The Decidability of Distributed Decision Tasks (Extended Abstract)

Maurice Herlihy*
Computer Science Department
Brown University, Providence RI 02912
herlihy@cs.brown.edu

Sergio Rajsbaum†
Instituto de Matemáticas
U.N.A.M., D.F. 04510, México
rajsbaum@servidor.unam.mx

Abstract

A *task* is a distributed coordination problem in which each process starts with a private input value taken from a finite set, communicates with the other processes by applying operations to shared objects, and eventually halts with a private output value, also taken from a finite set. A *protocol* is a distributed program that solves a task. A protocol is *t-resilient* if it tolerates failures by t or fewer processes. A task is *solvable* in a given model of computation if it has a t-resilient protocol in that model. A set of tasks is *decidable* in a given model of computation if there exists an effective procedure for deciding whether any task in that set has a t-resilient protocol.

This paper gives the first necessary and sufficient conditions for task decidability in a range of different models and resilience levels. We prove undecidability by exploiting classical decidability results from algebraic topology, and we prove decidability by explicit construction.

1 Introduction

A *task* is a distributed coordination problem in which each process starts with a private *input value* taken from a finite set, communicates with the other processes by applying operations to shared objects, and eventually halts with a private *output value*, also taken from a finite set. Tasks model "reactive" distributed systems such as databases, file systems, or flight control systems. Input values represent information entering the system from the outside world, such as a character typed at a keyboard, a message from another computer, or a signal from a sensor. Output values model effects on the outside world, such as an irrevocable decision to commit a transaction, to dispense cash, or to launch a missile.

A *protocol* is a distributed program that solves a task. A task is *solvable* if it has a protocol. An extensive literature has investigated the solvability of particular tasks, such as *consensus* [2, 16, 18], *renaming* [4, 27, 28], and *set agreement* [11, 13, 26, 27, 28, 36]. Whether a task is solvable depends on two important aspects of the model of computation: the degree of *resilience* required, and the underlying *communication model*. We now describe each of these aspects in more detail.

Modern multiprocessors are inherently *asynchronous*: processes may be delayed without warning for a variety of reasons, including interrupts, preemption, cache misses, communication delays, or failures. We say that a protocol is *t-resilient* it tolerates unbounded delay or failure by t or fewer processes, and it is *wait-free* if it tolerates delay or failure by all but one of the processes.

Early work focused on models in which processes communicate either by message-passing or by reading and writing a shared memory. Although read/write models retain theoretical interest, they are not an accurate reflection of real distributed and concurrent systems. Modern multiprocessor architectures provide a variety of more powerful synchronization primitives, including test-and-set [21], fetch-and-add [20], compare-and-swap [30] and load-linked/store-conditional [15, 32, 38]. It is known that each of these primitives is more powerful than simple read/write memory [23]. Formally, we model such synchronization primitives as a parameterized family of (m,k)-*set agreement* objects, where m is the number of processes that share an object, and k is a measure of the object's power: the smaller k with respect to m, the more powerful the object.

We say that a set of tasks is *decidable* in a given model of computation if there exists an effective procedure for deciding whether any task in that set has a protocol in that model. This paper proves the first necessary and sufficient conditions for task decidability in a wide range of models and resilience levels. We prove undecidability by exploiting classical results from algebraic topology and we prove decidability by explicit construction.

Our results are summarized in Figure 1. We find that tasks are decidable in some models, but not others. In read-write memory, it is known that tasks are decidable if and only if at most one process may fail [7, 19]. Although (m,k)-set agreement objects for $2 < k < m$ are strictly more powerful than read-write memory [11, 28, 36], we show that tasks in this model remain undecidable when more than one process can fail. When $k = 2$, however, there is an abrupt transition: we show that tasks are decidable if and only if t, the number of processes that can fail, is less than m, the number of processes that share a single object. Finally, when $k = 1$, $(m,1)$-set agreement is the same as m-process consensus [2, 16, 18], and we show that tasks are decidable if and only if $t < 2m$.

*Supported by NSF grants DMS-9505949 and CCR-9613785.
†Supported by DGAPA and CONACyT grants.

STOC '97 El Paso, Texas USA

To prove our decidability results, we draw a direct connection between decidability and the ability to solve a particular task, the 2-set agreement task of Chaudhuri [13]. The algorithms we present for the decidable models work for any tasks.

We can use these results to analyze multiprocessor architectures: for t-resilient tasks to be decidable, an architecture must support a synchronization primitive powerful enough to allow $\lceil t/2 \rceil$ processes to reach consensus, or t of them to reach $(m, 2)$-set agreement. For example, in an architecture that supports only *test-and-set* (equivalent to 2-process consensus), 3-resilient tasks are decidable, but 4-resilient tasks are not. In architectures such as Digital's Alpha AXP [38], and IBM's PowerPC [15], all processes can reach consensus, so t-resilient tasks are decidable for any t. Although tasks are undecidable in many of the models shown in Figure 1, they are likely to be decidable in most models of practical interest.

The proofs presented here establish a direct connection between fundamental issues of distributed computing and basic structures of algebraic topology, explaining why tasks are decidable in some models and not in others: in the decidable models the problem reduces to computing connectivity while in the undecidable models the problem reduces to computing contractibility (see next section). These results give further insight into the nature of distributed computing by illustrating the qualitative difference between the well-known computability theory for centralized sequential systems and the emerging computability theory for asynchronous distributed systems. In a centralized system, without failures or asynchrony, the tasks considered in this paper are not just decidable, they are easily seen to be solvable. The need for coordination, however, can make even simple tasks unsolvable, and simple families of tasks undecidable. Indeed, our undecidability results hold even if the task specification is finite, and each process has access to its own private oracle capable of solving undecidable problems.

model	decidable	undecidable
read-write memory	$t = 1$	$t > 1$
(m, k)-set agreement for $k > 2$	$t = 1$	$t > 1$
$(m, 2)$-set agreement	$t < m$	$t \geq m$
m-consensus	$t < 2m$	$t \geq 2m$

Figure 1: Summary of Results

In this extended abstract we omit some generalizations of the results, and many details about the model and proofs, which can be found in the full version [22].

2 Related Work

Perhaps the first paper to investigate the solvability of distributed tasks was the landmark 1985 paper of Fischer, Lynch, and Paterson [18] which showed that *consensus*, then considered an abstraction of the database commitment problem, had no 1-resilient message-passing protocol. Other tasks that attracted attention include *renaming* [4, 27, 28] and *set agreement* [11, 13, 26, 27, 28, 36]. The first decidability result for decision tasks is due to Biran, Moran, and Zaks [7], who, extending [34], showed that tasks are decidable in the 1-resilient message-passing model (later shown to be equivalent to 1-resilient read-write memory [6]). The same authors also showed that deciding 1-solvability is NP-complete [8], and give an optimal algorithm [9]. Taubenfeld et al. [41] showed that tasks with initial failures only are decidable.

Surprisingly, perhaps, there were no further decidability results for decision tasks until 1996, when Gafni and Koutsoupias [19], recognizing the important link between task decidability and contractibility, showed that wait-free tasks for three or more processes are undecidable in the wait-free read-write model. Our results extend this line of research in two directions: they apply to arbitrary levels of resilience (not just wait-free protocols), and to a variety of more realistic computational models (not just read-write memory).

In a related area, Chor and Moscovici [14] give decidable conditions characterizing the tasks that can be solved using randomized consensus.

Herlihy and Shavit [28, 29] introduced a formalism for tasks based on the classical concept of simplicial complexes. They used elementary homology theory to show certain impossibility results for set agreement and renaming. (The set agreement results were shown independently by Borowsky and Gafni [11] and Saks and Zaharoglou [36] using different techniques.) More recently, Herlihy and Rajsbaum used homology theory to derive further impossibility results for set agreement [25, 26], and to unify a variety of known impossibility results in terms of the theory of chain maps and chain complexes [27]. Using the same simplicial model, Attiya and Rajsbaum [5] give purely combinatorial proofs of several impossibility results that had required algebraic techniques.

Computational topology encompasses a number of celebrated undecidable problems. For finitely-presented groups, the *word problem* (whether an expression reduces to the unit element) was shown to be undecidable by S.P. Novikov in 1955, and the *isomorphism problem* (whether two such groups are isomorphic) was shown to be undecidable by M.O. Rabin in 1958. In this paper, we exploit the well-known undecidability of the *contractibility* problem. Contractibility requires deciding whether a loop λ in a finite simplicial complex $\mathcal{K}$ can be continuously deformed to a point. From a finite description of $\mathcal{K}$ one can construct a finite presentation of its fundamental group. Any finitely-presented group can be realized as the fundamental group of some complex. A loop λ is contractible in $\mathcal{K}$ if and only if λ represents the identity element of the fundamental group of $\mathcal{K}$. It follows that deciding contractibility is equivalent to deciding the word problem for finitely-presented groups, which is known to be undecidable. (For a more complete discussion of these issues, see Miller [33], Stillwell [40], or Sergeraert [37].)

In Section 7, we give a generic algorithm for any solvable task in the (m, k)-set agreement models where tasks are decidable. Our algorithm builds on earlier work of Biran, Moran, and Zaks [7] and of Borowsky and Gafni [10], who outline an algorithm for solving decision tasks in all the (m, k)-set agreement models. Our algorithm differs from that of Borowsky and Gafni in several respects. Most importantly, our algorithm is constructive: given nothing but the task specification, we can test certain graph connectivity conditions to decide whether the task is solvable, and if so, give an algorithm. By contrast, the Borowsky-Gafni algorithm is designed to work in all cases (not just the decidable ones) and hence is not constructive. Their conditions are given in terms of the existence of a certain map from one simplicial complex to another. The existence of this map is, in general, not decidable.

An *object* is a long-lived data structure subject to a potentially unbounded sequence of operations (unlike a task, in which each process participates once). Jayanti and Toueg

[31] construct a class of objects for which it is undecidable whether an object has a wait-free two-process implementation in read/write memory. This construction, however, appears to depend in an essential way on the use of infinite complexes and non-recursive specifications. By contrast, our undecidability results here apply even when all complexes are finite and all task specifications are recursive.

3 Model

A set of $n+1$ sequential threads of control, called *processes*, communicate by applying operations to shared objects. In every model considered here, objects share a set of variables called *read-write memory*, possibly augmented by more powerful objects. An (m,k)-*set* object x is a set of k or fewer values, initially empty. At most m distinct processes may apply operations to an (m,k)-set object. The object provides two operations: *insert* and *choose*. If $|x| < k$, *insert*(x,v) inserts v into x, and if $|x| = k$, *insert*(x,v) has no effect. If $|x| > 0$, *choose*(x) returns an arbitrary element from x. Given an (m,k)-set object, one can easily solve the (m,k)-set agreement task ([13]), and vice-versa.

An initial or final state of a process is modeled as a *vertex*, $\vec{v} = \langle P, v\rangle$, a pair consisting of a process id P and a value v (either input or output). We speak of the vertex as being *colored* with the process id. A set of $d+1$ mutually compatible initial or final states is modeled as a *d-dimensional simplex* (or d-simplex) $S^d = (\vec{s}_0, \ldots, \vec{s}_d)$. We say that $\vec{s}_0, \ldots, \vec{s}_d$ *span* S^d. Simplex S is a (proper) *face* of T if the vertexes of S form a (proper) subset of the vertexes of T. Where convenient, we use superscripts to indicate dimensions of simplexes. By convention, a dimension $d < 0$ denotes an empty simplex. The set of process ids associated with simplex S^n is denoted by *ids*(S^n), and the set of values by *vals*(S^n).

The complete set of possible initial (or final) states is represented by a set of simplexes, closed under containment, called a *simplicial complex* (or complex) $\mathcal{K}$. The *dimension* of a complex is the highest dimension of any of its simplexes. In this paper all the complexes of dimension n are *full* in the sense that every simplex is contained in some n-simplex. The k-th *skeleton* of a complex, *skel*$^k(\mathcal{K})$, is the subcomplex consisting of all simplexes of dimension k or less. A map $\mu : \mathcal{K} \to \mathcal{L}$ carrying vertexes to vertexes is *simplicial* if it also carries simplexes to simplexes. It is *surjective* if every vertex in $\mathcal{L}$ is the image of a vertex in $\mathcal{K}$. A complex is *colored* if each vertex is labeled with a process id, and each m-simplex is labeled with $m+1$ distinct colors. If $\mathcal{K}$ and $\mathcal{L}$ are colored complexes, then μ is *color-preserving* if $id(\vec{v}) = id(\mu(\vec{v}))$.

A *task specification* is given by a colored *input complex* $\mathcal{I}$, a colored *output complex* $\mathcal{O}$, and a recursive relation Δ carrying each m-simplex of $\mathcal{I}$ to a set of m-simplexes of $\mathcal{O}$, for each $0 \le m \le n$. Δ has the following interpretation: if the $(m+1)$ processes named in S^m start with the designated input values, and the remaining $n-m$ processes fail without taking any steps, then each simplex in $\Delta(S^m)$ corresponds to a legal final state. Note that a task specification makes no mention of resilience.

A *protocol* is a program in which each process receives a private input value, communicates with the other processes via shared objects, and eventually halts with with a private output value. As many as t processes may fail by halting, and any non-faulty process chooses an output and halts after a finite number of steps; Any process that completes the protocol has an associated *view*, which is the record of operations applied to the shared objects. For example, if the processes share a read/write memory, a process view is the sequence of variables and values read and written.

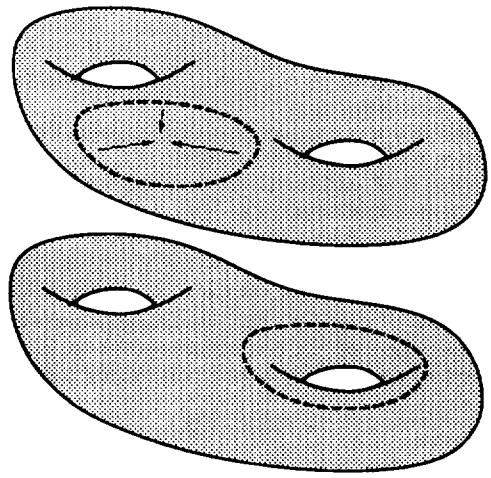

Figure 2: Contractible and Non-Contractible Loops

Any protocol has an associated *protocol complex* $\mathcal{P}$, defined as follows. Each vertex is labeled with a process id and a possible view for that process. A set of vertexes $\langle P_{i_1}, v_{i_1}\rangle, \ldots, \langle P_{i_d}, v_{i_d}\rangle$ span a simplex of $\mathcal{P}$ if and only if there is some execution in which $P_{i_1}, \ldots, P_{i_d}$ finish the protocol with respective views $v_{i_1}, \ldots, v_{i_d}$. Each simplex thus corresponds to an equivalence class of executions that "look the same" to the processes at its vertexes. The subcomplex of $\mathcal{P}$ corresponding to executions starting from a fixed simplex S^m is denoted $\mathcal{P}(S^m)$. The protocol complex $\mathcal{P}$ depends both on the protocol and the degree of resilience. A protocol *solves* a task if there exists a color-preserving simplicial *decision map* $\delta : \mathcal{P} \to \mathcal{O}$ such that for each simplex $R^m \in \mathcal{P}(S^m)$, $\delta(R^m) \in \Delta(S^m)$. Informally, when a process finished the protocol, it applies the decision map to its view to choose its output. We say that a process *participates* in a protocol execution if it takes any steps.

Although simplexes and complexes can be defined in a purely combinatorial way, it is frequently useful to give them a geometric interpretation. An *embedding* of a complex in Euclidean space maps each vertex to a point, each simplex to the convex hull of its affinely-independent vertexes, and each complex to a set of simplexes where each pair of simplexes intersect (as point sets) in a common face, or not at all. Any complex can be embedded in Euclidean space of sufficiently high dimension [40, p.22]. The *polyhedron* for a complex $\mathcal{K}$, denoted $|\mathcal{K}|$, is the point set underlying its embedding. Let $S^n = (\vec{s}_0, \ldots, \vec{s}_n)$. The *star* of S^m in $\mathcal{K}$, written $st(S^m, \mathcal{K})$, is the set of simplexes of $\mathcal{K}$ containing S^m, together with their faces. The *link* of S^m in $\mathcal{K}$, written *link*$(S^m, \mathcal{K})$ is the set of all simplexes in $st(S^m, \mathcal{K})$ that do not contain S^m.

Consider a point $k_0 \in |\mathcal{K}|$. We call k_0 the *base point*. A *loop* κ in $\mathcal{K}$ with base point k_0 is a continuous map from the unit interval $I = [0,1]$ to $|\mathcal{K}|$:

$$\kappa : I \to |\mathcal{K}|$$

such that $\kappa(0) = \kappa(1) = k_0$. A loop is *simple* if $\alpha(t)$ is unique for all $0 < t < 1$. Two loops κ and λ with base point v_0 are *homotopic* if one can be continuously deformed to the other while leaving the base point fixed. More precisely, there exists a continuous map $h : I \times I \to |\mathcal{C}|$, called a *homotopy*, such that $h(s,0) = \kappa(s)$, $h(s,1) = \lambda(s)$, and

$h(0,t) = h(1,t) = v_0$ for all $t \in [0,1]$. Homotopy is an equivalence relation. A loop is *contractible* if it is homotopic to the trivial loop k_0. A complex is *simply connected* if every loop is contractible.

An *edge loop* κ is a sequence of vertices such that each two consecutive vertices span a 1-simplex of $\mathcal{K}$, and the initial is equal to the final vertex. Every loop whose base point is a vertex is homotopic to an edge loop, so we can go back and forth between loops and edge loops at will.

Throughout this paper, $S^2 = (\vec{s}_0, \vec{s}_1, \vec{s}_2)$ is a 2-simplex, S^1_{ij} its face $(\vec{s}_i, \vec{s}_j)$, $\mathcal{S}^2$ the complex consisting of all faces of S^2 (including S^2), and $\dot{\mathcal{S}}^1$ the *boundary complex* consisting of all proper faces of S^2 (not including S^2). It is sometimes convenient to treat a loop κ as a continuous map

$$\kappa : |\dot{\mathcal{S}}^1| \to |\mathcal{K}|.$$

Lemma 0.1 ([39, 1.3.12]) *The following two conditions are equivalent:*

1. *$\kappa(|\dot{\mathcal{S}}^1|)$ is contractible, and*
2. *κ can be extended to a continuous map $f : |\mathcal{S}^2| \to |\mathcal{K}|$.*

4 Loop Agreement

Loop agreement is a family of tasks that includes generalizations of k-set agreement and approximate agreement. Let $\mathcal{K}$ be a finite (uncolored) 2-dimensional simplicial complex, κ a loop of $\mathcal{K}$, and $\vec{k}_0, \vec{k}_1, \vec{k}_2$ three distinguished vertexes in κ. For distinct i, j, and k, let κ_{ij} be the subpath of κ linking $\vec{k}_i$ to $\vec{k}_j$ without passing through $\vec{k}_k$, and assume these three paths are simple. In the $(\mathcal{K}, \lambda)$-loop agreement task for $n+1$ processes, each process has an input value in $\{0,1,2\}$. For each input simplex S^m, for $0 \le m \le n$, let $vals(S^m)$ be the set of input values, and define Δ by:

$vals(S^m)$	$\Delta(S^m)$
$\{i\}$	all decide $\vec{k}_i$
$\{i,j\}$	vertexes span simplex in κ_{ij}
$\{0,1,2\}$	vertexes span simplex in $\mathcal{K}$

In other words, the processes converge on a simplex in $\mathcal{K}$. If all processes have the same input value, they converge on the corresponding vertex. If they have only two distinct input vertexes, they converge on some simplex (either a vertex or an edge) along the path linking their vertexes. Finally, if the processes have all three input vertexes, they converge to any simplex of $\mathcal{K}$.

Here are some examples of loop agreement tasks.

- In the $(3,2)$-*set agreement* task [12], each of $n+1$ processes has an input taken from a set of 3 possible values, and each chooses an output value such that (1) each output is some process's input, and (2) no more than 2 distinct values are chosen. This task is the loop agreement task $(skel^1(\mathcal{S}^2), \zeta)$, where ζ is the edge loop $(\vec{s}_0, \vec{s}_1), (\vec{s}_1, \vec{s}_2), (\vec{s}_2, \vec{s}_0)$.
- Let $\sigma(\mathcal{S}^2)$ be an arbitrary subdivision of $\mathcal{S}^2$. In the 2-dimensional *uncolored simplex agreement* task, each process starts with a vertex in $\mathcal{S}^2$. If $S \in \mathcal{S}^2$ is the face spanned by the vertexes corresponding to inputs of participating processes, then the processes converge on a simplex in $\sigma(S)$. (This task is the uncolored version of simplex agreement [29].) This task is the loop agreement $(\sigma(\mathcal{S}^2), \sigma(\zeta))$, where ζ is the loop described in the previous task.
- The 2-dimensional *barycentric agreement* task is uncolored simplex agreement for the barycentric subdivision [35, p.96] $bary(\mathcal{S}^2)$ of $\mathcal{S}^2$.
- An *ϵ-approximate agreement task* [17] can be defined as a 1-dimensional version of barycentric agreement, $bary^\ell(\mathcal{S}^1)$, for large enough ℓ, where two of the vertices of the loop are equal. Processes start with 0 or 1, and they have to decide real numbers which differ by at most ϵ.

We will show that $(\mathcal{K}, \lambda)$-loop agreement is solvable in some models if and only if λ is contractible in $\mathcal{K}$, implying that tasks are undecidable in those models.

Formally, the output complex of loop agreement is defined in terms of the *colorized* complex $\widetilde{\mathcal{K}}$ as follows.

- The vertexes are all combinations of the form $\langle P_i, \vec{v} \rangle$, where P_i is a process id and $\vec{v}$ a vertex of $\mathcal{K}$.
- Vertexes $\langle P_0, \vec{v}_0 \rangle, \ldots, \langle P_k, \vec{v}_k \rangle$ lie on a simplex in $\widetilde{\mathcal{K}}$ if and only if the P_i are distinct process ids, and $(\vec{v}_0, \ldots, \vec{v}_k)$ is a simplex in $\mathcal{K}$. (Note that the $\vec{v}_i$ need not be distinct.)

Let $\pi : \widetilde{\mathcal{K}} \to K$ be the projection map. We require that for every input simplex S^m (of the colorized input complex) and output simplex $K^m \in \Delta(S^m) \subseteq \widetilde{\mathcal{K}}$, $\pi(K^m) \subseteq \mathcal{K}$ satisfies the conditions given above.

5 Contractible Implies Solvable

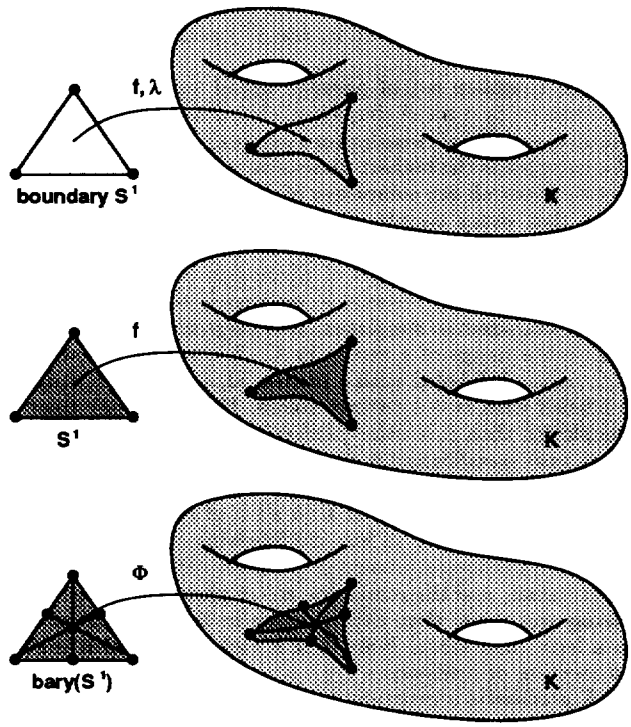

Figure 3: Proof of Theorem 1

Theorem 1 *If κ is contractible in $\mathcal{K}$, then $(\mathcal{K}, \kappa)$-loop agreement has an $(n+1)$-process wait-free read/write protocol.*

Note that if loop agreement has a wait-free read/write protocol, it also has a t-resilient read/write protocol, for $0 < t \leq n$, as well as a t-resilient protocol in any model combining read/write memory with other objects.

We first observe that the barycentric agreement task described above has a simple wait-free read-write protocol. Let S be the simplex of $\mathcal{S}$ spanned by the input vertexes. Each process writes its input to a shared array, snapshots the array [1], and chooses the barycenter of the simplex spanned by the vertexes read. By iterating this protocol N times, the processes can choose vertexes on a single simplex of $bary^N(S^2)$, the iterated barycentric subdivision of S^2.

We can view κ as a continuous map $|\dot{S}^1| \rightarrow |\mathcal{K}|$ (see top of Figure 3). Lemma 0.1 implies that because κ is contractible, it can be extended to a continuous map

$$f : |S^2| \rightarrow |\mathcal{K}|$$

such that $f(\vec{s}_i) = \vec{k}_i$, and f carries each face S^1_{ij} to κ_{ij} (see middle of Figure 3).

A *simplicial approximation* of f is a simplicial map $\phi : bary^N(\mathcal{K}) \rightarrow \mathcal{L}$, for some large enough N, such that $\phi(\vec{s}_i) = \vec{k}_i$, and ϕ carries every simplex in $bary^N(S^1_{ij})$ to a simplex in κ_{ij} (see bottom of Figure 3). The classical Simplicial Approximation Theorem [35, Th. 16.1] states that f has a simplicial approximation.

If κ is contractible in $\mathcal{K}$, we can now give a wait-free read-write protocol for $(\mathcal{K}, \kappa)$-loop agreement. Each process starts with a vertex in S^2, applies iterated barycentric agreement to choose a vertex in $bary^N(S^2)$, and then applies the decision map ϕ to choose a vertex in $\mathcal{K}$.

An unusual aspect of this construction is that it exploits a classical theorem of algebraic topology to derive an algorithm. Most other applications of such theorems [25, 26, 27, 28] have been used to derive impossibility results.

6 Solvable Implies Contractible

Consider a t-resilient $(\mathcal{K}, \kappa)$-loop agreement protocol Π in which $n+1$ processes communicate by a read/write memory possibly augmented by other objects. Call $P_0, \ldots, P_t$ the *low-order* processes, and $P_{t+1}, \ldots, P_n$ the *high-order* processes (wait-free protocols have no high-order processes). Any such protocol Π for loop agreement can be transformed into a protocol Π' *insensitive* to high-order inputs:

- If all low-order processes have input $\vec{s}_i$, then all processes decide $\vec{s}_i$.
- If all low-order processes have input $\vec{s}_i$ or $\vec{s}_j$, then all vertexes chosen lie on a 1-simplex in κ_{ij}.

Each process writes its input value to an array, waits for at least $n - t + 1$ inputs to appear, and then replaces its own input with that of the process with the least id. It then executes the protocol Π. This "preprocessing" ensures that every process starts the protocol with the input of some low-order process. Henceforth, without loss of generality, we restrict our attention to such protocols.

Define the *join* of simplexes $R^m = (\vec{r}_0, \ldots, \vec{r}_m)$ and $S^n = (\vec{s}_0, \ldots, \vec{s}_n)$ to be the $(m+n+1)$ simplex

$$R^m \cdot S^n = (\vec{r}_0, \ldots, \vec{r}_m, \vec{s}_0, \ldots, \vec{s}_n).$$

A *2-decomposition of* $t+1$ is a sequence (d_0, d_1, d_2) of non-negative integers such that $\sum_{i=0}^{2}(d_i + 1) = t + 1$. A (d_0, d_1, d_2)-*decomposition* of an input simplex S^n is any expression of the form

$$S^n = S_0 \cdot S_1 \cdot S_2 \cdot S^{n-t-1},$$

where $dim(S_i) = d_i$, and S^{n-t-1} is labeled with the high-order processes. Hence, each S_i has at least one vertex.

Definition 6.1 A t-faulty model of computation is 2-*weak* if there exists a 2-decomposition (d_0, d_1, d_2) of $t+1$, such that for every protocol $\mathcal{P}$, every input simplex S^n, and every (d_0, d_1, d_2)-decomposition of S^n,

$$S^n = S_0 \cdot S_1 \cdot S_2 \cdot S^{n-t-1},$$

- each $\mathcal{P}(S_i \cdot S_j \cdot S^{n-t})$ is connected, and
- $\mathcal{P}(S^n)$ is simply connected.

Notice that in order for a 2-decomposition (d_0, d_1, d_2) of $t+1$ to exist, t must be at least 2.

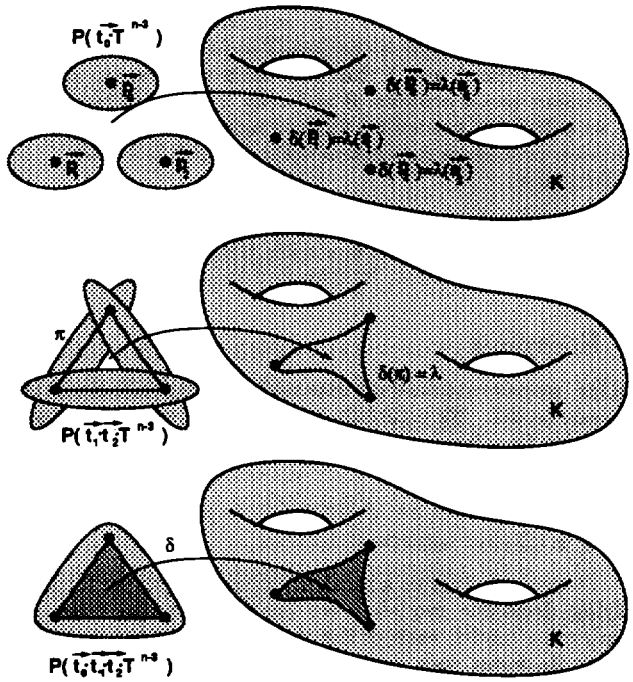

Figure 4: Proof of Theorem 2

Theorem 2 *If $(\mathcal{K}, \kappa)$-loop agreement has a t-resilient protocol in a 2-weak model, then κ is contractible in $\mathcal{K}$.*

Proof: The proof is illustrated in Figure 4 for the wait-free case. Pick the input simplex where each process in S_i has input i. As noted above, we can ignore all high-order inputs. Because each S_i is non-empty, we can pick a vertex $\vec{s}_i$ from each S_i. We now define a continuous map

$$f : |\mathcal{S}^2| \rightarrow |\mathcal{P}(S^n)|.$$

Define $\vec{q}_i = f(\vec{s}_i)$ to be any vertex in $\mathcal{P}(\vec{s}_i \cdot S^{n-t-1})$

Because $\mathcal{P}(S_i \cdot S_j \cdot S^{n-t-1})$ is connected, there is a path

$$\lambda_{ij} : I \rightarrow |\mathcal{P}(S_i \cdot S_j \cdot S^{n-t-1})|$$

linking $\vec{q}_i$ and $\vec{q}_j$. Define $f((1-t)\cdot\vec{q}_i + t\cdot\vec{q}_j) = \lambda_{ij}(t)$. Applying this construction to each face of S^2, we have

$$f : |\dot{S}^1| \to |\mathcal{P}(S_0 \cdot S_1 \cdot S_2 \cdot S^{n-t-1})|.$$

Because $\mathcal{P}(S^n)$ is simply connected, $f(|\dot{S}^1|)$ is contractible. By Lemma 0.1, f can be extended to $|S^2|$.

Let $\delta : \mathcal{P} \to \mathcal{K}$ be the decision map composed with the projection map π. Let $g : |S^2| \to |\mathcal{K}|$ be the composition:

$$|S^2| \xrightarrow{f} |\mathcal{P}(S^n)| \xrightarrow{\delta} |\mathcal{K}|$$

We claim that $g(|\dot{S}^1|) = |\kappa|$. In $\vec{s}_i \cdot S^{n-t-1}$, all low-order processes have input i, so all processes in $\mathcal{P}(\vec{s}_i \cdot S^{n-t-1})$ choose $\vec{k}_i$ (top of Figure 4), and $g(\vec{s}_i) = \vec{k}_i$. In $S_i \cdot S_j \cdot S^{n-t-1}$, all low-order processes have inputs i or j, so all processes in $\mathcal{P}(\vec{s}_i \cdot S^{n-t-1})$ choose vertexes along κ_{ij} (middle of Figure 4). Let i, j, and k be distinct. Because κ_{ij} is a simple path, $g(|S^2_{ij}|) = |\kappa_{ij}|$. It follows that κ is contractible (bottom of Figure 4). ■

Theorem 3 *Tasks are undecidable in any 2-weak model.*

Proof: It is easy to show that contractibility is also undecidable even for simple loops (proof in full paper). It follows from Theorems 1 and 2 that in 2-weak models, $(\mathcal{K}, \kappa)$-loop agreement has a protocol if and only if κ is contractible in $\mathcal{K}$, so a decision procedure for loop agreement would yield a decision procedure for contractibility, which is known to be undecidable. ■

Each of the models shown as undecidable in Figure 1 is 2-weak. (For read/write memory, see Herlihy and Shavit [28], and for the other cases, Herlihy and Rajsbaum [24, 27].)

7 Decidability

So far, we have established that tasks are undecidable in any 2-weak model. We establish a remarkably simple sufficient condition for a family of tasks to be decidable, via Theorem 6 of the next section.

Theorem 4 *Tasks are decidable in any model that admits an $(n+1, 2)$-set agreement protocol.*

Theorem 4 gives us a straightforward technique for identifying decidable models. For example, in the 1-resilient read/write model, each of the two low-order process simply writes its input to a register and halts, while each of the $n-1$ high-order processes simply waits and decides the first low-order value to appear in a register.

Corollary 4.1 *Tasks are decidable in any 1-resilient model that includes read/write registers.*

In the t-resilient $(m, 2)$-set agreement model, where $t < m$, each low-order process participates in an $(m, 2)$-set agreement protocol, writes its decision to a register, and halts. As before, the high-order processes wait for any low-order process to decide.

Corollary 4.2 *Tasks are decidable in the t-resilient $(m, 2)$-set agreement model when $t < m$.*

In the t-resilient m-consensus model, where $t < 2m$, $P_0, \ldots, P_{m-1}$ participate in one consensus protocol, while $P_m, \ldots, P_t$ participate in another. On completion, each low-order process writes its decision to a register, and halts, while the high-order processes wait for a low-order process to decide.

Corollary 4.3 *Tasks are decidable in the t-resilient m-consensus model when $t < 2m$.*

Note that these three corollaries establish the decidability claims of Figure 1.

7.1 Protocol Complexes

We now examine some important structural properties of protocol complexes in the decidable models.

Definition 7.1 A *k-path* in an n-complex $\mathcal{K}$ is a sequence of n-simplexes $S^n_0, \ldots, S^n_N$ such that $dim(S^n_i \cap S^n_{i+1}) \geq n-k$. A complex $\mathcal{K}$ is *k-path connected* if any pair of n-simplexes can be linked by a k-path.

Notice that if the dimension n of $\mathcal{K}$ is less than k, then $\mathcal{K}$ is trivially k-path connected, since by definition the dimension of an empty simplex is -1. When $k < n$, if $\mathcal{K}$ is k-path connected then it is connected in the usual sense.

Lemma 4.1 *If $\phi : \mathcal{K} \to \mathcal{L}$ is a color-preserving simplicial map, and $\mathcal{K}$ is k-path connected, then so is $\phi(\mathcal{K})$.*

Definition 7.2 A complex $\mathcal{K}$ is *k-link connected* if, for each $K^\ell \in \mathcal{K}$, $link(K^\ell, \mathcal{K})$ is k-path connected.

Since $dim(link(K^\ell, \mathcal{K})) = n - \ell - 1$, if $\mathcal{K}$ is k-link connected, then the link of any K^ℓ is connected in the usual sense if $\ell < n - k - 1$.

Lemma 4.2 *If $\mathcal{P}$ is a protocol complex then for every input simplex S^ℓ, $n-1 \leq \ell \leq n$, $\mathcal{P}(S^\ell)$ has the following properties in the following models:*

Model	$\mathcal{P}(S^\ell)$
read/write	*1-link connected for* $n-1 \leq \ell \leq n$
$(m, 2)$*-set agree*	*1-link connected for* $n-t \leq \ell \leq n$
m-consensus	*m-link connected for* $n-t+m \leq \ell \leq n$

7.2 Output Complexes

Most earlier work in this area [10, 25, 26, 27, 28] was concerned with proving the impossibility of specific tasks in specific models. If a task has a protocol in a particular model, then there exists a map from the protocol complex to that task's output complex. This map preserves certain topological properties. If all protocol complexes in a model can be shown to be topologically "incompatible" with a specific task's output complex, then that task has no protocol in that model.

Here, however, we must work in the opposite direction: given an arbitrary task specification, we must decide whether it is the image under a simplicial map of some protocol complex. As noted above, protocol complexes in the decidable models are k-link connected, for model-specific values of k. Unfortunately, link connectivity is not conserved by simplicial maps: the decision map is free to "glue" together arbitrary simplexes of matching color, so output simplexes of solvable tasks may have disconnected links. Nevertheless, we show in the full paper that it is possible to "pre-process" output complexes of solvable tasks in a way that restores the link connectivity of the underlying protocol complex. Informally, the preprocessing step discards "small" simplicial intersections in a way that ensures that every simplex of dimension $n-t$ or greater in the preprocessed output complex has a connected link. We then show that these two tasks are equivalent in the following sense.

Theorem 5 *Given a model of computation in which protocol complexes are k-link connected, and given a task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$, one can construct a "preprocessed" task $\langle \mathcal{I}, \mathcal{O}^*, \Delta^* \rangle$ such that*

1. *for each input simplex S of dimension greater than or equal $n-t$, $\Delta^*(S)$ is k-link connected,*
2. *$\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ has a t-resilient protocol if and only if $\langle \mathcal{I}, \mathcal{O}^*, \Delta^* \rangle$ does, and*
3. *if $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ has a t-resilient protocol with a surjective decision map, then so does $\langle \mathcal{I}, \mathcal{O}^*, \Delta^* \rangle$.*

Moreover, this preprocessing is computable.

8 Conditions

A *restriction* of a task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ is a task $\langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$, where for every input simplex S, $n - t \leq \dim(S) \leq n$, $\Gamma(S) \subseteq \Delta(S)$. To solve a task, it suffices to solve any restriction of that task. A task $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ is *extensible* if any set of at least $n-t+1$ processes can choose their outputs without knowledge of the remaining processes' inputs.

Definition 8.1 A t-resilient model of computation has *consensus number k* if it admits a t-resilient k-process consensus protocol, but no t-resilient $(k+1)$-process consensus protocol.

Lemma 5.1 *Any model of computation with consensus number greater than one admits an $(n+1)$-process* counting *protocol, in which each process takes a unique non-negative integer, and if some process takes i, then at least $i+1$ processes are participating in the protocol.*

Proof: Use 2-consensus protocols as balancers in a counting network [3]. ■

Lemma 5.2 *If a t-resilient model of computation has consensus number k, then for any protocol $\mathcal{P}$, $\mathcal{P}(s^\ell)$ is k-path connected, for $n - t \leq \ell \leq n$,*

Theorem 6 *In any t-resilient model of computation with consensus number k that admits an $(n+1, 2)$-set agreement protocol, $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ has a t-resilient protocol if and only if it has a restriction Γ such that the preprocessed task $\langle \mathcal{I}, \mathcal{O}^*, \Gamma^* \rangle$ satisfies the following conditions:*

Extensibility *If $S_0 \subset S_1$, and $\dim(S_0) \geq n - t$, then $\Gamma^*(S_0) \subseteq \Gamma^*(S_1)$.*

Connectivity *If $\dim(S) \geq n - t$, then $\Gamma^*(S)$ is k-path connected.*

These conditions are computable: there are only a finite set of restrictions Γ, extensibility and k-path connectivity are decidable, and Γ^* is computable by Theorem 5.

8.1 Necessity

Assume we have a t-resilient protocol $\mathcal{P}$ for $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ with decision map $\delta : \mathcal{P} \to \mathcal{O}$. Define the restriction $\Gamma(S) = \delta(\mathcal{P}(S))$. Theorem 5 states that there exists t-resilient protocol $\mathcal{P}^*$ for $\langle \mathcal{I}, \mathcal{O}^*, \Gamma^* \rangle$ with decision map δ^*. Moreover, because the decision map δ for $\langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ is surjective, so is the decision map δ^* for $\langle \mathcal{I}, \mathcal{O}^*, \Gamma^* \rangle$.

Lemma 6.1 *A task has a t-resilient protocol only if the extensibility condition is satisfied.*

```
shared vertex INPUTS[n] := [⊥,...,⊥];
shared simplex FIRST[t][n] := [⊥,...,⊥];
shared simplex SECOND[t][n] := [⊥,...,⊥];
simplex S_i, U_i, T_i = ∅, ∅, ∅

protocol(vertex i⃗)
1:  INPUTS[i] := i⃗;
2:  wait until scan(INPUTS) has ≥ n − t + 1 vertexes;
3:  S_i :=join of all vertexes from scan(INPUTS)
4:  FIRST[0][i] := agree(S_i)
    for r in 0..t do
5:      SECOND[r][i] := (∩T | T ∈ scan(FIRST[r]));
6:      view_i := {T | T ≠ ⊥ ∈ scan(SECOND[r])}
7:      if (∃v⃗ ∈ (∩T | T ∈ view_i)) | id(v⃗) = P_i
          then decide v⃗;
          end if ;
8:      U_i := (v⃗ | v⃗ ∈ T ∈ view_i and id(v⃗) ≠ P_i)
9:      T_i := U_i · (v⃗ ∈ T_i | id(v⃗) ∉ ids(U_i))
10:     S_i := (s⃗ | s⃗ ∈ scan(INPUTS))
11:     FIRST[r + 1][i] := link-agree(T_i, S_i);
        end for ;
    end protocol
```

Figure 5: A Generic Protocol

Proof: Because δ^* is surjective, some execution of the protocol leads to a state in which the processes in $ids(S^\ell)$ halt with values from T^ℓ. If $\Gamma(S^\ell)$ is not contained in $\Gamma(S^k)$, then if the remaining processes then start the protocol, they cannot be assigned output values consistent with T^ℓ. ■

Lemma 6.2 *A task has a t-resilient protocol only if the connectivity condition condition is satisfied.*

Proof: The model has consensus number k, so by Lemma 5.2, $\mathcal{P}^*(S^\ell)$ is k-path connected. Because δ^* is surjective, $\delta^*(\mathcal{P}^*(S^\ell)) = \Gamma^*(S^\ell)$ is also k-path connected by Lemma 4.1. ■

8.2 Sufficiency

We now define a "generic" protocol that solves any extensible task. This protocol calls two model-specific tasks as subroutines. We give explicit protocols for these tasks in the decidable models of computation.

The generic protocol appears in Figure 5. Let $\langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ be the task, $\langle \mathcal{I}, \mathcal{O}, \Gamma \rangle$ the restriction, and $\langle \mathcal{I}, \mathcal{O}^*, \Gamma^* \rangle$ the preprocessed task satisfying the Extensibility and Connectivity conditions. By Theorem 5, it is sufficient to show that $\langle \mathcal{I}, \mathcal{O}^*, \Gamma^* \rangle$ has a protocol.

The first model-specific task is *basic agreement*:

$$T_i := agree(S_i).$$

The input to each P_i is an input simplex S_i of dimension at least $n-t$, where the S_i are faces of a single input simplex. Each output is a simplex T_i in $\Gamma^*(\cup S_i)$, such that $\cap T_i \neq \emptyset$.

The second model-specific task is *link agreement*:

$$R_i := link\text{-}agree(T_i, S_i).$$

The input to each P_i is a pair of simplexes T_i and S_i, where each S_i is an input simplex, $T_i \in \Gamma^*(S_i)$, and $n - t \leq$

$dim(T_i) < dim(S_i)$, and the S_i are faces of a single simplex. The outputs R_i are simplexes in $link(\cap T_i, \Gamma^*(\cup S_i))$ such that $\cap R_i \neq \emptyset$.

Informally, the generic protocol works as follows. The processes share an $(n+1)$-element array of vertexes INPUTS, and two $(t+1)$ by $(n+1)$-element arrays of simplexes FIRST and SECOND. We use "$scan$(A[r])" to denote an atomic snapshot scan [1] of the r-th row of array A. Variables marked by subscripts are local variables.

At the start of the protocol, each P_i writes its input vertex to INPUTS[i] (Line 1), and waits until it observes at least $n - t + 1$ input vertexes (Line 2). P_i then constructs a simplex S_i by taking the join of all the input vertexes it observes (Line 3). P_i then calls the basic agreement protocol with argument S_i, and writes the resulting simplex to FIRST[0][i] (Line 4).

P_i then enters the loop. It atomically scans the FIRST[0] array (line 5), takes the intersection of the non-empty simplexes that it scans, and writes this simplex to SECOND[0][i], and scans SECOND[0] to construct a set of simplexes $view_i$ (Line 6). Because FIRST[0] is scanned atomically, the simplexes in $view_i$ are ordered by inclusion. If every simplex in $view_i$ has a vertex $\vec{v}$ with process id P_i, then P_i decides $\vec{v}$ (Line 7). Because all simplexes returned by basic agreement have at least one vertex in common, at least one process halts in round 0.

If P_i does not halt, then it constructs a simplex T_i of vertexes of processes that may have halted. Any process (other than P_i itself) with a vertex in $view_i$ may have halted, so P_i constructs U_i by joining the vertexes in $view_i$ other than its own (Line 8). Any process with a vertex already in T_i (empty in the first round) but not in $view_i$ is joined with U_i to form T_i (Line 9). P_i recomputes S_i by rescanning the INPUTS array (Line 10), calls a link agreement protocol with arguments T_i and S_i, and writes the result to FIRST[1][i].

Each subsequent iteration of the loop proceeds in the same way, except that simplexes written to FIRST[r] are generated by the link agreement protocol instead of basic agreement. The first vertex written to each FIRST[r] appears in every simplex in the round-r version of $view_i$, so at least one process halts in each subsequent round.

In the following lemmas, $T_i^{(r)}$ denotes the value of T_i at the end of round r. and $\cap T^{(r)}$ denotes the intersection of all non-empty $T_i^{(r)}$. Let r range over the rounds completed by some process.

Lemma 6.3 *$\cap T^{(r)}$ is a simplex.*

Proof: All simplexes written to FIRST[0] are chosen by link agreement, so they have a common intersection. Because scans are atomic, if P_i's scan of FIRST[0] is ordered before P_j's, then P_i's observed set of simplexes is included in P_j's, and therefore P_i's intersection is included in P_j's. All simplexes written to SECOND[0] are faces of a common simplex, and so is T_i.

Assume inductively that $\cap T^{(r)}$ is a simplex, for $r > 0$. All $\vec{v}$ in $\cap T^{(r+1)}$ but not in $\cap T^{(r)}$ are chosen by link agreement, and so lie on a common simplex in $link(\cap T^{(r)}, \Gamma^*(S^n))$. ■

Lemma 6.4 *$dim(\cap T^{(r)}) \geq r$.*

Proof: Let T be the first simplex written to FIRST[0] (Line 4). T appears in every scan of FIRST[0] (Line 5), and is a face of every simplex written to SECOND[0] (Line 5) and scanned into $view_i$ (Line 6). Every process with a vertex in T halts with that vertex (Line 7), and T is a face of the U_i constructed by each non-halted process. It follows that $T \subset \cap T^{(0)}$, so $dim(\cap T^{(0)}) \geq 0$.

In each subsequent round, let R be the first simplex written to FIRST[r]. Because R is returned by a link agreement protocol, $R \notin \cap T^{(r)}$, but the same argument shows that $R \in \cap T^{(r+1)}$. ■

Lemma 6.5 *The protocol of Figure 5 is correct.*

Proof: If P_i has a vertex $\vec{v}$ in $\cap T^{(r)}$ then it halts with $\vec{v}$ before round $r + 1$. Conversely, if P_i halts at round r with $\vec{v}$, then $\vec{v} \in \cap T^{(r+1)}$. Since every $\cap T^{(r)}$ is a simplex (Lemma 6.3), all processes halt on a common simplex. At least one process halts at round 0 (Lemma 6.4), and at least one process halts in each subsequent round (Lemma 6.4), so all processes halt by the end of round $n + 1$. ■

To show that the conditions of Theorem 6 are sufficient, we need only show that basic agreement and link agreement are solvable in any model that admits an $(n+1, 2)$-set agreement protocol.

8.2.1 Basic Agreement

```
shared simplex INPUTS[n] := [⊥, ..., ⊥];
basic-agree(simplex S_i)
  INPUTS[i] := S_i
  r_i := counter()
  if (r_i < k)
    then S_i := consensus(S_i)
    else S_i := (∪S | S ∈ scan(INPUTS));
    end if
  S := 2-set-agree(S_i)
  return (basic-graph-agree(S))
  end basic-agree
```

Figure 6: Basic Agreement Protocol for $k > 1$

We show how to solve basic agreement in any model of computation that admits a $(n+1, 2)$-set agreement protocol.

Definition 8.2 Let $\mathcal{K}$ be a complex. Define the *adjacency graph* $adj(\mathcal{K})$ to be a graph where each vertex is labeled with a simplex of $\mathcal{K}$, and there is an edge from one vertex to another if and only if both simplexes are faces of a common simplex.

To avoid cumbersome notation, we use a simplex S to denote both the simplex and its corresponding vertex in the adjacency graph, relying on context to disambiguate. Now consider the following task,

$$\langle adj(\mathcal{I}), adj(\mathcal{O}^*), B \rangle,$$

where

$$\begin{aligned} B(S) &= S \\ B(S_0, S_1) &= adj(\Gamma^*(S_0 \cup S_1)). \end{aligned}$$

The *basic graph agreement* task is defined in terms of the colorized version: Each process receives as input a simplex from $\mathcal{I}$. Like basic agreement, all are faces of a common simplex in $\mathcal{I}$. Unlike basic agreement, there may be at most two distinct inputs, S_0 and S_1. The processes choose simplexes from $\Gamma^*(S_0 \cup S_1)$. There are at most two distinct

```
shared pair INPUTS[n] := [⊥, ..., ⊥];
link-agree(simplex T_i, S_i)
   INPUTS[i] := ⟨S_i, T_i⟩
   r_i := counter()
   if (r_i < k)
     then ⟨S_i, T_i⟩ := consensus(S_i, T_i)
     else S_i := (∪S | ⟨S, T⟩ ∈ scan(INPUTS))
          T_i := (∩T | ⟨S, T⟩ ∈ scan(INPUTS))
     end if
   ⟨S_i, T_i⟩ := 2-set-agree(S_i, T_i)
   return (link-graph-agree(S_i, T_i))
   end link-agree
```

Figure 7: Link Agreement Protocol

outputs, and one must be a face of the other. In the full paper, we show that the Connectivity condition ensures that basic graph agreement has a wait-free read-write protocol if all inputs are S, or if $dim(S_0 \cup S_1) \geq k$.

We now describe a basic agreement protocol. If $k = 1$, then the processes run $(n+1,2)$-set agreement to choose at most two input simplexes, S_0 and S_1, which are used as input to basic graph agreement. Since $dim(S_0 \cup S_1) \geq 1$, basic graph agreement has a wait-free read/write protocol.

If $k > 1$, the protocol is illustrated in Figure 6. Each process starts by writing its input simplex to an INPUTS array. By Lemma 5.1, this model admits a *counter* protocol with the property that if it returns i to a process, then no other process receives i, and at least $i+1$ processes have already invoked the protocol. The processes invoke the counter protocol. All processes that receive a number less than k execute a k-consensus protocol to agree on an input simplex. They use this simplex as input to an $(n+1,2)$-set agreement protocol. The remaining processes rescan the INPUTS array, taking the union of all simplexes present, and use that union as input to the $(n+1,2)$-set agreement protocol. Since all input simplexes are faces of a single input simplex, these unions are well-defined. Moreover, because more than k processes have invoked the counter protocol, each process has a vertex labeled with its own color in its input, the dimension of each such union must be k or greater. Let S_0 and S_1 be the simplexes returned by the set agreement protocol. These simplexes are used as input to a basic graph agreement protocol, and the output of the graph protocol is returned as the output of the basic agreement protocol. Either S_0 and S_1 are the same, or if they are different, then at least one has dimension greater than k, and so does $S_0 \cup S_1$. In either case, basic agreement has a wait-free read/write protocol.

8.2.2 Link Agreement

Just as for basic agreement, we argue by reduction to a graph problem. Define the ***link graph agreement*** task in terms of the colorized version of $[\mathcal{L}, adj(\mathcal{O}^*), \Lambda]$ as follows. The input complex $\mathcal{L}$ is a graph where each vertex is labeled with a pair $\langle S_0, T_0 \rangle$, where S_0 is an input simplex of dimension at least $n-t$, $T_0 \in \Gamma^*(S_0)$, and $dim(T_0) < dim(S_0)$. There is an edge $(\langle S_0, T_0 \rangle, \langle S_1, T_1 \rangle)$ in $\mathcal{L}$ if S_0 and S_1 are faces of a common simplex in $\mathcal{I}$, and $T_0 \cap T_1$ is non-empty. The relation Λ is given by

$$\Lambda(S_0, T_0) \quad = \quad R_i \in adj(link(T_i, \Gamma^*(S_0)))$$

$$\Lambda(\langle S_0, T_0 \rangle, \langle S_1, T_1 \rangle) \quad = \quad adj(link(T_0 \cap T_1, S_0 \cup S_1)).$$

On a single input, the task chooses a fixed simplex R_i in $link(T_i, \Gamma^*(S_i))$. On two inputs, the task returns at most two intersecting simplexes in $link(T_i \cap T_j, S_i \cup S_j)$. This task is essentially the same as link agreement, except that at most two distinct inputs are permitted.

The k-link connectivity of Γ^* ensures that this task has a wait-free read/write protocol if all inputs are the same, or if

$$dim(link(T_0 \cap T_1, S_0 \cup S_1)) \geq k.$$

We are now ready to present a link agreement protocol. If $k = 1$, then the processes run $(n+1,2)$-set agreement to choose at most two pairs, $\langle S_0, T_0 \rangle$ and $\langle S_1, T_1 \rangle$, which are used as inputs to link graph agreement. Since

$$dim(link(T_0 \cap T_1, \Gamma^*(S_0 \cup S_1))) \geq 1,$$

link graph agreement has a wait-free read/write protocol.

If $k > 1$, the protocol is illustrated in Figure 7. Each process starts by writing its pair to an INPUTS array. The processes invoke a counter protocol, and all processes that receive a number less than k execute a k-consensus protocol to agree on a pair. They use this pair as input to an $(n+1,2)$-set agreement protocol. The remaining processes rescan the INPUTS array, taking the union of all the non-empty S_i, the intersection of all the non-empty T_i, and use the resulting pair as input to the $(n+1,2)$-set agreement protocol. Because more than k processes have invoked the counter protocol, and each such process has a vertex in its T_i but not its S_i, the difference in dimension of the S_i and T_i must be k or greater. Let $\langle S_0, T_0 \rangle$ and $\langle S_1, T_1 \rangle$ be the simplexes returned by the set agreement protocol. These simplexes are used as input to a link graph agreement protocol, whose result is returned. It remains to check that link graph agreement has a protocol. Either $\langle S_0, T_0 \rangle$ and $\langle S_1, T_1 \rangle$ are the same, or if they are different, then the dimensions of $S_0 \cup S_1$ and $T_0 \cap T_1$ differ by at least k. In either case, link graph agreement has a wait-free read/write protocol.

9 Conclusions

An interesting problem is to identify interesting classes of decidable tasks. For example, loop agreement is decidable if and only if $\mathcal{K}$ has a fundamental group with a decidable word problem (e.g,, if $\mathcal{K}$ is a surface). Are there similar characterizations for arbitrary tasks?

References

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873–890, September 1993.

[2] J. Aspnes and M.P. Herlihy. Fast randomized consensus using shared memory. *Journal Of Algorithms*, 11(3):441–460, September 1990.

[3] J. Aspnes, M.P. Herlihy, and N. Shavit. Counting networks. Journal of the ACM, 41(5):1020–1048, September 1994.

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

[5] H. Attiya and S. Rajsbaum. A combinatorial topology framework for wait-free computability. In *Proceedings of the Workshop on Distributed Algorithms and Graphs*, 1996.

[6] A. Bar-Noy and D. Dolev. Shared memory vs. message passing in an asynchronous distributed environment. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing*, pages 371–382, August 1989.

[7] O. Biran, S. Moran, and S. Zaks. A combinatorial characterization of the distributed 1-solvable tasks. *Journal of Algorithms*, 11:420–440, 1990.

[8] O. Biran, S. Moran, and S. Zaks. Deciding 1-solvability of distributed tasks is NP hard. In *proc.* 16^{th} *International Workshop on Graph-Theoretic Concepts in Computer Science, Berlin*, pages 206–220, June 1990.

[9] O. Biran, S. Moran, and S. Zaks. Tight bounds on the round complexity of distributed 1-solvable tasks. *Theoretical Computer Science*, 145:271–290, 1995. Preliminary version appeared in Proc. 4th WDAG, 1990.

[10] E. Borowsky. Capturing the power of resiliency and set consensus in distributed systems. Technical report, University of California Los Angeles, Los Angeles, California, 1995. and personal communications.

[11] 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.

[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*, pages 311–234, August 1990.

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

[14] B. Chor and L. Moscovici. Solvability in asynchronous environments. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1989.

[15] IBM Corporation. *The PowerPC Architecture*. Morgan Kaufman, San Francisco, 1994.

[16] D. Dolev, C. Dwork, and L Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.

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

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

[19] E. Gafni and E. Koutsoupias. Three-processor tasks are undecidable. **daphne.cs.ucla.edu/eli/undec.ps**, 1996.

[20] A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolph, and M. Snir. The NYU Ultracomputer - designing an MIMD parallel computer. *IEEE Transactions on Computers*, C-32(2):175–189, February 1984.

[21] A. Gottlieb and C. P. Kruskal. Coordinating parallel processors: A partial unification. *Computer Architecture News*, 9(6):16–24, [10] 1981.

[22] M. Herlihy and S. Rajsbaum. The decidability of distributed decision tasks. **www.cs.brown.edu/people-/mph/decide.html**.

[23] M.P. Herlihy. Wait-free synchronization. *ACM Transactions On Programming Languages And Systems*, 13(1):123–149, January 1991.

[24] M.P. Herlihy and S. Rajsbaum. On the decidability of distributed decision tasks. Full version of 1996 Herlihy and Rajsbaum PODC paper *op. cit.*

[25] M.P. Herlihy and S. Rajsbaum. Set consensus using arbitrary objects. Full version of 1994 Herlihy and Rajsbaum PODC paper *op. cit.*

[26] M.P. Herlihy and S. Rajsbaum. Set consensus using arbitrary objects. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, August 1994.

[27] M.P. Herlihy and S. Rajsbaum. Algebraic spans. In *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing*, pages 90–99. ACM, August 1995.

[28] 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.

[29] M.P. Herlihy and N. Shavit. A simple constructive computability theorem for wait-free computation. In *Proceedings of the 1994 ACM Symposium on Theory of Computing*, May 1994.

[30] IBM. System/370 principles of operation. Order Number GA22-7000.

[31] P. Jayanti and S. Toueg. Some results on the impossibility, universality, and decidability of consensus. In *Proceedings of the Workshop on Distributed Algorithms and Graphs*, pages 69–84, 1992.

[32] E.H. Jensen, G.W. Hagensen, and J.M. Broughton. A new approach to exclusive data access in shared memory multiprocessors. Technical Report UCRL-97663, Lawrence Livermore National Laboratory, November 1987.

[33] C.F. Miller. Decision problems for groups – survey and reflections. In G. Baumslag and C.F. Miller, editors, *Algorithms and Classification in Combinatorial Group Theory*, volume 23 of *Mathematical Sciences Research Institute Publications*, pages 1–60. Springer-Verlag, New York, 1989.

[34] S. Moran and Y. Wolfstahl. Extended impossibility results for asynchronous complete networks. *Inform. Process. Lett.*, 26:145–151, 1987/88.

[35] J.R. Munkres. *Elements Of Algebraic Topology*. Addison Wesley, Reading MA, 1984. ISBN 0-201-04586-9.

[36] 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.

[37] F. Sergeraert. The computability problem in algebraic topology. *Advances in Mathematics*, 104(1):1–29, March 1994.

[38] R.L. Sites. *Alpha Architecture Reference Manual*. Digital Press, Maynard, MA, 1992.

[39] E.H. Spanier. *Algebraic Topology*. Springer-Verlag, New York, 1966.

[40] J. Stillwell. *Classical Topology and Combinatorial Group Theory*, volume 72 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1980.

[41] G. Taubenfeld, S. Katz, and S. Moran. Initial failures in distributed computations. *International Journal of Parallel Programming*, 18:255–276, 1989.