Solvability Characterization for General Three-Process Tasks

Hagit Attiya
Technion
Haifa, Israel
hagit@cs.technion.ac.il

Pierre Fraigniaud
Université Paris Cité and CNRS
Paris, France
pierre.fraigniaud@irif.fr

Ami Paz
LISN - CNRS and Paris Saclay University
Paris, France
ami.paz@lisn.fr

Sergio Rajsbaum
Instituto de Matematicas, U.N.A.M.
Mexico City, Mexico
rajsbaum@im.unam.mx

Abstract

A key result of distributed computing in asynchronous systems is a characterization for the wait-free solvability of *colorless* tasks by the existence of a continuous map from the task's input complex (representing the valid input configurations) to its output complex (representing the valid output configurations) which respects that task's specification. This natural characterization led to many proofs, mainly of impossibility: showing that a colorless task is not wait-free solvable, can be done by proving that there is no continuous map (respecting the task's specification) between two simplicial complexes, which can be done using classical topological machinery.

The seminal work of Herlihy and Shavit (JACM '99) characterized the solvability of general (not necessarily colorless) tasks. However, this characterization is much more involved than the colorless one, as it uses the new notions of chromatic subdivisions and color-preserving maps. The characterization asks whether there exists a chromatic subdivision of the input complex and a color-preserving map from the resulting subdivided complex to the output complex. This characterization is much harder to check as there are no ready-made topological tools for it, and in fact, finding such a subdivision and map is related to finding an algorithm for the task.

This work presents a new and simpler characterization for the solvability of general tasks with three processes. In our characterization, the output complex undergoes a bounded number of simple combinatorial transformations. Then, we check if there is a continuous map from the input complex to the resulting output complex; we show that this suffices for determining whether the original task is solvable. Our approach provides a new and more direct way for deciding the solvability of a task, and also for proving impossibility and possibility of wait-free solutions for general tasks.

CCS Concepts

• **Theory of computation → Concurrent algorithms**; **Distributed computing models**.

PODC '25, Huatulco, Mexico

ACM ISBN 979-8-4007-1885-4/25/06
https://doi.org/10.1145/3732772.3733548

Keywords

topology, chromatic tasks, read / write shared memory, wait-free

ACM Reference Format:
Hagit Attiya, Pierre Fraigniaud, Ami Paz, and Sergio Rajsbaum. 2025. Solvability Characterization for General Three-Process Tasks. In *ACM Symposium on Principles of Distributed Computing (PODC '25), June 16–20, 2025, Huatulco, Mexico.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3732772.3733548

1 Introduction

For more than thirty years, distributed computing has been studied through the lens of topology, developing a deep understanding of the *solvability of tasks*. In this approach, a *simplex* represents a configuration as a set of vertices, each representing the state of one process. In general, each vertex has an associated process *id*, sometimes referred to as its *color*. *Tasks* are triples $(\mathcal{I}, \mathcal{O}, \Delta)$, where $\mathcal{I}$ and $\mathcal{O}$ are *simplicial complexes* modelling the inputs and outputs of the task, and Δ is a relation specifying the possible valid outputs, $\Delta(\sigma)$, for each input simplex $\sigma \in \mathcal{I}$. For any initial configuration σ of $\mathcal{I}$, each process starts with an input vertex of σ colored by its id, and must decide on an output vertex with its color, respecting Δ, that is, such that the vertices decided by the processes form a simplex τ of $\Delta(\sigma)$.

Tasks are *colorless* [9, 19], if they can be defined only in terms of input and output values, regardless of the number of processes involved, and regardless of which process has a particular input or output value; accordingly, the simplices of $\mathcal{I}$ and $\mathcal{O}$ consist of sets of values, without process ids. Well-known examples are the consensus task [12] and its generalization to set consensus [10].

In general, not all tasks can be stated as colorless; such tasks are called *chromatic* or *colored*. Several such tasks have been studied, notably *renaming* [3]. A simple example is the *majority consensus* task (see Figure 1): three processes start with binary input values, and each must decide on a value that appeared as an input; additionally, if all three processes participate then they must either decide on the same value, or more processes must decide 0 than 1.

A major contribution of the topological approach is a set of novel impossibility results and algorithms for specific tasks, through fundamental characterizations of solvable tasks in various distributed computing models. Beyond telling us what is solvable and what is not, characterization results tell us *what makes tasks solvable or not*, indicating the obstructions to solvability, and sometimes indicating how these obstructions can be avoided.

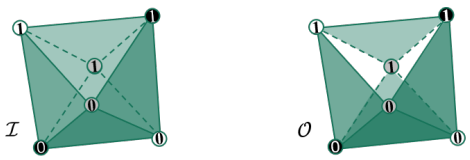

Figure 1: The majority consensus task. Gray levels represent process ids, and their inputs and outputs are written inside the vertices.

1.1 The challenge: Wait-free solvability of chromatic tasks

The focus of this work is on *wait-free* protocols for solving a task in a read/write shared memory system.

A cornerstone result for this model is the *asynchronous computability theorem* (ACT) of Herlihy and Shavit [22] (see also [18, Theorem 11.2.1]). The key insight of the ACT theorem is that a protocol solving a task in this model closely corresponds to a color-preserving mapping from a chromatic subdivision of the input complex into allowable outputs in the output complex. This requires checking, *for an arbitrarily large r,* whether there is a *mapping from the complex that results from r repeated chromatic subdivisions of the input complex to the output complex* that is color-preserving and respects Δ. Thus, the ACT does not provide us with easy-to-use techniques that work solely by relating the topology of the input and the output complexes.

The usefulness of such results is demonstrated by the characterization known for *colorless* tasks, which are much simpler to analyze. There is an elegant solvability characterization for colorless tasks only in terms of their input and output complexes [18, 21]: *a colorless task is wait-free solvable if and only if there is a continuous map from $|\mathcal{I}|$ to $|\mathcal{O}|$ respecting Δ* (where $|K|$ is the geometric realization of a simplicial complex).

Researchers have tried to find a solvability characterization of chromatic tasks in terms of continuous maps, analogous to the colorless characterization. The value of such a characterization would not only be due to its direct nature from the input complex to the output complex, but also due to the direct connection to topology: continuous maps between spaces.

The quest for such a characterization has failed, because *it is not true* that a chromatic task is solvable if and only if there is a continuous map from the input to the output complex. This has been demonstrated with the *hourglass* task, depicted in Figure 2. This task has a single input configuration. Each process running solo decides on value 0. If process P_0 (black) runs concurrently with either P_1 or P_2, they can also (in addition to their solo values) decide on their respective vertices, with output 1. While P_1 and P_2 running concurrently can additionally decide their respective vertices with value 2. When all three run concurrently, any triangle is a valid output simplex. As discussed in [18, Section 11.1], despite there being a continuous map from the input to the output complex for the hourglass task, the hourglass task is nevertheless unsolvable. The same holds for the majority consensus task.

1.2 Our results and techniques

This paper proves a *necessary and sufficient* condition for the solvability of a *three-process* task, $(\mathcal{I}, \mathcal{O}, \Delta)$. Rather then relating the input complex $\mathcal{I}$ to the output complex $\mathcal{O}$, which is impossible (as demonstrated by the hourglass task), our condition relates the input complex to an output complex $\mathcal{O}'$, effectively derived from $\mathcal{O}$. The condition applies to all tasks, but its main novelty is in showing the critical role of a new type of obstructions for chromatic tasks. (See Section 5.)

Our condition goes through identifying a notion of a *local articulation point* in the output complex: a vertex whose neighborhood, its *link* in the topological parlance, is disconnected. The right-hand side of Figure 2 depicts the link of a vertex in the hourglass task, which is a graph consisting of two connected components: the output edges compatible with the vertex of P_1 deciding 1.

We show a novel method for dealing with each local articulation point by *splitting* the output complex around it, eventually creating a *link-connected* output complex $\mathcal{O}'$. See the center-right side of Figure 2, for splitting the hourglass task.

To deal with multiple input configurations, we ensure that all tasks are in a *canonical* form; this means that each output vertex is the image, by Δ, of a unique input vertex. To this end, we require each process to output its input in addition to the output it decides on (see Section 3.) This is essential since the splitting of an output vertex crucially depends on its *unique pre-image* under Δ.

Consecutive splitting operations yield a task T', with the same input complex $\mathcal{I}$ as the original task and an adjusted relation Δ' between $\mathcal{I}$ and $\mathcal{O}'$. We show (Section 4) that the solvability of the original general task is equivalent to the solvability of T'. This results in showing T is wait-free solvable if and only if there is a continuous map from the geometric realization of $\mathcal{I}$ to that of $\mathcal{O}'$, which respects Δ'.

As already mentioned, continuous maps of these kind are known to imply protocols for colorless tasks, but we have to adapt them to deal with chromatic tasks. Specifically, we first employ a colorless protocol as a *color-agnostic* solution to the chromatic task: a protocol that handles vertices with colors (id), but is not *color-preserving*, namely, a *process might decide on a vertex with a non-matching color.* We then present a concrete algorithm that allows three processes to turn such a solution into a properly colored one, where each process decides on an output vertex with its id, all on the same output simplex. Similar protocols have appeared in earlier work (e.g., [13, 22, 27]), but they were, at least partially, topological in nature (see Section 5.2). This yields implicit algorithms that are harder to understand, while our algorithm relies on standard synchronization techniques. Section 6 discusses several examples, including majority consensus and the hourglass task.

1.3 Additional related work

Link connectivity has showed up in all previous papers about chromatic tasks, since the work of Herlihy and Shavit [22]. However, our paper is the first to identify the precise role of link connectivity in a necessary *and sufficient* characterization of solvable chromatic tasks, *for three processes.*

The case of three processes has played an important role in past research, because it is the smallest dimension where topological

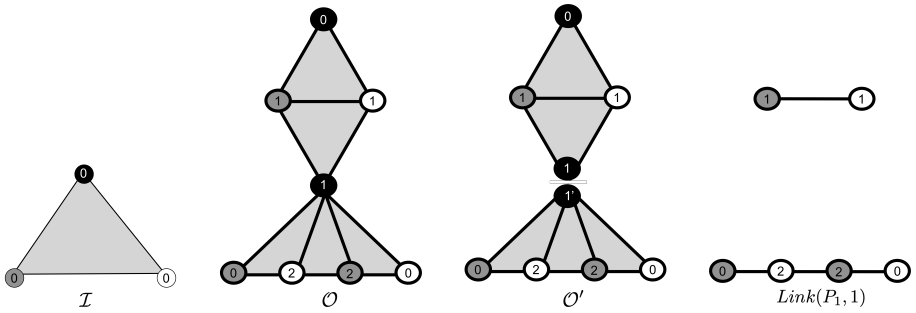

Figure 2: The hourglass task: input complex (left), output complex (center left), output complex after splitting (center right) and link (right).

properties beyond graph connectivity appear. In this case, two types of obstructions—local articulation points and contractibility—are neatly identifiable. In smaller dimensions, e.g., for two-process tasks, if the output complex has a local articulation point then it is also not connected (in the graph-theoretic sense), and hence, it is not solvable. On the other hand, in dimensions higher than 2, i.e., with four or more processes, a disconnected link may be connected (in the graph-theoretic sense).

Note that undecidability results [13, 20] are essentially proved in the three-process case, using *loop agreement* task [19, 20, 25]. A loop agreement task is defined by a given two-dimensional output complex O, together with a loop in it. Roughly, each process starts on one of *three* distinguished vertices of the loop; if they start on the same vertex, they decide on this vertex; if they start on two distinct distinguished vertices, then they decide on vertices belonging to the same edge along the path linking their starting vertices; finally, if they start on all three distinguished vertices, then they can decide on vertices belonging to any simplex of O. While a loop agreement task can be solved for any number of processes, because it is colorless, its essential behavior is captured by considering *only three processes*, each starting on one of the distinguished vertices. In fact, some results for loop agreement are known to have no generalization for more than three processes [19, Theorem 5.4] .

Our approach is different from the one used in [13], where the undecidability of three-process tasks was proved by a reduction from the *contractibility problem* to the task-solvability problem. The contractibility problem asks whether a given loop of a simplicial complex can be continuously transformed into a point, a problem which is known to be undecidable even for 2-dimensional simplicial complexes (see, e.g., [28]). Gafni and Koutsoupias prove the reduction by showing that contractibility is undecidable for the special case of chromatic complexes and loops of length 3. To do so, they first show that the contractibility problem is undecidable for link-connected two-dimensional complexes. We instead transform the output complex to be link-connected, and can then argue directly about colorless tasks.

There are two previous approximations to continuous characterization. First, when assuming link connectivity, there are sufficiency results for general tasks ([27], [14], and [18, Section 11.5]), without a matching necessary condition. Another related notion is that of a *continuous task* [16], which has an input/output specification that is a continuous function between the geometric realizations of the input and output complex. The characterization is that a task is solvable if and only if there exists a *chromatic function* (a notion introduced in [16]) from the input complex $\mathcal{I}$ to the output complex O respecting the task specification. Our characterization is in contrast more explicit about the obstructions (since it exposes the role of local articulation points), and establishes a direct connection with colorless solvability: after removing articulation points, chromatic and colorless solvability are the same.

Our splitting deformation draws upon work on modeling of real-world objects, used in computer-aided design (CAD) [23]. There is a long line of papers studying splitting deformations mostly of two and three-dimensional simplicial complexes, because these are the dimensions of most graphics applications (and for technical reasons, as discussed in, e.g. [7]), although there is also work on higher dimensional complexes. The same splitting we do has been used (e.g. [11, Fig.1]), but not to fix a disconnected link; instead, the interest has been in doing additional splittings, even of edges, because the goal in this research line is to decompose a non-manifold complex into an assembly of manifolds, or at least into components that belong to simpler, well-understood class of complexes for which efficient data structures are known.

2 Preliminaries

We consider the standard *read-write shared memory* model of distributed computing [5], and adopt a standard framework for modeling distributed computation through algebraic topology [18].

2.1 Read-write asynchronous shared-memory model

We consider distributed systems with $n \geq 2$ processes, labeled by distinct integers from 0 to $n-1$. Every process i initially knows its identity i, called *identifier* (id), as well as the total number n of processes in the system.

A *process* is a deterministic (infinite) state machine. Processes exchange information through a shared memory. The system is *asynchronous*, and all interleavings of the atomic operations of processes are possible. We do not limit the computational power of the processes, the size of their private memories or the size of the registers in the shared memory.

The shared memory has n distinct atomic single-writer multiple-reader *registers* $R[0], \ldots, R[n-1]$. Each process i can store data

in the shared memory by *writing* in its register $R[i]$, and no other processes can write in $R[i]$. On the other hand, for every $i \in [n]$, $R[i]$ can be *read* by any process j.

A *collect* by process i results in this process reading all registers $R[j]$, $j = 0, \ldots, n-1$, in arbitrary sequential order. While read and write operations are atomic, a collect operation is not. A stronger variant is a *scan* operation [1], which is an atomic collect, in which all registers are read simultaneously at once, in an atomic manner. An even stronger variant is an *immediate snapshot* [8], in which the write-snapshot sequence of operations is itself atomic, that is, not only all registers are read simultaneously at once, but the scan occurs "immediately" after the write operation.

In parts of this work, we assume processes communicate by immediate snapshots on a single array of read/write registers, without loss of generality (see, e.g. [15]). The reason is that the executions of an algorithm solving a task are represented by a simplicial complex that is a chromatic subdivision, as discussed next.

2.2 Elements of algebraic topology

A (simplicial) *complex* is a collection $\mathcal{K}$ of non-empty sets, closed under inclusion, i.e., if $\sigma \in \mathcal{K}$ then, for every non-empty set $\sigma' \subseteq \sigma$, $\sigma' \in \mathcal{K}$. Every set in $\mathcal{K}$ is called a *simplex*. A subset of a simplex is called a *face*, and a *facet* of $\mathcal{K}$ is a face that is maximal under inclusion in $\mathcal{K}$. The *dimension* of a simplex σ is $|\sigma| - 1$, where $|\sigma|$ denotes the cardinality of σ. The dimension of a complex is the maximal dimension of its facets. A complex in which all facets are of the same dimension is called *pure*. The *vertices* of $\mathcal{K}$ are all simplices with a single element (i.e., of dimension 0). The set of vertices of a complex $\mathcal{K}$ are denoted by $V(\mathcal{K})$. As is common in literature, we do not always distinguish a vertex v from the singleton set $\{v\}$ containing it.

Recall that $|K|$ denotes the geometric realization of a simplicial complex K. Definitions and discussion about the geometrical and topological view of simplicial complexes appear in [18, Sections 3.2.2 and 3.2.3].

In a *chromatic* complex, every vertex is a pair $v = (i, x)$ where $i \in [n] = \{1, \ldots, n\}$ for some $n \geq 1$ is the *color* of v (used to denote a process id), and x is some value (e.g., an input value, an output value, or a value representing the data acquired after some computation). Moreover, in a chromatic complex, a "color" i appears at most once in every simplex, that is, a simplex is of the form $\sigma = \{(i, v_i) : i \in I\}$ for some non-empty set $I \subseteq [n]$. For such simplex σ, $\mathrm{id}(\sigma)$ is the set of colors in σ, i.e., $\mathrm{id}(\sigma) = I$.

A *simplicial map* from a complex $\mathcal{K}$ to a complex $\mathcal{K}'$ is a map $f : V(\mathcal{K}) \to V(\mathcal{K}')$ preserving simplices, i.e., for every simplex σ, the set $f(\sigma) = \{f(v) : v \in \sigma\}$ is a simplex of $\mathcal{K}'$. A *chromatic* simplicial map preserves the colors of the vertices, i.e., for every $(i, x) \in V(\mathcal{K})$, $f(i, x) = (i, y) \in V(\mathcal{K}')$; note that $\mathrm{id}(f(\sigma)) = \mathrm{id}(\sigma)$.

A *carrier map* from a complex $\mathcal{K}$ to a complex $\mathcal{K}'$ is a map $\Delta : \mathcal{K} \to 2^{\mathcal{K}'}$ that maps every simplex $\sigma \in \mathcal{K}$ to a pure subcomplex of $\mathcal{K}'$ with the same dimension; moreover, it must be *monotonic*, that is, for every $\sigma' \subseteq \sigma$, $\Delta(\sigma') \subseteq \Delta(\sigma)$, i.e., $\Delta(\sigma')$ is a subcomplex of $\Delta(\sigma)$. When the complexes are chromatic, we only consider *chromatic* carrier maps, i.e., for every $\sigma \in \mathcal{K}$, $\Delta(\sigma)$ has the same colors as σ. Specifically, for a single vertex $(i, x) \in \mathcal{K}$, all vertices in $\Delta(i, x)$ have color i.

For a vertex $v \in V(\mathcal{K})$, the *link of v in $\mathcal{K}$*, denoted $\mathrm{lk}_{\mathcal{K}}(v)$, is a simplicial complex defined as

$$\mathrm{lk}_{\mathcal{K}}(v) = \{\sigma \subseteq V(\mathcal{K}) \setminus \{v\} : (\sigma \cup \{v\}) \in \mathcal{K}\} .$$

We focus on 2-dimensional complexes, in which case, $\mathrm{lk}_{\mathcal{K}}(v)$ is a 1-dimensional complex, i.e., a graph. We say that $\mathcal{K}$ is *link connected* if $\mathrm{lk}_{\mathcal{K}}(v)$ is a connected graph, for every vertex $v \in V(k)$.

2.3 Tasks

A *task* for n processes is a triple $(\mathcal{I}, \mathcal{O}, \Delta)$ where $\mathcal{I}$ and $\mathcal{O}$ are $(n-1)$-dimensional complexes, respectively called *input* and *output* complexes, and $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$ is an input-output specification. Every simplex $\sigma = \{(i, x_i) : i \in I\}$ of $\mathcal{I}$ defines a legal input state corresponding to the scenario in which, for every $i \in I$, process i starts with input value x_i. Similarly, every simplex $\tau = \{(i, y_i) : i \in I\}$ of $\mathcal{O}$ defines a legal output state corresponding to the scenario in which, for every $i \in I$, process i outputs the value y_i. The map Δ is an input-output relation specifying, for every input state $\sigma \in \mathcal{I}$, the set of output states $\tau \in \mathcal{O}$ with $\mathrm{id}(\tau) = \mathrm{id}(\sigma)$ that are legal with respect to σ. That is, assuming that only the processes in $\mathrm{id}(\sigma)$ participate in the computation (the set of participating processes is not known a priori to the processes in σ), the processes are allowed to output any simplex $\tau \in \Delta(\sigma)$. $\Delta(\sigma)$ can be viewed as a collection of simplices, each with the same set of ids as σ, or, alternatively, as the complex induced by this set of simplices and containing also all their faces. Recall that we assume that the carrier map Δ is *monotonic*, that is, for every $\sigma, \sigma' \in \mathcal{I}$, if $\sigma' \subseteq \sigma$ then $\Delta(\sigma') \subseteq \Delta(\sigma)$ as subcomplexes. If Δ was not monotonic, then some output vertices would have never been decided by an algorithm; such values can be omitted from Δ. Fig. 3 present a simple task, which is used as a running example below.

2.4 Task Solvability

Given a simplex $\sigma = \{(i, x_i) : i \in I\} \in \mathcal{I}$ for some $I = \mathrm{id}(\sigma) \subseteq [n]$, we can consider what happens after some interleaving of a write and a collect operation by processes in I. Such a simplex is of the form $\tau = \{(i, V_i) : i \in I\}$, where $V_i = \{(j, x_j) : j \in J_i\}$ is the *view* of process i. These simplices induce a complex $\mathcal{P}^{(1)}(\sigma)$. In the case of immediate snapshots assumed in this paper, $\mathcal{P}^{(1)}(\sigma)$ is a chromatic subdivision of σ [22].

In general, it is known that a full-information protocol, where processes communicate by immediate snapshots, remembering all the past and repeatedly writing it in each immediate snapshot, a finite number of times, the views at the end form a chromatic subdivision of the input complex, denoted $\mathcal{P}$, the *protocol complex*. Furthermore, there is a structure of a monotonic carrier map from $\mathcal{I}$ to $\mathcal{P}$, which identifies the subcomplexes of executions starting with each input simplex σ of $\mathcal{I}$. Overloading notation, we use $\mathcal{P}$ also as a carrier map, denoting by $\mathcal{P}(\sigma)$ the subcomplex of $\mathcal{P}$ of all executions starting with σ. It is also well-known that $\mathcal{P}(\sigma)$ is a chromatic subdivision of σ.

Finally, a task $\Pi = (\mathcal{I}, \mathcal{O}, \Delta)$ is solvable if and only if there exists a simplicial chromatic map $f : \mathcal{P} \to \mathcal{O}$ from the protocol complex to the output complex that is *carried by* Δ, i.e., for every $\sigma \in \mathcal{I}$, $f(\mathcal{P}(\sigma)) \subseteq \Delta(\sigma)$, informally called *respecting* Δ. The mapping f is defined by the vertices, i.e., for $\sigma = \{(i, V_i) : i \in I\}$, and corresponds

Figure 3: A simple task with its relation Δ specified for some input simplices. Vertices' colors represent their ids, and their shape represent their Δ. The green facet in O is in $\Delta(\sigma)$ and in $\Delta(\sigma')$; its black vertex is similarly in Δ of both black vertices of $\mathcal{I}$.

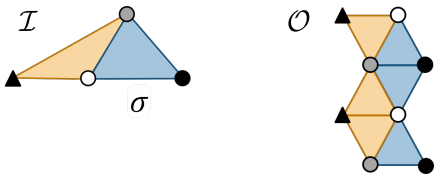

Figure 4: Canonical task corresponding to the task of Fig. 3.

to the decisions of output values taken by the processes individually, based on their views at the end of an execution. But the map f extends from vertices to simplices, being a simplicial map, and we have $f(\sigma) = \{f(i, V_i) : i \in I\}$ is a simplex of O.

3 Canonical tasks

As a first step, we show how to transform a task $T = (\mathcal{I}, O, \Delta)$ to a task $T^* = (\mathcal{I}, O^*, \Delta^*)$, in a way that facilitates analyzing it as if it was *inputless*, that is, as if its input complex contains a single facet, following [17]. Intuitively, facets of the input complex of T^* can be considered independently of each other, because Δ^* is "one-to-one", that is, for two facets σ_1, σ_2, no facet is in $\Delta^*(\sigma_1) \cap \Delta^*(\sigma_2)$. The intersection $\Delta^*(\sigma_1) \cap \Delta^*(\sigma_2)$ may be non-empty if $\sigma_1 \cap \sigma_2$ is non-empty, in which case it would contain simplices of size $|\sigma_1 \cap \sigma_2|$. This result holds for a task T defined for any number n of processes.

At a high level, T^* is the task that results from T by requiring each process to output its input in addition to the output it produces for T. We define O^* formally as a product of simplicial complexes, as follows (see Fig. 4):

Given two pure chromatic simplicial complexes C and T of dimension $n-1$, with vertices $\mathcal{V}(C), \mathcal{V}(T)$ colored with n colors by χ (each facet is colored with n distinct ids in our applications), the *product* $C \times T$ is the following pure chromatic simplicial complex of dimension $n-1$. Its vertices are of the form (u, v) with $u \in \mathcal{V}(C)$ and $v \in \mathcal{V}(T)$ such that $\chi(u) = \chi(v)$; the color of (u, v) is $\chi((u, v)) = \chi(u) = \chi(v)$. Its simplices are of the form $X \times Y = \{(u_0, v_0), \ldots, (u_k, v_k)\}$ where $X = \{u_0, \ldots, u_k\} \in C$, $Y = \{v_0, \ldots, v_k\} \in T$ and $\chi(u_i) = \chi(v_i)$.

For a task $T = (\mathcal{I}, O, \Delta)$ whose complexes $\mathcal{I}, O$ are colored with process ids, let O^* be the sub-complex of the Cartesian product $\mathcal{I} \times O$ induced by all $X \times Y$ such that $Y \in \Delta(X)$. The task T^* is then $(\mathcal{I}, O^*, \Delta^*)$, where $\Delta^*(X)$ contains all the simplices $Y^* = X \times Y$, for $Y \in \Delta(X)$. Intuitively, an output simplex $Y \in \Delta(X)$ is replaced by a pair $X \times Y$, i.e., each process $y \in Y$ also outputs its input value in X; this is done both in Δ and in O.

Theorem 3.1. *A task $T = (\mathcal{I}, O, \Delta)$ is solvable iff its canonical form $T^* = (\mathcal{I}, O^*, \Delta^*)$ is solvable.*

Proof. If T is solvable, then there is a simplicial map δ from a protocol complex $\mathcal{P}$ starting in $\mathcal{I}$ to O. Each process can add its input value to its decision, thus defining a simplicial map δ^* from $\mathcal{I}$ to O^*. This indeed defines a simplicial map that solves T: consider an input simplex $X \in \mathcal{I}$ and a simplex $\tau \in \mathcal{P}(X)$. We have that $\delta(\tau) \in O$ is a simplex satisfying $\delta(\tau) \in \Delta(X)$ since δ solves T. Hence, $\delta^*(\tau) \in O^*$ and also $\delta^*(\tau) \in \Delta^*(X)$, by the mare definitions of δ^*, O^* and Δ^*.

Conversely, if T^* is solvable by a simplicial map δ^*, then projecting out the input values gives a simplicial map δ solving T. □

4 Resolution of Local Articulation Points

For an input facet $\sigma \in \mathcal{I}$, a vertex $y \in \Delta(\sigma)$ is called a *local articulation point w.r.t. σ* (*LAP*) if its link $\mathrm{lk}_{\Delta(\sigma)}(y)$ has at least two connected components. Here, $\mathrm{lk}_{\Delta(\sigma)}(y)$ is the link of y in the complex $\Delta(\sigma) \subseteq O$. Fig. 5 depicts a disconnected link of a vertex $y \in \Delta(\sigma)$.

This section presents our first technical contribution, a topological deformation that removes local articulation points from the output complex of a task. The deformation splits each such point and re-defines the input-output relation, which results in a new task with the same solvability as the original task.

To address the challenge of dealing with multiple input facets, we henceforth assume that all the tasks are canonical. We further assume that Δ is a carrier map, and that all of O is reachable, i.e. $O = \cup_{\sigma \in \mathcal{I}} \Delta(\sigma)$ (other simplices can clearly be omitted from O).

4.1 The Splitting Deformation

Let $T = (\mathcal{I}, O, \Delta)$ be a canonical task. Consider a facet $\sigma \in \mathcal{I}$ and a vertex $y \in \Delta(\sigma)$, such that y is a local articulation point w.r.t. σ, i.e., $\mathrm{lk}_{\Delta(\sigma)}(y)$ is a graph with $r \geq 2$ connected components $C_1, \ldots, C_r$. We define the *splitting deformation of O w.r.t. y and σ* by replacing O with a new output complex $O_{\sigma,y}$, and simultaneously redefine the carrier map Δ into a new carrier map $\Delta_{\sigma,y} : \mathcal{I} \to 2^{O_{\sigma,y}}$. For brevity, we henceforth omit σ from the subscript, and denote the newly defined task by $T_y = (\mathcal{I}, O_y, \Delta_y)$; we now detail the construction of O_y and Δ_y (see Fig. 5).

For the vertices, omit y from O and replace it by r vertices $y_1, \ldots, y_r$ in O_y, where $\mathrm{id}(y_i) = \mathrm{id}(y)$ for all i. For each $\tau \in \mathcal{I}$, the vertices of the complex $\Delta_y(\tau)$ will be implicitly defined in the following, by including in $V(\Delta_y(\tau))$ every vertex which belongs to some face of $\Delta_y(\tau)$. For each simplex $\tau \in \mathcal{I}$, of any dimension, if $\rho \in \Delta(\tau)$ satisfies $y \notin \rho$ then we add ρ to $\Delta_y(\tau)$.

Note that Δ is a carrier map, so any face in O must be contained in some facet in the image of Δ. Thus, we henceforth define Δ_y by considering facets in the image of Δ.

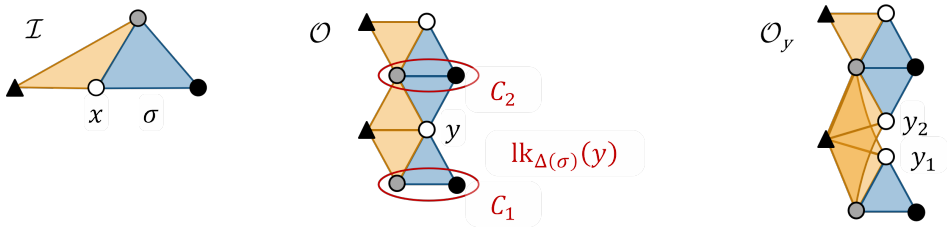

Figure 5: A link $\mathrm{lk}_{\Delta(\sigma)}(y)$ with two connected components, and after splitting w.r.t. y and σ

For a facet $\{z, z', y\} \in \Delta(\sigma)$, we must have that $z, z' \in C_i$ for some i. Add the facet $\{z, z', y_i\}$ and all its faces to O_y and to $\Delta_y(\sigma)$. We extend this to the faces of σ in a natural way, as follows. For each edge $e \subseteq \sigma$, if $\{z, y\} \in \Delta(e)$ then add $\{z, y_i\}$ and all its faces to $\Delta_y(\sigma)$; act analogously if $\{z', y\} \in \Delta(e)$. For each vertex $x \in \sigma$, if $\{y\} \in \Delta(x)$ then add $\{y_i\}$ to $\Delta_y(x)$. Note that the above guarantees that Δ_y is a carrier map, at least for σ and its faces.

Consider a facet $\{z, z', y\} \in \Delta(\sigma')$ in the image $\Delta(\sigma')$ of another facet $\sigma' \in \mathcal{I}$, $\sigma' \neq \sigma$. As the task is canonical, we have $\{z, z', y\} \notin \Delta(\sigma)$. Add all the facets $\{z, z', y_i\}$ to $\Delta_y(\sigma')$, for all i, as well as all their faces. For each edge $e \subseteq \sigma'$, if $\{z, y\} \in \Delta(e)$ then add all the edges $\{z, y_i\}$ and all their faces to $\Delta_y(\sigma')$; act analogously if $\{z', y\} \in \Delta(e)$. For each vertex $x \in \sigma'$, if $\{y\} \in \Delta(x)$ then add all of $\{y_i\}$ to $\Delta_y(x)$. The above guarantees, again, that Δ_y is a carrier map, at least as far as σ' and its faces are concerned.

Claim 1. *If T is a canonical task then so is T_y.*

Proof. Let $x \in \mathcal{I}$ be the unique input vertex s.t. $(x, y) \in \Delta$. The input-output relation Δ on any input vertex other than x does not change in Δ_y. For x, we have $(x, y_i) \in \Delta_y$ for some set of indices i, but no other vertex $x' \in \mathcal{I}$ satisfies $(x', y_i) \in \Delta_y$, for any i. □

Lemma 4.1. *The number of local articulation points in O_y w.r.t. σ is strictly smaller than in O.*
Moreover, if O contains no articulation point w.r.t. some facet $\sigma' \in \mathcal{I}$, then O_y also contains no articulation point w.r.t. σ'.

Proof. Focus first on σ. The LAP y is omitted from $\Delta(\sigma)$, while for each y_i its link $\mathrm{lk}_{\Delta(\sigma)}(y_i) = C_i$ is connected in $\Delta(\sigma)$. We now show that if a vertex $z \in \Delta(\sigma)$ is not no a LAP w.r.t. σ, it does not become such. For a vertex $z \in \Delta(\sigma)$ such that $z \notin \mathrm{lk}_{\Delta(\sigma)}(y)$, we have that $y \notin \mathrm{lk}_{\Delta(\sigma)}(z)$, hence $\mathrm{lk}_{\Delta(\sigma)}(z)$ in O is the same as $\mathrm{lk}_{\Delta_y(\sigma)}(z)$ in O_y.

Consider a vertex $z \in \mathrm{lk}_{\Delta(\sigma)}(y)$ which is not a LAP w.r.t. σ; we show it does not became a LAP by the transformation. Note that the edge $\{z, y\}$ in $\Delta(\sigma)$ is replaced by a single edge $\{z, y_i\}$ in $\Delta_y(\sigma)$, so only a single copy y_i of y is adjacent to z in $\Delta_y(\sigma)$. We have that $\mathrm{lk}_{\Delta(\sigma)}(z)$ is a connected graph H, and $\mathrm{lk}_{\Delta_y(\sigma)}(z)$ is a graph isomorphic to H where y is replaced by some node y_i, and hence, it is also connected.

No new LAP is created by the deformation, while y is removed, and the first claim follows.

For the second claim, consider a facet $\sigma' \in \mathcal{I}$ such that O contains no articulation point w.r.t. σ'. For every vertex $z \in \Delta(\sigma')$, if $y \notin \mathrm{lk}_{\Delta(\sigma')}(z)$ then $\mathrm{lk}_{\Delta(\sigma')}(z) = \mathrm{lk}_{\Delta_y(\sigma')}(z)$ and we are done.

Consider next a vertex $z \in \Delta(\sigma')$ such that $y \in \mathrm{lk}_{\Delta(\sigma')}(z)$. The edge $\{y, z\}$ exists in $\Delta(\sigma')$, and since Δ is a carrier map, there is a facet $\{y, z, z'\}$ in $\Delta(\sigma')$. By assumption, $\mathrm{lk}_{\Delta(\sigma')}(z)$ is connected, so $\mathrm{lk}_{\Delta(\sigma')}(z)$ contains a path from every node w to either y or z' that does not go through the edge $\{y, z'\}$. Consider such a path. If it reaches z', it remains intact in the deformation and exists in $\mathrm{lk}_{\Delta_y(\sigma')}(z)$. If it reaches y, its last edge is of the form $\{w', y\}$ for some w'. If $w' \in \Delta(\sigma)$ then $w' \in C_i$ for some connected component C_i of $\Delta(\sigma)$, and the edge $\{w', y_i\}$ exists in $\mathrm{lk}_{\Delta_y(\sigma')}(z)$ while the other path edges remain intact. If $w' \notin \Delta(\sigma)$ then all the edges of the form $\{w', y_i\}$ (for all i) exist in $\mathrm{lk}_{\Delta_y(\sigma')}(z)$ while the other path edges remain intact. Finally, we note that all the edges of the form $\{z', y_i\}$ exist in $\mathrm{lk}_{\Delta_y(\sigma')}(z)$. Hence, in $\mathrm{lk}_{\Delta_y(\sigma')}(z)$ each vertex w has a path to either z' or to some y_i, and all the vertices y_i are connected to z', which proves the connectivity of $\mathrm{lk}_{\Delta_y(\sigma')}(z)$.

Finally, we consider y itself, and the case where $y \in \Delta(\sigma')$ is not an articulation point w.r.t. σ'. In this case $\mathrm{lk}_{\Delta(\sigma')}(y) = \mathrm{lk}_{\Delta_y(\sigma')}(y_i)$ for all y_i, hence no y_i is an articulation point in Δ_y w.r.t. σ'. □

4.2 Splitting Maintains Solvability

A core property of the splitting deformation is that it maintains the solvability or insolvability of the original task, as stated next.

Lemma 4.2. *Let $T = (\mathcal{I}, O, \Delta)$ be a task in a canonical form, and let y be a LAP w.r.t. σ for some $\sigma \in \mathcal{I}$. Then, the task $T_y = (\mathcal{I}, O_y, \Delta_y)$ derived from T by eliminating y is solvable iff T is solvable.*

Proof. First, an algorithm A_y for T_y implies an algorithm A for T: execute A_y, then let each process with output $y_i \in O_y$ output y in A; each other process keeps its output.

We show that A indeed solves T. Fix $\tau \in \mathcal{I}$ (of any dimension) and consider an execution of A with input τ, and the resulting set $\rho \subseteq V(O)$ of outputs (technically, a set of id-value pairs). Let $\rho_y \in O_y$ be the output of the execution of A_y within the execution of A, and note that $\rho_y \in \Delta_y(\tau)$. If none of the vertices of ρ_y was created by the splitting deformation then $\rho = \rho_y$, and we have $\rho \in \Delta(\tau)$ and $\rho \in O$ as desired. Otherwise, there is some i such that $y_i \in \rho_y$. We may have $\rho_y = \{y_i\}$, $\rho_y = \{y_i, z\}$ or $\rho_y = \{y_i, z, z'\}$. In all the cases, ρ_y is created by taking a simplex $\rho \in \Delta(\tau)$ with $y \in \rho$, replacing y by y_i and adding the resulting simplex ρ_y to $\Delta_y(\tau)$ and to O_y. Hence, the original simplex ρ satisfies $\rho \in \Delta(\tau)$ and $\rho \in O$ and we are done.

For the opposite direction, we have to use the link $\mathrm{lk}(y)$, and since this is a topological structure the argument will be topological as well. Assume that T is solvable, then there is a protocol for T which induces a protocol complex $\mathcal{P}$, and a simplicial map $\delta : \mathcal{P} \to O$

carried by Δ. Let $\mathcal{P}(\sigma)$ be the sub-complex of $\mathcal{P}$ resulting from subdividing σ. We stress that our solvability characterization does not use $\mathcal{P}$ and it only appears here for the correctness proof.

We prove that T_y is solvable by presenting a mapping $\delta_y : \mathcal{P} \to O_y$ carried by Δ_y; note that $\mathcal{P}$ remains unchanged. For every vertex $w \in \mathcal{P}$, if $\delta(w) \neq y$ then define $\delta_y(w) = \delta(w)$. On the other hand, if $\delta(w) = y$, consider two cases. If $w \in \mathcal{P}(\sigma)$ then choose a neighbor w' of w in $\mathcal{P}(\sigma)$, check to which connected component C_i of $\mathrm{lk}_{\Delta(\sigma)}(y)$ this neighbor w' is mapped by δ, and define $\delta_y(w) = y_i$. Otherwise, define $\delta_y(w) = y_1$.

We show that the above δ_y is well defined in the case of $w \in \mathcal{P}(\sigma)$. First, $\mathcal{P}(\sigma)$ is connected (see, e.g., [18, Theorem 10.2.11.]) and contains more than a single vertex, so a neighbor w' exists. Second, $\mathcal{P}(\sigma)$ is link connected (see, e.g., [26, Theorem 5]) and δ preserves the connectivity of each link (see, e.g., [26, Lemma 4]), hence all neighbors w' of w are mapped by δ to the same connected component of $\mathrm{lk}_{\Delta(\sigma)}(y)$, so the choice of w' is insignificant.

Finally, we show that δ_y is simplicial and that it is carried by Δ_y. We consider a simplex $\tau \in \mathcal{I}$ (of any dimension) and show that $\delta_y(\mathcal{P}(\tau)) \subseteq \Delta_y(\tau)$; since any simplex in Δ_y was also added to O_y, this suffices. Consider a simplex $\xi \in \mathcal{P}(\tau)$.

We start with the simple case, when $y \notin \delta(\xi)$. We assume A is correct so $\delta(\xi) \in \Delta(\tau)$; since Δ_y is the same as Δ on simplices that do not contain y, we also have $\delta(\xi) \in \Delta_y(\tau)$. Finally, the definition of A_y gives $\delta(\xi) = \delta_y(\xi)$, so $\delta_y(\xi) \in \Delta_y(\tau)$ and we are done.

Consider now $\xi \in \mathcal{P}(\tau)$ such that some $w \in \xi$ satisfies $\delta(w) = y$. Since $y \in \Delta(\sigma)$ and the task is canonical, we have $\tau \cap \sigma \neq \emptyset$. We consider two sub-cases, as depicted in Fig. 6.

(1) $\tau \subseteq \sigma$. Here, $\xi \in \mathcal{P}(\tau) \subseteq \mathcal{P}(\sigma)$. Since A is correct, $\delta(\xi) \subseteq \{y, z, z'\}$ for some facet $\{y, z, z'\} \in \Delta(\sigma)$, and $\{z, z'\} \in C_i$ for some connected component C_i of $\mathrm{lk}_{\Delta(\sigma)}(y)$. In this case, $\Delta_y(\tau)$ is defined to contain the same simplices as $\Delta(\tau)$ but with y_i instead of y. Since A is correct $\delta(\xi) \in \Delta(\tau)$, so the simplex resulting from $\delta(\xi)$ by replacing y by y_i is in $\Delta_y(\tau)$. Finally, $\delta_y(\xi)$ contains all the vertices of $\delta(\xi)$, except for y which is replaced by $\delta_y(w) = y_i$, and this simplex is in $\Delta_y(\tau)$ by the above.

(2) $\tau \cap \sigma \neq \emptyset$ while $\tau \not\subseteq \sigma$. In this case, there is a facet $\sigma' \in \mathcal{I}$, $\sigma' \neq \sigma$, such that $\tau \subseteq \sigma'$. For each simplex $\rho \in \Delta(\tau)$ with $y \in \rho$, the carrier map Δ_y is defined so that $\Delta_y(\tau)$ contains all the simplices resulting from ρ by replacing y by y_i, for all i. Since A is correct $\delta(\xi) \in \Delta(\tau)$ is a simplex, thus all the simplices resulting from $\delta(\xi)$ by replacing y (if $y \in \delta(\xi)$) by some y_i are in $\Delta_y(\tau)$. If $w \in \mathcal{P}(\sigma)$ then $\delta_y(w) = y_i$ for some i, and otherwise $\delta_y(w) = y_1$. However, a crucial point is that in both cases $\delta_y(\xi)$ is the outcome of taking $\delta(\xi)$ and replacing y by some y_i. Hence, we have $\delta_y(\xi) \in \Delta_y(\tau)$. □

4.3 Creating a Link-Connected Task

By repeatedly eliminating LAPs, we can take a canonical task T and create a new task T' without LAPs while maintaining solvability:

Theorem 4.3. *Given a canonical task $T = (\mathcal{I}, O, \Delta)$, we can apply a finite sequence of LAP elimination steps to obtain a link-connected task $T' = (\mathcal{I}, O', \Delta')$, which is solvable iff T is solvable.*

Proof. We start from T and go over all the facets $\sigma \in \mathcal{I}$. For each σ, we repeat the above process for all the articulation points $y^1, y^2, \ldots$ w.r.t. σ, one at a time, to create a sequence $O, O_{y^1}, (O_{y^1})_{y^2}, \ldots$ of complexes and a sequence $\Delta, \Delta_{y^1}, (\Delta_{y^1})_{y^2}, \ldots$ of carrier maps, with a decreasing number of LAPs w.r.t. σ. This can be done since all the intermediate tasks are canonical by Claim 1. After no LAPs w.r.t. σ are left, we move to the next facet, and so on. Lemma 4.1 guarantees that no further LAPs w.r.t. σ are created after they are eliminated, and the same will hold for any later facet in the sequence. This guarantees that the process terminates with an output complex O' and a carrier map Δ' such that there is no LAP w.r.t. any facet, or equivalently, a task T' that is link-connected. Lemma 4.2 guarantees that the solvability is preserved throughout the process, so T' is solvable iff T is solvable, as claimed. □

5 Solvability Characterization

Once we have shown Theorem 4.3, stating that articulation points can be eliminated without changing solvability, we can derive the main characterization result. We continue to denote $T' = (\mathcal{I}, O', \Delta')$ the task after splitting a general task $T = (\mathcal{I}, O, \Delta)$ (which was first transformed into a canonical form if necessary). Note that by construction, T' is link connected, meaning that for each facet $\sigma \in \mathcal{I}$, $\Delta'(\sigma)$ is link connected. Recall that $|K|$ denotes the geometric realization of a simplical complex K.

Theorem 5.1. *A task $T = (\mathcal{I}, O, \Delta)$ is solvable iff there is a continuous map from $|\mathcal{I}|$ to $|O'|$ carried by Δ'.*

The theorem follows from Lemma 5.2 ($\Rightarrow$) and Lemma 5.3 ($\Leftarrow$).

5.1 From Solvability to a Continuous Map

Lemma 5.2. *If a task $T = (\mathcal{I}, O, \Delta)$ is solvable then there is a continuous map from $|\mathcal{I}|$ to $|O'|$ carried by Δ'.*

Proof. Assume that T is solvable, then so is $T' = (\mathcal{I}, O', \Delta')$ by Theorem 3.1 and Theorem 4.3. Since we consider the wait-free model and a finite input complex, the solvability of T implies, by the Asynchronous Computability Theorem (ACT) [22], that there is a chromatic subdivision $\mathrm{Ch}^{(r)}(\mathcal{I})$ of $\mathcal{I}$ and a chromatic simplicial map $\delta : \mathrm{Ch}^{(r)}(\mathcal{I}) \to O'$ carried by Δ'.

The simplicial map δ and the chromatic subdivision $\mathrm{Ch}^{(r)}$ induce a continuous map $|\delta \circ \mathrm{Ch}^{(r)}| : |\mathcal{I}| \to |O'|$ (see equation (3.2.2) in [18, Section 3.2.3]), which is carried by Δ'. □

5.2 From a Color-Agnostic Protocol to Chromatic Solvability

To prove the converse direction, note that by Theorem 4.3 the task T' is link-connected. Informally, if there is a continuous map $f : |\mathcal{I}| \to |O'|$ carried by Δ', then by the colorless ACT [18, Theorem 5.2.7], the task $(\mathcal{I}, O', \Delta')$ *when viewed as a colorless task*, is solvable. This gives a *color-agnostic* algorithm by which processes starting with input values in some $\sigma \in \mathcal{I}$ decide on output values in the same simplex τ of $\Delta'(\sigma)$, *but not necessarily each process on a vertex of its own color (*id*)*. However, if processes start in a face $\tau \in \mathcal{I}$ then they decide on a face of $\Delta(\tau)$.

Lemma 5.3 shows that, since T' is link-connected, the processes can find vertices of their own colors, forming a simplex still in $\Delta'(\sigma)$. The lemma describes an explicit algorithm specialized for three processes. The existence of such an algorithm was suggested in the past ([22, Lemma 4.21], [13, Lemma 3]). An algorithm with

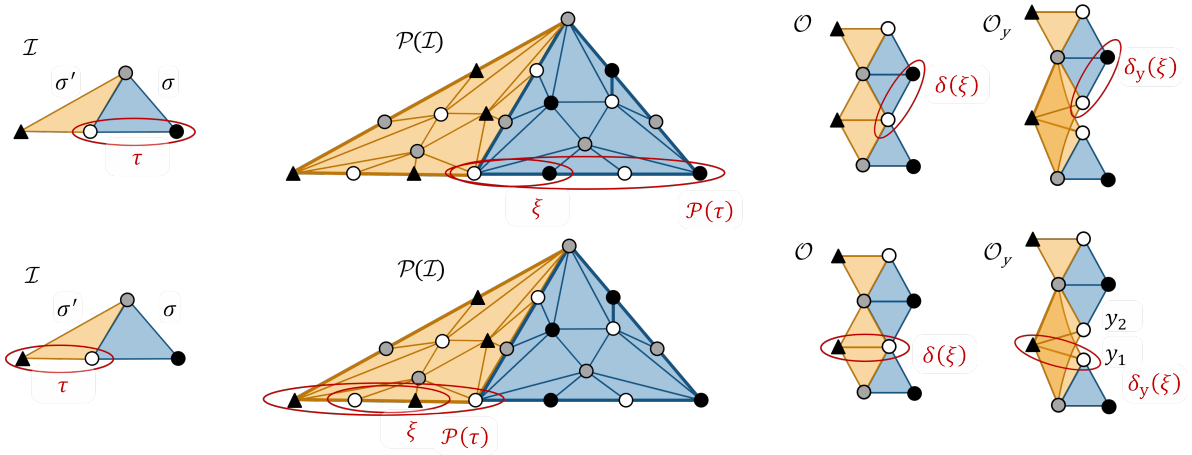

Figure 6: Proof of Lemma 4.2: case 1 (top) and case 2 (bottom).

some more explicit steps was presented in [27], but a key part of it, solving *link-based non-chromatic simplex agreement*, also eventually applies the *non-constructive* simplicial approximation theorem [24].

Lemma 5.3. *If the task $T = (\mathcal{I}, \mathcal{O}, \Delta)$ is link-connected and there is a color-agnostic algorithm $\mathcal{A}_C$ solving its colorless variant, then the algorithm in Fig. 7 is a chromatic algorithm for solving T.*

The processes first run the color-agnostic algorithm $\mathcal{A}_C$, which produces chromatic vertices in the same output simplex, but does not assure that a process ends with a vertex of its own color. So, assume process p_i ends with vertex $v_i \in \mathcal{V}(\mathcal{O})$. As all the decided vertices belong to the same simplex in the output complex, although several processes may decide on the same vertex, different decided vertices have different colors.

Process p_i then updates v_i in a snapshot object and scans it to obtain a set V_i of vertices (a *view*). The process then updates V_i in another snapshot object, and scans this object to get a set of views V_0, V_1, V_2, some of which may be empty. Since the non-empty views among V_0, V_1, V_2 are results of scanning the same snapshot object, the non-empty views among V_0, V_1, V_2 are comparable, i.e., contained in one another. Furthermore, not all the views are empty, since p_i's scan includes p_i's own view V_i, and $\{v_i\} \subseteq V_i$.

In the next step, p_i picks V^* to be the smallest nonempty set among V_0, V_1, V_2 (this is its initial *core*). Note that by comparability, V^* is contained in all the nonempty sets among V_0, V_1, V_2. In fact, V^* can be seen as the intersection of the non-empty views among V_0, V_1, V_2 (as done in [27]). If V^* includes a vertex v such that $\mathrm{id}(v) = i$, then p_i *decides on v and terminates*. (Such processes are called *pivots* in the proof.) Otherwise, the color i of p_i does not appear in V^*, thus V^* contains only one or two vertices.

If V^* contains two vertices (necessarily with different colors), i.e., $V^* = \{v^*, w^*\}$, then p_i picks a vertex v' such that (a) $\mathrm{id}(v') = i$ and (b) $\{v', v^*, w^*\}$ is a facet. Then, p_i writes (v', V^*) in yet another snapshot object, and scans this object. If p_i does not find a smaller view in this object, then p_i *decides on v' and terminates*. Otherwise, it finds a new minimal view *of size 1*, which is a subset of V^*, say $\{v^*\}$, and proceeds as if its core had a single vertex (see next paragraph).

If V^* contains a single vertex v^*, i.e., $V^* = \{v^*\}$, then note that the process whose color is $\mathrm{id}(v^*)$ will observe v^* in its core and will decide on v^*. (The minimal view in A may only decrease, and it cannot get smaller than $\{v^*\}$, so all processes observe v^* in their core, including $p_{\mathrm{id}(v^*)}$ itself.) Thus, p_i is going to decide on some vertex in the link of v^*, possibly after negotiating with p_j, the third process other than p_i and $p_{\mathrm{id}(v^*)}$. This negotiation is done by "jumping" towards the other process on a predetermined path in $\mathrm{lk}(v^*)$. It ends when the two processes are on an edge of the link, which means that the three processes are in the same facet.

A more detailed pseudocode appears in Figure 7. Assume that the processes start on a chromatic input simplex $\sigma = (x_0, x_1, x_2)$, such that $\mathrm{id}(x_i) = i$, i.e., the input of process p_i is x_i. Recall that Δ is a carrier map, hence every vertex or edge in $\Delta(\tau)$, for any $\tau \subseteq \sigma$, belongs to a facet. The concrete algorithm is slightly more complex than described above, since if the processes start in a face $\tau \in \mathcal{I}$ then they must decide on a face of $\Delta(\tau)$ and cannot decide on any face of $\mathcal{O}$ as described above. This is a crucial point, which was overlooked in prior work. We remark that some of the updates and scans could be replaced with more efficient writes and collects, but we avoid such optimizations for simplicity.

Proof of Lemma 5.3. The algorithm's code guarantees that if a process returns, then it returns a vertex with its own color.

To argue that the algorithm is correct, we need to prove that (a) when processes start in $\tau \in \mathcal{I}$, all output vertices belong to the same simplex in $\Delta(\tau)$, and (b) each process eventually returns. In fact, each process returns in time which is at most proportional to the length of the longest link in the output complex.

The set V^*, first set by p_i in (5), is called its *core*; its value in (5) is called the *initial core*. Processes that decide in (6) are called *pivots*, and all other processes are *non-pivots*.

Claim 2. *At least one process is a pivot, and all pivots return vertices in the same simplex.*

Proof. The views written in M_{snap} are comparable (by containment), and in particular, the minimal view picked by a process (in (5) or (7e)) must include at least one common vertex, say v^*. The

(1) Update $M_{\text{in}}[i] \leftarrow x_i$
(2) Run a color-agnostic algorithm $\mathcal{A}_C$ for T with input x_i, producing output y_i // $\text{id}(y_i)$ *is not necessarily* i
(3) Update $M_{\text{cless}}[i] \leftarrow y_i$ and scan M_{cless} to obtain a view V_i
(4) Update $M_{\text{snap}}[i] \leftarrow V_i$ and scan M_{snap} to obtain (V_0, V_1, V_2)
(5) Let V^* be the minimal non-empty set among V_0, V_1, V_2
(6) If V^* contains a vertex v such that $\text{id}(v) = i$, decide on v and terminate
(7) If $V^* = \{u^*, w^*\}$ for some $u \neq w$, then
  (a) Scan M_{in} into τ // *Here* $|\tau| = 3$*:* u^* *and* w^* *have different colors that are both different from* i*. Since* $\mathcal{A}_C$ *respects* Δ*, it must have been executed by processes with these three colors*
  (b) Pick a vertex v_i such that $\text{id}(v_i) = i$ and $\{v_i, u^*, w^*\} \in \Delta(\tau)$ // *Note that* v_i *is in the links of both* u^* *and* w^*
  (c) Update $M_{\text{decisions}}[i] \leftarrow (v_i, v_i, V^*)$ and scan $M_{\text{decisions}}$
  (d) If no other value is in $M_{\text{decisions}}$, decide on v_i and terminate
  (e) Otherwise, $M_{\text{decisions}}$ contains a pair $(\_, \_, W^*)$, with $|W^*| = 1$ // *Another process does not write in* $M_{decisions}$ *if it also has* V^* *as core*
  Set $V^* \leftarrow W^*$
(8) Let v^* be the single vertex such that $V^* = \{v^*\}$
(9) Scan M_{in} into τ. // *Here* $|\tau| \geq 2$ *since at least* p_i *and a process with color* $\text{id}(v^*)$ *participate*
(10) If $v_i \neq \perp$, then pick a vertex v_i such that $\text{id}(v_i) = i$ and $\{v_i, v^*\} \in \Delta(\tau)$ // *(7) was not executed*
(11) Update $M_{\text{decisions}}[i] \leftarrow (v_i, v_i, V^*)$ and scan $M_{\text{decisions}}$
(12) If $M_{\text{decisions}}$ does not contain any other process, decide on v_i and terminate
(13) Let $j \neq i$ and v_j, v and V_j such that $M_{\text{decisions}}[j] = (v_j, v, V_j)$
Let Π be the lexicographically-smallest shortest (v_i, v_j)-path in $\text{lk}_{\Delta(\tau)}(v^*)$ // *We assign a unique number to each vertex in the output complex, and identify each path with the (unordered) set of unique numbers of the vertices in the path*
$v' \leftarrow v_i$
(14) While (v', v) is not an edge in $\text{lk}_{\Delta(\tau)}(v^*)$ do
  (a) $v' \leftarrow$ the vertex adjacent to v on Π // $\text{id}(v') = i$ *must hold*
  (b) Update $M_{\text{decisions}}[i] \leftarrow (v_i, v', V^*)$ and scan $M_{\text{decisions}}$
  (c) Update v such that $M_{\text{decisions}}[j] = (v_j, v, V_j)$
(15) Decide on v' and terminate

Figure 7: Algorithm for Process p_i with Input x_i

process whose color is $\text{id}(v^*)$ terminates in (6). Thus, at least one process is a pivot. All pivots (who terminate in (6)) return vertices in the same simplex, due to the correctness of $\mathcal{A}_C$. □

The claim implies that at most two processes are non-pivots and proceed beyond (6). Moreover, the cores of non-pivots contain the minimal core, and thus, the include the vertices decided by pivots.

For non-pivots, we need to prove that the algorithm is well-defined, namely, they can pick vertices satisfying the relevant constraints. The vertex picked in (7b) must exist since the results of $\mathcal{A}_C$ belong to a facet in the chromatic output complex. For the same reasons, it is in $\Delta(\tau)$. The vertex picked in (10) exists since it depends only on the single vertex in the core.

When executing the loop in (14), the non-pivot processes use the same path Π. This is because the first components of their entries in $M_{\text{decisions}}$ are set only once, and they determine the path Π.

Finally, each iteration of the loop in (14) reduces the distance between the two proposed vertices, written in $M_{\text{decisions}}$. First, note that if a non-pivot process p_i performs two consecutive updates to $M_{\text{decisions}}$, without an interleaved update by the other non-pivot process, then the process decides. (This also covers the case where one non-pivot process runs solo.) Otherwise, the vertex it picks is inside the sub-path of Π between their prior vertices. □

5.3 Corollaries of Theorem 5.1

For two processes, a task is solvable if and only if there is a continuous map from $|\mathcal{I}|$ to $|\mathcal{O}|$ carried by Δ, and note that for two processes this is not restricted to colorless tasks. This is a consequence of [18, Theorem 2.5.2]; it is also analogous to the seminal result for message-passing models with a single process failure [6]. This 1-dimensional statement can be used to prove that tasks are unsolvable after splitting.

Proposition 5.4. *A two-process task $(\mathcal{I}, \mathcal{O}, \Delta)$ is solvable if and only if there is a continuous map from $|\mathcal{I}|$ to $|\mathcal{O}|$ carried by Δ.*

The following two corollaries are useful for showing that a task is unsolvable due to local articulation points; they are used in the next section. In the following, we say a path *crosses through a LAP* y if it has three consecutive vertices w_1, y, w_2 such that w_1 and w_2 belong to two different connected components of the link of y.

Corollary 5.5. *A task $(\mathcal{I}, \mathcal{O}, \Delta)$ is not solvable if there is an input simplex $\sigma = (x, x', x'') \in \mathcal{I}$ such that for every pair of vertices $y \in \Delta(x)$ and $y' \in \Delta(x')$ any path from y to y' in $\Delta(x, x')$ crosses through a LAP w.r.t. σ.*

Majority consensus (defined in Section 1) satisfies the conditions of the colorless ACT, but it is not wait-free solvable. After splitting the articulation points, the output complex $\mathcal{O}'$ consists of two disconnected components, with the solo output vertex of P_0 in one component, where it decides 0. The other connected component contains the edge where the other two processes start with input 1, which makes the task wait-free unsolvable, by Corollary 5.5.

Recall that $\text{Skel}^n \mathcal{I}$ denotes the subcomplex of $\mathcal{I}$ of all simplices of dimension at most n. When $\mathcal{I}$ is a triangle, $\text{Skel}^1 \mathcal{I}$ is the *boundary of the triangle*, the cycle consisting of the three input edges.

Corollary 5.6. *A task $(\mathcal{I}, \mathcal{O}, \Delta)$ where $\mathcal{I}$ is a single triangle is not solvable if every cycle in $\Delta(\text{Skel}^1 \mathcal{I})$ goes through a LAP.*

6 Examples

Here we describe two tasks that demonstrate two types of obstructions, which can be used to show them unsolvable: a decidable one, due to articulation points, and another one, due to contractibility (in general, undecidable). Both examples take a task and remove some of the triangles of the output complex but no edges or vertices. Thus, as long as only one or two processes participate, the task remains unchanged. When considered as a colorless task, where a process can output any one of three (not necessarily distinct) values v_0, v_1, v_2, and the task is often solvable. However, when considered as chromatic tasks, some of these decisions are disallowed, making the tasks harder to solve.

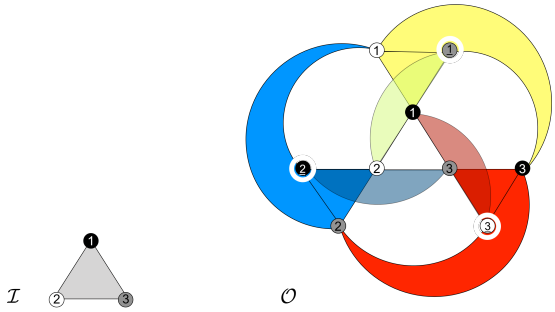

Figure 8: The pinwheel task

The first example (the hourglass task) is solvable as a colorless task, since there is continuous map from its input complex to its output complex, while the second example (the pinwheel task) has no such continuous map. For both examples, however, articulation points can be used to prove that the tasks are not wait-free solvable.

6.1 The hourglass task

The *hourglass* task, mentioned in the introduction (see Figure 2), was presented in [18]. The three processes P_0, P_1, and P_2 are denoted by black, white, and gray, respectively. This task has input complex $\mathcal{I}$ with a single simplex (facet), and an output complex $\mathcal{O}$. Each process running solo decides value 0. Process P_0 running concurrently with either P_1 or P_2 can additionally (to their solo values) decide on their respective vertices, with output 1. While P_1 and P_2 running concurrently can additionally decide their respective vertices with value 2. When all three processes run concurrently, any triangle is a valid output simplex. Thus, informally, this task is constructed by taking the standard chromatic subdivision and "pinching it at the waist" to identify P_0's vertices on the edges representing its two-process executions. This gluing creates an articulation point, the vertex of P_0 with output 1.

The Hourglass task satisfies the conditions of the colorless ACT, but it is not wait-free solvable. This is proved by reduction from 2-set agreement [18]. We show another way to prove the impossibility of the hourglass task. As shown in Figure 2, after splitting the articulation point, the "dimension" of the impossibility proof is reduced: the new task is unsolvable by reduction from consensus, using Corollary 5.5, instead of by reduction from 2-set agreement.

6.2 The pinwheel task

Another example is the *pinwheel* task in Figure 8. Recall that the three processes P_0, P_1, and P_2 are denoted by black, white, and gray, respectively. This task is obtained from (inputless) 2-set agreement by removing some output triangles. But it leaves intact the outputs for the edges, i.e., for executions where two processes participate. For example, notice that when P_0 starts with 1 and P_1 starts with 2, the four output edges of combinations of 1 and 2 are possible.

Being a subtask of 2-set agreement, the pinwheel task is as hard as 2-set agreement (but not as hard as consensus), and hence, unsolvable. However, the impossibility can be derived from our theorem, by splitting articulation points, to obtain three disconnected components, identified by three different colors in the figure.

Notice that now the splitting affects the relation Δ in all three dimensions, vertices, edges and triangles. Each input vertex now is allowed to decide on two different output vertices, one copy per connected component of $\mathcal{O}'$. For instance, when P_0 starts with input value 1, it can decide the copy in the yellow component or the copy in the red component.

At the edge level, in 2-set agreement, a cycle of four edges can be decided for each input edge, and this cycle remains in the pinwheel task, except that it is broken in the split version $\mathcal{O}'$. However, to prove the impossibility we cannot directly use Corollary 5.5, because there is still a path between vertices in $\Delta(x)$ and $\Delta(x')$ for each input edge x, x'.

We use Corollary 5.6 to see why the split version of the pinwheel task is unsolvable. Suppose P_0 decides in a solo execution the copy of output 1 in the yellow component. Then P_1 must decide a copy of 2 in the same component, because there is an edge joining their respective inputs in $\mathcal{I}$, and due to Theorem 5.1. By the same reason, P_2 should also decide a vertex on the same component, but neither of the copies of output vertex 3 is in the yellow component.

7 Discussion

This paper provides a new characterization for wait-free solvability of general three-process tasks. We present a new topological deformation on the output complex of a task T, which eliminates *local articulation points* by splitting them. We prove that the task T is wait-free solvable if and only if there is a continuous map from the geometric realization of the input complex of T to the geometric realization of the deformed output complex, which respects the deformed task mapping.

We consider the restriction to three processes as a critical stepping-stone for future investigation of solvable chromatic tasks, for an arbitrary number of processes. While some of our techniques apply to any number of processes, others are specific to three processes, most notably, the splitting deformation described in Section 4.

Our results expose two types of obstructions to solvability: The first, which exists only in chromatic tasks, are local articulation points; these obstructions can be effectively detected (and removed). The second is the one present in colorless tasks, namely, the non-existence of a continuous map from the geometric realization of $\mathcal{I}$ to that of $\mathcal{O}'$ carried by Δ'. The locality of the former obstruction makes it an ideal target for extension-based impossibility proofs [2, 4]. The latter obstruction is not in general decidable, as it is related to the topological notion of loop contractibility [13, 19]. The hourglass task impossibility can be obtained by either of these two obstructions, by contractibility, as is shown in [18, Section 11.1], or by articulation points, as shown in Section 6.1.

Acknowledgments

Hagit Attiya is supported by the Israel Science Foundation (grant number 22/1425). Pierre Fraigniaud is partially supported by ANR projects DUCAT (ANR-20-CE48-0006), PREDICTIONS (ANR-23-CE48-0010), and ENEDISC (ANR-24-CE48-7768). Part of the work of Sergio Rajsbaum was done while visiting IRIF, partially supported by ANR Project DUCAT (ANR-20-CE48-0006).

References

[1] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. 1993. Atomic Snapshots of Shared Memory. *J. ACM* 40, 4 (1993), 873–890. doi:10.1145/153724.153741

[2] Dan Alistarh, James Aspnes, Faith Ellen, Rati Gelashvili, and Leqi Zhu. 2023. Why Extension-Based Proofs Fail. *SIAM J. Comput.* 52, 4 (2023), 913–944. doi:10.1137/20M1375851

[3] Hagit Attiya, Amotz Bar-Noy, Danny Dolev, David Peleg, and Rüdiger Reischuk. 1990. Renaming in an Asynchronous Environment. *J. ACM* 37, 3 (1990), 524–548. doi:10.1145/79147.79158

[4] Hagit Attiya, Armando Castañeda, and Sergio Rajsbaum. 2023. Locally solvable tasks and the limitations of valency arguments. *J. Parallel Distributed Comput.* 176 (2023), 28–40. doi:10.1016/J.JPDC.2023.02.002

[5] Hagit Attiya and Jennifer Welch. 2004. *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* John Wiley & Sons, Hoboken, NJ, USA.

[6] Ofer Biran, Shlomo Moran, and Shmuel Zaks. 1990. A combinatorial characterization of the distributed 1-solvable tasks. *Journal of algorithms* 11, 3 (1990), 420–440.

[7] Dobrina Boltcheva, David Canino, Sara Merino Aceituno, Jean-Claude Léon, Leila De Floriani, and Franck Hétroy. 2011. An iterative algorithm for homology computation on simplicial shapes. *Computer-Aided Design* 43, 11 (2011), 1457–1467. doi:10.1016/j.cad.2011.08.015 Solid and Physical Modeling 2011.

[8] Elizabeth Borowsky and Eli Gafni. 1993. Generalized FLP impossibility result for t-resilient asynchronous computations. In *25 ACM Symposium on Theory of Computing (STOC)*. 91–100. doi:10.1145/167088.167119

[9] Elizabeth Borowsky, Eli Gafni, Nancy A. Lynch, and Sergio Rajsbaum. 2001. The BG distributed simulation algorithm. *Distributed Comput.* 14, 3 (2001), 127–146. doi:10.1007/PL00008933

[10] Soma Chaudhuri. 1993. More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. *Inf. Comput.* 105, 1 (1993), 132–158. doi:10.1006/INCO.1993.1043

[11] Leila De Floriani, Mostefa M. Mesmoudi, Franco Morando, and Enrico Puppo. 2003. Decomposing non-manifold objects in arbitrary dimensions. *Graphical Models* 65, 1 (2003), 2–22. doi:10.1016/S1524-0703(03)00006-7

[12] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (1985), 374–382. doi:10.1145/3149.214121

[13] Eli Gafni and Elias Koutsoupias. 1998. Three-Processor Tasks Are Undecidable. *SIAM J. Comput.* 28, 3 (1998), 970–983. doi:10.1137/S0097539796305766

[14] Eli Gafni, Petr Kuznetsov, and Ciprian Manolescu. 2014. A generalized asynchronous computability theorem. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, Magnús M. Halldórsson and Shlomi Dolev (Eds.). ACM, 222–231. doi:10.1145/2611462.2611477

[15] Eli Gafni and Sergio Rajsbaum. 2010. Distributed Programming with Tasks. In *14th International Conference on Principles of Distributed Systems (OPODIS) (LNCS 6490)*. Springer, 205–218. doi:10.1007/978-3-642-17653-1_17

[16] Hugo Rincon Galeana, Sergio Rajsbaum, and Ulrich Schmid. 2022. Continuous Tasks and the Asynchronous Computability Theorem. In *13th Innovations in Theoretical Computer Science Conference, ITCS (LIPIcs, Vol. 215)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 73:1–73:27. doi:10.4230/LIPICS.ITCS.2022.73

[17] Éric Goubault, Jérémy Ledent, and Sergio Rajsbaum. 2021. A simplicial complex model for dynamic epistemic logic to study distributed task computability. *Inf. Comput.* 278 (2021), 104597. doi:10.1016/J.IC.2020.104597

[18] Maurice Herlihy, Dmitry N. Kozlov, and Sergio Rajsbaum. 2013. *Distributed Computing Through Combinatorial Topology.* Morgan Kaufmann.

[19] Maurice Herlihy and Sergio Rajsbaum. 1997. The decidability of distributed decision tasks. In *Proceedings of the 29th annual ACM symposium on Theory of computing*. 589–598.

[20] Maurice Herlihy and Sergio Rajsbaum. 2003. A classification of wait-free loop agreement tasks. *Theoretical Computer Science* 291, 1 (2003), 55–77. doi:10.1016/S0304-3975(01)00396-6

[21] Maurice Herlihy, Sergio Rajsbaum, Michel Raynal, and Julien Stainer. 2017. From wait-free to arbitrary concurrent solo executions in colorless distributed computing. *Theor. Comput. Sci.* 683 (2017), 1–21.

[22] Maurice Herlihy and Nir Shavit. 1999. The topological structure of asynchronous computability. *J. ACM* 46, 6 (1999), 858–923. doi:10.1145/331524.331529

[23] Annie Hui and Leila De Floriani. 2007. A two-level topological decomposition for non-manifold simplicial shapes. In *Proceedings of the 2007 ACM symposium on Solid and Physical Modeling*. 355–360.

[24] James R Munkres. 1984. *Elements of algebraic topology.* Addison Wesley.

[25] Vikram Saraph and Maurice Herlihy. 2015. The Relative Power of Composite Loop Agreement Tasks. In *19th International Conference on Principles of Distributed Systems, OPODIS (LIPIcs, Vol. 46)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:16. doi:10.4230/LIPICS.OPODIS.2015.13

[26] Vikram Saraph, Maurice Herlihy, and Eli Gafni. 2016. Asynchronous Computability Theorems for t-Resilient Systems. In *30th International Symposium on Distributed Computing (DISC) (LNCS, Vol. 9888)*. Springer, 428–441. doi:10.1007/978-3-662-53426-7_31

[27] Vikram Saraph, Maurice Herlihy, and Eli Gafni. 2018. An algorithmic approach to the asynchronous computability theorem. *J. Appl. Comput. Topol.* 1, 3-4 (2018), 451–474.

[28] John C. Stillwell. 1980. *Classical topology and combinatorial group theory.* Graduate Texts in Mathematics, Vol. 72. Springer-Verlag, New York. xii+301 pages.