Combinatorial Topology and Distributed Computing

Maurice Herlihy Dmitry Kozlov Sergio Rajsbaum

February 22, 2011

Contents

Part I: Undergraduate Course

Chapter 4

Connectivity

In Chapter 3, we used Sperner's Lemma to show that k-set agreement has no protocol in any model of computation where protocol complexes are manifolds. We saw one specific model, the round-by-round immediate snapshot model, whose protocol complexes are manifolds. Unfortunately, however, protocol complexes in naturally-arising models of computation are typically not manifolds. In this chapter, we show how to use more powerful mathematical techniques to establish when a model permits a protocol for k-set agreement and related tasks.

4.1 Consensus and Path-Connectivity

We start with a simple, model-independent topological condition that ensures that a particular model of computation cannot solve consensus.

Recall that in the consensus task, each process starts with a private *input value* and halts with an *output value* such that (1) all processes choose the same output value, and (2) the output value chosen was some process's input. For lower bounds, it is enough to consider a *fixed-input* form of the task. The task is given by $(\sigma, \text{skel}^0\,\sigma, \text{skel}^0(\cdot))$, where $\sigma = \{\vec{s}_0, \ldots, \vec{s}_n\}$ is an n-simplex and $\text{skel}^0\,\sigma$ is the set of vertices of σ. Each process P_i has $\vec{s}_i$ as its only possible input. Consider an execution where exactly $m \leq n$ processes participate. If τ is the face of σ defined by their inputs, then all processes halt with the same vertex of σ^m. It is easy to see that any lower bounds for the fixed-input form apply to the general case as well.

Informally, consensus requires that all participating processes "commit" to a single value. Expressed as a protocol complex, executions in which they commit to one value must be separate, in some sense, from executions in

which they commit to another value. We will now make this notion more precise.

Let $\mathcal{K}$ be a complex.

Definition 4.1.1. An *edge path* (or simply a *path*) between vertices $\vec{u}$ and $\vec{v}$ in $\mathcal{K}$ is a sequence of vertices $\vec{u} = \vec{v}_0, \ldots, \vec{v}_\ell = \vec{v}$ such that $\mathcal{K}$ contains an edge between each $\vec{v}_i$ and $\vec{v}_{i+1}$, $0 \leq i < \ell$.

Definition 4.1.2. A complex $\mathcal{K}$ is *path-connected* if there is a path between every two vertices in $\mathcal{K}$.

In the next theorem, we consider a protocol in an arbitrary model that admits at least one process failure. If all processes participate, the protocol complex must be path-connected. If one process does not participate, then the protocol complex need only be non-empty. (We do not need to say anything about how the protocol behaves for fewer participants.)

Theorem 4.1.3. If the protocol complex $\mathcal{P}(\sigma)$ is path-connected for every input n-simplex, and non-empty for every input $(n-1)$-simplex, then $\mathcal{P}(\cdot)$ cannot solve consensus.

Proof. Recall that $\mathrm{Face}_0\,\sigma$ is the input simplex that excludes $\vec{s}_0$. The subcomplex $\mathcal{P}(\mathrm{Face}_0\,\sigma)$ represents all executions in which P_0 does not participate. By hypothesis, this complex is non-empty, so we can pick an arbitrary vertex $\vec{p}$ in $\mathcal{P}(\mathrm{Face}_0\,\sigma)$. (The vertex $\vec{p}$ represents the final state of a process after some protocol execution in which P_0 does not participate.) The decision map $\delta : \mathcal{P}(\sigma) \rightarrow \mathrm{skel}^0\,\sigma$ sends $\vec{p}$ to a vertex $\vec{s}_i$ of σ, where $i \neq 0$. Informally, P_i is the "winner" of the execution that led to $\vec{p}$. P_0 cannot be the winner because it did not participate in this execution.

In the same way, pick a vertex $\vec{q}$ in $\mathcal{P}(\mathrm{Face}_i\,\sigma)$, where P_i is the winning process in $\vec{p}$. The decision map $\delta : \mathcal{P}(\sigma) \rightarrow \mathrm{skel}^0\,\sigma$ sends $\vec{q}$ to a vertex $\vec{s}_j$ of σ, where P_i and P_j are distinct. (The process P_i cannot be the "winner" of the execution leading to $\vec{q}$ because it did not participate.)

Because $\mathcal{P}(\sigma)$ is path-connected, there is a path $\vec{p} = \vec{p}_0, \ldots, \vec{p}_\ell = \vec{q}$ such that each successive pair $\{\vec{p}_k, \vec{p}_{k+1}\}$ forms an edge of the complex. Because the decision map δ is a simplicial map, it carries each edge to an edge in the output complex. For consensus, however, each output simplex consists of a single vertex of σ, so δ maps both $\vec{p}_k$ and $\vec{p}_{k+1}$ to the same vertex. By construction, the decision map δ carries $\vec{p}_0$ to $\vec{s}_i$. By a simple inductive argument, it sends every vertex in the path to $\vec{s}_i$, contradicting our hypothesis that $\delta(\vec{q}) = \vec{s}_j$, where $i \neq j$. □

This impossibility result is model-independent: it requires only that the model of computation permit at least one failure, so the protocol complex is defined on an $(n-1)$-simplex. We can use this theorem to derive three kinds of lower bounds.

- In asynchronous models, the adversary can typically enforce these conditions for every protocol complex. For these models, we can prove *impossibility*: consensus cannot be solved by any protocol.
- In synchronous models, the adversary can typically enforce these conditions for every r-round protocol, where r is a parameter of the model. For these models, we can prove *communication lower bounds*: consensus cannot be solved by any protocol that runs in r or fewer rounds.
- In semi-synchronous models, the adversary can typically enforce these conditions for every protocol that runs in less than a particular time T, where T is a parameter of the model. For these models, we can prove *time lower bounds*: consensus cannot be solved by any protocol that runs in time less than T.

In the next section, we apply this theorem to prove the impossibility of consensus in any asynchronous read-write model that permits at least one failure.

4.2 Consensus in Asynchronous Read-Write Memory

In this section, we show how to apply the general claim of Theorem 4.1.3 to a specific model of computation. We consider the asynchronous *read-write* model, in which processes share an array M of single-writer, multi-reader variables. Each process P_i has a dedicated array entry, $M[i]$, that it alone can write. Any process can read any other's array entry.

This model seems to be much less structured than the immediate snapshot model considered in earlier chapters. Indeed, the protocol complexes for this model are not manifolds. (Later on, we will prove the surprising fact that the wait-free read-write model is, in fact, equivalent to the immediate snapshot model.)

This section introduces a style of proof that we will use several times, called a *critical state* argument. This argument is useful in asynchronous models, where processes can take steps independently. As noted earlier, we can think of the system as a whole as a state machine, where each local

process state is a component of the global state. Each input n-simplex σ encodes a possible initial system state, the protocol complex $\mathcal{P}(\sigma)$ encodes all possible protocol executions starting from σ, and each facet of $\mathcal{P}(\sigma)$ encodes one possible final state. In the beginning, all interleavings are possible, and the entire protocol complex is reachable. At the end, an execution has been chosen, and only a single simplex remains reachable. In between, as the execution unfolds, we can think of the reachable part of the protocol complex as "shrinking" over time, as each step renders certain final states inaccessible.

We want to show that a particular property, such as having a path-connected reachable protocol complex, that holds in each final protocol state, also holds in the initial state. We argue by contradiction. We assume the property does not hold at the start, and manoevre the protocol into a *critical state* where the property still does not hold, but where any further step by any process will make it hold from that point on ("henceforth"). We then do a case analysis of each of the process's possible next steps, and use a combination of model-specific reasoning and basic topological results to show that the desired property must already have held in the critical state, a contradiction.

Let σ be an input m-simplex, $0 \leq m \leq n$, and let s be a global state reached by running $\mathcal{P}$ from the initial state given by σ. A simplex τ of $\mathcal{P}(\sigma)$ is *reachable* from s if there is an execution starting from s in which each process in $\mathrm{id}(\tau)$ completes the protocol with the local state specified in τ. The *reachable complex* from s, written $\mathcal{P}(s)$, is the union of the reachable simplices from s.

Notice that any input simplex σ defines an initial state, from which the reachable complex is just $\mathcal{P}(\sigma)$. For brevity, we say a state is *reachable* from input simplex σ if it is reachable from the initial state whose process IDs and inputs are given by σ.

Definition 4.2.1. Formally, a *property* is a predicate on isomorphism classes of simplicial complexes. A property is *eventual* if it holds for any complex consisting of a single n-simplex and its faces.

For brevity, we say that a property $\wp$ *holds* in global state s if $\wp$ holds for $\mathcal{P}(s)$, the reachable complex from s.

Definition 4.2.2. A global state s is *critical* for an eventual property $\wp$ if $\wp$ does not hold in s, but holds for every state reachable from s.

Informally, a critical state is the last state in an execution where $\wp$ fails to hold.

Lemma 4.2.3. Every eventual property either holds in every state, or it has a critical state.

Proof. A process is *non-critical* if its next step will not make an eventual $\wp$ henceforth hold. Starting from state s, repeatedly pick a non-critical pending process and run it until it is no longer non-critical. Because the protocol must eventually terminate in a state where $\wp$ holds, advancing non-critical processes in this way will eventually leave the protocol in a state where $\wp$ does not hold, but all processes are either decided or about to make $\wp$ henceforth hold. This state is the desired critical state. □

We need a way to reason about the path-connectivity of a complex from the path-connectivity of its components. The lemma that follows is a special case of the more powerful *Nerve Lemma* used later on to reason about higher-dimensional notions of connectivity.

Let $\mathcal{K}$ be a complex that can be expressed as the union of components over some finite index set I:

$$\mathcal{K} = \bigcup_{i \in I} \mathcal{K}_i.$$

Lemma 4.2.4. If each $\mathcal{K}_i$ is path-connected, and each pair-wise intersection $\mathcal{K}_i \cap \mathcal{K}_j$ is non-empty, then $\mathcal{K}$ itself is path-connected.

Proof. Left as an exercise. □

We now show that every wait-free read-write protocol satisfies the conditions of Theorem 4.1.3. We will prove a stronger property than necessary: for any n, if σ is an input n-simplex, then $\mathcal{P}(\sigma)$ is path-connected. (This condition implies that $\mathcal{P}(\sigma)$ is non-empty if $\dim \sigma = n - 1$.)

Lemma 4.2.5. If σ is an input n-simplex, then $\mathcal{P}(\sigma)$ is path-connected.

Proof. We argue by induction on n. When $n = 0$, the protocol is deterministic, and $\mathcal{P}(\sigma)$ is a single vertex.

For the induction hypothesis, assume that $\mathcal{P}(\sigma')$ is path-connected for every input m-simplex σ', for $m < n$. By way of contradiction, assume that $\mathcal{P}(\sigma)$ is not path-connected for some n-simplex σ. Path-connectivity is an eventual property, so by Lemma 4.2.3 it has a critical state c, such that $\mathcal{P}(c)$ is not path-connected, but each $\mathcal{P}(c_i)$ is path-connected, where c_i is the new state if P_i takes the next step.

$$\mathcal{P}(c) = \bigcup_{i \in \Pi} \mathcal{P}(c_i).$$

Because c is a critical state for path-connectivity, each $\mathcal{P}(c_i)$ is path-connected. We will show that for any distinct P_i and P_j, the complex $\mathcal{P}(c_i) \cap \mathcal{P}(c_j)$ is non-empty. By Lemma 4.2.4, it follows that $\mathcal{P}(c)$ was already path-connected, contradicting the assumption c is a critical state.

The rest is a case analysis considering which combinations of operations P_i and P_j could be about to do in c.

1. Suppose P_j is about to read. Consider the execution in which P_i runs to completion before P_j takes a step. Because P_i moved first, this execution leads to a simplex in $\mathcal{P}(c_i)$. Next, consider the same execution except that P_j reads, then P_i runs to completion before P_j takes another step. Because P_j moved first, this execution leads to a simplex in $\mathcal{P}(c_j)$. These two executions are indistinguishable to P_i, so both produce the same vertex for P_i, which lies in $\mathcal{P}(c_i) \cap \mathcal{P}(c_j)$.

2. Suppose P_i and P_j are about to write to distinct variables. Consider the execution in which P_i writes, P_j writes, and P_i runs to completion before P_j takes another step. Next, consider the execution in which P_i writes, P_j writes, and P_i runs to completion before P_j takes another step. These two executions are indistinguishable to P_i, so both produce the same vertex for P_i, which lies in $\mathcal{P}(c_i) \cap \mathcal{P}(c_j)$.

dmitry: something is wrong with 2 and 3 here

3. Suppose P_i and P_j are about to write to distinct variables. Consider the execution in which P_i runs to completion before P_j takes a step. Because P_i moved first, this execution leads to a simplex in $\mathcal{P}(c_i)$. Consider the execution in which P_j writes, and P_i runs to completion before P_j takes another step. These two executions are indistinguishable to P_i, so both produce the same vertex for P_i, which lies in $\mathcal{P}(c_i) \cap \mathcal{P}(c_j)$.

□

We have just shown the following.

Theorem 4.2.6. Let $\mathcal{P}(\cdot)$ be a wait-free read-write protocol complex. For every input simplex σ, $\mathcal{P}(\sigma)$ is path-connected.

Corollary 4.2.7. It is impossible to solve consensus using wait-free read-write memory.

4.3 Set Agreement and Connectivity in Higher Dimensions

In the previous section, we drew a connection between a topological property, path-connectivity, and the impossibility of solving a particular coordination problem, consensus. In this section, we draw a similar connection between a family of topological properties, called k-connectivity, and the impossibility of solving a family of coordination problems, k-set agreement.

We can rephrase notions of connectivity in terms of spheres and disks. If a complex is path-connected, then there is a path between any two vertices. Think of these two vertices as the image, under a continuous map, of a 0-dimensional sphere (the points ± 1 on the real line). The existence of the path means that this map from the 0-sphere can be extended to a continuous map of the 1-disk (the closed interval $[-1, 1]$). We say that a path-connected complex is *0-connected*.

This notion generalizes to higher dimensions in a natural way. A *loop* in a complex $\mathcal{K}$ is a path whose starting and end vertices are the same. A loop can be considered a continuous map $f : S^1 \to |\mathcal{K}|$, carrying the 1-sphere S^1 to the polyhedron of $\mathcal{K}$. A complex is 1-*connected* (or *simply-connected*) if any such map can be extended to the 2-disk: $F : D^2 \to |\mathcal{K}|$. In general, a complex is k-connected if any continuous map $f : S^k \to |\mathcal{K}|$ can be extended to $F : D^{k+1} \to |\mathcal{K}|$. One way to think about this property is that that any map f that cannot be filled in represents an n-dimensional "hole" in the complex. We will prove that the wait-free read-write protocol complex has no "holes" in dimension n or lower.

We have already seen that there is no protocol for k-set agreement in any model of computation where every protocol complex is a k-manifold. We now prove a stronger result: there is no protocol for k-set agreement in any model of computation where certain protocol complexes are $(k-1)$-connected. First, we must introduce a few new mathematical concepts.

maurice: Some of this material is duplicated in the "geometric view" part of the topology chapter

Although we have defined vertices, simplices, and complexes as abstract sets, it is sometimes convenient to treat them as point sets in Euclidean space. Let $\mathcal{K}$ be a complex. Each vertex $\vec{v}$ of $\mathcal{K}$ corresponds to a point $|\vec{v}|$. For each simplex $\sigma = \{\vec{s}_0, \ldots, \vec{s}_t\}$, the vertices correspond to affinely-independent points[1] $\{|\vec{s}_0|, \ldots, |\vec{s}_t|\}$, the *geometric simplex* $|\sigma|$ is their convex hull. The geometric complex $|\mathcal{K}|$ corresponds to a set of geometric simplices arranged so that that every two simplices intersect either in a common face,

[1] Points $x_0, \ldots, x_n$ are affinely independent if $x_1 - x_0, \ldots, x_n - x_0$ are linearly independent.

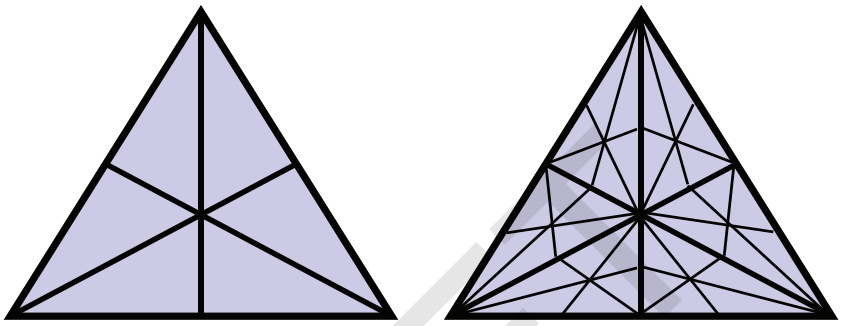

Figure 4.1: The complexes $\mathrm{Bary}\,\sigma$ and $\mathrm{Bary}^2\,\sigma$.

or not at all. The point set $|\mathcal{K}|$ occupied by a geometric complex $\mathcal{K}$ is called its *polyhedron*. For ease of presentation, we sometimes omit the distinction between abstract and geometric complexes when there is no danger of ambiguity.

Any point x of $|\mathcal{K}|$ has a unique expression in terms of *barycentric coordinates*:

$$x = \sum_{i=0}^{t} t_i \cdot |\vec{s}_i|$$

where the $\vec{s}_i$ are the vertices of a t-simplex σ^t of $\mathcal{K}$, and for $0 \leq i \leq t$, $\sum_i t_i = 1$, $0 < t_i \leq 1$.

maurice: We mention subdivisions earlier, although we don't use them. Do we want to define them in Chapter 2?

A geometric complex is *subdivided* by partitioning each of its simplices into smaller simplices without changing the complex's polyhedron. More formally, a complex $\mathcal{B}$ is a *subdivision* of $\mathcal{A}$ if

- For each simplex β of $\mathcal{B}$, there is a simplex α of $\mathcal{A}$ such that $|\beta| \subseteq |\alpha|$.

- For each simplex α of $\mathcal{A}$, $|\alpha|$ is the union of a finite set of polyhedrons of simplices of $\mathcal{B}$.

We would like to go back and forth between simplicial maps of complexes and continuous maps of polyhedrons. One direction is easy. Any simplicial map $\phi : \mathcal{A} \to \mathcal{B}$ can be turned into a piece-wise linear map $|\phi| : |\mathcal{A}| \to |\mathcal{B}|$ by extending over barycentric coordinates:

$$|\phi|(x) = \sum_i t_i \cdot \phi(\vec{s}_i).$$

Going from a continuous map to a simplicial map is more involved. We would like to "approximate" a continuous map from one polyhedron to another with a simplicial map on related complexes. Let $\mathcal{A}$ and $\mathcal{B}$ be complexes,

$$\begin{aligned}\phi &: \mathcal{A} \to \mathcal{B} \\ f &: |\mathcal{A}| \to |\mathcal{B}|\end{aligned}$$

where ϕ is a simplicial map of complexes and f a continuous map of their polyhedrons. We say that ϕ is a *simplicial approximation* to f if

$$f(\operatorname{St} \vec{v}) \subseteq \operatorname{St} \phi(\vec{v})$$

for every vertex $\vec{v}$ of $\mathcal{A}$.

The *diameter* $\operatorname{diam} \sigma$ of a geometric simplex $|\sigma|$ is the length of its longest edge. The *diameter* $\operatorname{diam} \mathcal{K}$ of a geometric complex $|\mathcal{K}|$ is the maximum diameter of any of its simplices. A subdivision is *diameter-shrinking* if

$$\operatorname{diam} \operatorname{Div} \mathcal{K} < c \cdot \operatorname{diam} \mathcal{K}$$

for some constant $0 < c < 1$ (which may depend on $\mathcal{K}$).

Not every continuous map $f : |\mathcal{A}| \to |\mathcal{B}|$ has a simplicial approximation mapping $\mathcal{A}$ to $\mathcal{B}$. The following theorem, however, states we can always find a simplicial approximation defined over a sufficiently refined subdivision of $\mathcal{A}$.

Fact 4.3.1 (Simplicial Approximation Theorem). Let $\mathcal{A}$ and $\mathcal{B}$ be complexes, and Div a diameter-shrinking subdivision. Given a continuous map $f : |\mathcal{A}| \to |\mathcal{B}|$, there is an N such that f has a simplicial approximation $\phi : \operatorname{Div}^N \mathcal{A} \to \mathcal{B}$.

It is often convenient to use the following specific subdivision.

Definition 4.3.2. The *barycentric* subdivision of a simplex σ, written $\operatorname{Bary} \sigma$, is the complex whose vertices are indexed by faces of σ. A set of vertices $\sigma_0, \ldots, \sigma_k$ forms a simplex if $\sigma_0 \subset \cdots \subset \sigma_k$.

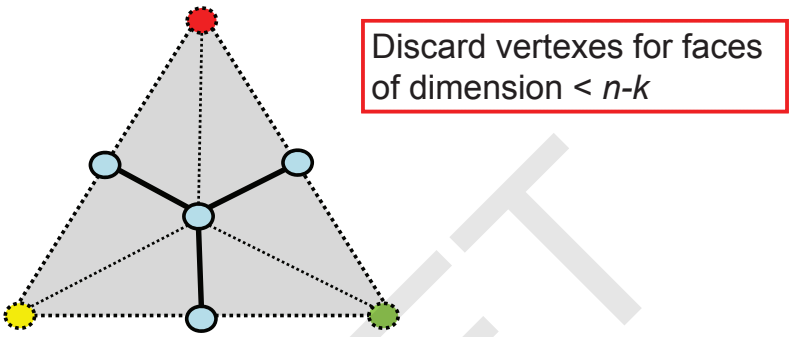

Figure 4.2: Truncated Barycentric Subdivision $\mathrm{Bary}_1 \sigma^2$

Sometimes it is useful to apply repeated subdivisions: $\mathrm{Bary}^N \mathcal{K}$ is the complex constructed by taking N repeated barycentric subdivisions. Figure 4.1 shows two complexes: $\mathrm{Bary}\, \sigma^2$ and $\mathrm{Bary}^2 \sigma^2$, where σ^2 is a complex consisting of a single 2-simplex. The vertex corresponding to $\tau \subseteq \sigma$ is usually placed at the barycenter (centroid) of τ.

maurice: redraw Fig figure:connect:truncated

Just as for consensus, it is convenient to recast k-set agreement in the following fixed-input form. The task is $(\sigma, \mathrm{skel}^{k-1}\sigma, \mathrm{skel}^{k-1}(\cdot))$, where the input complex consists of a single simplex $\sigma = \{\vec{s}_0, \ldots, \vec{s}_n\}$, where process P_i has $\vec{s}_i$ as its only possible input. Consider an execution of the task where exactly $m \leq n$ processes participate. If τ is the face of σ defined by their inputs, then each process halts with a vertex of τ, and together the processes choose at most $k-1$ distinct vertices. Collectively, the processes choose vertices on a simplex in $\mathrm{skel}^{k-1}(\tau)$. Using essentially the same reductions as for consensus, we can show this fixed-input formulation of k-set agreement is equivalent to the usual multi-input definition. Later, we will extend k-set agreement to an arbitrary colored input complex as the task

$(\mathcal{I}, \text{skel}^{k-1}(\mathcal{I}), \text{skel}^{k-1}(\cdot))$.

Definition 4.3.3. The *truncated* barycentric subdivision of a simplex σ, $\text{Bary}_k \sigma$, is the barycentric subdivision omitting vertices corresponding to faces of dimension less than $n-k$.

For example, $\text{Bary}_n \sigma$ is just $\text{Bary}\,\sigma$, while $\text{Bary}_0 \sigma$ is a single vertex, the *barycenter* of σ. (See Fig. 4.2.) Because every n-simplex of $\text{Bary}\,\sigma$ loses exactly $n-k$ vertices, this complex is a pure k-dimensional complex. We are interested in arbitrary subdivisions of this complex, written $\text{Div}\,\text{Bary}_k \sigma$. The *carrier* of a simplex τ in $\text{Div}\,\text{Bary}_k \sigma$ is the carrier it inherits from $\text{Div}\,\text{Bary}\,\sigma$.

maurice: Do we want to use "carrier" or "support"? Is carrier confusing with carier map?

When $k = n$, the following technical lemma is the same as Sperner's Lemma.

Lemma 4.3.4. If $\gamma :\ \text{Div}\ \ \text{Bary}_k \sigma \ \rightarrow\ \sigma$ is a simplicial map that sends each vertex to a vertex in its carrier, then γ sends some k-simplex onto a k-face of σ.

Proof. We "put back" the vertices discarded by $\text{Bary}_k \sigma$ to construct a new subdivision of σ. Take $\text{Bary}\,\sigma$, and apply the subdivision Div to the subcomplex $\text{Bary}_k \sigma \subseteq \text{Bary}\,\sigma$.

Every n-simplex τ of $\text{Bary}\,\sigma$ can be written as $\alpha \cup \beta$, where β is a k-simplex of $\text{Bary}_k \sigma$. The subdivision Div induces a subdivision of τ by subdividing β and taking the join with α: $\alpha \cdot \text{Div}\,\beta$. Define the subdivision $\text{Div}^*_k \sigma$ by applying this subdivision to each simplex of $\text{Bary}\,\sigma$. Note that $\text{Div}\,\text{Bary}_k \sigma \subset \text{Div}^*_k \sigma$, and every n-simplex in $\text{Div}^*_k \sigma$ can be expressed as $\alpha \cup \gamma$, where γ is an k-simplex in $\text{Div}\,\text{Bary}_k \sigma$.

Next, extend ϕ to $\phi^* :\ \text{Div}^*_k \sigma \ \rightarrow\ \sigma$ to send every vertex to a vertex in its carrier. Sperner's Lemma implies that ϕ^* sends some n-simplex τ onto σ. Because $\tau = \alpha \cup \gamma$, where γ is an k-simplex in $\text{Div}\,\text{Bary}_k \sigma$, ϕ sends a k-simplex γ of $\text{Div}\,\text{Bary}_k \sigma$ onto an k-face of σ. □

We can reformulate Theorem 4.1 as follows: a protocol cannot solve 1-set agreement (consensus) if the image of each n-simplex is 0-connected (path-connected), and each $(n-1)$-simplex is (-1)-connected (non-empty). This formulation suggest the following generalization: a protocol cannot solve k-set agreement if the image of each n-simplex is $(k-1)$-connected, each $(n-1)$-simplex is $(k-2)$-connected, and so on down to dimension $n-k$.

Theorem 4.3.5. Let $\mathcal{P}(\cdot)$ be an $(n+1)$-process protocol complex. If $\mathcal{P}(\sigma)$ is $(k-(n-\dim\sigma)-1)$-connected for every input simplex σ, then $\mathcal{P}(\cdot)$ cannot solve k-set agreement.

Proof. We exploit the connectivity of the protocol complex to construct a continuous map from the truncated barycentric subdivision to the protocol complex,

$$f : |\operatorname{Bary}_k \sigma| \to |\mathcal{P}(\sigma)|.$$

This map is *carrier-preserving*: for each simplex $\beta \in \operatorname{Bary}_k \sigma$, $f(\beta) \subseteq |\mathcal{P}(\operatorname{Car}(\beta, \sigma))|$. We construct this map inductively on the skeleton. Recall that each vertex $\vec{v}$ of $\operatorname{Bary}_k \sigma$ is indexed by a face $\sigma_{\vec{v}}$ of σ of dimension at least $n-k$. For each vertex $\vec{v}$ of $\operatorname{Bary}_k \sigma$, $\mathcal{P}(\sigma_{\vec{v}})$ is non-empty by hypothesis, so we can define $f_0(\vec{v})$ to be any vertex in $\mathcal{P}(\sigma_{\vec{v}})$.

maurice: too many subscripts? Maybe σ_i instead of $\sigma_{\vec{b}_i}$?

For the induction step, assume we have a carrier-preserving continuous map,

$$f_{\ell-1} : |\operatorname{skel}^{\ell-1} \operatorname{Bary}_k \sigma| \to |\mathcal{P}(\sigma)|.$$

Let $\beta = \left\{\vec{b}_0, \ldots, \vec{b}_\ell\right\}$ be an ℓ-simplex of $\operatorname{Bary}_k \sigma$ with carrier κ. We can reindex the vertices of β so that $\sigma_{\vec{b}_0} \subset \cdots \subset \sigma_{\vec{b}_\ell}$. Since $\dim \sigma_{\vec{b}_0} = n-k$, $\dim \sigma_{\vec{b}_\ell} \geq n-k+\ell$, so $\dim \kappa \geq n-k+\ell$. By hypothesis, $\mathcal{P}(\kappa)$ is $(\ell-1)$-connected, so we can extend $f_{\ell-1}$ on $\operatorname{skel}^{\ell-1} \beta$ to

$$f_\ell : |\beta| \to |\mathcal{P}(\kappa)|.$$

In the same way, we can extend $f_{\ell-1}$ over each ℓ-simplex of $\operatorname{Bary}_k \sigma$. Each of these extensions agree on the $(\ell-1)$-skeleton, so together they define a continuous map,

$$f_\ell : |\operatorname{skel}^{\ell} \operatorname{Bary}_k \sigma| \to |\mathcal{P}(\sigma)|,$$

sending the image of each simplex to its carrier. Note that $\operatorname{skel}^k \operatorname{Bary}_k \sigma = \operatorname{Bary}_k \sigma$, so the desired map f is just f_k.

The map f has a carrier-preserving simplicial approximation

$$\phi : \operatorname{Ch}^n \operatorname{Bary}_k \sigma \to \mathcal{P}(\sigma).$$

Composing ϕ with the decision map δ yields a map

$$\gamma : \operatorname{Ch}^n \operatorname{Bary}_k \sigma \xrightarrow{\phi} \mathcal{P}(\sigma) \xrightarrow{\delta} \sigma.$$

The map γ, the composition of ϕ and δ, is carrier-preserving, so by Lemma 4.3.4, it sends some k-simplex τ of $\operatorname{Ch}^n \operatorname{Bary}_k \sigma$ onto a k-face of σ. It follows that the decision map δ sends the simplex $\phi(\tau)$ onto a k-face of σ, implying that there is some execution in which $k+1$ processes choose $k+1$ distinct values, contradicting the specification for k-set agreement. □

4.4 Set Agreement and Read-Write memory

In this section, we use Theorem 4.3.5 to show that there is no $(n+1)$-process n-set agreement protocol for wait-free read-write memory. This result implies Theorem 4.2.6. Our arguments in this section are higher-dimensional analogs of the critical-state arguments used in Section 4.2.

To satisfy the conditions of Theorem 4.3.5, and to show that n-set agreement is impossible in wait-free read-write memory, it is enough to show that for any wait-free read-write protocol $\mathcal{P}(\cdot)$ and any input simplex σ, $\mathcal{P}(\sigma)$ is $(\dim \sigma)$-connected.

The Nerve Lemma

To compute the connectivity of a complex, we would like to break it down into simpler components, compute the connectivity of each of the components, and then "glue" those components back together in a way that permits us to to deduce the connectivity of the original complex from the connectivity of the components.

To this end, we use the *Nerve Lemma*, a basic theorem of combinatorial topology,

Definition 4.4.1. The *nerve* of a family of sets $\{A_0, \ldots, A_n\}$ captures how its members intersect. It is the simplicial complex whose vertex set is $\{0, \ldots, n\}$ with simplices given by:

$$\mathcal{N}(A_0, \ldots, A_n) = \left\{ \sigma \subseteq \{0, \ldots, n\} : \bigcap_{i \in \sigma} A_i \neq \emptyset \right\}.$$

For example, $\{0, 1, 2\}$ is a simplex in $\mathcal{N}(A_0, \ldots, A_n)$ if $A_0 \cap A_1 \cap A_2$ is non-empty.

Definition 4.4.2. Let I be a finite index set. A set of complexes $\{\mathcal{K}_i | i \in I\}$ is a *cover* for $\mathcal{K}$ if $\mathcal{K} = \cup_{i \in I} \mathcal{K}_i$.

Informally, the nerve of a covering describes how the elements of the covering "fit together" to form the original complex. Note that the nerve is determined by the covering, not just the complex.

Lemma 4.4.3 (Nerve Lemma). Let $\{\mathcal{K}_i | i \in I\}$ be a cover for a complex $\mathcal{K}$. For any index set $J \subset I$, define $\mathcal{K}_J = \cap_{j \in J} \mathcal{K}_j$. If each $\mathcal{K}_U$ is either $(k - |U| + 1)$-connected or empty, then $\mathcal{K}$ is k-connected if and only if $\mathcal{N}(\mathcal{K}_i | i \in I)$ is k-connected.

Reachable Complexes

To apply the Nerve Lemma to wait-free shared-memory computation, we need some additional concepts. If s is a protocol state, then for each process P_i, s_i denotes the state if P_i takes the next step from s. (If P_i has decided, define s_i to be s.) Let $\mathcal{P}(s)$ be the reachable complex from s, and $\mathcal{P}(s_i) = \mathcal{P}_i(s)$ the reachable complex from s_i. The $\mathcal{P}_i(s)$ form a cover for $\mathcal{P}(s)$.

For $U \subseteq \Pi$, define

$$\mathcal{P}_U(s) = \bigcap_{i \in U} \mathcal{P}_i(s).$$

By convention, $\mathcal{P}_\emptyset = \mathcal{P}(s)$.

For brevity, we often write $\mathcal{P}_i$ and $\mathcal{P}_U$ when the state s is clear from context. Informally, each simplex in $\mathcal{P}_U$ corresponds to an execution from s in which some process in U went first, but no process can tell which. For ease of exposition, when we say that U *contains* an operation, (or that the operation is *in* U) we mean that for some $i \in U$, P_i has that operation pending in s. We also say that an execution e is in $\mathcal{P}(s)$ if the simplex defined by the decision values of the processes in e is a simplex in $\mathcal{P}(s)$.

Here is a simple example. Let $n = 2$, and suppose s has three pending operations: P_0 and P_1 are about to read from $M[2]$, which currently holds value u, and P_2 is about to write value v to $M[2]$. Let s_0 be the state reached from s by letting P_0 take the first step, and so on.

First, let us focus on read operations alone. Let $U = \{0, 1\}$, so

$$\mathcal{P}_U = \mathcal{P}_0 \cap \mathcal{P}_1.$$

In every execution in $\mathcal{P}_0$, P_0 reads the old value u, and similarly for $\mathcal{P}_1$ and P_1, so in every execution in $\mathcal{P}_U$, both P_0 and P_1 read value u. There is no execution in $\mathcal{P}_U$ where P_0 or P_1 reads the new value v. Let s' be the state reached from s by letting P_0 and P_1 take the next steps (in either order). Any execution starting in s where it is ambiguous whether P_0 or P_1 went first is equivalent to an execution starting from s', i.e., $\mathcal{P}_U = \mathcal{P}(s')$.

Next, let us focus on the interaction between a read and a write. Let $V = \{0, 2\}$, so

$$\mathcal{P}_V = \mathcal{P}_0 \cap \mathcal{P}_2.$$

We have seen that in every execution in $\mathcal{P}_0$, P_0 reads the old value u. In every execution in $\mathcal{P}_2$, however, P_2 writes v before P_0 reads, so P_0 reads v. Of course, P_0 cannot read both values in a single execution. It follows that in every execution in $\mathcal{P}_V(s)$, P_0 takes no further steps after reading. Let

$\mathcal{P}_2^0(\cdot)$ be the protocol, starting from s_2, identical to $\mathcal{P}$ except that P_0 does not participate. We conclude that $\mathcal{P}_V(s) = \mathcal{P}_2^0(s)$.

In both cases, we were able to identify the intersection of certain reachable complexes from s with the (simpler) reachable complex for a related protocol, either with fewer pending operations, or fewer participating processes.

We are now ready to turn these examples into lemmas. A pending read (or write) operation in s by a process in U is *conflicting* if there is another pending write (or read) operation to the same memory entry. Otherwise, it is *non-conflicting*. Two operations *commute* if, whenever they are adjacent in an execution, reversing their order does not change any operation's results.

Lemma 4.4.4. If U contains a non-conflicting operation by P_i, then

$$\mathcal{P}_U = \mathcal{P}_{U\setminus\{i\}}(s_i).$$

Proof. Let x_i be P_i's pending operation in s. Because every execution e in $\mathcal{P}_U$ is equivalent to an execution in which x_i appears first, x_i must commute with every operation that precedes it in e

First, we show that $\mathcal{P}_U(s) \subseteq \mathcal{P}_{U\setminus\{i\}}(s_i)$. Let e be an execution in $\mathcal{P}_i(s) \cap \mathcal{P}_j(s)$, where P_j's pending operation is x_j. Because e is an execution in $\mathcal{P}_j(s)$, it is equivalent to an execution $x_j \cdot e'$. Because x_i commutes with its predecessors in $x_j \cdot e'$, this execution is equivalent to $x_j \cdot x_i \cdot e''$, which is equivalent to $x_j \cdot x_i \cdot e''$. It follows that e is in $\mathcal{P}_j(s_i)$.

$$\begin{aligned}
\mathcal{P}_i(s) \cap \mathcal{P}_j(s) &\subseteq \mathcal{P}_j(s_i) \\
\mathcal{P}_i(s) \cap \bigcap_{j\in U\setminus\{i\}} \mathcal{P}_j(s) &\subseteq \bigcap_{j\in U\setminus\{i\}} \mathcal{P}_j(s_i) \\
\mathcal{P}_U(s) &\subseteq \mathcal{P}_{U\setminus\{i\}}(s_i)
\end{aligned}$$

For the reverse inclusion, let e be an execution in $\mathcal{P}_j(s_i)$, for $j \neq i$. This execution is equivalent to an execution $x_j \cdot e'$. It is also equivalent to executions $x_i \cdot x_j \cdot e'$ and $x_j \cdot x_i \cdot e'$ starting from s. It follows that e'' is in $\mathcal{P}_i(s) \cap \mathcal{P}_j(s)$.

$$\begin{aligned}
\mathcal{P}_j(s_i) &\subseteq \mathcal{P}_i(s) \cap \mathcal{P}_j(s) \\
\bigcap_{j\in U\setminus\{i\}} \mathcal{P}_j(s_i) &\subseteq \mathcal{P}_i(s) \cap \bigcap_{j\in U\setminus\{i\}} \mathcal{P}_j(s) \\
\mathcal{P}_{U\setminus\{i\}}(s_i) &\subseteq \mathcal{P}_U(s)
\end{aligned}$$

□

A simple inductive argument on the number of operations yields:

Corollary 4.4.5. If U consists entirely of non-conflicting operations, then

$$\mathcal{P}_U = \mathcal{P}(s'),$$

where s' is a state reachable from s.

Let $\mathcal{P}^i(s)$ denote the reachable complex from state s through executions in which P_i takes no steps. Note that $\mathcal{P}^i(\cdot)$ is itself a wait-free read-write protocol complex for n (instead of $n+1$) processes.

Lemma 4.4.6. If r_i is a conflicted read in U, then

$$\mathcal{P}_U(s) = \mathcal{P}^i_{U\setminus\{i\}}(s).$$

Proof. First, we show that $\mathcal{P}_U(s) \subseteq \mathcal{P}^i_{U\setminus\{i\}}(s)$. Let e be an execution in $\mathcal{P}_i(s) \cap \mathcal{P}_j(s)$, where P_j's pending operation is x_j. Because e is an execution in $\mathcal{P}_j(s)$, it is equivalent to an execution $x_j \cdot e'$. Because w_j overwrites the value read by r_i, P_i takes no steps in e', so $\mathcal{P}_i(s) \cap \mathcal{P}_j(s) \subseteq \mathcal{P}^i_j(s)$. Because P_i takes no steps in $\mathcal{P}_i(s) \cap \mathcal{P}_j(s)$, it takes no steps in $\mathcal{P}_U(s)$.

$$\begin{aligned}
\mathcal{P}_i(s) \cap \mathcal{P}_j(s) &\subseteq \mathcal{P}^i_j(s) \\
\mathcal{P}_i(s) \cap \bigcap_{j\in U\setminus\{i\}} \mathcal{P}_j(s) &\subseteq \bigcap_{j\in U\setminus\{i\}} \mathcal{P}^i_j(s) \\
\mathcal{P}_U(s) &\subseteq \mathcal{P}^i_{U\setminus\{i\}}(s)
\end{aligned}$$

For the reverse inclusion, let e be an execution in $\mathcal{P}^i_j(s)$. This execution is equivalent to executions $w_j \cdot e'$ in $\mathcal{P}_j(s)$ and $r_i \cdot w_j \cdot e'$ in $\mathcal{P}_i(s)$, where P_i takes no steps in e'. (It is permissible for P_i to have different views in the two executions because it never decides, and these executions generate no vertices for P_i.)

$$\begin{aligned}
\mathcal{P}^i_j(s) &\subseteq \mathcal{P}_i(s) \cap \mathcal{P}_j(s) \\
\bigcap_{j\in U\setminus\{i\}} \mathcal{P}^i_j(s) &\subseteq \mathcal{P}_i(s) \cap \bigcap_{j\in U\setminus\{i\}} \mathcal{P}_j(s) \\
\mathcal{P}^i_{U\setminus\{i\}}(s) &\subseteq \mathcal{P}_U(s)
\end{aligned}$$

□

A simple inductive argument on the number of conflicted operations yields:

Corollary 4.4.7. If U includes conflicting operations, then

$$\mathcal{P}_U = \mathcal{P}'(s'),$$

where $\mathcal{P}'(\cdot)$ is a protocol for $m+1$ processes, where $n - |U| \leq m < n$.

Corollaries 4.4.5 and 4.4.7 imply that:

Corollary 4.4.8. In any state s and any $U \subseteq \Pi$, $\mathcal{P}_U(s)$ is non-empty.

The subcomplexes $\mathcal{P}_i(s)$ form a cover of $\mathcal{P}(s)$. By Corollary 4.4.8, every $\mathcal{P}_U(s)$ is non-empty, implying that this covering has a simple nerve:

Corollary 4.4.9. The nerve complex $\mathcal{N}(\mathcal{P}_0, \ldots, \mathcal{P}_n)$ is just the n-simplex Δ^n.

Knowing that the nerve complex of the covering has a simple structure does not by itself say anything about the connectivity of $\mathcal{P}(s)$. We will need to compute the connectivity of each $\mathcal{P}_U(s)$ before we can draw conclusions about the connectivity of $\mathcal{P}(s)$.

4.4.1 Critical States

Theorem 4.4.10. For every wait-free read-write protocol, $\mathcal{P}(\sigma)$ is $(\dim \sigma - 1)$-connected.

Proof. We will show a stronger property: for every state s reachable from an initial state σ, where $n = \dim \sigma$, $\mathcal{P}(s)$ is $(n-1)$-connected.

We argue by induction on $n = \dim \sigma$. For the base case, when $n = 0$, $\mathcal{P}(\sigma)$ is a single vertex, which is (-1)-connected (non-empty).

For the induction hypothesis, assume $\mathcal{P}(s)$ is $(m-1)$-connected for $(m+1)$-process protocols, where $0 \leq m < n$.

Being $(n-1)$-connected is an eventual property, so it has a critical state c such that $\mathcal{P}(c)$ is not $(n-1)$-connected, but $\mathcal{P}(s)$ is $(n-1)$-connected for every state reachable from c. In particular, each $\mathcal{P}_i(c)$ is $(n-1)$-connected, where the $\mathcal{P}_i(c)$ are a covering of $\mathcal{P}(c)$.

Now consider each $\mathcal{P}_U$. If U contains only non-conflicting operations, then by Corollary 4.4.5, it is equivalent to $\mathcal{P}(s)$, where s is a state reachable from c. Because c is critical for $(n-1)$-connectivity, $\mathcal{P}(s) = \mathcal{P}_U(c)$ is $(n-1)$-connected.

If U contains conflicting operations, then by Corollary 4.4.7, it is equivalent to $\mathcal{P}'(s')$, where $\mathcal{P}'(\cdot)$ is an $(m+1)$-process wait-free read-write protocol for $n - |U| \leq m < n$ processes. By the induction hypothesis for n, $\mathcal{P}'(s') = \mathcal{P}_U(c)$ is $(n - |U|)$-connected.

Either way, each $\mathcal{P}_U(c)$ is $(n - |U|)$-connected, so by the Nerve Lemma, $\mathcal{P}(s)$ is $(n-1)$-connected if and only if the Nerve $\mathcal{N}(\mathcal{P}_0(c), \ldots, \mathcal{P}_n(c))$ is

$(n-1)$-connected. By Corollary 4.4.9 $\mathcal{N}(\mathcal{P}_0(c),\ldots,\mathcal{P}_n(c))$ is just the n-simplex Δ^n, which is $(n-1)$-connected. It follows that $\mathcal{P}(c)$ is $(n-1)$-connected, contradicting the assumption that c is a critical state for $(n-1)$-connectivity. □

Theorem 4.4.11. There is no protocol for n-set agreement in wait-free read-write memory.

Proof. We have shown that for every protocol complex $\mathcal{P}(\cdot)$, and every input simplex σ, $\mathcal{P}(\sigma)$ is $(\dim\sigma-1)$-connected. The claim follows from Theorem 4.3.5, setting $k=n$. □

4.5 Chapter Notes

Michael Fischer, Nancy Lynch, and Michael Paterson [11] were the first to prove that consensus is impossible in a message-passing system where a single thread can halt. They introduced the critical state style of impossibility argument. M. Loui and H. Abu-Amara [22] and Herlihy [**?**] extended this result to shared memory. Biran, Moran, and Zaks [5] were the first to draw the connection between path-connectivity and consensus.

Chaudhuri [8] was the first to study the k-set agreement task. The connection between connectivity and k-set agreement appears in Chaudhuri, Herlihy, Lynch and Tuttle [9], Saks and Zaharoglou [26], Borowsky and Gafni [6], and Herlihy and Shavit [18].

4.6 Exercises

Exercise 4.1. Prove Lemma 4.2.4.

Exercise 4.2. Defend or refute the claim that "without loss of generality", it is enough to prove that k-set agreement is impossible when inputs are taken only from a set of size $k+1$.

Exercise 4.3. Use the Nerve lemma to prove that if $\mathcal{A}$ and $\mathcal{B}$ are n-connected, and $\mathcal{A}\cap\mathcal{B}$ is $(n-1)$-connected, then $\mathcal{A}\cup\mathcal{B}$ is n-connected.

Exercise 4.4. Let σ be an n-simplex. Recall that Sperner's Lemma states that any map $\phi:\ \mathrm{Div}\ \sigma\ \to\ \sigma$ that sends each vertex to a vertex in its carrier must send an *odd* number of n-simplices onto σ. The proof of Lemma 4.3.4 uses Sperner's Lemma to show that any simplicial $\gamma:\ \mathrm{Div}\ \mathrm{Bary}_k\,\sigma\ \to\ \sigma$ that sends each vertex to a vertex in its carrier must send *some* n-simplex

onto σ. Explain why the proof of Theorem 4.3.4 does *not* imply that γ sends an odd number of n-simplices onto σ.

Exercise 4.5. Extend the proof of Theorem 4.2.5 to a model in which processes share multi-writer variables. Hint: the case analysis must consider two pending writes to the same variable.

Bibliography

[1] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873–890, 1993.

[2] Hagit Attiya, Amotz Bar-Noy, Danny Dolev, David Peleg, and Rudiger Reischuk. Renaming in an Asynchronous Environment. *Journal of the ACM*, July 1990.

[3] Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *J. ACM*, 41:122–152, January 1994.

[4] Hagit Attiya and Jennifer Welch. *Distributed Computing Fundamentals, Simulations, and Advanced Topics Second Edition.*

[5] Ofer Biran, Shlomo Moran, and Shmuel Zaks. A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor. In *PODC '88: Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, pages 263–275, New York, NY, USA, 1988. ACM.

[6] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 91–100, New York, NY, USA, 1993. ACM.

[7] Elizabeth Borowsky and Eli Gafni. A Simple Algorithmically Reasoned Characterization of Wait-Free Computations (Extended Abstract). In *PODC '97: Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, pages 189–198, New York, NY, USA, 1997. ACM.

[8] S. Chaudhuri. Agreement Is Harder Than Consensus: Set Consensus Problems in totally asynchronous systems. In *Proceedings Of The Ninth Annual ACM Symosium On Principles of Distributed Computing*, pages 311–234, August 1990.

[9] Soma Chaudhuri, Maurice Herlihy, Nancy A. Lynch, and Mark R. Tuttle. A Tight Lower Bound for k-Set Agreement. In *In Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pages 206–215, 1993.

[10] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Andreas Tielmann. The Disagreement Power of an Adversary. In Idit Keidar, editor, *Distributed Computing*, volume 5805 of *Lecture Notes in Computer Science*, chapter 6, pages 8–21. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2009.

[11] M. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility Of Distributed Commit With One Faulty Process. *Journal of the ACM*, 32(2), April 1985.

[12] Eli Gafni and Elias Koutsoupias. Three-Processor Tasks Are Undecidable. *SIAM J. Comput.*, 28(3):970–983, 1999.

[13] Eli Gafni, Sergio Rajsbaum, and Maurice Herlihy. Subconsensus Tasks: Renaming Is Weaker Than Set Agreement. In *Distributed Computing, 20th International Symposium, Stockholm, Sweden, September 18-20, 2006, Proceedings(DISC)*, volume 4167 of *Lecture Notes in Computer Science*, pages 329–338. Springer, 2006.

[14] Maurice Herlihy and Sergio Rajsbaum. The decidability of distributed decision tasks (extended abstract). In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 589–598, New York, NY, USA, 1997. ACM.

[15] Maurice Herlihy and Sergio Rajsbaum. The topology of shared-memory adversaries. In *Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, PODC '10, pages 105–113, New York, NY, USA, 2010. ACM.

[16] Maurice Herlihy, Sergio Rajsbaum, and Mark Tuttle. An Axiomatic Approach to Computing the Connectivity of Synchronous and Asynchronous Systems. *Electron. Notes Theor. Comput. Sci.*, 230:79–102, 2009.

[17] Maurice Herlihy, Sergio Rajsbaum, and Mark R. Tuttle. Unifying synchronous and asynchronous message-passing models. In *PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 133–142, New York, NY, USA, 1998. ACM.

[18] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999.

[19] Flavio Junqueira and Keith Marzullo. A framework for the design of dependent-failure algorithms: Research Articles. *Concurr. Comput. : Pract. Exper.*, 19(17):2255–2269, 2007.

[20] Flavio P. Junqueira and Keith Marzullo. Designing Algorithms for Dependent Process Failures. Technical report, 2003.

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

[22] M. C. Loui and H. H. Abu-Amara. *Memory requirements for agreement among unreliable asynchronous processes*, volume 4, pages 163–183. JAI press, 1987.

[23] Yoram Moses and Sergio Rajsbaum. A Layered Analysis of Consensus. *SIAM J. Comput.*, 31:989–1021, April 2002.

[24] James Munkres. *Elements of Algebraic Topology*. Prentice Hall, 2 edition, January 1984.

[25] Michael Saks and Fotios Zaharoglou. Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449–1483, 2000.

[26] Michael Saks and Fotios Zaharoglou. Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000.

[27] Francis Sergeraert. The Computability Problem In Algebraic Topology. *Adv. Math*, 104:1–29, 1994.

[28] Edwin H. Spanier. *Algebraic topology*. Springer-Verlag, New York, 1981.

[29] John Stillwell. *Classical Topology and Combinatorial Group Theory*. Springer, 2nd edition, March 1993.