}(\mathcal{A})$ . The adversarial snapshot task is the task given by  $(\Delta^n, Ch_{\mathcal{A}}(\Delta^n), Ch_{\mathcal{A}})$ .*

Note that adversarial protocol complex induces an *adversarial snapshot operator*  $Ch_{\mathcal{A}}$  on arbitrary input complexes  $\mathcal{I}$ , given by applying  $Ch_{\mathcal{A}}$  to each  $n$ -simplex  $\sigma = \Delta^n$  in  $\mathcal{I}$ . This is due to the boundary consistency of  $Ch_{\mathcal{A}}$ . Recalling the adversary  $\mathcal{B}$  as a running example, see how  $Ch_{\mathcal{B}}(\Delta^n)$  is constructed in Figure 7.2.

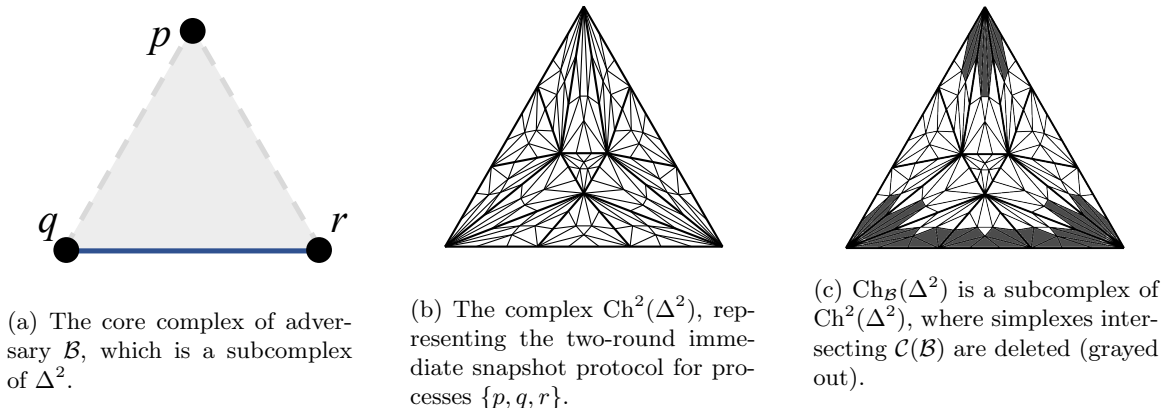


Figure 7.2: Adversary  $\mathcal{B}$  has cores  $\{p\}$  and  $\{q, r\}$ . Pictorially,  $Ch_{\mathcal{B}}(\Delta^2)$  is obtained by overlaying  $\mathcal{C}(\mathcal{B})$  on top of  $Ch^2(\Delta^2)$  and deleting simplexes in their intersection.

We define a two-round immediate snapshot protocol for this task, where the first and second snapshots are separated by the following barrier. If a survivor set of processes has not completed the first snapshot, then the processes must wait for a survivor set before proceeding to the second

snapshot. Once a survivor set has been observed by some process, this process opens the barrier allowing all other processes to continue wait-free. Pseudocode is given in Algorithm 3.

```

1 shared mem0[ $n+1$ ];
2 shared mem1[ $n+1$ ];
3 shared done;
4 done := false;
5 protocol AdversarialSnapshot( $id$ ):
6   immediate
7   |   mem0[ $id$ ] :=  $id$ ;
8   |   snap0 := snapshot(mem0);
9   |   if  $\exists$  core  $c$  :  $snap0 \cap c = \emptyset$  then
10  |   |   while not done
11  |   |   |   skip
12  |   immediate
13  |   |   mem1[ $id$ ] := snap0;
14  |   |   snap1 := snapshot(mem1);
15  |   done := true;
16  |   return snap1;

```

**Algorithm 3:** The adversarial snapshot protocol.

**Theorem 7.2.2.** *The adversarial snapshot protocol solves the adversarial snapshot task.*

*Proof.* Let  $\mathcal{A}$  be the adversary in question. To show correctness, we prove two statements: (1) the protocol terminates, and (2) it returns outputs compliant with the adversarial snapshot task. To verify termination, consider an execution in the presence of  $\mathcal{A}$ . The set of non-faulty processes in the execution must contain a survivor set  $S$ . The last process  $p \in S$  to perform its first immediate snapshot observes the entire survivor set  $S$  in its snapshot, so it proceeds past the wait barrier, allowing all waiting and subsequent processes to continue wait-free. Therefore all non-faulty processes eventually return with outputs, proving (1).

To prove (2), first notice that processes executing the protocol choose a simplex in  $\text{Ch}_2(\Delta^n)$ , since the protocol consists of two successive immediate snapshots on clean memory (albeit separated by a wait barrier). We must further show that each process chooses a vertex in  $\text{Ch}_{\mathcal{A}}(\Delta^n)$ . Toward a contradiction, suppose process  $q$  completes the protocol, but choose a vertex not contained in  $\text{Ch}_{\mathcal{A}}(\Delta^n)$ . Then its carrier is colored only by processes in  $\Pi - C$ , for some core  $C$  of  $\mathcal{A}$ . Operationally, this means that in its entire execution,  $q$  did not observe any process in  $C$ . In particular, if  $V_q$  is the set of processes contained in the first snapshot of  $q$ , then we have  $V_q \cap C = \emptyset$ . However, for  $q$  to have proceeded past the wait barrier,  $V_q$  must have contained some survivor set  $S$ , or  $S \subseteq V_q$ . From the definition of a survivor set,  $S$  intersects every core  $C$ , so  $S \cap C \neq \emptyset$ . But from the containment



$S \subseteq V_q$ , it follows that  $S \cap C \subseteq V_q \cap C$ , which is a contradiction, since  $V_q \cap C$  is empty and  $S \cap C$  is not. This proves property (2).

We conclude that processes executing the adversarial snapshot protocol against  $\mathcal{A}$  will collectively choose a simplex in  $\text{Ch}_{\mathcal{A}}(\Delta^n)$ , thus solving the adversarial snapshot task. □

As with other snapshot algorithms, one can sequentially compose several instances of the adversarial snapshot to obtain an *iterated adversarial snapshot protocol*. The operator for  $N$  instances of adversarial snapshot is denoted by  $\text{Ch}_{\mathcal{A}}^N$ .

If the adversarial snapshot protocol is restricted to the  $t$ -resilient model, we get precisely the *delayed snapshot protocol* by Saraph *et al.* [39]. In their work, they show that  $t$ -resilient protocols can be built from successive delayed snapshots. In the process, they prove that one delayed snapshot can wait-free simulate several rounds of the protocol. One may conjecture that their simulation easily translates to the adversaries; however as we will see, it fails to hold in general. The next section exploits adversary  $\mathcal{B}$  to demonstrate this failure.

### 7.3 Impossibility of Single-Round Waiting

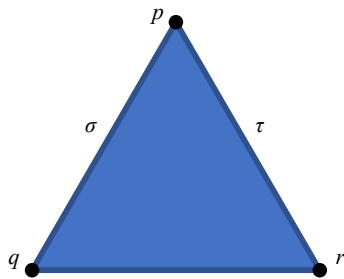
Recall that any  $t$ -resiliently solvable task can be solved with a  $t$ -resilient protocol that uses only one wait barrier. Stated another way, any number of rounds of delayed snapshots may be simulated in shared memory from just one round. This result, called the *wait reduction theorem*, was used to prove a topological statement of the  $t$ -resilient asynchronous computability theorem, one which only used one wait barrier. Unfortunately, the wait reduction theorem does not immediately generalize to adversaries; that is, there are adversaries for which iterated adversarial snapshot cannot be simulated from one round. The wait reduction theorem for the  $t$ -resilient model is given below, in both combinatorial and operational terms:

**Theorem 7.3.1** (*t-resilient wait reduction*). *There is a wait-free simulation of the  $N$ -round delayed snapshot protocol from one round of the protocol. Combinatorially, for any input complex of  $\mathcal{I}$  of some task, and any  $N > 0$ , there exists a color-preserving simplicial map  $\phi : \text{Ch}^M(\text{Ch}_{\mathcal{A}_t}(\mathcal{I})) \rightarrow \text{Ch}^N(\mathcal{I})$ .*

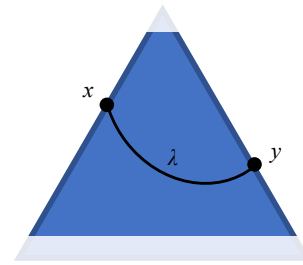
It is then natural to ask whether this theorem still holds if the  $t$ -resilient adversary  $\mathcal{A}_t$  is replaced

with an arbitrary adversary  $\mathcal{A}$ . However as previously noted, the general case fails to hold. As a counterexample, recall the three-process adversary  $\mathcal{B}$  characterized by cores  $\{p\}$  and  $\{q, r\}$ . Then one  $\mathcal{B}$ -resilient snapshot cannot simulate two using only wait-free immediate snapshots. This statement can be posed as a question of whether a distributed task is solvable wait-free in read-write memory.

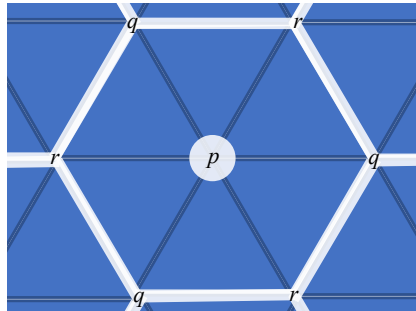
**Theorem 7.3.2** (one-round impossibility). *There is no wait-free read-write protocol that solves the task  $(\text{Ch}_{\mathcal{B}}(\Delta^2), \text{Ch}_{\mathcal{B}}^2(\Delta^2), \text{Ch}_{\mathcal{B}})$ .*



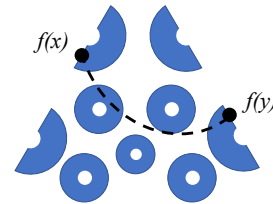
(a) Three-process input complex  $\Delta^2$ , with simplexes  $\sigma = \{p, q\}$  and  $\tau = \{p, r\}$ .



(b) Points  $x \in \text{Ch}_{\mathcal{B}}(\sigma)$  and  $y \in \text{Ch}_{\mathcal{B}}(\tau)$ , and a path  $\lambda$  in  $\text{Ch}_{\mathcal{A}}^2(\Delta^2)$  that join them. The lightly shaded regions are those removed from  $|\Delta^2|$  after applying  $\text{Ch}_{\mathcal{B}}$ .



(c) Zooming in on the image around a vertex  $v \in \text{Ch}_{\mathcal{B}}(\Delta^2)$  colored by  $p$ , after applying the second  $\text{Ch}_{\mathcal{B}}$ . The components of  $\text{Ch}_{\mathcal{B}}^2(\Delta^2)$  are punctured disks.



(d) Points  $f(x) \in \text{Ch}_{\mathcal{B}}^2(\sigma)$  and  $f(y) \in \text{Ch}_{\mathcal{B}}^2(\tau)$ . The dashed path between  $f(x)$  and  $f(y)$  signifies its non-existence.

Figure 7.3: Impossibility of simulating two rounds of  $\mathcal{B}$ -resilient snapshot from one round.

*Proof.* Towards a contradiction, suppose that this task is wait-free solvable. Then by the wait-free asynchronous computability theorem, there is a color-preserving simplicial map  $\delta : \text{Ch}^N(\text{Ch}_{\mathcal{B}}(\Delta^n)) \rightarrow \text{Ch}_{\mathcal{B}}^2(\Delta^n)$  carried by  $\text{Ch}_{\mathcal{B}}$ .

Consider the induced continuous map  $f = |\delta|$ , and let  $\sigma = \{p, q\}$  and  $\tau = \{p, r\}$  be 1-simplexes in  $\Delta^2$  as shown in Figure 7.3a. Choose any points  $x \in |\text{Ch}_{\mathcal{B}}(\sigma)|$  and  $y \in |\text{Ch}_{\mathcal{B}}(\tau)|$ . Then as illustrated in Figure 7.3b, there is a path  $\lambda$  in  $|\text{Ch}_{\mathcal{B}}(\Delta^2)|$  connecting  $x$  and  $y$ . Now consider the points  $f(x)$  and  $f(y)$  in  $|\text{Ch}_{\mathcal{B}}^2(\Delta^2)|$ . Since  $f$  is carried by  $\text{Ch}_{\mathcal{B}}$ ,  $f$  must map  $x$  into  $|\text{Ch}_{\mathcal{B}}^2(\sigma)|$  and  $y$  into  $|\text{Ch}_{\mathcal{B}}^2(\tau)|$ . Moreover,  $f$  maps the path  $\lambda$  to a path in  $\text{Ch}_{\mathcal{B}}^2(\Delta^2)$ , connecting  $f(x)$  and  $f(y)$ . We argue that such a path cannot exist.

See the schematic in Figure 7.3d, depicting  $|\text{Ch}_{\mathcal{B}}^2(\Delta^2)|$ . It is technically not an accurate depiction of  $\text{Ch}_{\mathcal{B}}^2(\Delta^2)$ , since  $\text{Ch}_{\mathcal{B}}^2(\Delta^n)$  is a subcomplex  $\text{Ch}^4(\Delta^2)$ . But  $\text{Ch}^4(\Delta^2)$  has 28,561 simplexes, so clearly it is not feasible to accurately illustrate it within the confines of this paper. Instead, Figure 7.3d captures essential topological properties of  $\text{Ch}_{\mathcal{B}}^2(\Delta^2)$ , namely that the complex is disconnected. This can be understood by looking at Figure 7.3c, which illustrates a zoomed-in view of a hypothetical component in  $\text{Ch}_{\mathcal{B}}^2(\Delta^n)$ . If  $v$  is a vertex in  $\text{Ch}_{\mathcal{B}}(\Delta^2)$  colored by process  $p$ , then its neighborhood becomes disconnected when applying  $\text{Ch}_{\mathcal{B}}$  to  $\text{Ch}_{\mathcal{B}}(\Delta^n)$  (yielding  $\text{Ch}_{\mathcal{B}}^2(\Delta^n)$ ), since the  $\{q, r\}$  edges have been stripped away.

Looking at Figure 7.3d, notice that there is no connected component intersecting both  $|\text{Ch}_{\mathcal{B}}^2(\sigma)|$  and  $|\text{Ch}_{\mathcal{B}}^2(\tau)|$ , since the only possible such component was deleted from  $\Delta^2$  upon applying the first  $\text{Ch}_{\mathcal{B}}$  operator to it. So  $f(x)$  and  $f(y)$  must be in different components, so there cannot be a path between  $f(x)$  and  $f(y)$ . This contradicts the existence of the constructed path  $f(\lambda)$ . Thus one round of  $\mathcal{B}$ -resilient snapshot fails to simulate two rounds of  $\mathcal{B}$ -resilient snapshot.

□

Therefore for the irregular adversary  $\mathcal{B}$ , one adversarial snapshot cannot simulate multiple snapshots wait-free. This is fundamentally due to the topological asymmetry of the protocol complex in question. If one were to follow the argument in the  $t$ -resilient wait reduction theorem, then we would retract the one-round protocol complex to a low-dimensional subspace, and then map it into higher-round complexes by using topological connectivity. But in this instance, we can only retract  $\text{Ch}_{\mathcal{B}}(\Delta^2)$  to a one-dimensional subspace, due to the constraint that the retraction must be carrier-preserving. However, when looking at the higher round complexes, we have a space that is only  $(-1)$ -connected, and not one that possesses the 0-connectivity required to allow the argument to go through. In summary, there is a mismatch in the lowest dimensional retraction of  $\text{Ch}_{\mathcal{B}}(\Delta^n)$ , and the connectivity of  $\text{Ch}_{\mathcal{B}}^N(\Delta^n)$ .

### 7.3.1 More Rounds

The results in this section motivate the following question: if a single round of adversarial snapshots is not sufficient to simulate arbitrarily many snapshots wait-free, then how many rounds are enough? Evidence suggests that in all cases, *two* rounds are enough to simulate any larger number of rounds of adversarial snapshot. Indeed, for the adversary  $\mathcal{B}$ , two rounds of adversarial snapshots can simulate  $N$ -rounds.

In order to prove such a result, one cannot take the retraction approach for  $t$ -resilient adversaries. Instead, studying the “hole” spaces of the two-round and  $N$ -round complexes, or their respective complements within  $\Delta^n$ , may be more useful. For the adversary  $\mathcal{B}$ , the key intuition is that the two-round  $\text{Ch}_{\mathcal{B}}^2(\Delta^n)$  has the same connectivity properties as any higher-round complex. Informally, after applying  $\text{Ch}_{\mathcal{A}}$  twice, no new *kinds* of holes are produced in any subsequent round, the only difference is that *more* holes appear in higher rounds. Thus we are able to take a characteristic hole in  $\text{Ch}_{\mathcal{B}}^2(\Delta^n)$  and “wrap it around” every hole in  $\text{Ch}_{\mathcal{B}}^N(\Delta^n)$ . Contrast this with the one-round  $\text{Ch}_{\mathcal{B}}(\Delta^n)$ , which is always connected in every dimension, thus restricting the protocols it is capable of simulating.

While the adversary  $\mathcal{B}$  provides an example of a task which is solvable with two rounds of adversarial snapshots, even two rounds may not suffice in all cases. In systems with more processes than three, there are adversaries that require at least three rounds of adversarial snapshot before one may proceed wait-free.

## 7.4 Concluding Remarks

This chapter discussed how the delayed snapshot for the  $t$ -resilience model of fault-tolerance may be generalized to a building block protocol for adversaries, called the adversarial snapshot. Previous work on cores and survivor sets was used to develop the adversarial snapshot. In future work, one may consider how to characterize *stability* of the adversarial snapshot, or after how many rounds of adversarial snapshots are necessary before processes proceed wait-free. In the  $t$ -resilient model, the question of stability is trivial, since it was shown that processes only require one wait barrier before they can continue wait-free. However, in this chapter, we gave an example of an irregular adversary where this is not the case. Furthermore, it is hypothesized that there are adversaries for which not even two rounds suffice. Intuitively, this is because more subdivisions are required to realize the full

amount of disconnectedness that the adversarial snapshot is able to produce.

# Bibliography

- [1] Hagit Attiya, Amotz Bar-Noy, Danny Dolev, Daphne Koller, David Peleg, and Rudiger Reischuk. Achievable cases in an asynchronous environment. pages 337–346, October 1987.
- [2] Hagit Attiya, Armando Castañeda, Maurice Herlihy, and Ami Paz. Upper bound on the complexity of solving hard renaming. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 190–199, New York, NY, USA, 2013. ACM.
- [3] O. Biran, S. Moran, and S. Zaks. A combinatorial characterization of the distributed 1-solvable tasks. *Journal of Algorithms*, 11:420–440, 1990.
- [4] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for  $t$ -resilient asynchronous computations. May 1993.
- [5] Elizabeth Borowsky and Eli Gafni. A simple algorithmically reasoned characterization of wait-free computations. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, pages 189–198, August 1997.
- [6] Elizabeth Borowsky, Eli Gafni, Nancy A. Lynch, and Sergio Rajsbaum. The BG distributed simulation algorithm. *Distributed Computing*, 14(3):127–146, 2001.
- [7] Armando Castañeda and Sergio Rajsbaum. New combinatorial topology upper and lower bounds for renaming. In *PODC '08: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 295–304, New York, NY, USA, 2008. ACM.
- [8] Soma Chaudhuri. Agreement is harder than consensus: set consensus problems in totally asynchronous systems. pages 311–234, August 1990.

- [9] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Andreas Tielmann. The disagreement power of an adversary. In *Distributed Computing, 23rd International Symposium, DISC 2009, Elche, Spain, September 23-25, 2009. Proceedings*, pages 8–21, 2009.
- [10] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, June 1982.
- [11] E. Gafni and E. Koutsoupias. Three-processor tasks are undecidable. *SIAM J. Comput.*, 28(3):970–983, 1999.
- [12] Eli Gafni, Yuan He, Petr Kuznetsov, and Thibault Rieutord. Read-Write Memory and k-Set Consensus as an Affine Task. In Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone, editors, *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, volume 70 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:17, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [13] Eli Gafni and Petr Kuznetsov. On l-resilience, hitting sets, and colorless tasks. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 81–82, New York, NY, USA, 2010. ACM.
- [14] Eli Gafni and Petr Kuznetsov. Turning adversaries into friends: Simplified, made constructive, and extended. In *Proceedings of the 14th International Conference on Principles of Distributed Systems*, OPODIS'10, pages 380–394, Berlin, Heidelberg, 2010. Springer-Verlag.
- [15] Eli Gafni, Petr Kuznetsov, and Ciprian Manolescu. A generalized asynchronous computability theorem. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 222–231, 2014.
- [16] Rachid Guerraoui and Petr Kuznetsov. Two faces of the asynchronous computability theorem. In *The 4th Workshop in Geometry and Topology in Concurrency and Distributed Computing*, 2004.
- [17] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.
- [18] M. P. Herlihy and S. Rajsbaum. The decidability of distributed decision tasks. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 589–598, 1997.

- [19] M. P. Herlihy and S. Rajsbaum. A classification of wait-free loop agreement tasks. *Theor. Comput. Sci.*, 291(1):55–77, 2003.
- [20] M. P. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 111–120, 1993.
- [21] M. P. Herlihy and N. Shavit. A simple constructive computability theorem for wait-free computation. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 243–252, 1994.
- [22] Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Elsevier, 2013.
- [23] Maurice Herlihy and Sergio Rajsbaum. The topology of distributed adversaries. *Distributed Computing*, 26(3):173–192, 2013.
- [24] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999.
- [25] Maurice P. Herlihy and Nir Shavit. The asynchronous computability theorem for t-resilient tasks. pages 111–120, May 1993.
- [26] Flavio Junqueira and Keith Marzullo. A framework for the design of dependent-failure algorithms. *Concurrency and Computation: Practice and Experience*, 19(17):2255–2269, 2007.
- [27] D. N. Kozlov. *Combinatorial Algebraic Topology*, volume 21 of *Algorithms and computation in mathematics*. Springer, 2008.
- [28] Dmitry N. Kozlov. Chromatic subdivision of a simplicial complex. *Homology, Homotopy, and Applications*, 1(14):1–13, 2012.
- [29] Dmitry N. Kozlov. Structure theory of flip graphs with applications to weak symmetry breaking. *CoRR*, abs/1511.00457, 2015.
- [30] Dmitry N. Kozlov. Topology of the view complex. *Homology, Homotopy, and Applications*, 17, 2015.



- [31] Petr Kuznetsov, Thibault Rieutord, and Yuan He. An asynchronous computability theorem for fair adversaries. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 387–396, New York, NY, USA, 2018. ACM.
- [32] S. Mac Lane. *Categories for the Working Mathematician*. Springer Verlag, 1998.
- [33] X. Liu, J. Pu, and J. Pan. A classification of degenerate loop agreement. In *Fifth IFIP International Conference On Theoretical Computer Science*, volume 273 of *IFIP*, pages 203–213. Springer, 2008.
- [34] X. Liu, Z. Xu, and J. Pan. Classifying rendezvous tasks of arbitrary dimension. *Theor. Comput. Sci.*, 410(21-23):2162–2173, 2009.
- [35] J. R. Munkres. *Elements Of Algebraic Topology*. Addison Wesley, Reading MA, 1984.
- [36] Michel Raynal and Julien Stainer. Increasing the power of the iterated immediate snapshot model with failure detectors. In *Structural Information and Communication Complexity - 19th International Colloquium, SIROCCO 2012, Reykjavik, Iceland, June 30-July 2, 2012, Revised Selected Papers*, pages 231–242, 2012.
- [37] Michael Saks and Fotis Zaharoglou. Wait-free  $k$ -set agreement is impossible: The topology of public knowledge. May 1993.
- [38] Vikram Saraph and Maurice Herlihy. The relative power of composite loop agreement tasks. In *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*, pages 13:1–13:16, 2015.
- [39] Vikram Saraph, Maurice Herlihy, and Eli Gafni. Asynchronous computability theorems for  $t$ -resilient systems. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, pages 428–441, 2016.
- [40] Vikram Saraph, Maurice Herlihy, and Eli Gafni. An algorithmic approach to the asynchronous computability theorem. *Journal of Applied and Computational Topology*, 1(3):451–474, Jun 2018.