A Wait-Free Classification of Loop Agreement Tasks

(Extended Abstract)★

Maurice Herlihy[1] and Sergio Rajsbaum[2]

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

Abstract. Loop agreement is a family of wait-free tasks that includes set agreement and approximate agreement tasks. This paper presents a complete classification of loop agreement tasks. Each loop agreement task can be assigned an *algebraic signature* consisting of a finitely-presented group G and a distinguished element g in G. This signature completely characterizes the task's computational power. If $\mathcal{G}$ and $\mathcal{H}$ are loop agreement tasks with respective signatures $\langle G, g\rangle$ and $\langle H, h\rangle$, then $\mathcal{G}$ implements $\mathcal{H}$ if and only if there exists a group homomorphism $\phi : G \to H$ carrying g to h.

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. Examples of tasks include *consensus* [9], *renaming* [2], and *set agreement* [6]. A *protocol* is a program that solves a task. A protocol is *wait-free* if it tolerates failures or delays by n out of $n+1$ processes. One task *implements* another if one can transform (as described below) any protocol for one into a protocol for the other.

Informally, a *classification* of a set of tasks $\mathcal{T}$ is a way of partitioning $\mathcal{T}$ into disjoint *classes* $\mathcal{C}_0, \mathcal{C}_1, \ldots$ so that tasks in the same class share some interesting property. A classification is *complete* if all tasks in a class are computationally equivalent: each task in a class implements any other task in that class. Computational equivalence is an equivalence relation in the ordinary sense.

Loop agreement is a family of tasks that includes set agreement [6] and approximate agreement [8] tasks. This paper presents the first complete classification of loop agreement tasks. Each loop agreement task can be assigned an *algebraic signature* consisting of a finitely-presented group G and a distinguished element g in G. Remarkably, this signature completely characterizes the

★ This work has been supported by a Conacyt-NSF grant.

task's computational power. If task $\mathcal{G}$ has signature $\langle G, g\rangle$ and $\mathcal{H}$ has signature $\langle H, h\rangle$, then $\mathcal{G}$ implements $\mathcal{H}$ if and only if there exists a group homomorphism $\phi : G \to H$ carrying g to h. In short, the discrete, algorithmic problem of determining how to implement one loop agreement task in terms of another reduces to a problem of abstract algebra.

We believe these results are interesting for several reasons. First, loop agreement tasks are interesting in themselves, as they generalize a number of well-known problems (see Section 4) and they have a rich combinatorial structure. They already proved to be essential in obtaining a variety of results concerning the decidability of distributed tasks [15]. Second, the techniques introduced here differ in important ways from earlier approaches to understanding the computational power of wait-free tasks. Much recent work on task composition and robustness has focused on proving that two tasks are inequivalent by constructing specific counterexamples for which equivalence fails. Although the resulting constructions are often ingenious, we would like to replace the search for counterexamples with a more systematic method of analysis. We identify specific (algebraic) properties of tasks that, in a sense, capture their relative power. Evaluating whether one task is stronger, weaker, or incomparable to another reduces to the problem of analyzing certain group homomorphisms. Finally, while most earlier applications of topological techniques focused on impossibility results, we have been able to identify general *sufficient* topological conditions for one task to implement another.

We are hopeful that the techniques introduced here can be used to derive more general classification results. Moreover, because any more general classification scheme must encompass the algebraic structure described here, these results suggest to us that algebraic and topological techniques remain the most promising approach toward classifying tasks in general.

As an application of this classification, we show that loop agreement protocols can be divided into *torsion classes* which also have a nice mathematical structure. This is a coarser partition which still shows how rich the structure of a task classification can be, but is defined in terms of simpler algebraic properties.

2 Related Work

Perhaps the first paper to investigate the solvability of distributed tasks was the landmark 1985 paper of Fischer, Lynch, and Paterson [9] which showed that *consensus*, then considered an abstraction of the database commitment problem, had no 1-resilient message-passing solution. Other tasks that attracted attention include *approximate agreement* [8], *renaming* [2, 14, 16] and *set agreement* [4, 6, 13, 14, 16, 25].

Herlihy and Shavit [16, 17] introduced the use of homology theory to show certain impossibility results for set agreement and renaming. (The set agreement results were shown independently by Borowsky and Gafni [4] and Saks and Zaharoglou [25] using different techniques.) More recently, Herlihy and Rajsbaum used homology theory to derive further impossibility results for set agreement

[12, 13], and to unify a variety of known impossibility results in terms of the theory of chain maps and chain complexes [14].

A novel aspect of the work described here is the extensive use of the fundamental group; that is, homotopy instead of homology theory. Gafni and Koutsoupias [10] were the first to use the fundamental group to analyze distributed tasks, showing that it is undecidable whether certain wait-free read-write tasks have a protocol. Herlihy and Rajsbaum [12] used the fundamental group to prove undecidability results in a variety of other models.

It should be emphasized that our results apply to short-lived *tasks*, not to long-lived objects. Classifying objects raises the complex question of *robustness*: the question whether one can combine "weak" objects to derive "strong" objects. Jayanti [18] was the first to raise the question of robustness with respect to the consensus hierarchy [11]. Since then, a variety of researchers have contributed a variety of ingenious counterexamples showing that different restrictions of the consensus hierarchy are not robust [5, 19, 21, 24, 26]. Neiger et al. [23] and Borowsky et al. [7] claim that the consensus hierarchy is robust for objects having deterministic sequential specifications.

3 Background

3.1 Distributed Computing

A model *of computation* is characterized by a set of $n+1$ processes, a set of objects shared by the processes, and a set of assumptions about timing and failures. Examples of shared objects include message-passing channels, read/write memory, or read/write memory augmented by more powerful synchronization primitives, such as *test-and-set* or *compare-and-swap* variables. All models considered in this paper are *asynchronous*: there is no bound on relative process speed.

A *protocol* is a program that solves a task in a particular model of computation. A protocol is *wait-free* if it tolerates failures by n out of $n+1$ processes. All protocols considered in this paper are wait-free.

An initial (or final) local state of a process is modeled as a *vertex* $\boldsymbol{v} = \langle P, v \rangle$, a pair consisting of a process id P and an input (or output) value v. We speak of the vertex as being *colored* with the process id. A set of $d+1$ mutually compatible initial (or final) local states is modeled as a *d-dimensional simplex* (or d-simplex) $S^d = (\boldsymbol{s}_0, \ldots, \boldsymbol{s}_d)$.

The complete set of possible initial (or final) states of a protocol is represented by a set of simplexes, closed under containment, called a *simplicial complex* (or complex) $\mathcal{K}$. A map $\mu : \mathcal{K} \to \mathcal{L}$ carrying vertexes to vertexes is *simplicial* if it also carries simplexes to simplexes. If vertexes of $\mathcal{K}$ and $\mathcal{L}$ are labeled with process ids, then μ is *color-preserving* if $id(\boldsymbol{v}) = id(\mu(\boldsymbol{v}))$.

A *task* specification for $n+1$ processes is given by an *input complex* $\mathcal{I}$, an *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 $0 \leq m \leq n$. Informally, Δ has the following interpretation: if the $(m+1)$ processes named in S^m start with the designated input values, and only they participate, then each simplex in $\Delta(S^m)$ designates a legal final set of values.

Any protocol has an associated *protocol complex* $\mathcal{P}$, in which each vertex is labeled with a process id and that process's final state (called its *view*). Each simplex thus corresponds to an equivalence class of executions that "look the same" to the processes at its vertexes. The protocol complex corresponding to executions starting from a fixed simplex S^m is denoted $\mathcal{P}(S^m)$. A protocol *solves* a task if there exists a a simplicial *decision map* $\delta : \mathcal{P} \to \mathcal{O}$ such that for each simplex $R^m \in \mathcal{P}(S^m)$, $\delta(R^m) \in \Delta(S^m)$.

Task $\mathcal{L}$ *implements* task $\mathcal{K}$ if one can construct a wait-free protocol for $\mathcal{K}$ by interleaving calls to any number of instances of protocols for $\mathcal{L}$ with operations on any number of read-write registers. In this paper we concentrate on the following, more restricted, implementation notion. An *instance* of $\mathcal{L}$ implements $\mathcal{K}$ if one can construct a protocol for $\mathcal{K}$ by interleaving a call to a single instance of a protocol for $\mathcal{L}$ with any number of operations of read-write registers. Two tasks are *equivalent* if each implements the other, and a task is *universal* for some set of tasks if it implements any task in that set.

3.2 Algebraic Topology

We now informally review a number of undergraduate level notions from algebraic topology; more complete descriptions can be found in any standard topology text (such as [1, 20, 22]).

Let $\mathcal{C}$ be a connected simplicial complex. The union of the simplexes which make up the complex can be regarded as a subset of Euclidean space, called the *polyhedron,* $|\mathcal{C}|$. The *barycentric subdivision* [1, p.125], is inductively obtained by introducing a new vertex at the barycenter (center of mass) of each i-simplex, and then introducing all simplexes of dimension $\leq i$ determined by the additional vertexes. The ℓ*-skeleton* $skel^\ell(\mathcal{C})$ of $\mathcal{C}$ is the subcomplex consisting of simplexes of $\mathcal{C}$ of dimension ℓ or less.

Consider a point $x_0 \in |\mathcal{C}|$. We call x_0 the *base point.* A *loop* α in $\mathcal{C}$ with base point x_0 is a continuous map from the unit interval $I = [0, 1]$ to $|\mathcal{C}|$:

$$\alpha : I \to |\mathcal{C}|$$

such that $\alpha(0) = \alpha(1) = x_0$. A loop is *simple* if $\alpha(t)$ is unique for all $0 \leq t < 1$. Two loops α and β with base point x_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 $h(s, 0) = \alpha(s)$, $h(s, 1) = \beta(s)$, and $h(0, t) = h(1, t) = x_0$ for all $s, t \in [0, 1]$. Homotopy is an equivalence relation. Let $[\alpha]$ denote the equivalence class of loops homotopic to α.

The *fundamental group* [1, p.87] $\pi_1(\mathcal{C}, x_0)$ of $\mathcal{C}$ is defined as follows. The elements of the group are equivalence classes under homotopy of loops with base point x_0. The group operation on these equivalence classes is defined as follows. For loops α and β, $\alpha * \beta$ is the loop obtained by traversing first α and then β:

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

Define $[\alpha] \cdot [\beta] = [\alpha * \beta]$. It is easy to check that this defines a group, with the equivalence class of the constant loop $\alpha(s) = x_0$ as the identity, and for an arbitrary loop α based at x_0, $[\alpha]^{-1} = [\alpha^{-1}]$, where α^{-1} is obtained by traversing α in the opposite direction. The identity element consists of the loops that can be continuously deformed to the point x_0, called *null-homotopic,* or *contractible.*

Since we are assuming that $\mathcal{C}$ is connected, the fundamental group is independent of the base point, so we simply write $\pi_1(\mathcal{C})$. To simplify notation, we sometimes write α instead of $[\alpha]$, relying on context to distinguish between a loop and its equivalence class.

Let $f : |\mathcal{K}| \to |\mathcal{L}|$ be a continuous map. Notice that if α is a loop in $|\mathcal{K}|$ then $f \cdot \alpha$ is a loop in $|\mathcal{L}|$. We can define the *homomorphism induced* by f, $f_* : \pi_1(\mathcal{K}) \to \pi_1(\mathcal{L})$, by $f_*(\langle\alpha\rangle) = \langle f \cdot \alpha\rangle$.

The group $\pi_1(\mathcal{C})$ can also be defined combinatorially, in terms of the *edge loops* of $\mathcal{C}$. An edge loop κ is a sequence of vertices such that each two consecutive vertices span a 1-simplex of $\mathcal{C}$, 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.

4 Loop Agreement

Let $\mathcal{K}$ be a finite (uncolored) 2-dimensional simplicial complex, κ a simple loop of $\mathcal{K}$, and $\boldsymbol{k}_0$, $\boldsymbol{k}_1$, $\boldsymbol{k}_2$ three *distinguished* vertexes in κ. For distinct i, j, and k, let κ_{ij} be the subpath of κ linking $\boldsymbol{k}_i$ to $\boldsymbol{k}_j$ without passing through $\boldsymbol{k}_k$.

Define the $(\mathcal{K}, \kappa)$-*loop agreement* task as follows. Each of $n+1$ processes has an input value in $\{0, 1, 2\}$. Let S^n be an input simplex with input values $\mathit{vals}(S^n)$.

$$\text{If } \mathit{vals}(S^n) = \begin{cases} \{i\} & \text{every process chooses } \boldsymbol{k}_i \\ \{i,j\} & \text{all vertexes chosen span a simplex that lies in } \kappa_{ij} \\ \{0,1,2\} & \text{all vertexes chosen span a simplex in } \mathcal{K}. \end{cases} \tag{1}$$

In other words, the processes converge on a simplex in $\mathcal{K}$. If all processes have the same input vertex, they converge on it (Figure 1). If the processes have only two distinct input vertexes, they converge on some simplex along the path linking them (Figure 2). Finally, if the processes have all three input vertexes, they converge to any simplex of $\mathcal{K}$ (Figure 3).

We now present some examples of interesting loop agreement tasks. Let S^2 be the 2-simplex $(\boldsymbol{s}_0, \boldsymbol{s}_1, \boldsymbol{s}_2)$, and $\mathcal{S}^2$ is the complex constructed from S^2 and its faces.

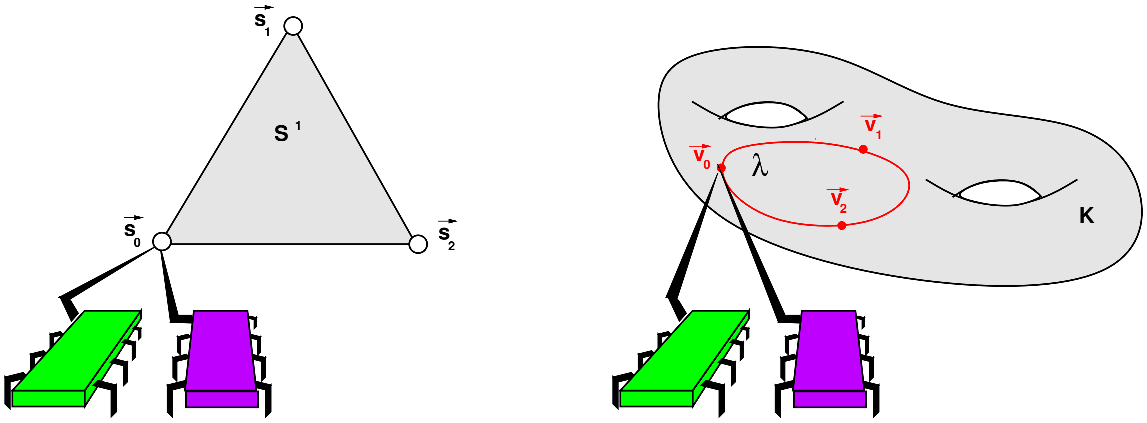

Fig. 1. Loop Agreement with a Single Input

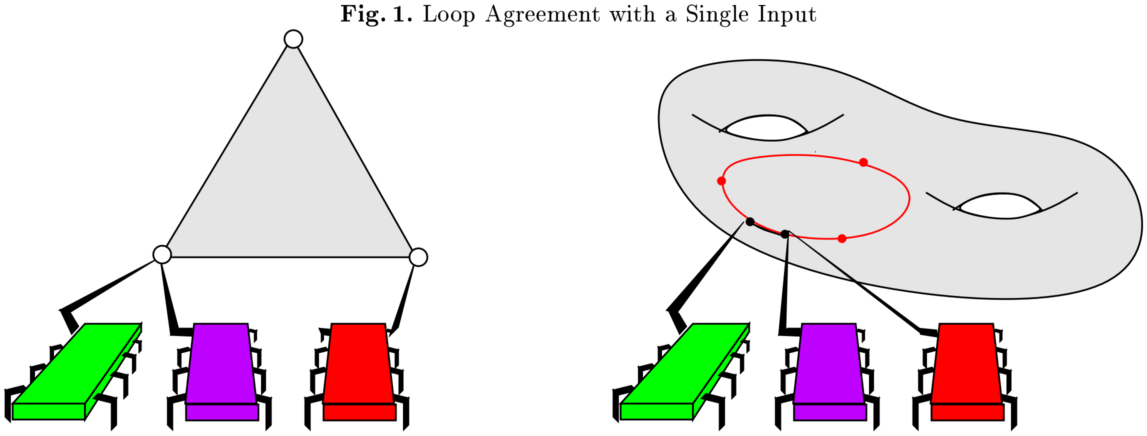

Fig. 2. Loop Agreement with Two Distinct Inputs

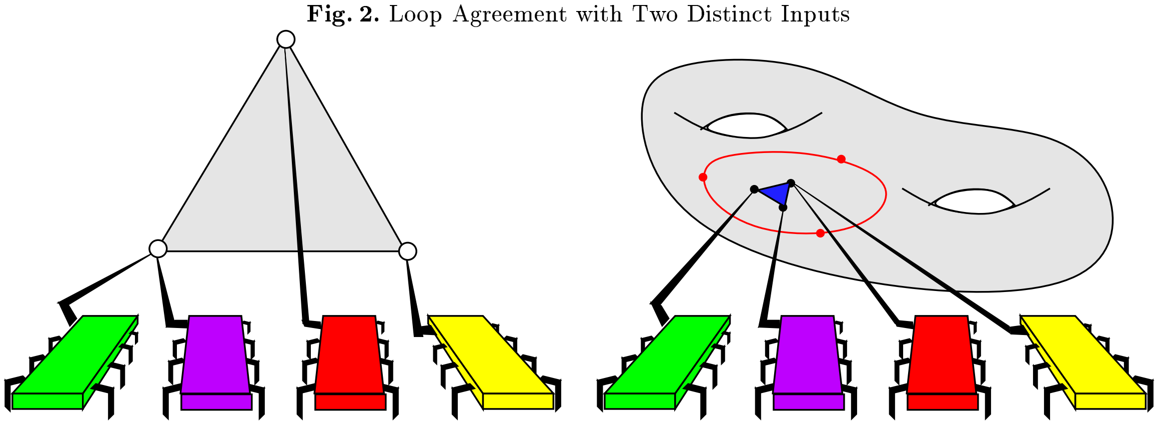

Fig. 3. Loop Agreement with Three Distinct Inputs

- In the $(3,2)$-*set agreement* task [6], 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 $(\boldsymbol{s}_0, \boldsymbol{s}_1), (\boldsymbol{s}_1, \boldsymbol{s}_2), (\boldsymbol{s}_2, \boldsymbol{s}_0)$.
- 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 starting vertexes, then the processes converge on a simplex in $\sigma(S)$. (This task is the uncolored version of simplex agreement [17].) This task is the loop agreement $(\sigma(\mathcal{S}^2), \sigma(\zeta))$, where ζ is the loop described above.
- The 2-dimensional N-*barycentric agreement* task is uncolored simplex agreement for the N-th iterated barycentric subdivision. Notice that 0-barycentric agreement is just the trivial loop agreement task $(\mathcal{S}^2, \zeta)$ where a process with input i can directly decide $\boldsymbol{s}_i$.
- In the 2-dimensional ϵ-*agreement* task [3], input values are vertexes of a simplex S of $\mathcal{S}^2$, and output values are points of $|S|$ that lie within $\epsilon > 0$ of one another in the convex hull of the input values. It is easily seen that this task can be solved by a protocol for N-barycentric agreement, for suitably large N.
- In the 1-dimensional *approximate agreement* task [8] input values are taken from the set $\{0, 1\}$, and output values are real numbers that lie within $\epsilon > 0$ of one another in the convex hull of the input values. It is easily seen that this task can be solved by a 2-dimensional ϵ-agreement protocol.
- A *torus loop agreement* task can be defined by taking a triangulation of a torus, $\mathcal{K}^2$, and a loop κ.
- A *projective plane loop agreement* task can be defined by taking a triangulation of a 2-dimensional disk, and considering the boundary loop. Identify opposite points on the boundary to obtain a projective plane $\mathcal{K}$, and the resulting loop κ.

5 Algebraic Signatures

Consider the $(\mathcal{K}, \kappa)$-loop agreement task, and another loop agreement task $(\mathcal{L}, \lambda)$ with distinguished vertices $\boldsymbol{\ell}_i$. The loop κ represents an element $[\kappa]$ of $\pi_1(\mathcal{K})$.

Definition 1. *The* signature *of a loop agreement task* $(\mathcal{K}, \kappa)$ *is the pair* $(\pi_1(\mathcal{K}), [\kappa])$.

We use the notation

$$\phi : (\pi_1(\mathcal{K}), [\kappa]) \to (\pi_1(\mathcal{L}), [\lambda])$$

to denote a group homomorphism from $\pi_1(\mathcal{K})$ to $\pi_1(\mathcal{L})$ that carries $[\kappa]$ to $[\lambda]$, i.e., with $\phi([\kappa]) = [\lambda]$. Similarly, we use the notation

$$f : (\mathcal{K}, \kappa) \to (\mathcal{L}, \lambda).$$

to indicate a continuous (sometimes simplicial) map from $|\mathcal{K}|$ to $|\mathcal{L}|$ carrying each $\boldsymbol{k}_i$ to $\boldsymbol{\ell}_i$ and each κ_{ij} to λ_{ij}.

We are now ready to present the paper's principal technical result: the ability of one loop agreement task to implement another is completely determined by its algebraic signature.

Theorem 1. *A single instance of* $(\mathcal{K}, \kappa)$ *implements* $(\mathcal{L}, \lambda)$ *if and only if there exists a group homomorphism* $\phi : (\pi_1(\mathcal{K}), [\kappa]) \to (\pi_1(\mathcal{L}), [\lambda])$.

Before discussing the proof, we give a few applications. We start with a known ([4, 16, 25]) result, but which illustrates the power of the theorem.

Proposition 1. $(3, 2)$*-set agreement cannot be implemented wait-free.*

Proof. Recall that (3,2)-set agreement is $(skel^1(\mathcal{S}^2), \zeta)$, where ζ is the loop (s_0, s_1, s_2). It is a standard result that $\pi_1(skel^1(\mathcal{S}^2))$ is infinite cyclic with generator ζ. Implementing $(3, 2)$-set agreement wait-free is the same as implementing it with 0-barycentric agreement $(\mathcal{S}^2, \zeta)$. Because $\pi_1(\mathcal{S}^2)$ is trivial, the only homomorphism $\phi : (\mathcal{S}^2, \zeta) \to (skel^1(\mathcal{S}^2), \zeta)$ is the trivial one that carries ζ to the identity element of the group, and not to ζ.

It is now easy to identify the most powerful and least powerful tasks in this lattice.

Proposition 2. $(3, 2)$*-set agreement is* universal *among loop agreement tasks: it implements any loop agreement task whatsoever.*

Proof. Recall that (3,2)-set agreement is $(skel^1(\mathcal{S}^2), \zeta)$, where ζ is the loop (s_0, s_1, s_2). It is a standard result [1, p.136] that $\pi_1(skel^1(\mathcal{S}^2))$ is the infinite cyclic group $\mathbf{Z}$ with generator ζ. To implement any $(\mathcal{L}, \lambda)$, let $\phi(\zeta) = \lambda$.

Proposition 3. *Uncolored simplex agreement is implemented by any loop agreement task.*

Proof. The complex for uncolored simplex agreement has trivial fundamental group [1, p.96], because it is a subdivided simplex, and hence its poliyhedron is a convex subset of Euclidean space. To implement this task with any $(\mathcal{L}, \lambda)$, let ϕ of every element be 0.

The remainder of this section gives an informal overview of the proof. Recall that a single-instance implementation is allowed to use read/write registers at will. We first check that algebraic signatures are invariant under such compositions. Two algebraic signatures (G, g) and (H, h) are *isomorphic* if there exists an isomorphism between G and H carrying g to h.

Lemma 1. *If* $(\mathcal{K}', \kappa')$ *is a protocol constructed from a single instance of* $(\mathcal{K}, \kappa)$ *and operations on read/write registers, then their signatures are isomorphic.*

It is now easy to obtain the necessary part of Theorem 1. Let $(\mathcal{K}', \kappa')$ be any protocol constructed by composing a protocol for $(\mathcal{K}, \kappa)$ with read/write operations. By Lemma 1, $\pi_1(\mathcal{K}) = \pi_1(\mathcal{K}')$ and $[\kappa] = [\kappa']$. If a single instance of

$(\mathcal{K}, \kappa)$ implements $(\mathcal{L}, \lambda)$, then the decision map $\delta : (\mathcal{K}', \kappa') \to (\mathcal{L}, \lambda)$ induces the desired homomorphism $(\pi_1(\mathcal{K}), [\kappa]) \to (\pi_1(\mathcal{L}), [\lambda])$.

The other direction requires more work. We proceed in three steps. First, we show that the conditions of the theorem suffice to guarantee a *continuous* map

$$f : (\mathcal{K}, \kappa) \to (\mathcal{L}, \lambda).$$

Second, we show how to make this continuous map simplicial. Third, we show how to get the protocol which solves $(\mathcal{L}, \lambda)$.

Assume we have

$$\phi : (\pi_1(\mathcal{K}), [\kappa]) \to (\pi_1(\mathcal{L}), [\lambda]). \tag{2}$$

We need to prove the following lemma.

Lemma 2. *If $\mathcal{K}$ and $\mathcal{L}$ are 2-dimensional simplicial complexes with a homomorphism $\phi : (\pi_1(\mathcal{K}), [\kappa]) \to (\pi_1(\mathcal{L}), [\lambda])$, then there exists a continuous map $f : (|\mathcal{K}|, \kappa) \to (|\mathcal{L}|, \lambda)$.*

We make use of the following standard result:

Lemma 3. *If $\mathcal{K}$ and $\mathcal{L}$ are 2-dimensional simplicial complexes, then every homomorphism $\phi : \pi_1(\mathcal{K}) \to \pi_1(\mathcal{L})$ is induced by some continuous map $f : |\mathcal{K}| \to |\mathcal{L}|$.*

This lemma is not quite enough to prove Lemma 2, because f is constrained only to carry certain equivalence classes of loops to others, while Lemma 2 requires f to carry specific paths and vertexes to others. Nevertheless, in the full paper, we show how to adapt the standard proof of Lemma 3 to prove Lemma 2.

Once we have established the existence of a continuous map $f : (|\mathcal{K}|, \kappa) \to (|\mathcal{L}|, \lambda)$, it remains to be shown that this map can be transformed into a simplicial map. The classical Simplicial Approximation Theorem [1, p.128] guarantees that for sufficiently large $N > 0$, f can be "approximated" by a simplicial map

$$F : (bary^N(\mathcal{K}), \kappa) \to (\mathcal{L}, \lambda).$$

Once we have this map it is a simple matter to obtain a protocol for $(\mathcal{L}, \lambda)$ by running N rounds of barycentric agreement ([15]). This completes the proof of Theorem 1.

6 Application: Torsion Classes

The *torsion number* of $(\mathcal{K}, \kappa)$ is the least positive integer k such that $[\kappa]^k$ is the identity element of $\pi_1(\mathcal{K})$ (related notions appear, for example, in [1, p.178] or [22, p. 22]). If no such k exists, then the order is infinite. Every loop agreement protocol has a well-defined torsion number. Define *torsion class k* to be the tasks with torsion number k.

How much information does a task's torsion number convey? The following properties follow directly from Theorem 1.

– If a task in class k (finite) implements a task in class ℓ, then $\ell|k$ (ℓ divides k).
– Each torsion class includes a *universal* task that solves any loop agreement task in that class.
– The universal task for class k (finite) is also universal for any class ℓ where $\ell|k$.
– The universal task for class k (finite) does not implement any task in class ℓ if ℓ does not divide k.
– The universal task for class ∞ is universal for any class k.

Torsion classes form a coarser partitioning than our complete classification, but they are defined in terms of simpler algebraic properties, and they have an interesting combinatorial structure.

Acknowledgments

We are grateful to Tom Goodwillie of the Brown Mathematics Department for drawing our attention to Lemma 3.

References

1. M.A. Armstrong. *Basic Topology.* Undergraduate Texts In Mathematics. Springer-Verlag, New York, 1983.
2. 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.
3. E. Borowsky and E. Gafni. Consensus as a form of resilience. private communication.
4. 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*, pages 91–100, May 1993.
5. T. Chandra, V. Hadzilacos, P. Jayanti, and S. Toueg. Wait-freedom vs. t-resiliency and the robustness of the wait-free hierarchies. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, pages 334–343, 1994.
6. 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.
7. Y. Afek E. Borowsky, E. Gafni. Consensus power makes (some) sense! In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, pages 363–372, August 1994.
8. A. Fekete. Asymptotically optimal algorithms for approximate agreement. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*, pages 73–87, August 1986.
9. 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.
10. E. Gafni and E. Koutsoupias. Three-processor tasks are undecidable. `http://daphne.cs.ucla.edu/eli/undec.ps`, 1996.
11. M.P. Herlihy. Wait-free synchronization. *ACM Transactions On Programming Languages And Systems*, 13(1):123–149, January 1991.

12. M.P. Herlihy and S. Rajsbaum. Set consensus using arbitrary objects. Full version of 1994 Herlihy and Rajsbaum PODC paper *op. cit.*
13. 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.
14. 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.
15. M.P. Herlihy and S. Rajsbaum. The decidability of distributed decision task. In *Proceedings of the 1997 ACM Symposium on Theory of Computing*, pages 589–598, May 1997. Brief announcement in PODC 1996.
16. 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*, pages 111–120, May 1993.
17. 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*, pages 243–252, May 1994.
18. P. Jayanti. On the robustness of herlihy's hierarchy. In *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing*, pages 145–158, 1993.
19. W-K. Lo and V. Hadzilacos. All of us are smarter than any of us: more on the robustness of the consensus hierarchy. In *Proceedings of the 1997 ACM Symposium on Theory of Computing*, pages 579–588, 1997.
20. W.S. Massey. *Algebraic Topology: An Introduction.* Graduate Texts In Mathematics. Springer-Verlag, New York, 1977.
21. S. Moran and L. Rappoport. On the robustness of h_m^r. In *Proceedings of the 10th International Workshop on Distributed Algorithms*, pages 344–361. ACM, 1996.
22. J.R. Munkres. *Elements Of Algebraic Topology.* Addison Wesley, Reading MA, 1984. ISBN 0-201-04586-9.
23. G.L. Peterson, R. A. Bazzi, and G. Neiger. A gap theorem for consensus types. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, pages 344–353, 1994.
24. O. Rachman. Anomalies in the wait-free hierarchy. In *Proceedings of the 8th International Workshop on Distributed Algorithms*, pages 156–163, 1994.
25. 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*, pages 101–110, May 1993.
26. Eric Schenk. *Computability and Complexity Results for agreement problems in shared memory systems.* PhD thesis, University of Toronto, 1996.