New Perspectives in Distributed Computing

Maurice Herlihy[1] and Sergio Rajsbaum[2]

[1] Computer Science Department, Brown University, Providence RI 02912,
herlihy@cs.brown.edu ***
[2] Instituto de Matemáticas, U.N.A.M., Ciudad Universitaria, D.F. 04510, México,
rajsbaum@math.unam.mx †

Abstract. This is an informal introduction to recent developments in the theory of distributed computing, showing how notions from combinatorial and algebraic topology can be used to capture essential aspects of distributed computing.

1 Introduction

This paper gives an informal description of some recent developments in the theory of distributed computing. These developments came about through the realization that techniques borrowed from combinatorial and algebraic topology could be used to capture essential aspects of distributed computing. We take a historical approach, tracing how these developments emerged from earlier work. In a sense, topological notions were present from the start, but they were not recognized as such, because early work relied on the most elementary of topological properties: connectivity. Only later was it realized that more advanced topological notions could be used to attack harder problems.

This paper is not intended to be a comprehensive survey, and it omits many important results. Instead, our goal is to provide an intuitively appealing introduction to the new approach, and to make it more accessible to non-specialists. We will also describe how the topological approach has motivated new models and techniques as well as providing a deeper understanding of certain classical results.

Perhaps the most important goal of the Theory of Distributed Computing is to understand which tasks can be solved by a distributed system, and at what cost. The answer depends on the task itself, as well as the specific assumptions made about the system, such as the type and number of faults, the degree of asynchrony, and the communication mechanisms (such as message passing, read/write memory, or other shared memory synchronization operations). This paper is organized as follows. In the next section, we describe two of the best-studied distributed computing models. In Section 3, we give an overview of the classical connectivity-based results. These results were the precursors of the modern topological approach, which is described in Section 4.

*** Supported by NSF grant DMS-9505949.
† Supported by CONACyT and DGAPA-UNAM grants.

2 Models of Distributed Computing

A distributed system consists of $n + 1$ processes that communicate with one another to solve a task. In this paper, we consider *decision* tasks, in which each process starts with a private *input value*, the processes communicate among themselves, and then each process chooses an *output value* and halts.

The first model we consider is the *synchronous message-passing* model. In this model, computation proceeds in a sequence of rounds. In each round, a process sends messages to the other processes, receives the messages sent to it by the other processes in that round, and changes state. All processes take steps at exactly the same rate, and all messages are delivered with exactly the same time. Historically, this model was the first to be studied, and many of the classical results were derived in this context. Although this model is now widely considered to be unrealistic, it is still worth studying because lower bounds for this model extend to more realistic models, and the simplicity of the model severs to illuminate a number of basic principles of distributed computing.

We also consider the *asynchronous shared-memory* model. In this model, there is no bound on the amount of time that can elapse between process steps, and processes communicate by reading and writing variables in a shared memory. While this model is more realistic than the synchronous message-passing model, it can still be criticized as an imperfect reflection of current practice. Modern multiprocessors typically provide synchronization operations, such as *test-and-set* and *load-locked/store-conditional*, that are more powerful than simple read and write operations. Nevertheless, this model also rewards study, as many of the modern topological results are best illustrated in this context. We will discuss below the effects of appending more powerful (and more realistic) primitives to this model.

The *state* of a system consists of a *local state* for each process, a *global state* consisting of each of the local states augmented by an *environment* that captures other relevant information, such as the state of the shared memory, or messages in transit. or the messages in transit. An *event* occurs when a process executes some significant action, such as sending or receiving a message, or reading or writing a shared variable. An *execution* is an alternating sequence of states and events.

A *crash failure* occurs when a process halts without warning. An important parameter of a model of computation, typically denoted by f, is the maximal number of processes that can crash. A *protocol* is a program that solves a task. We are interested in protocols that tolerate f or fewer crash failures; such protocols are called *f-resilient*. When f is n out of $n + 1$ processes, the protocol is called *wait-free*, because any process can finish without waiting for any other process.

3 Connectivity

The *consensus* task is perhaps the best-studied problem in theoretical distributed computing (e.g. [17, 18, 20, 27, 42]). In the simplest form of this problem, each process starts with a private binary input (either 0 or 1). The processes decide on a binary output satisfying the following conditions:

- *Termination*: Every non-faulty process eventually chooses a value,
- *Agreement*: All non-faulty processes decide on the same value, and
- *Validity*: The value chosen is some process's input value.

The validity condition implies that if all initial values are the same, then every non-faulty process decides that value.

3.1 Impossibility and Indistinguishability

The distributed computing literature encompasses a dazzling variety of results characterizing the circumstances under which consensus is or is not solvable, and lower bounds on the complexity of solutions when they exist. At the heart of nearly all such impossibility arguments is the notion of the *indistinguishability* of distinct global states to a process. Briefly put, two global states x and y are *indistinguishable* to process p if p has the same local state in both, denoted

$$x \sim_p y.$$

For example, consider global states x and y in the synchronous message-passing model, where two distinct executions lead to those states. If p has the same initial state in both executions, and receives the same sequence of messages, then x and y are indistinguishable to p ($x \sim_p y$). A *similarity chain* is a sequence of states such that any two consecutive states are indistinguishable to some process.

Similarity chains are central to the classical analysis of asynchronous consensus. If $x \sim_p y$, where p is a non-faulty process and x and y are final global states of a consensus protocol, then p must decide the same value in both x and y (because it cannot distinguish between them). The agreement condition on consensus implies that all non-faulty processes decide the same value in both states. A simple inductive argument shows that if x and y are related by a similarity chain, then all non-faulty processes must decide the same value in both states.

Fischer, Lynch and Paterson [20] proved in 1985 that there is no consensus protocol in an asynchronous message-passing system where even one process can fail (i.e., $f = 1$). We follow established custom by referring to this result as the *FLP* proof. (Later, Dwork et al. [14] and Loui and Abu-Amara [37] extended this result to asynchronous read/write memory.)

The fundamental idea underlying the FLP proof is the following. The agreement condition ensures that if any process decides a value v, they all do, so we can speak of an execution deciding value v. Assume by way of contradiction that we have a consensus protocol. A global state is 0-*valent* if every execution that passes through that state decides 0, and similarly for 1. A global state is *univalent* if every execution passing through it decides the same value, and it is *bivalent* otherwise. Clearly, no protocol can terminate in a bivalent state.

The first step is to show that any consensus protocol must have an initial bivalent state. Let x_0 (and x_1) be the initial state in which all processes have input 0 (and 1). Clearly, x_0 is 0-valent, and x_1 is 1-valent. Now suppose by way of contradiction that all initial global states are are univalent, Let y_i be the initial state in which processes $0, \ldots, n-i$ start with input 0, and the rest start with input 1. By construction, $y_0 = x_0$ and $y_{n+1} = x_1$. We now argue by induction. As noted above, y_0 is 0-valent. Assume, as induction hypothesis, that y_i is 0-valent. Let z_i be a global state reached by starting in global state y_i, crashing process i before it takes any steps, and running the protocol to completion by execution e. Let z_{i+1} be a global state reached by starting in global state y_{i+1}, crashing process i before it takes any steps, and running the protocol to completion by the same execution e. Since the only process that can distinguish the initial states y_i and y_{i+1} crashes before sending any messages (or writing to shared memory), the global states z_i and z_{i+1} must be indistinguishable to every non-faulty process, so all such processes must decide the same value in both. It follows that all processes must decide the same value in $y_0 = x_0$ and $y_n = x_1$, a clear contradiction.

The second step in the FLP proof is to show that any bivalent state x can always be extended to another bivalent state, implying that the protocol can be made to run forever, a clear violation of the termination condition. Start the protocol in a bivalent initial state, and run it for as long as possible in a bivalent state. Say that an operation O is *pending* in global state x is some process is about to execute O. For brevity, we restrict our attention to shared-memory reads and writes (the analysis for message send and receive is essentially the same). If x is bivalent, then as long as some pending operation carries the protocol to a bivalent state, execute that operation. Because the protocol must eventually terminate, every execution must eventually reach a bivalent state where every pending operation carries it to a univalent state. Because x is still bivalent, some pending operation O_0 carries the protocol to a 0-valent state, and another pending operation O_1 to a 1-valent state.

The rest is a case analysis. For example, if O_0 and O_1 are both reads, then the 0-valent state reached by executing O_0 followed by O_1 is indistinguishable from the 1-valent state reached by executing the operations in the reverse order. The other cases are left as an exercise for the reader (hint: you can fail at most one process).

This elegant argument, due to FLP, may appear to need no further elaboration. Nevertheless, it is instructive to recast this proof in the following geometric way. Each global state is a vertex in a graph G. Two global states are linked by an edge if they are indistinguishable to some process. If every initial state is univalent, then we can color each vertex with its eventual decision value The global state x_0 in which all processes have input zero is colored with zero, and the corresponding global state x_1 is colored with one. The FLP proof shows that there is a path in G linking x_0 and x_1.

Lemma 1 (1-dimensional Sperner). *Consider a graph in which each vertex is colored with a binary color. If the graph encompasses a path from vertex x_0 to vertex x_1, and x_1 and x_0 are colored with different colors, then two adjacent vertexes in the path are colored with different colors.*

The proof is a simple counting argument, showing that an odd number of edges in the path have two colors. This lemma implies that if all initial states are univalent, then there are two initial global states, y_i and y_{i+1}, differing only in the input to some process p, such that y_i is 0-valent, and y_{i+1} is 1-valent. As we have seen, this observation leads to a contradiction.

The second part of the FLP proof can also be recast in in graph-theoretic terms. Each global state (not just the initial states) is a vertex in a graph. Two vertexes (global states) are linked by an edge if they are indistinguishable to some process. As argued above, if vertex x is univalent (or bivalent), then so is every vertex connected to x by a path in the graph. The FLP proof (reinterpreted) argues that if vertex x is bivalent, but some pending operation leaves system in 0-valent global state x_0, while another leaves the system in 1-valent global state x_1, then x_0 and x_1 are connected by a path in the graph, implying that they have the same valence (or none), a contradiction.

3.2 Consensus in synchronous systems

In a synchronous message passing system, consensus can be solved in $f+1$ rounds. In 1982 Dolev and Strong [16] proved that it is impossible to solve consensus in fewer than $f+1$ rounds. The argument also uses indistinguishability with similarity chains, but in a different way. The proof assumes by way of contradiction that some protocol finishes in f rounds, and finds a similarity chain from a failure-free execution where all processes start with 0, to a failure-free execution where all processes start with 1. We illustrate this construction with an example in which $n \geq 2$ and $f = 1$. Consider a one-round execution e in which all processes start with input 0. Now consider the execution e' identical to e except that p_0 fails to send a message to p_1. If e leaves the protocol in global state x, and e' in global state x', then x and x' are indistinguishable to p_2. Continuing in this way, we remove one by one, each of the messages sent by p_0, constructing a similarity chain from the failure-free execution to the execution in which p_0 fails cleanly (without sending any messages). Now consider the execution in which p_0 fails cleanly with input 1. One-by-one, replace each of the messages sent by p_0, until we reach an execution e_1 in which p_0 starts with input 1, the rest start with input 0, and no failures occur. All processes decide 0 at the end of e, and e is linked to e_1 by a similarity chain, so all processes must decide 0 at the end of e_1. Continuing this construction, however, we can replace each 0 input with a 1,ending up with a failure-free execution e_n in which all processes start with 1. Since e and e_n are linked by a similarity chain, the processes must decide the same way in both executions, a contradiction.

A slightly more complicated construction is needed for the multi-round case, but the basic idea is to establish *connectivity* between the failure-free execution in which all processes start with 0, and the one in which they start with 1.

3.3 Approximate Agreement

Before turning into the more general question of characterizing the tasks that are solvable in the presence of a single failure, let us consider a relaxed form of consensus, called ϵ-*approximate agreement* (e.g. [15]). In this task, the inputs to the processes are real values. Processors must choose real decision values within ϵ of each other, and each process's decision must lie within the range of the initial inputs.

A simple and elegant algorithm for approximate agreement can be described in the shared memory model using *atomic snapshots.* An atomic snapshot object is an array of registers, in which each process is assigned an array element. A process can write its own element, and atomically read the entire array. Atomic snapshot objects can be implemented using read/write variables in the presence of any number of failures [5]. In the approximate agreement algorithm, a process repeatedly writes its current proposed decision value (initially its input), and takes a snapshot. For the next iteration, its proposed value is the average of the proposed values it read in the last snapshot. Different atomic snapshot objects are used for each iteration. The number of iterations needed depends on ϵ. It is easy to check that in each iteration the range of values held by the processes is divided by 2.

The Number of Different Values. This algorithm works with any number of failures. If f is known, however, then a process can take repeated snapshots until it observes at least $n-f+1$ proposed values, ensuring that at most $f+1$ distinct decision values are chosen in any execution. Thus, the first $n-f$ processes to finish an iteration will see each other proposed values, and will compute the same value for the next iteration. The remaining f processes may see additional estimates, and hence may compute different values. The total number of values will thus be at most $f+1$.

An interesting way of interpreting the approximate agreement algorithm for $f = 1$ is the following. Consider a graph which is a simple path. Label the endpoints 0 and 1, and the other vertices, moving from the 0-vertex to the 1-vertex, with evenly distributed, increasing values between 0 and 1. The processes start with inputs either 0 or 1. The processes repeatedly write their proposed values, which are vertices of the graph, and take snapshots. In each iteration, if a process saw two different vertices, it proposes to the next round a vertex in the middle of the sub-path joining the two vertices. If it saw only one vertex (its own), it stays with the same value. Under this perspective, ϵ-approximate agreement is just agreement on a sufficiently long path. Each process ends up deciding on a vertex, such that the decided vertices are either the same or are joined by an edge.

Notice that by the FLP impossibility result, any 1-resilient algorithm most have at least one execution where at least two different values are decided, and hence, in this sense, the above approximate agreement algorithm is optimal.

3.4 Solvability of tasks with one failure

Starting in 1987, in a series of papers, Moran and Wolfstahl [40] and Biran, Moran and Zaks [11] embarked in a general study of solvability of tasks in asynchronous message-passing systems with a single crash failure. This work generalized the FLP result from consensus to arbitrary tasks, and gave additional insights into the role of connectivity. In [40] a necessary condition for a task to be 1-solvable was given. The corresponding sufficient condition appeared in [11]. This characterization is based on connectivity, again using similarity chains. As a consequence, the problem of deciding when a decision task has a 1-resilient solution in the asynchronous message-passing model is computable. Using simulations [3] between shared memory and message passing, or using the direct approaches of Section 3.6, we get similar results for shared memory.

Formally, a *decision task* consists of a set of *input vectors,* $\mathcal{I}$ a set of *output vectors,* $\mathcal{O}$, and an input/output *task relation,* Δ. An input vector specifies an initial value for each process; an output vector specifies its output value. The relation Δ associates to each input vector $x \in \mathcal{I}$, the set $\Delta(x)$ of allowable output vectors. Now, we say that a set of vectors is *connected* if it is possible to get from any vector to any other vector by a sequence of vectors, such that each two consecutive vectors differ in exactly one component.

The characterization theorem states that the task is solvable with one failure if and only if there exists a restriction Δ' of Δ such that for every connected $X \subseteq \mathcal{I}$, $\Delta'(X)$ is connected. The introduction of the restriction Δ' is due to the fact that the decisions taken by a protocol have to span only a subset of the decision vectors allowed by the task specification Δ.

The idea of the necessity in [40] is by reduction to consensus. It is shown that if there is a protocol P that solves a disconnected task, then it can be used to produce another protocol P' that solves consensus, which is impossible, by the FLP result. The protocol P' decides 0 if P ends up in one connected component, and decides 1 if P ends up in a different component.

To show sufficiency, Biran et al. [11] use a form of approximate agreement protocol, with $f = 1$. As noted above, this protocol has the property that the processes decide on at most two different values. Given a connected task, a process first writes its input to a shared memory ([11] is described using message passing, but the ideas are similar), and then takes snapshots until it sees at all but one process's input values. Now, the process chooses an output vector as input for the approximate agreement task. If it saw all process's inputs, that is, a full input vector $x \in \mathcal{I}$, then it chooses a default output vector $y \in \Delta(x)$ as input for the approximate agreement. Otherwise, it chooses an output vector that is allowable no matter what the remaining input value is; that is, a default output vector in the intersection of $\Delta(x)$, for every x that extends the n-vector it knows. Finally, the processes execute approximate agreement on the path (as in Section 3.3) that joins the two proposed output vectors.

3.5 New Perspectives with a Single Failure

Following the new topological perspective [34], the single-failure ($f = 1$) characterization can be described without a reduction to consensus. Consider the set of *protocol vectors,* $\mathcal{P}$, one for each final state of the protocol in some execution. Each component of a protocol vector consists of a final local state of some process. We can view $\mathcal{P}$ as a relation that gives the set of protocol vectors, $\mathcal{P}(x)$, for each input vector x.

The main theorem for the characterization consists of showing that *any* protocol that tolerates one failure has a connected set of protocol vectors, $\mathcal{P}(x)$, for every $x \in \mathcal{I}$. There are several ways of proving such a theorem. Herlihy and Shavit [34] use a critical state argument, similar to FLP. Another approach is to consider a well structured subset of the executions [6, 8, 44], and on these prove the connectivity property. These proofs are most naturally expressed for the wait-free case. A reduction to $f = 1$ is done using a simulation [9]. Another technique is to use a combination of the structured subset of executions, with the bivalency argument [41], as described in next section.

By working with protocol vectors, one learns an important lesson:

> *The protocol vectors preserve the connectivity structure of the input vectors.*

This means, roughly, that the connectivity graph of the input vectors and the one of the protocol vectors are *homeomorphic.* Here we clearly encounter topological notions for the first time. Two geometric objects are homeomorphic if it is possible to deform one into the other by continuous transformations (e.g., stretching and bending, but not tearing). Speaking informally, our protocol graph is a stretched version of the input graph; a path in the input graph becomes a longer path in the protocol graph, but it is still connected.

In more detail, the decisions taken by the processes induce a *decision map* δ from $\mathcal{P}$ to $\mathcal{O}$, satisfying $\delta(\mathcal{P}(x)) \subseteq \Delta(x)$ for every $x \in \mathcal{I}$. Moreover, δ is "continuous" in the following sense: it maps vertexes to vertexes, and also edges to edges. Therefore, it sends each connected subgraph into a connected subgraph. The FLP impossibility result can now be clearly understood: the input graph is connected, and therefore so is the protocol graph. Thus, also the image of the protocol graph under δ is connected. However, consensus requires sending the protocol vectors to two disconnected output vectors, the all 0 vector and the all 1 vector. This is impossible, and hence consensus is unsolvable.

Remarkably, this is just what Lemma 1 says. Consider a graph which is a simple path, P, and a graph C which consist of two isolated vertices, denoted $0, 1$. The coloring of vertices is described by a function f from vertices of P to vertices of C. We view P and C as geometric objects, and f as instructions on how to "bend" P into C. If it is required that f sends the endpoints of P to different vertices of C, the Lemma says that at least one edge of P will have to "jump" from one vertex of C to the other. As an exercise, it is easy to verify that all these arguments still hold if C consists of two disconnected graphs, called 0 and 1.

3.6 Unifying Consensus-Style Results

We mentioned at the end of Section 2 that one of the new insights gained by the topological approach is to better understand the relationship between different models of distributed computation, in particular in three papers [21, 33, 41]. We describe here the one-dimensional approach of [41]. Moses and Rajsbaum [41] presented a unified framework to study solvability questions with 1 failure in asynchronous message passing and shared memory systems, and the relations to synchronous systems, simplifying and unifying previous results. In particular, they show that task solvability in all these situations depends only on connectivity, and hence is decidable. Inspired by the more recent perspectives, the work is based on the notions of

- sub-models— considering only a well-structured subset of the executions,
- round-by-round analysis— the structure of the sub-model is synchronous, and can be seen as consisting of rounds,
- connectivity vs. bivalency notions— using FLP arguments in a synchronous setting.

Initially, the present an abstract model of computation, which is later instantiated to various classic models of message-passing and shared memory. The model is based on the notion of a *system* which is simply a set of *runs*, i.e., sequences of states with a specification of which process are failed in each run. As in Section 2, an environment encompasses the state of the communication mechanisms. The general argument is: (i) if a set of states is "connected," and (ii), it contains a 0-valent and a 1-valent state, then it must contain a bivalent state.

Besides the usual connectivity notion in terms of similarity chains, another notion based on "potential" value of a state is used, which considers possible future decision values from that state. This notion is useful in unifying the treatment for the different concrete models of computation. The previous general argument can be used to get an abstract impossibility proof for consensus. Show (as in Section 3) that there is an initial bivalent state. Then, that the set of successors of every state is connected. Therefore, by induction, construct a run that consists of bivalent states. Finally, show that consensus is not solved in this run, because a consensus algorithm cannot terminate while in a bivalent state. There are various technical details that need to be taken care of, but this is the general idea. For example, this immediately yields a very simple bivalency proof of the $f+1$ synchronous lower described above.

Now we can revisit the FLP bivalency argument. Recalling the protocol vectors framework of the previous section, and in particular property 3.5, we have that an asynchronous consensus algorithm cannot terminate in a bivalent state x because the states in the future of x are connected and but have to be sent to something disconnected: an all 0's and an all 1's vectors. However, notice that this argument fails in models for which consensus can be solved, because a bivalent state may have successor states that are not connected.

To facilitate impossibility proofs for concrete models, and to expose the similarities between them, the notion of *layering* is introduced. In particular, layering avoids the need to ensure that at most one process crashes in any bivalent run. Instead of trying to prove connectivity properties of all executions in a model, only a well-structured subset of the executions of the model is considered, which include a very small degree of asynchrony. Given a state x of an algorithm, consider a set $S(x)$ of, not necessary immediate, successor states. These are thought of as "the next layer" of computation. For instance, in the asynchronous shared memory model, layering facilitates the fairness arguments to guarantee at most one failure. One such layering is called *permutation layering,* and is obtained by having either $n-1$ or all n processes execute their pending operations, in a linear ordering. Thus, in this layering, $S(x)$ consists of all states obtained by performing at least $n-1$ pending operations in all possible linear orderings. The two properties needed of a layering to give the consensus impossibility are very easy to show. First, that $S(x)$ is connected, and second, that the sub-model defined by these runs still allows at most one process to crash.

This ideas are used to obtain the necessity part of characterization theorems for solvability in the asynchronous models, and in the f-resilient synchronous model; sufficiency is obtained using approximate agreement.

4 Higher Degrees of Connectivity

When we go from 1-resilient computing, to f-resilient, for $f > 1$, we must replace the one-dimensional notion of connectivity with higher-dimensional notions. In topology, these notions arise in the study of geometric objects which can be continuously deformed. That is, for a topologist, a disk and a triangle are essentially the same object, i.e. *homeomorphic,* because one can be continuously deformed into the other without tearing it apart.

In particular, the size of an object is not interesting because it can always be shrunk or expanded to the desired size. On the other hand, an object which consists of two separate pieces is never homeomorphic to one which is connected. A sphere and a torus cannot be homeomorphic, because one has holes, and the other does not. Holes can be of different dimensions, as we shall see. Whether two objects are homeomorphic depends on the dimension and structure of their holes.

The ability of a system to solve tasks depends on the topological structure of the runs of the system, and if it is "compatible" with the topological structure of the task itself. As we have seen, in the case of $f = 1$, "compatible" means that a system can solve a task if and only if both have the same connectivity structure. For $f > 1$, the compatibility has to be at connectivity in higher dimensions. As we shall see, the notions described in the previous section are nicely generalized for the case of $f > 1$.

4.1 Set-consensus

In 1990, Soma Chaudhuri [12] proposed a simple generalization of consensus in which more than one distinct decision value can be chosen. This problem later triggered the discovery of the fundamental role of topology in distributed computing. The termination and validity conditions of consensus are maintained, while the agreement condition is relaxed. In the *k-set-consensus* problem, $k < n$, each process gets a value from the set $\{0, 1, \ldots, n-1\}$, and has to decide on a value satisfying:

- *Termination*: Every non-faulty process must at some point irreversibly decide some value;
- *Agreement*: All non-faulty processes decide on at most k different values; and
- *Validity*: Every decided value must be the input of some process.

Set-consensus has the interesting property that it can be solved in a sufficiently reliable system: $k \geq f+1$. Moreover, it can be solved with one iteration of the approximate agreement algorithm of Section 3.3: the processes take snapshots until at least $n-f$ input values are read, and then decide on one of those values, say the median. Clearly, this algorithm terminates in a system with at most f failures, and validity is satisfied. As discussed in Section 3.3, the number of different values is at most $f+1$, and hence the agreement condition is satisfied. Also, for $f=1$, it follows that 2-set-consensus is solvable, and as mentioned in that section, by the FLP result, 1-set-consensus, i.e. consensus, is not solvable. To show that, in general, k-set-consensus is not solvable if $k < f+1$, we need to deal with higher-dimensional versions of connectivity, and of Lemma 1. We do this next.

4.2 Topology

A graph can be seen as a collection of 1 element sets (vertices), and of 2 element sets (edges), closed under contention, since if $\{u, v\}$ is in the collection, so are $\{u\}$ and $\{v\}$. A *simplicial complex* (or simply a *complex*) is a collection of sets, called *simplexes,* closed under intersection. Thus, the subsets of a simplex, called *faces,* are included in the complex. The *dimension* of a simplex, d, is one less than its cardinality, denoted d-simplex. The 0-simplexes are also called *vertices.* In an N-dimensional complex the maximum dimension of a simplex is N, and every simplex is contained in an N-simplex.

A simplex has a simple geometric interpretation, as the convex hull of $n+1$ independent points in a Euclidean space. A complex is a discrete approximation to a geometric object such as a surface or solid. For example, a 2-dimensional disk is homeomorphic to a 2-simplex (a solid triangle).

It is often useful to *subdivide* a complex, replacing each simplex with a complex that occupies the same point set. Subdivisions play an important role in distributed computing, because a subdivision of a complex preserves all its structure. We start with the generalized version of Lemma 1. Notice that in an N-complex which is a subdivided N-simplex every $N-1$-simplex is contained in either one or two N-simplexes. For example, in a subdivided triangle, every edge is contained in either one or two triangles. The $N-1$ simplexes contained in exactly one N-simplex are the *boundary* of the complex. We state only the two dimensional version of Lemma 1, because it gives very precisely the intuition of the higher dimensional version. In this case, we consider an arbitrarily subdivided triangle. It has three vertices in its corners, and its boundary is a graph consisting of three paths connected at the corners, forming a cycle. The inside of the triangle consist of many little triangles.

Lemma 2 (2-dimensional Sperner). *Consider a 2-dimensional subdivided simplex, K^2, such that every vertex is assigned a color from $\{0,1,2\}$, (i) the corners are assigned different colors, and (ii) the vertices in the path connecting two corners are colored with the colors of these corners. Then there is at least one 2-simplex colored with all three colors.*

The proof of Lemma 2 is an elementary parity counting argument, and uses the fact that there is an odd number of edges in the boundary colored 0 and 1, as implied by Lemma 1. The proof shows that the number of triangles colored $0,1,2$ is odd. For example, see [6, 10].

Although Lemma 2 can be proved using just combinatorics, a nice intuition for this lemma comes by proving it using topological arguments. This proof is by viewing it as folding one object into another, as in Section 3.5. The first object is the subdivided simplex, K^2, and it is folded into a triangle, S^2, whose three vertices are called $0,1,2$, subject to the restriction that the boundary of K^2 is mapped to the boundary S^2. The lemma says that, since K^2 has no holes, then at least one of its triangles will have to go to the 2-simplex of S^2.

Holes. The role of the holes of a complex is very important for distributed computing (and for topology). A complex has no holes of dimension d if any continuous map of the d-sphere into that complex can be "filled in" by extending it to a $(d+1)$-disk. For example, a complex has no holes of dimension 0 if it is connected: a 0-sphere consists of two disconnected points, and the 1-disk is a path between them. A subdivided simplex has no holes of any dimension, and this is important in the proof of Lemma 2. In general, a complex is *solid* if it has no holes of any dimension.

4.3 Asynchronous Wait-free Solvability

We now describe how to use topological notions to model decision tasks and protocols. A *task* consists of an *input complex,* $\mathcal{I}$, an *output complex,* $\mathcal{O}$, and an input-output relation Δ, from $\mathcal{I}$ to $\mathcal{O}$. Each simplex in $\mathcal{I}$ specifies a set of input values to the processes, and each output simplex in $\mathcal{O}$ specifies a set of output values. If $S \in \mathcal{I}$, then $\Delta(S) \subseteq \mathcal{O}$ specifies the allowable outputs when the processes start with inputs from S. If $S^d \in \mathcal{I}$ is of dimension d, then it specifies inputs for $d+1$ processes, when they run to completion before the other processes take any steps. In this case, $\Delta(S^d) \subseteq \mathcal{O}$ is a subcomplex of dimension d.

The *protocol complex,* $\mathcal{P}$, consists of an $n-1$-simplex for each final state of the system, and all their faces. Each vertex of a simplex is labeled with a local state of a process in the execution corresponding to that simplex, representing the local view of that process in the execution. A simplex corresponds to a set of executions, which are indistinguishable to the processes of that simplex. If the simplex is of dimension d, it represent a set of executions where $d+1$ processes observe operations from each other, but not from the other $n-(d+1)$ processes.

When $d+1$ processes start with inputs from $S^d \in \mathcal{I}$, $\mathcal{P}(S^d)$ is the subcomplex of $\mathcal{P}$ of all states corresponding to executions starting in S^d. Now, the decision of the processes induce a *simplicial decision map* δ from $\mathcal{P}$ to $\mathcal{O}$. This map sends a vertex of $\mathcal{P}$ of some process with some decision value, to a vertex of $\mathcal{O}$ labeled with the same process and output value. The map δ is *simplicial* because it sends simplexes to simplexes, i.e., intuitively, it is continuous, and deforms $\mathcal{P}$ into $\mathcal{O}$, with the restriction:

For every $S \in \mathcal{I}$, $\delta(\mathcal{P}(S)) \subseteq \mathcal{O}(S)$.

The following fundamental result is implied by [8, 34, 44]. The protocol complex $\mathcal{P}(S)$, for every $S \in \mathcal{I}$, of an asynchronous wait-free system is solid. This implies that $\mathcal{P}$ preserves all the structure of $\mathcal{I}$; $\mathcal{P}$ looks like a subdivision of $\mathcal{I}$. This claim is the "asynchronous computability theorem" of Herlihy and Shavit [34, 35]:

Theorem 1 (HS). *A task* $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ *is wait-free solvable if and only if there is a subdivision of* $\mathcal{I}$*, and a simplicial map* f *from* $\mathcal{I}$ *to* $\mathcal{O}$*, such that* f *sends vertices of one process to vertices of the same process, and* $f(S) \in \Delta(S)$*, for every* $S \in \mathcal{I}$.

That is, $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ is wait-free solvable if and only if $\mathcal{I}$ can be stretched and bent into $\mathcal{O}$ satisfying the requirement of Δ. Depending on how difficult it is to solve the task, how fine the subdivision must be, and how large is the time complexity of the algorithms.

Herlihy and Shavit proved the necessity of Theorem 1 in [34], using critical state arguments in the style of FLP. A alternative approach is to consider structured subsets of executions (as in [6, 8, 44]) where it can be directly shown that a subdivision is induced. The necessity part [35] is using a form of approximate agreement.

In a sense, the wait-free case is fundamental. The general case of $1 < f < n$ states that the protocol complex has no holes below dimension f. Two approaches for proving this result have been suggested in [34] with a critical state argument [34] and via a reduction [9] to the wait-free case using simulations [8].

Analogous properties in other models of distributed computation, like synchronous and asynchronous message passing, and stronger communication primitives in shared memory like set-consensus, see for example, [21, 28, 33, 41].

4.4 Applications

Set-consensus. We are ready to see why k-set-consensus is unsolvable, when $k < f+1$. We first consider the wait-free case, i.e., $f = n$, and $k < n+1$. This case captures the main ideas; the general case can be proved by reduction [9]. For this example, we consider only the three-process case.

Consider an input 2-simplex, S^2, where processes p_0, p_1, p_2 start with values $0, 1, 2$. As described in the previous section, $\mathcal{P}(S^2)$ is a subdivision of S^2. For each i, if p_i runs solo, i.e., finishes its computation before seeing operations by the other processes, it has to decide i. Thus, the i-th corner of $\mathcal{P}(S^2)$ is colored with decision value i. Similarly, in any execution where p_i, p_j see only each other, they have to decide either i or j, and hence all vertices along that boundary of $\mathcal{P}(S^2)$ are colored either i or j. All internal vertices of $\mathcal{P}(S^2)$ are colored with any of the decision values $0, 1, 2$. It follows that this coloring induced by the decision values satisfies Lemma 2. It follows that there must be at least one simplex colored with all three colors, and hence an execution where three values are decided. Thus, k cannot be less than 3. This argument readily generalizes to n processes, $f = n - 1$, and a Sperner argument for $\mathcal{P}(S^{n+1})$ shows that in at least one execution $n + 1$ values are decided, implying that $k \geq f + 1$.

4.5 Simplex Agreement and Convergence Tasks

Recall the approximate agreement task of Section 3.3, where processes start with real values and have to decide on real values ϵ apart from each other. In the 1-dimensional discrete version of the problem, processes start at the end-points of a path, and have to converge to a single vertex or to two vertices joined by an edge. In the d-dimensional version, processes start at the corners of a subdivided d-simplex, and have to decide on (not necessarily distinct) vertices contained in a simplex of the subdivision. Thus, processes start with at most $d + 1$ input values (vertices in the corners of the subdivision), and have to decide on at most $d + 1$ output values that form a simplex. This task is wait-free solvable using the algorithm described in Section 3.3, by executing rounds of snapshots; the number of rounds depends on the level of the subdivision. It is a consequence of the set-consensus impossibility result that processes can become arbitrarily close, but can never agree exactly: in at least one execution, at least $d+1$ different values are decided.

Most of the results obtained using topology are impossibility proofs. Various possibility results [30, 31] have to do with the solvability of approximate agreement on more arbitrary spaces. In a *convergence task* [30], processes start on vertices of a given complex, and have to converge on vertices of the complex contained in a simplex, subject to some restrictions. The restrictions specify the part of the complex where they can converge, depending on where they started. Herlihy and Rajsbaum [30] presented a generic algorithm to solve this task, and gave conditions specifying when the algorithm solves the convergence task, depending on the complex and its restrictions, under various asynchronous shared memory systems. The idea is to use the discrete approximate agreement algorithm described above, and then use topological arguments to show that the processes can map the values obtained by this algorithm to decisions in the given complex.

Undecidability. A nice application of the wait-free solvability characterization is proving that there is no algorithm that takes a decision task specification as input and tells if there is a wait-free algorithm that solves the task [22, 30] (the same result holds in other models [30]).

This undecidability result is by reduction to the classic *contractibility* problem in topology: it is undecidable whether an arbitrary loop in an arbitrary finite complex can be contracted to a point. The result holds even if the complex is 2-dimensional, and the loop is simple.

The idea of the reduction is to show that a the loop is contractible if and only if there is a wait-free solution to a convergence task on the complex with the following restrictions. Take three distinct, distinguished vertices on the given loop. If all processes start on the same distinguished vertex, they all must decide that same vertex. If they all start with two distinguish vertices, they have to decide on vertices spanning a simplex on the sub-path of the loop connecting the two vertices. If the processes start with three different distinguished vertices, they can decide on any vertices, as long as they are contained in a simplex of the given complex.

The proof of this claim follows from the wait-free solvability characterization, and the fact that if a loop is contractible then there is a simplicial map δ from a sufficiently fine subdivision of a simplex to the complex, sending the boundary of the simplex to the loop. Then, solving approximate agreement on the subdivided simplex, and then following the map δ to decide vertices of the given complex. On the other hand, if the convergence task is solvable, then the fact that the complex is solid implies that there is a contraction of the loop, because the restrictions of the convergence task force the boundary of the protocol complex to be mapped to the loop.

References

1. H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics,* McGraw–Hill, 1998.
2. Baruch Awerbuch, "Complexity of Network Synchronization," *Journal of the ACM,* Vol. 32, No. 4, Oct. 1985, pp. 804-823.
3. H. Attiya, A. Bar-Noy, and D. Dolev, "Sharing Memory Robustly in message Passing Systems," *Journal of the ACM,* Vol. 42, No. 1 (January 1995), pp. 124–142.
4. Hagit Attiya, Amotz Bar-Noy, Danny Dolev, David Peleg, and Rudiger Reischuk, "Renaming in an asynchronous environment," *Journal of the ACM* 37, 3, July 1990, 524–548.
5. Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt and N. Shavit, "Atomic snapshots of shared memory," *J. of the ACM,* 40(4), (Sept. 1993), 873–890.
6. Hagit Attiya and Sergio Rajsbaum, "The Combinatorial Structure of Wait-free Solvable Tasks," *10th International Workshop on Distributed Algorithms (WDAG)*, O. Babaoglu and K. Marzullo, Eds., October 1996, 321–343. Lecture Notes in Computer Science #1151, Springer-Verlag.
7. M.A. Armstrong, *Basic Topology*, Undergraduate Texts In Mathematics, Springer-Verlag, New York, 1983.

8. E. Borowsky and E. Gafni, "Generalized FLP impossibility result for t-resilient asynchronous computations," in *Proceedings of the 1993 ACM Symposium on Theory of Computing*, May 1993, 91–100.
9. E. Borowsky, E. Gafni, N. Lynch, and S. Rajsbaum, "The BG Distributed Simulation Algorithm," Technical Memo MIT/LCS/TM-573, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, December, 1997. Submitted for publication. Includes the results of [8, 39].
10. J.A. Bondy and U.S.R. Murty, *Graph theory with applications,* North-Holland, 1979.
11. O. Biran, S. Moran, S. Zaks, "A combinatorial characterization of the distributed 1-solvable tasks," *Journal of Algorithms,* 11, 1990, 420-440.
12. S. Chaudhuri, "Agreement is harder than consensus: set consensus problems in totally asynchronous systems," in *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, 1990, 311–234, 1990.
13. S. Chaudhuri, M.P. Herlihy, N. Lynch, and M.R. Tuttle, "A tight lower bound for k-set agreement," in *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, October 1993, 206–215.
14. D. Dolev and C. Dwork and L Stockmeyer, "On The Minimal Synchronism Needed For Distributed Consensus", in Journal of the ACM, 34(1), January 1987, 77–97.
15. D. Dolev, N. Lynch, S. Pinter, E. Stark and W. Weihl, "Reaching approximate agreement in the presence of faults," *Journal of the ACM,* 33 (3), 1986, 499–516.
16. D. Dolev, H.R. Strong, " Polynomial algorithms for multiple processor agreement," in *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pp. 401-407, May 1982.
17. C. Dwork, Y. Moses, "Knowledge and common knowledge in a byzantine environment: crash failures," *Information and Computation,* vol. 8, no. 2, pp. 156–186, October 1990.
18. M. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," Research Report YALE/DCS/RR-273, Yale University, Department of Computer Science, New Haven, Conn., June 1983.
19. M.J. Fischer, N.A. Lynch, "A lower bound for the time to assure interactive consistency," *Information Processing Letters,* vol. 14, no. 4, pp. 183–186, June 1982.
20. 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, 374–382.
21. E. Gafni, "Round-by-round fault detectors: unifying synchrony and asynchrony," *Proc. of the 17th ACM Symp. on Principles of Dist. Comp.,* 1998, pp. 199–208.
22. E. Gafni, E. Koutsoupias, "3-processor tasks are undecidable," *Proceedings of the 14-th Annual ACM Symposium on Principles of Distributed Computing*, page 271, August 1995.
23. E. Goubault, "Schedulers as abstract interpretations of HDA," *Proc. of PEPM'95*, 1995.
24. E. Goubault, "The dynamics of wait-free distributed computations," *LIENS, Ecole Normale Superieure*, 26 (1996), 1–40.
25. E. Goubault, "A semantic view on distributed computability and complexity," *Proc. of the 3rd Theory and Formal Methods Section Workshop*, 1996.
26. J. Gunawardena, "Homotopy and concurrency," *Bulletin of the EATCS*, 54 (1994), 184–193.
27. M.P. Herlihy, "Wait-Free Synchronization," *ACM Transactions on Programming Languages and Systems,* vol. 13, no. 1, pp. 123-149, January 1991.

28. M.P. Herlihy and S. Rajsbaum, "Set Consensus Using Arbitrary Objects," in *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, pages 324–333, August 1994.
29. M.P. Herlihy and S. Rajsbaum, "Algebraic Spans," in *Proc. of the 14th Annual ACM Symp. on Principles of Dist. Comp.*, 90–99, 1995.
30. M.P. Herlihy and S. Rajsbaum, "The decidability of distributed decision task," in *Proc. of the 29th ACM Symposium on Theory of Computing*, 589–598, 1997.
31. M.P. Herlihy and S. Rajsbaum, "A Wait-Free Classification of Loop Agreement Tasks," S. Kutten (Ed.) *Lecture Notes in Computer Science # 1499,* Springer Verlag, 175–185: 12th Int. Symp. on Dist. Comp. (DISC before WDAG), Sept. 24–26, 1998.
32. M.P. Herlihy and S. Rajsbaum, "A Primer on Algebraic Topology and Distributed Computing," in *Computer Science Today,* Jan van Leeuwen (Ed.), LNCS Vol. 1000, Springer–Verlag, 1995, p. 203–217.
33. M.P. Herlihy, S. Rajsbaum, and M. Tuttle, "Unifying Synchronous and Asynchronous Message-Passing Models," in *Proc. of the 17th ACM Symp. on Principles of Dist. Comp.* (PODC), 1998, 133–142.
34. M.P. Herlihy and N. Shavit, "The asynchronous computability theorem for t-resilient tasks," In *Proceedings of the 1993 ACM Symposium on Theory of Computing*, May 1993, 111–120.
35. M.P. Herlihy and N. Shavit, "A simple constructive computability theorem for wait-free computation," *Proceedings of the 1994 ACM Symposium on Theory of Computing*, May 1994, 243–252.
36. G. Hoest and N. Shavit, "Towards a Topological Characterization of Asynchronous Complexity," *Proc. of the 16th ACM Symp. on Principles of Dist. Comp.* (PODC), 1997, 199–208.
37. M. C. Loui and H.H. Abu-Amara, "Memory requirements for agreement among unreliable asynchronous processes," In *Parralel and Distributed Computing*, F. P. Preparata, editor, vol. 4 of *Advances in Computing Research*, pages 163–183. JAI Press, 1987.
38. N.A. Lynch, *Distributed Algorithms,* Morgan Kaufmann Publishers, Inc. 1996.
39. N.A. Lynch and S. Rajsbaum, "On the Borowsky-Gafni Simulation Algorithm," In *Proceedings of the Fourth Israel Symposium on Theory of Computing and Systems*, June 1996, 4–15.
40. S. Moran and Y. Wolfstahl, "Extended impossibility results for asynchronous complete networks," *Information Processing Letters,* 26 (1987/88), 145–151.
41. Y. Moses and S. Rajsbaum, "The unified structure of consensus: a layered analysis approach," in *Proc. of the 17th ACM Symp. Principles of Dist. Comp.,* 1998, pp. 123-132.
42. M., Pease, R. Shostak and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM,* Vol. 27, No. 2, (April 1980),. 228-234.
43. John H. Wensley et al. "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," *Proceedings of the IEEE,* Vol. 66, No. 10, pp. 1240–1255, October 1978.
44. M. Saks and F. Zaharoglou, "Wait-free k-set agreement is impossible: The topology of public knowledge," In *Proceedings of the 1993 ACM Symposium on Theory of Computing*, May 1993, 101–110.