# Contents

1 Introduction ......................................................... 7
   1.1 Decision Tasks ............................................. 8
   1.2 Communication .............................................. 9
   1.3 Failures .................................................... 9
   1.4 Timing ...................................................... 10
      1.4.1 Processes and Protocols ............................ 10
   1.5 Chapter Notes .............................................. 12

2 Elements of Combinatorial Topology ............................... 13
   2.1 Basic Concepts ............................................. 13
   2.2 Distributed Computing ..................................... 15
   2.3 Chapter Notes .............................................. 16
   2.4 Exercises ................................................ 16

3 Manifolds, Impossibility, and Separation .......................... 17
   3.1 Manifold Complexes ....................................... 17
   3.2 Immediate Snapshots ...................................... 18
   3.3 Manifold Protocols ........................................ 23
   3.4 Set Agreement .............................................. 24
   3.5 Anonymous Protocols ...................................... 28
   3.6 Weak Symmetry-Breaking .................................. 29
   3.7 Anonymous Set Agreement versus Weak Symmetry Breaking 30
   3.8 Chapter Notes .............................................. 35
   3.9 Exercises ................................................ 35

4 Connectivity ....................................................... 37
   4.1 Consensus and Path-Connectivity .......................... 37
   4.2 Consensus in Asynchronous Read-Write Memory ........... 39
   4.3 Set Agreement and Connectivity in Higher Dimensions .... 43
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 vertexes of $\sigma$. Each process $P_i$ has $\vec{s}_i$ as its only possible input. Consider an execution where exactly $m \leq n$ processes participate. If $\tau$ is the face of $\sigma$ defined by their inputs, then all processes halt with the same vertex of $\sigma^m$. 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 \( \mathcal{C} \) be a complex.

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

**Definition 4.1.2.** A complex \( \mathcal{C} \) is path-connected if there is a path between every two vertexes in \( \mathcal{C} \).

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 \( \text{Face}_0 \sigma \) is the input simplex that excludes \( \vec{s}_0 \). The subcomplex \( \mathcal{P}(\text{Face}_0 \sigma) \) 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 \( \mathcal{P}(\text{Face}_0 \sigma) \). (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 \( \delta : \mathcal{P}(\sigma) \to \text{skel}^0 \sigma \) sends \( \vec{p} \) to a vertex \( \vec{s}_i \) of \( \sigma \), 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 \( \mathcal{P}(\text{Face}_1 \sigma) \), where \( P_i \) is the winning process in \( \vec{p} \). The decision map \( \delta : \mathcal{P}(\sigma) \to \text{skel}^0 \sigma \) sends \( \vec{q} \) to a vertex \( \vec{s}_j \) of \( \sigma \), 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 \( \delta \) 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 \( \sigma \), so \( \delta \) maps both \( \vec{p}_k \) and \( \vec{p}_{k+1} \) to the same vertex. By construction, the decision map \( \delta \) 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 \( i \neq j \). \( \square \)
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 encodes a possible initial system state, the protocol complex \(P(\sigma)\) encodes all possible protocol executions starting from \(\sigma\), and each facet of \(P(\sigma)\) 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 manoeuvre 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 \(\sigma\) be an input \(m\)-simplex, \(0 \leq m \leq n\), and let \(s\) be a global state reached by running \(P\) from the initial state given by \(\sigma\). A simplex \(\tau\) of \(P(\sigma)\) is reachable from \(s\) if there is an execution starting from \(s\) in which each process in \(\text{ids}(\tau)\) completes the protocol with the local state specified in \(\tau\). The reachable complex from \(s\), written \(P(s)\), is the complex generated by reachable simplexes from \(s\).

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

**Definition 4.2.1.** Formally, a property is a predicate on 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 \(\varphi\) holds in global state \(s\) if \(\varphi\) holds for \(P(s)\), the reachable complex from \(s\).

**Definition 4.2.2.** A global state \(s\) is critical for an eventual property \(\varphi\) if \(\varphi\) 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 \(\varphi\) 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 \( \varphi \) 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 \( \varphi \) holds, advancing non-critical processes in this way will eventually leave the protocol in a state where \( \varphi \) does not hold, but all processes are either decided or about to make \( \varphi \) 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 \( C \) be a complex that can be expressed as the union of components over some finite index set \( I \):

\[
C = \bigcup_{i \in I} C_i.
\]

Lemma 4.2.4. If each \( C_i \) is path-connected, and each pair-wise intersection \( C_i \cap C_j \) is non-empty, then \( C \) 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 \( \sigma \) is an input \( n \)-simplex, then \( P(\sigma) \) is path-connected. (This condition implies that \( P(\sigma) \) is non-empty if \( \dim \sigma = n - 1 \).)

Lemma 4.2.5. If \( \sigma \) is an input \( n \)-simplex, then \( P(\sigma) \) is path-connected.

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

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

\[
P(c) = \bigcup_{i \in I} P(c_i).
\]
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 \( \mathcal{P}(c_i) \cap \mathcal{P}(c_j) \).

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 \( \sigma \), \( \mathcal{P}(\sigma) \) 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 vertexes. Think of these two vertexes as the image, under a continuous map, of a 0-dimensional sphere (the points $\pm 1$ 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 $C$ is a path whose starting and end vertexes are the same. A loop can be considered a continuous map $f : S^1 \to |C|$, carrying the 1-sphere $S^1$ to the polyhedron of $C$. A complex is 1-connected (or simply-connected) if any such map can be extended to the 2-disk: $F : D^2 \to |C|$. In general, a complex is $k$-connected if any continuous map $f : S^k \to |C|$ can be extended to $F : D^{k+1} \to |C|$. 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.

Although we have defined vertexes, simplexes, and complexes as abstract sets, it is sometimes convenient to treat them as point sets in Euclidean space. Let $C$ be a complex. Each vertex $\vec{v}$ of $C$ corresponds to a point $|\vec{v}|$. For each simplex $\sigma = \{\vec{s}_0, \ldots, \vec{s}_{t}\}$, the vertexes correspond to affinely-independent point $\bar{\vec{v}} \in \{|\vec{s}_0|, \ldots, |\vec{s}_{t}|\}$, the geometric simplex $|\sigma|$ is their convex hull. The geometric complex $|C|$ corresponds to a set of geometric simplexes arranged so that that every two simplexes intersect either in a common

---

1 points $x_0, \ldots, x_n$ are affinely independent if $x_1 - x_0, \ldots, x_n - x_0$ are linearly independent.
face, or not at all. The point set $|C|$ occupied by a geometric complex $C$ 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 $|C|$ 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 vertexes of a $t$-simplex $\sigma^t$ of $C$, and for $0 \leq i \leq t$, $\sum_i t_i = 1$, $0 < t_i \leq 1$.

A geometric complex is subdivided by partitioning each of its simplexes into smaller simplexes without changing the complex’s polyhedron. More formally, a complex $B$ is a subdivision of $A$ if

- For each simplex $\beta$ of $B$, there is a simplex $\alpha$ of $A$ such that $|\beta| \subseteq |\alpha|$.
4.3. SET AGREEMENT AND CONNECTIVITY IN HIGHER DIMENSIONS

• For each simplex $\alpha$ of $\mathcal{A}$, $|\alpha|$ is the union of a finite set of polyhedrons of simplexes of $\mathcal{A}$.

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:

$$|\phi|(x) = \sum_i t_i \cdot \phi(s_i).$$

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,

$$\phi : \mathcal{A} \to \mathcal{B}$$
$$f : |\mathcal{A}| \to |\mathcal{B}|$$

where $\phi$ is a simplicial map of complexes and $f$ a continuous map of their polyhedrons. We say that $\phi$ is a simplicial approximation to $f$ if

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

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

The diameter $\text{diam} \sigma$ of a geometric simplex $-\sigma|$ is the length of its longest edge. The diameter $\text{diam} C$ of a geometric complex $||C||$ is the maximum diameter of any of its simplexes. A subdivision is diameter-shrinking if

$$\text{diam} \text{Div} C < c \cdot \text{diam} C$$

for some constant $0 < c < 1$ (which may depend on $C$).

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 $\mathcal{A}$.

**Theorem 4.3.1 (Simplicial Approximation).** Let $\mathcal{A}$ and $\mathcal{B}$ be complexes, and $\text{Div}$ 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 $\phi : \text{Div}^N \mathcal{A} \to \mathcal{B}$.

It is often convenient to use the following specific subdivision.

**Definition 4.3.2.** The barycentric subdivision of a simplex $\sigma$, written Bary $\sigma$, is the complex whose vertexes are indexed by faces of $\sigma$. A set of vertexes $\sigma_0, \ldots, \sigma_k$ forms a simplex if $\sigma_0 \subset \cdots \subset \sigma_k$. 
Sometimes it is useful to apply repeated subdivisions: Bary$^N$C is the complex constructed by taking $N$ repeated barycentric subdivisions. Figure 4.1 shows two complexes: Bary$^1$ and Bary$^2$, where $\sigma^2$ is a complex consisting of a single 2-simplex. The vertex corresponding to $\tau \subseteq \sigma$ is usually placed at the barycenter (centroid) of $\tau$.

Just as for consensus, it is convenient to recast $k$-set agreement in the following fixed-input form. The task is $(\sigma, \text{ske}l^{k-1}\sigma, \text{ske}l^{k-1}(.))$, where the input complex consists of a single simplex $\sigma = \{\bar{s}_0, \ldots, \bar{s}_n\}$, where process $P_i$ has $\bar{s}_i$ as its only possible input. Consider an execution of the task where exactly $m \leq n$ processes participate. If $\tau$ is the face of $\sigma$ defined by their inputs, then each process halts with a vertex of $\tau$, and together the processes choose at most $k - 1$ distinct vertexes. Collectively, the processes choose vertexes on a simplex in $\text{ske}l^{k-1}(\tau)$. 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
4.3. SET AGREEMENT AND CONNECTIVITY IN HIGHER DIMENSIONS

\((I, \text{skek}^{k-1}(I), \text{skek}^{k-1}(\cdot))\).

**Definition 4.3.3.** The **truncated** barycentric subdivision of a simplex \(\sigma\), \(\text{Bary}_k \sigma\), is the barycentric subdivision omitting vertexes 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 \(\sigma\). (See Fig. 4.2.) Because every \(n\)-simplex of \(\text{Bary} \sigma\) loses exactly \(n - k\) vertexes, 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 \(\tau\) in \(\text{Div} \text{Bary}_k \sigma\) is the carrier it inherits from \(\text{Div} \text{Bary} \sigma\).

When \(k = n\), the following technical lemma is the same as Sperner’s Lemma.

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

**Proof.** We “put back” the vertexes discarded by \(\text{Bary}_k \sigma\) to construct a new subdivision of \(\sigma\). Take \(\text{Bary} \sigma\), and apply the subdivision \(\text{Div}\) to the subcomplex \(\text{Bary}_k \sigma \subseteq \text{Bary} \sigma\).

Every \(n\)-simplex \(\tau\) of \(\text{Bary} \sigma\) can be written as \(\alpha \cup \beta\), where \(\beta\) is an \(k\)-simplex of \(\text{Bary}_k \sigma\). The subdivision \(\text{Div}\) induces a subdivision of \(\tau\) by subdividing \(\beta\) and taking the join with \(\alpha\): \(\alpha \cdot \text{Div} \beta\). 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 \(\gamma\) is an \(k\)-simplex in \(\text{Div} \text{Bary}_k \sigma\).

Next, extend \(\phi\) to \(\phi^* : \text{Div}^*_k \sigma \to \sigma\) to send every vertex to a vertex in its carrier. Sperner’s Lemma implies that \(\phi^*\) sends some \(n\)-simplex \(\tau\) onto \(\sigma\). Because \(\tau = \alpha \cup \gamma\), where \(\gamma\) is an \(k\)-simplex in \(\text{Div} \text{Bary}_k \sigma\), \(\phi\) sends a \(k\)-simplex \(\gamma\) of \(\text{Div} \text{Bary}_k \sigma\) onto an \(k\)-face of \(\sigma\). \(\square\)

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 \(n\)-simplex \(\sigma\), 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,

\[ f : \text{Bary}_k \sigma \to |\mathcal{P}(\sigma)|. \]

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

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

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

Let \( \beta = \{ \tilde{b}_0, \ldots, \tilde{b}_L \} \) be an \( \ell \)-simplex of \( \text{Bary}_k \sigma \) with carrier \( \kappa \). We can reindex the vertexes of \( \beta \) so that \( \sigma_{\tilde{b}_0} \subset \cdots \subset \sigma_{\tilde{b}_L} \). Since \( \dim \sigma_{\tilde{b}_0} = n - k \), \( \dim \sigma_{\tilde{b}_l} \geq n - k + \ell \), so \( \dim \kappa \geq n - k + \ell \). By hypothesis, \( \mathcal{P}(\kappa) \) is \((\ell - 1)\)-connected, so we can extend \( f_{\ell - 1} \) on \( \text{skel}^{\ell - 1} \beta \) to

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

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

\[ f_{\ell} : |\text{skel}^{\ell} \text{Bary}_k \sigma| \to |\mathcal{P}(\sigma)|, \]

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

The map \( f \) has a carrier-preserving simplicial approximation

\[ \phi : \text{Ch}^n \text{Bary}_k \sigma \to \mathcal{P}(\sigma). \]

Composing \( \phi \) with the decision map \( \delta \) yields a map

\[ \text{Ch}^n \text{Bary}_k \sigma \xrightarrow{\phi} \mathcal{P}(\sigma) \xrightarrow{\delta} \sigma. \]

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

\[ \square \]
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 \(P(\cdot)\) and any input simplex \(\sigma\), \(P(\sigma)\) 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 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 simplexes given by:

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

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

**Definition 4.4.2.** Complexes \(C_0, \ldots, C_q\) cover \(C\) if \(C = \bigcup_{i=0}^q C_i\).

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 \(C_0, \ldots, C_n\) cover a complex \(C\). For any index set \(U \subset [0..n]\), define \(C_U = \bigcap_{i \in U} C_i\). If each \(C_U\) is \((k - |U| + 1)\)-connected, then \(C\) is \(k\)-connected if and only if \(N(C_0, \ldots, C_n)\) 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 with 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

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

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

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
4.4. SET AGREEMENT AND READ-WRITE MEMORY

$P_0^0(\cdot)$ be the protocol, starting from $s_2$, identical to $P$ except that $P_0$ does not participate. We conclude that $P_V(s) = P_2^0(s)$.

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

$$P_U = P_U \setminus \{i\}(s_i).$$

Proof. Let $x_i$ be $P_i$’s pending operation in $s$. Because every execution $e$ in $P_U$ 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 $P_U(s) \subseteq P_U \setminus \{i\}(s_i)$. Let $e$ be an execution in $P_i(s) \cap P_j(s)$, where $P_j$’s pending operation is $x_j$. Because $e$ is an execution in $cP_j(s)$, 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 $P_j(s_i)$.

$$P_i(s) \cap P_j(s) \subseteq P_j(s_i)$$

$$P_i(s) \cap \bigcap_{j \in U \setminus \{i\}} P_j(s) \subseteq \bigcap_{j \in U \setminus \{i\}} P_j(s_i)$$

$$P_U(s) \subseteq P_U \setminus \{i\}(s_i)$$

For the reverse inclusion, let $e$ be an execution in $P_j(s_i)$, 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 $P_i(s) \cap P_j(s)$.

$$P_j(s_i) \subseteq P_i(s) \cap P_j(s)$$

$$\bigcap_{j \in U \setminus \{i\}} P_j(s_i) \subseteq P_i(s) \cap \bigcap_{j \in U \setminus \{i\}} P_j(s)$$

$$P_U \setminus \{i\}(s_i) \subseteq P_U(s)$$
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 $\mathcal{P}_i(s)$ 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

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

**Proof.** First, we show that $\mathcal{P}_U(s) \subseteq \mathcal{P}_{U\setminus\{i\}}(s)$. 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 $cP_j(s)$, 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 $\mathcal{P}_i(s) \cap \mathcal{P}_j(s) \subseteq \mathcal{P}_j(s)$. Because $P_i$ takes no steps in $\mathcal{P}_i(s) \cap \mathcal{P}_j(s)$, it takes no steps in $\mathcal{P}_U(s)$.

$$\mathcal{P}_i(s) \cap \mathcal{P}_j(s) \subseteq \mathcal{P}_j(s)$$

$$\mathcal{P}_i(s) \cap \bigcap_{j \in U\setminus\{i\}} \mathcal{P}_j(s) \subseteq \bigcap_{j \in U\setminus\{i\}} \mathcal{P}_j(s)$$

$$\mathcal{P}_U(s) \subseteq \mathcal{P}_{U\setminus\{i\}}(s)$$

For the reverse inclusion, let $e$ be an execution in $\mathcal{P}_j(s)$. 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 not vertexes for $P_i$.)

$$\mathcal{P}_j(s) \subseteq \mathcal{P}_i(s) \cap \mathcal{P}_j(s)$$

$$\bigcap_{j \in U\setminus\{i\}} \mathcal{P}_j(s) \subseteq \mathcal{P}_i(s) \cap \bigcap_{j \in U\setminus\{i\}} \mathcal{P}_j(s)$$

$$\mathcal{P}_{U\setminus\{i\}}(s) \subseteq \mathcal{P}_U(s)$$

A simple inductive argument on the number of conflicted operations yields:
Corollary 4.4.7. If $U$ includes conflicting operations, then

$$P_U = P'(s'),$$

where $P(\cdot)$ 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 $U \subseteq \Pi$, $P_U(s)$ is non-empty.

The subcomplexes $P_i(s)$ form a cover of $P(s)$. By Corollary connect non-empty, every $P_U(s)$ is non-empty, implying that this covering has a simple nerve:

Corollary 4.4.9. The nerve complex $\mathcal{N}(P_0, \ldots, P_n)$ is just the $n$-simplex $\Delta^n$.

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

4.4.1 Critical States

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

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

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

For the induction hypothesis, assume $P(s)$ 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 $P(c)$ is not $(n - 1)$-connected, but $P(s)$ is $(n - 1)$-connected for every state reachable from $c$. In particular, each $P_i(c)$ is $(n - 1)$-connected, where the $P_i(c)$ are a covering of $P(c)$.

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

If $U$ contains conflicting operations, then by Corollary [4.4.7] it is equivalent to $P'(s')$, where $P'(\cdot)$ is an $(m + 1)$-process wait-free read-write protocol for $n - |U| \leq m < n$ processes. By the induction hypothesis for $n$, $P'(s') = P_U(c)$ 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 $\Delta^n$, 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 $\sigma$, $\mathcal{P}(\sigma)$ 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 [7] 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 [?] and Herlihy [?] extended this result to shared memory. Biran, Moran, and Zaks [2] were the first to draw the connection between path-connectivity and consensus.

Chaudhuri [5] 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 [6], Saks and Zaharoglou [18], Borowsky and Gafni [4], and Herlihy and Shavit [12].

### 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 $\sigma$ be an $n$-simplex. Recall that Sperner’s Lemma states that any map $\phi : \text{Div } \sigma \to \sigma$ that sends each vertex to a vertex in its carrier must send an *odd* number of $n$-simplexes onto $\sigma$. The proof of Lemma 4.3.4 uses Sperner’s Lemma to show that any simplicial $\gamma : \text{Div } \text{Bary}_k \sigma \to \sigma$
that sends each vertex to a vertex in its carrier must send some $n$-simplex onto $\sigma$. Explain why the proof of Theorem 4.3.4 does not imply that $\gamma$ sends an odd number of $n$-simplexes onto $\sigma$.

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


