How Exhaustive Does an Extension-Based Proof Need to Be?

Faith Ellen
University of Toronto, Canada

Shihao Liu
University of Toronto, Canada

Leqi Zhu
University of Manitoba, Winnipeg, Canada

Eli Gafni
University of California, Los Angeles, CA, USA

Rati Gelashvili
New Hampshire, USA

Abstract

The class of extension-based proofs encompasses traditional valency arguments. It has been shown that they are insufficient to establish the impossibility of $(n-1)$-set agreement among $n \geq 3$ processes in an asynchronous system with crash failures. We generalize this definition to *k-exhaustive* extension-based proofs, in which a prover can learn the maximum length of all executions involving a set of at most k processes from a specified configuration (which may be infinite). An upper bound on the length of these executions enables the prover to determine the outputs of all these executions. When $k = n$, this enables the prover to perform an exhaustive search of all reachable configurations, so it knows everything about the protocol. On the other hand, extension based proofs are as powerful as 1-exhaustive extension-based proofs. For any task with no deterministic, wait-free solution among $n \geq 2$ processes, we show that there is an $(n-1)$-exhaustive extension-based proof of its impossibility. This is done using a new characterization of such tasks. In contrast, we prove that for $1 \leq k \leq n-2$, there is no k-exhaustive extension-based proof of the impossibility of $(n-1)$-set agreement.

2012 ACM Subject Classification Theory of computation → Computability; Theory of computation → Interactive proof systems; Theory of computation → Distributed algorithms; Theory of computation → Distributed computing models

Keywords and phrases Extension-based proof, set agreement, valency argument, zero-one exclusion

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2025.29

Funding Faith Ellen and Shihao Liu were supported by the Natural Science and Engineering Research Council of Canada (NSERC) grant RGPIN-2020-04178 and Leqi Zhu was supported by the Natural Science and Engineering Research Council of Canada (NSERC) grant RGPIN-2025-07125.

1 Introduction

In the *k-set agreement problem*, there are $n \geq k+1$ processes that each have a private input value in $\{0, 1, \ldots, k\}$. The goal is for each process to output an input value so that, collectively, at most k different values are output. The special case of the problem when $k = 1$ is called *consensus*.

It is well known that, in asynchronous systems with crash failures, every deterministic protocol for k-set agreement must have a *bad* execution, in which some processes output incorrect values (i.e., the protocol is *incorrect*) or take infinitely many steps without outputting a value (i.e., the protocol is *not wait-free*). Fischer, Lynch, and Paterson [13] introduced

the valency argument and used it to first prove this result for the special case of consensus, i.e., when $k = 1$. The general result was independently obtained by Borowsky and Gafni [7], Herlihy and Shavit [17], and Saks and Zaharoglou [19].

The valency argument for consensus is simple and elegant: given a protocol, the proof provides a strategy to locate a non-empty prefix of a bad execution, which only requires examining the outputs of the processes from some carefully chosen initial configurations. Moreover, if the given protocol never produces incorrect outputs, then the strategy may be iterated to produce non-empty extensions of the prefix, allowing the construction of arbitrarily long, non-terminating executions.

In contrast, proofs for $k \geq 2$ are all nonconstructive and complex: they show that every wait-free protocol must be incorrect. The early proofs used tools from combinatorial topology to do so. Later, Attiya and Castañeda [3] and Attiya and Paz [5] gave more direct, combinatorial proofs of the same result. Ultimately, while these proofs show the existence of bad executions, they do not suggest a strategy to locate a non-empty prefix of one, let alone a strategy to iteratively find non-empty extensions.

Recently, Alistarh, Aspnes, Ellen, Gelashvili, and Zhu [1] defined the class of *extension-based proofs*, which attempt to model a constructive valency argument as an interaction between a fixed *prover* and a given *protocol*. The goal of the prover is to either build an infinite execution of the protocol or an execution in which the protocol outputs incorrect values. It works in phases, during which it is allowed to ask the protocol *finitely many* step queries, chains of step queries, and output queries. At the beginning of the first phase, the prover's current execution is empty and all initial configurations have been reached. In each phase, the prover is restricted to ask queries starting from configurations reachable from its current execution. Roughly speaking:

- A *step query* allows the prover to *reach* the configuration resulting from the step of some specified processes. The prover can then examine what processes have terminated and what outputs they have produced in the new configuration.
- A *chain of step queries* allows the prover to ask an adaptively chosen (and possibly infinite) sequence of step queries. For example, it allows the prover to ask what happens if the first three processes keep taking steps in round-robin order.
- An *output query* allows the prover to ask for an execution from a given configuration by a specified set of processes in which a specified value is output (or is told that no such execution exists).

At the end of each phase, if the prover has not found a bad execution of the protocol, then the prover must commit to a finite, non-empty extension of its current execution. They proved that, for every extension-based prover, there is some (adversarially constructed) protocol for $(n-1)$-set agreement among $n \geq 3$ processes that never outputs incorrect values and for which, at the end of the first phase, the prover commits to an execution that has no infinite extension. The idea is that, since the prover can only ask a finite number of queries, it cannot learn enough about the protocol before it ends the first phase by committing to a finite execution. Therefore, it is possible to adaptively define a protocol that appears to be correct and wait-free in all extensions of this execution, but may have bad executions elsewhere.

Brusse and Ellen [9] proved that if there is a many-one reduction from one problem to another and there is an augmented extension based proof for the impossibility of the first problem, then there is also an augmented extension based proof for the impossibility of the second problem. In an *augmented extension-based proof*, the prover is equipped with a slightly more powerful variant of an output query:

- An *assignment query* allows the prover to ask for an execution from a given configuration by a specified set of processes in which some of these processes output specified values (or is told that no such execution exists).

They showed that extension-based provers with access to assignment queries are still unable to reliably locate any bad execution in protocols that claim to solve $(n-1)$-set agreement among $n \geq 3$ processes. Currently, it is unclear whether augmented extension-based proofs are *strictly* more powerful than extension-based proofs.

When proving impossibility results about tasks using combinatorial techniques, it is often convenient to consider *uniform* protocols, where each process terminates after taking exactly M steps, for some non-negative integer M. It is known that a task has a wait-free protocol if and only if it has a uniform protocol. The set of all executions of a uniform protocol is easy to represent as a *static* combinatorial object, whose structure only depends on the value of M, the number of processes, and the set of initial configurations. It is easy for an extension-based prover to locate a bad execution of a uniform protocol, if any such execution exists: the prover can submit a chain of step queries to determine the value of M and use it to exhaustively search through the finitely many executions of the protocol until it finds one where processes output incorrect values. Attiya, Castañeda, and Rajsbaum [4] considered a restricted variant of extension-based proofs, known as *local valency proofs*, that removes the prover's ability to perform exhaustive search: Instead of performing step queries and chains of step queries, the prover must repeatedly choose a one-round extension of the current execution (in which each process takes exactly one step), based on the set of all possible output values in all executions starting from each of these extensions. They showed that local valency proofs are insufficient to establish the impossibility of $(n-1)$-set agreement and weak symmetry breaking. It is unknown if the set of tasks that have extension-based impossibility proofs against general protocols is the same as the set of tasks that have local valency impossibility proofs against uniform protocols.

One way to make extension-based proofs more powerful is to give an extension-based prover the ability to perform exhaustive search on all executions that involve at most ℓ processes, for some parameter $\ell \leq n$, starting from any configuration it has previously reached. Such an extension-based proof is called *ℓ-exhaustive*. We ask the following question: how exhaustive does an extension-based proof need to be to enable a prover to locate a bad execution of a protocol for a specific task, assuming at least one such execution exists? More formally, an *ℓ-exhaustive extension-based proofs* provides the extension-based prover with a new *length query*, which tells the prover the length of a longest execution involving a specified set of at most ℓ processes starting from a specified configuration it has previously reached.

If a protocol has a bad execution, then an n-exhaustive extension-based proofs can find one for exactly the same reason that a standard extension-based proof can easily find one when the protocol is known to be uniform. We show that $(n-1)$-exhaustive extension-based proofs can also find a bad execution in any protocol for a task that has no wait-free solution. This is shown via a new characterization of such tasks. In general, this characterization is nonconstructive, so it does not enable us to construct an $(n-1)$-exhaustive extension-based impossibility proof of an arbitrary task that has no wait-free solution. However, for some tasks, such as $(n-1)$-set agreement among $n \geq 2$ processes and zero-one exclusion tasks among $n = 3$ processes, we can construct explicit $(n-1)$-exhaustive extension-based proofs. These proofs are based on combinatorial properties that were used in earlier impossibility proofs.

On the other hand, an extension-based prover can easily simulate a length query of a 1-exhaustive extension-based proof by performing a single chain of step queries. Hence, for each $k \geq 2$, there is no 1-exhaustive extension-based proof that can find a bad execution

in any protocol for k-set agreement among $n > k$ processes. Using new techniques, we substantially strengthen this result: there is no $(k-1)$-exhaustive extension-based proof that can find a bad execution in any protocol for k-set agreement among $n > k \geq 2$ processes. To show this, given any prover whose length queries only involve subsets of at most $k-1$ processes, we adaptively construct a correct k-*obstruction-free* protocol that appears to be wait-free to the prover.

Note that any protocol for k-set agreement among more than k processes is also a protocol for k-set agreement among $k+1$ processes. Thus, the k-exhaustive extension-based proof of the impossibility of k-set agreement among $k+1$ processes is also a k-exhaustive extension-based proof of the impossibility of k-set agreement among more than k processes. A consequence of our results is that there are ℓ-exhaustive extension-based proofs for the impossibility of k-set agreement among $n > k$ processes if and only if $\ell \geq k$. This suggests a hierarchy of impossible tasks: a task is at level k if k is the smallest value such that there is a k-exhaustive extension-based proof of its impossibility. In particular, k-set agreement among $n > k$ processes is at level k. Moreover, for any task at level k of this hierarchy, we show that there is a k-exhaustive extension-based proof that finds an infinite execution (involving $k+1$ processes) in any correct k-obstruction-free protocol for this task.

Roadmap. In Section 2, we define the model and introduce necessary terminology and notation. In Section 3, we prove that $(n-1)$-exhaustive extension-based proofs are sufficient to establish the impossibility of any task that has no wait-free solution and give explicit constructions for set-agreement tasks and a particular family of zero-one exclusion tasks among three processes. In Section 4, we prove that $(k-1)$-exhaustive extension-based proofs are insufficient to establish the impossibility of k-set agreement. Finally, in Section 5, we conclude with a discussion of future work.

2 Preliminaries

We consider the wait-free solvability of tasks in the iterated immediate snapshot model. It is known that a task has a wait-free solution in the standard asynchronous shared memory model, where processes communicate using (read/write) registers, if and only if it has a wait-free solution in the iterated immediate snapshot model [17]. The main advantage of this model is that the set of reachable configurations of a wait-free protocol can be easily represented using combinatorial topology.

2.1 Model

In the *iterated immediate snapshot* model, there are $n \geq 2$ processes, $p_1, \ldots, p_n$, that communicate using an infinite sequence, $S_1, S_2, \ldots$, of shared *snapshot* objects. Each snapshot object has n components, which are all initially blank (denoted by $-$), and supports two operations, $\mathsf{update}(v)$, for values $v \neq -$, and $\mathsf{scan}()$. An $\mathsf{update}(v)$ operation by p_i sets the ith component of the object to have value v. A $\mathsf{scan}()$ operation returns a vector containing the value of each component of the object.

A (full-information) *protocol* is described using a function δ from the set of possible process states to $O \cup \{\perp\}$, where O is a set of possible output values that does not include $\perp$. The *initial* state of each process p_i is a pair, (i, x_i), which consists of its identifier, i, and its input, x_i. The t'th time that p_i is scheduled, it performs an update to S_t with its current state as the value and then performs a scan on S_t. The new state of p_i is the pair

$(i, r_{i,t})$, where $r_{i,t}$ is the result of this scan. If δ returns an output value for this state, then p_i outputs that value and *terminates*. Otherwise, δ returns $\perp$. In this case, p_i remains *active* and will access the next snapshot object, S_{t+1}, when it is next scheduled.

A *configuration* (of a protocol) consists of the states of each process and the contents of each snapshot object at some point during an execution when each process is either poised to perform an update or has already terminated. In an *initial* configuration, each process is in an initial state and the contents of all snapshot objects are blank. An important property of the iterated immediate snapshot model is that the contents of the snapshot objects in a configuration can be inferred from the states of the processes, so a configuration is often described by its set of process states.

An *execution* of a protocol is an alternating sequence $\alpha = C_0, Q_1, C_1, Q_2, C_2, \ldots$ of configurations and nonempty subsets of processes. For $t \geq 1$, the processes in Q_t are all poised to update the same snapshot object in C_{t-1}. Configuration $C_t = C_{t-1}Q_t$ results from configuration C_{t-1} when the processes in Q_t perform their pending updates (in any order), immediately followed by their pending scans (in any order, after processes in Q_t have all performed their updates). The execution is *finite* if it ends with a configuration and is *terminating* if it ends with a configuration in which every process is terminated. It is *P-only*, for some set of processes P, if $Q_t \subseteq P$ for all $t \geq 1$. A configuration C' is *reachable from* another configuration C if there is an execution α that starts with C and ends with C'. This is denoted by $C' = C\alpha$. Configuration C' is *reachable* if it is reachable from an initial configuration.

Given any execution α from an initial configuration, there is an execution α' from the same initial configuration that is indistinguishable from α to all processes, such that all operations on each snapshot object S_t occur before any operation on snapshot object $S_{t'}$, for all $1 \leq t < t'$. This is because operations on different snapshot objects commute with one another.

A *t-round execution* of a protocol is an execution from an initial configuration in which each process appears in exactly t subsets of processes within the execution (i.e. it has performed exactly one update and scan operation on each of the first t snapshot objects) or it has terminated after performing fewer than t update and scan operations. A configuration is *round-based* if it is reachable via a t-round execution, for some $t \geq 0$. If C is a round-based configuration, then a *1-round execution from C* is an execution starting from C in which each process that is not terminated in C performs exactly one update and one scan on the same snapshot object. More generally, if C is a configuration in which each process in P is poised to update the same snapshot object or has terminated before accessing that snapshot object, then a *t-round, P-only execution from C* is a P-only execution from C in which each process in P appears in exactly t subsets or has terminated before performing t update and scan operations.

For each $n \geq 2$ and each $1 \leq x \leq n$, a protocol among n processes is *x-obstruction-free* if, for each subset P consisting of at most x processes and for each configuration C reachable via some initial configuration, all P-only executions starting from C are finite. A protocol among n processes is *wait-free* if it is n-obstruction-free.

2.2 Combinatorial topology

An*(abstract) simplex* σ is a non-empty finite subset of some universe. Each element of σ is a *vertex* of σ and each non-empty subset of σ is a *face* of σ. An *(abstract) simplicial complex* $\mathbb{C}$ is a collection of simplices that is closed under subset: if $\sigma \in \mathbb{C}$ and σ' is a face of σ, then $\sigma' \in \mathbb{C}$. A simplex is a *facet* of $\mathbb{C}$ if it is not a face of any other simplex in $\mathbb{C}$. The set $V(\mathbb{C})$ of *vertices* of $\mathbb{C}$ is the union $\cup\{\sigma \mid \sigma \in \mathbb{C}\}$ of the vertices of simplices in $\mathbb{C}$. A *subcomplex* of

$\mathbb{C}$ is any subset of $\mathbb{C}$ that is itself a simplicial complex. Given a set of vertices $V \subseteq V(\mathbb{C})$, the subcomplex of $\mathbb{C}$ *induced by* V is the set of all simplices $\sigma \in \mathbb{C}$ such that $\sigma \subseteq V$. Notice that a simplex σ together with all its faces form a simplicial complex. The *boundary* of σ, denoted by $\partial\sigma$, is the subcomplex consisting of all *proper faces* $\sigma' \subsetneq \sigma$.

Let G be the graph with vertex set $V(\mathbb{C})$ whose edges are the 2-vertex simplices in $\mathbb{C}$. Two vertices of $\mathbb{C}$ are *adjacent* (or *neighbours*) if they are adjacent in G. The *distance* between two vertices of $\mathbb{C}$ is the length of a shortest path between them in G. A vertex is *adjacent* to (or is a neighbour of) a subcomplex $\mathbb{A} \subseteq \mathbb{C}$ if the vertex is not in $V(\mathbb{A})$, but is adjacent to some vertex in $V(\mathbb{A})$.

A task $\mathbb{T}$ may be formalized in the language of combinatorial topology as a triple $\mathbb{T} = (\mathbb{I}, \mathbb{O}, \Delta)$, where $\mathbb{I}$ and $\mathbb{O}$ are simplicial complexes, called the *input* and *output* complexes of the task, and $\Delta : \mathbb{I} \to 2^{\mathbb{O}}$ is a task specification map from $\mathbb{I}$ to the set of all subcomplexes of $\mathbb{O}$. Additionally, δ is required to be a *carrier map*. This means that $\Delta(\sigma') \subseteq \Delta(\sigma)$ for all $\sigma' \subseteq \sigma \in \mathbb{I}$.

The input complex $\mathbb{I}$ represents the set of initial configurations of any *protocol for* $\mathbb{T}$. Each simplex in $\mathbb{O}$ represents a correct (partial) assignment of outputs to processes. Δ encodes the input/output specifications of the task. A task is *finite* if its input complex is finite.

Let δ be a protocol in the iterated immediate snapshot model. For $t \geq 0$, it is also possible to represent the set of all configurations of δ reachable via t-round executions as a simplicial complex, $\mathbb{C}^t$. Specifically, each such configuration C is represented by an n-vertex simplex $\sigma \subseteq \mathbb{C}^t$ whose vertices are the states of the processes in C. The *color* of a vertex is the identifier of the process whose state it represents. We say that a vertex is *terminated* if it represents the state of a terminated process. Otherwise, we say that it is *unterminated*.

Given a simplex $\sigma \in \mathbb{C}^t$ with no terminated vertices, its *chromatic subdivision*, $\chi(\sigma, \delta)$, is the the subcomplex of $\mathbb{C}^{t+1}$ induced by the vertex set $V(\chi(\sigma, \delta)) = \{(i, \tau) \mid \tau \subseteq \sigma \text{ is a face that contains a vertex with color } i\}$. The simplices of $\chi(\sigma, \delta)$ are the nonempty subsets of $V(\chi(\sigma, \delta))$ with the property that, for any two vertices (i, τ) and (j, τ') that belong to the same subset, $i \neq j$ and either $\tau \subseteq \tau'$ or $\tau' \subseteq \tau$.

More generally, suppose σ is an arbitrary simplex in $\mathbb{C}^t$. Let $W = \{s \in V(\sigma) \mid \delta(s) \neq \perp\}$ be the terminated vertices in σ and let $\tau \subseteq \sigma$ be the simplex induced by the unterminated vertices of σ. If $W \neq \emptyset$, then $\chi(\sigma, \delta)$ is the simplicial complex whose vertex set is $W \cup V(\chi(\tau, \delta))$ and whose faces are of the form $W' \cup \rho$, where $W' \subseteq W$ and $\rho \in \chi(\tau, \delta)$ is a simplex.

Note that, if $\sigma' \subseteq \sigma$, then $\chi(\sigma', \delta) \subseteq \chi(\sigma, \delta)$. Moreover, if σ represents a round-based configuration, C, and $Q = \{p_i \mid i \text{ is the first component of some vertex in } \sigma'\}$, then each facet of $\chi(\sigma', \delta)$ is the set of states of processes in Q at the end of some 1-round, Q-only execution from C. (The processes in Q that are terminated in C do not take steps in this execution.) In particular, each n-vertex simplex of $\chi(\sigma, \delta)$ represents a configuration obtained from C via a 1-round execution.

For each subcomplex $\mathbb{A} \subseteq \mathbb{C}^t$, we define $\chi(\mathbb{A}, \delta) = \cup\{\chi(\sigma, \delta) \mid \sigma \in \mathbb{A} \text{ is a simplex}\} \subseteq \mathbb{C}^{t+1}$. Note that, if $\mathbb{A}' \subseteq \mathbb{A}$ is a subcomplex, then $\chi(\mathbb{A}', \delta) \subseteq \chi(\mathbb{A}, \delta)$. Hoest and Shavit [18] showed that $\mathbb{C}^{t+1} = \chi(\mathbb{C}^t, \delta)$, for all $t \geq 0$. Given a subcomplex $\mathbb{A} \subseteq \mathbb{C}^t$, we use $\chi^0(\mathbb{A}, \delta)$ to denote $\mathbb{A}$ and, for $k \geq 0$, we use $\chi^{k+1}(\mathbb{A}, \delta)$ to denote $\chi(\chi^k(\mathbb{A}, \delta), \delta) \subseteq \mathbb{C}^{t+k+1}$. In particular, for all $t \geq 0$, $\mathbb{C}^t = \chi^t(\mathbb{I}, \delta)$. For any $k \geq 0$ and any n-vertex simplex $\sigma \subseteq \mathbb{C}^t$ representing a configuration C reachable via a t-round execution, the set of all configurations reachable from C via k-round executions is represented by $\chi^k(\sigma, \delta)$.

For any n-vertex simplex $\sigma \in \mathbb{I}$, each $(n-1)$-vertex simplex $\tau \in \chi^t(\sigma, \delta)$ is a face of at most two n-vertex simplices in $\chi^t(\sigma, \delta)$ (see [16], Chapter 9). If τ is a face of exactly one n-vertex simplex in $\chi^t(\sigma, \delta)$, then we say that τ is *on the boundary* of $\chi^t(\sigma, \delta)$. Otherwise, we say that τ is *in the interior* of $\chi^t(\sigma, \delta)$.

A protocol δ for $\mathbb{T}$ is *safe* if the outputs of the terminated processes in each execution from an initial configuration represented by σ are represented by a simplex contained in $\Delta(\sigma)$. Note that a safe protocol is not necessarily wait-free. In particular, a protocol in which no process terminates is safe. Finally, δ is uniform if there is a constant M such that, in every execution, every process terminates after performing exactly M update operations and scan operations.

2.3 Colorless computation

In a *colorless* protocol, processes make decisions ignoring all information about process identifiers and the number of times the same information appears in a scan. The *colorless states* of processes are defined inductively from their states: If the initial state of process p_i is (i, x_i), then its initial colorless state is simply its input value, x_i. Assume that each process that has not terminated after its $(t-1)$'th step has a well-defined colorless state. Then, for each such process p_i, its new colorless state after obtaining a scan result r from snapshot object S_t is the set $\{c_j \mid r[j] \neq -$ and c_j is the colorless state of $r[j]\}$. In particular, a colorless state does not record the number of times each colorless state at the end of the previous round occurs in the result of a scan. In general, it is not possible to determine the previous colorless states of a process from its current colorless state. Thus, colorless protocols are not full-information protocols.

Given a configuration C of δ, its *colorless configuration*, D, is the set of colorless states of the process states in C. A colorless configuration D' is *reachable from* D if there exists a configuration reachable from C whose colorless configuration is D'. A colorless configuration is *reachable* if it is reachable from an initial colorless configuration.

Executions of a colorless protocol have a simpler combinatorial representation using simplicial complexes. For each $t \geq 0$, the simplicial complex, $\mathbb{B}^t$ represents the set of all colorless configurations of δ reachable via t-round executions. Specifically, each such colorless configuration is a facet of $\mathbb{B}^t$. If σ contains no terminated vertex, then the *barycentric subdivision*, $\beta(\sigma, \delta)$, of σ is a simplicial complex whose vertices are the non-empty subsets of $V(\sigma)$. Its simplices are the nonempty subsets of $V(\beta(\sigma, \delta))$ with the property that, for any two vertices τ and τ' that belong to the same subset, either $\tau \subseteq \tau'$ or $\tau' \subseteq \tau$. The *barycenter* of $\beta(\sigma, \delta)$ is the vertex $V(\sigma)$. If $\mathbb{A}$ is a simplicial complex such that $\delta(v) = \perp$ for all $v \in V(\mathbb{A})$, then the *barycentric subdivision*, $\beta(\mathbb{A}, \delta)$, of $\mathbb{A}$ is the union of $\beta(\sigma, \delta)$ for all simplices $\sigma \subseteq \mathbb{A}$. For *uniform* colorless protocols, δ, it is known that $\mathbb{B}^{t+1} = \beta(\mathbb{B}^t, \delta)$. (See Chapter 4 of [16].)

More generally, suppose σ is a simplex representing an arbitrary colorless configuration reachable via a t-round execution. Let $W = \{s \in V(\sigma) \mid \delta(s) \neq \perp\}$ be the terminated vertices in σ and let $\tau \subseteq \sigma$ be the simplex induced by the unterminated vertices of σ. If $W \neq \emptyset$, then $\beta(\sigma, \delta)$ is the simplicial complex whose vertex set is $W \cup V(\beta(\tau, \delta))$ and whose faces are of the form $W' \cup \rho$, where $W' \subseteq W$ and $\rho \subseteq \beta(\tau, \delta)$ is a simplex. Using essentially the same argument as in Hoest and Shavit [18], it is possible to show that $\mathbb{B}^{t+1} = \beta(\mathbb{B}^t, \delta)$, for all $t \geq 0$. Furthermore, β maps every subcomplex $\mathbb{A} \subseteq \mathbb{B}^t$ to a subcomplex $\beta(\mathbb{A}, \delta) \subseteq \mathbb{B}^{t+1}$. Given a subcomplex $\mathbb{A} \subseteq \mathbb{B}^t$, we use $\beta^0(\mathbb{A}, \delta)$ to denote $\mathbb{A}$ and, for $k \geq 0$, we use $\beta^{k+1}(\mathbb{A}, \delta)$ to denote $\beta(\beta^k(\mathbb{A}, \delta), \delta) \subseteq \mathbb{B}^{t+k+1}$.

2.4 Extension-based proofs and augmented extension-based proofs

Consider any task $T = (\mathbb{I}, \mathbb{O}, \Delta)$ among $n \geq 2$ processes that has no wait-free solution in the iterated immediate snapshot model. An *extension-based proof* of (the impossibility of) T is an interaction between a *prover* and a *protocol* δ that claims to solve T. The goal of the

prover is to find a *bad execution* of the protocol: an execution that is either infinite in length or one that where processes output values that violate the task specification. The interaction proceeds in *phases*.

At the start of each phase $\varphi \geq 1$, the prover has a set, $A(\varphi)$, of reachable configurations of protocol δ. These configurations differ from one another only in the states of processes that are still in their initial states. (In other words, the prover has committed to an execution and a set of initial configurations that are indistinguishable from the first configuration in the execution to all processes that appear in the execution.) $A(1)$ is the set of all initial configurations of δ. Throughout phase φ, the prover maintains a set of configurations, $A'(\varphi)$, that it has reached from configurations in $A(\varphi)$. This set is empty at the start of phase φ.

In each phase φ, the prover can ask queries to find out information about the protocol. For a *single-step query*, the prover specifies a configuration $C \in A(\varphi) \cup A'(\varphi)$ and a nonempty set Q of processes that are poised to update the same snapshot object in configuration C. The protocol responds to this query with information about the configuration $C' = CQ$. More specifically, for each process in Q, the protocol returns $\delta(v)$, where v is the state of that process in configuration C'. This is sufficient information for the prover to know everything about configuration C'. Then the prover adds C' to $A'(\varphi)$. If the outputs in configuration C' do not satisfy the specifications of T, then the prover wins, since it has demonstrated that δ is incorrect.

A *chain of (single-step) queries* is a (finite or infinite) sequence of consecutive single-step queries $(C_1, Q_1), (C_2, Q_2), \ldots$ such that $C_1, Q_1, C_2, Q_2, \ldots$ is an execution from C_1. If the prover constructs an infinite chain of queries, it wins, since it has demonstrated that δ is not wait-free.

For an *output query*, the prover specifies a configuration $C \in A(\varphi) \cup A'(\varphi)$, a set Q of processes that are poised to update the same snapshot object in configuration C, and an output value y. If there is a Q-only execution from C that ends in a configuration in which some process in Q outputs y, then the protocol responds with the sequence of sets of processes scheduled in some such execution. Otherwise, the protocol says that no such execution exists.

An *assignment query* is a generalization of an output query. Instead of specifying an output value y, the prover now needs to specify a function $f : Q' \to V(\mathbb{O})$, where Q' is some non-empty subset of Q. Similar to an output query, if there is a Q-only execution from C that ends in a configuration in which every process $p \in Q'$ outputs $f(p)$, then the protocol responds with the sequence of sets of processes scheduled in some such execution. Otherwise, the protocol says that no such execution exists.

The prover must end the current phase after asking a finite number of output queries and chains of queries. To end phase φ, the prover chooses a configuration $C \in A'(\varphi)$. If all processes are terminated in configuration C, then the prover loses. Otherwise, phase $\varphi + 1$ begins, with $A(\varphi + 1)$ consisting of C and any configuration that differs from C only in the states of processes that are still in their initial states.

The prover wins if there are an infinite number of phases in the interaction, since it has demonstrated that the protocol is not wait-free. It also wins if the protocol gives it inconsistent responses.

2.5 k-exhaustive extension-based proofs

For a *length query*, the prover specifies a configuration $C \in A(\varphi) \cup A'(\varphi)$ and a nonempty set Q of processes that are poised to update the same snapshot object in configuration C. The protocol responds with an upper bound on the length of a longest Q-only execution from C, or ∞, if there is an infinite Q-only execution from C.

A k-exhaustive extension-based proof is a generalization of an (augmented) extension-based proof that allows the prover to ask length queries for sets of processes of size at most k, in addition to chains of queries and assignment queries.

It is easy to obtain an n-exhaustive extension-based proof of the impossibility of any task among $n \geq 2$ processes that has a finite input complex, but no deterministic, wait-free solution. For each initial configuration C, the prover performs a length query for C and the set of all processes. If the protocol's response to this query is ∞, then a length query for some one-round extension of C and the set of all processes must also return ∞. In this case, the prover will be able to inductively construct an infinite execution of the protocol, which demonstrates that the protocol is not wait-free. If the protocol's response is r, the prover performs all possible chains of queries from C. If any of these chains continues for more than r steps, the prover wins, since the protocol has provided inconsistent information. For each configuration reached in which some processes are terminated, the prover checks that the outputs of these processes satisfy the specifications of the task. If not, the prover wins, since the protocol is incorrect. If at any point during this interaction the prover receives conflicting information (for example, a process producing different outputs in two configurations that are indistinguishable to this process), the prover wins. Since the protocol is not a deterministic, wait-free solution to the task, one of these conditions must eventually happen.

3 (n-1)-Exhaustive Extension-based Proofs are Powerful

In this section, we consider tasks among $n \geq 2$ processes that have finite input complexes. We show that, if there is no deterministic, wait-free solution to such a task, then its impossibility can be proved using an $(n-1)$-exhaustive extension-based proof.

Consider any task $\mathbb{T} = (\mathbb{I}, \mathbb{O}, \Delta)$ among $n \geq 2$ processes and any $(n-1)$-obstruction-free protocol δ with input complex $\mathbb{I}$. Let σ be an n-vertex simplex representing a round-based configuration C of the protocol δ. Since all executions from C that involve at most $n-1$ processes are finite, König's Lemma implies that there is a finite upper bound, r, on the length of any such execution. In this case, the *boundary assignment* $\partial(\sigma, \delta)$ of σ (or of C) is the subcomplex $\chi^r(\partial\sigma, \delta)$ of $\chi^r(\sigma, \delta)$ induced by the state of every process in P at the end of every r-round P-only execution from C, for all subsets P of $n-1$ processes. Each of these states, v, is terminated, so there is an output $\delta(v)$ assigned to it. In the case that δ is not $(n-1)$-obstruction-free, some round-based configurations may not have boundary assignments.

In an $(n-1)$-exhaustive extension-based proof, the prover has the ability to determine the boundary assignment of any round-based configuration C it has reached. To do so, it first performs length queries on C with respect to every possible subset of $n-1$ processes. If any query returns ∞, then C does not have a boundary assignment. Otherwise, there is an upper bound on the length of any execution from C involving at most $n-1$ processes. Hence, there are only finitely many such executions, so the prover can perform finitely many chains of step queries to determine the output assignment produced in each such execution.

All provers we describe in this section will follow the same general strategy. At the beginning of each phase, the prover has a set S of round-based configurations. For the first phase, S is the set of all initial configurations. The prover first attempts to compute the boundary assignment of each configuration in S using length queries. If any configuration C in S does not have a boundary assignment, then the prover knows that some extension of C leads to an infinite execution. In this case, the prover chooses C and ends the phase. In the next phase, S will be the set of all 1-round extensions of C. Otherwise, the prover

checks whether there is a configuration in S whose boundary assignment satisfies a certain task-specific bad boundary property. If so, then the prover chooses one such configuration C' and ends the phase. If C' is not a terminal configuration, then S will be the set of all 1-round extensions of C' in the next phase. For tasks that have no deterministic, wait-free solutions, we show that there will always be at least one configuration in S that does not have a boundary assignment or has a boundary assignment that satisfies the task-specific bad boundary property. The extension-based prover in our strategy will either produce an infinite execution of the protocol or reach a terminal configuration that satisfies the bad boundary property, which will guarantee that the outputs of the processes violate the task specification.

We begin with two examples, the 2-set agreement task and zero-one exclusion tasks among three processes. For each of these, we give a simple and computable bad boundary property. Combined with the general construction described above, we get extension-based proofs for the impossibility of these tasks. We then formally define the notion of bad boundary properties for tasks and conclude the section with a sketch of the proof for Theorem 4.

3.1 Extension-based proofs for set agreement tasks

Consider the 2-set agreement task among $n = 3$ processes. Let $\mathbb{I}$ denote the input complex for this task and let δ be an arbitrary protocol with input complex $\mathbb{I}$. Note that a round-based configuration is represented by a triangle and its boundary assignment (if it has one) can be obtained by performing chromatic subdivisions on its edges.

For the 2-set agreement task, we call an edge in a boundary assignment *special* if one of its endpoints is assigned output 0 by δ and its other endpoint is assigned output 1 by δ. Let B be the property such that, for any n-vertex simplex σ representing a round based configuration C that has a boundary assignment under δ, $B(\partial(\sigma, \delta))$ is true if and only if some output of $\partial(\sigma, \delta)$ does not satisfy validity or $\partial(\sigma, \delta)$ has an odd number of special edges.

Consider the initial configuration C represented by the simplex $\sigma = \{v_0, v_1, v_2\}$, where $v_i = (p_i, i)$ represents process p_i with input value i, for each $i \in \{0, 1, 2\}$. Suppose that C has a boundary assignment $\partial(\sigma, \delta)$ that satisfies validity. Vertices in $\partial(\sigma, \delta)$ that represent states of processes in terminating $\{p_i, p_j\}$-only executions are assigned either i or j as outputs. It follows that all special edges in $\partial(\sigma, \delta)$ lie on the subdivision of the edge $\{v_0, v_1\}$ of σ. The two vertices at the ends of this subdivided edge represent solo-executions by p_0 and by p_1, which must output 0 and 1, respectively. It follows that the outputs of the vertices along this subdivided edge switch between 0 and 1 an odd number of times, which gives an odd number of special edges.

In the full version of the paper, we show that any non-terminal round-based configuration whose boundary assignment satisfies B has a one-round extension whose boundary assignment also satisfies B. This is proved via a cancellation argument similar to the proof of two-dimensional Sperner's lemma [6].

Finally, if a configuration C' is terminal, then its boundary assignment defines the outputs of its three vertices. Notice that if σ' is the simplex representing C', then it is impossible for σ' to have three special edges. Furthermore, in the case that σ' has exactly one special edge, the set of outputs produced by the processes must be $\{0, 1, 2\}$, which violates agreement. Hence, if $B(\partial(\sigma', \delta))$ is true, then the outputs of processes in configuration C' violate either validity or agreement.

More generally, for $(n-1)$-set agreement with $n \geq 2$, we can define B to be the predicate such that the boundary assignment of a round-based configuration satisfies B if and only if its outputs do not satisfy validity for $(n-1)$-set agreement or it contains an odd number of $(n-1)$-vertex simplices whose vertices are assigned different values in $\{0, 1, \ldots, n-2\}$ by δ.

3.2 Extension-based proofs for zero-one exclusion tasks

In the *weak symmetry breaking* task, processes do not have inputs. Each process that does not crash must output either 0 or 1, with the restriction that, if no processes crash, at least one process must output 0 and at least one process must output 1. It is known that weak symmetry-breaking has a wait-free protocol from registers when the number of processes, n, is not a prime power [10].

The class of zero-one exclusion tasks was introduced by Gafni [14]. Each of the tasks in this class further restricts the outputs when some of the processes do not participate (i.e. take no steps). Thus, any solution to a zero-one exclusion task is a solution to weak symmetry breaking. More specifically, a zero-one exclusion task among $n \geq 2$ processes is specified by a sequence of $n-1$ bits $b(1), b(2), ..., b(n-1)$. In any execution where exactly $0 < k < n$ processes participate, the task requires that not all k processes output $b(k)$. Gafni gave a reduction from any zero-one exclusion task to set agreement and proved that some zero-one exclusion tasks are strictly weaker than set agreement. Gafni and Herlihy [15] gave a characterization of zero-one exclusion tasks that have wait-free protocols from registers using combinatorial topology.

We consider the zero-one exclusion tasks among 3 processes. The input complex $\mathbb{I}$ consists of a single triangle, $\sigma_0 = \{p_0, p_1, p_2\}$, where each vertex represents an inputless process. For $b(1) = b(2) = 0$, the task $\mathbb{E}_{00} = (\mathbb{I}, \mathbb{O}_{00}, \Delta_{00})$ requires that at least one process output 1 in any execution and, if all 3 processes participate, then at least one process must also output 0. In other words, a process must output 1 in a solo-terminating execution, if 2 processes run by themselves, they cannot both output 0, and if all 3 processes produce output values, both 0 and 1 are output. The task $\mathbb{E}_{01} = (\mathbb{I}, \mathbb{O}_{01}, \Delta_{01})$ is defined similarly. A process must output 1 in a solo-terminating execution, if 2 processes run by themselves, they cannot both output 1, and if all 3 processes participate, both 0 and 1 are output. There are two other zero-one exclusion tasks among 3 processes that can be obtained from $\mathbb{E}_{00}$ and $\mathbb{E}_{01}$ by interchanging 0 and 1 and are computationally equivalent to them.

Let δ be an arbitrary protocol with input complex $\mathbb{I}$ and binary output values. For each vertex v that represents the state of a terminated process p_i, define its *label* (with respect to δ) to be the quantity $i + \delta(v) \mod 3$. We call an edge of a boundary assignment *special* if one of its endpoints has label 0 and its other endpoint has label 1. Define the property B_{00} to be such that, for any n-vertex simplex σ representing a round-based configuration C that has a boundary assignment under δ, $B_{00}(\partial(\sigma, \delta))$ is true if and only if the output values in $\partial(\sigma, \delta)$ do not satisfy the specification of the task $\mathbb{E}_{00}$ or $\partial(\sigma, \delta)$ has an odd number of special edges.

Suppose that σ_0 has a boundary assignment $\partial(\sigma_0, \delta)$ that satisfies specification of $\mathbb{E}_{00}$. Each edge $\{p_i, p_j\}$ of σ_0 is subdivided into an odd number of edges in the protocol complex. This is because the vertices along the subdivided edge must alternate between states of p_i and states of p_j, where the vertices on the ends of the path represent states of p_i and p_j at the end of their solo-executions. From the specification of E_{00}, these two vertices both output 1. Furthermore, no edge in $\partial(\sigma_0, \delta)$ has output 0 at both of its endpoints. Hence, if there is an edge $\{u, v\}$ in the subdivision of $\{p_i, p_j\}$ such that $\delta(u) = 0$ and $\delta(v) = 1$, then there is an another edge $\{u, w\}$ in this subdivision such that $\delta(w) = 1$. The remaining edges in the subdivision of $\{p_i, p_j\}$ are assigned output 1 at both endpoints. There are an odd number of such edges.

In the subdivision of $\{p_0, p_1\}$, an edge is special if and only if both endpoints output 0, which does not occur. In the subdivision of $\{p_1, p_2\}$, an edge is special if and only if the endpoint corresponding to p_2 outputs 1 and the endpoint corresponding to p_1 outputs 0. As

above, such a special edge is adjacent to another special edge in the subdivision of $\{p_1, p_2\}$. This gives an even number of special edges in the subdivision of $\{p_1, p_2\}$. In the subdivision of $\{p_0, p_2\}$, an edge is special if and only if both processes output 1. As above, there are an odd number of such edges. In total, this gives an odd number special edges in $\partial(\sigma_0, \delta)$.

The argument used for set-agreement can also be applied here to show that any non-terminal round-based configuration whose boundary assignment satisfies B_{00} has a 1-round extension whose boundary assignment also satisfies B_{00}.

Finally, if σ' is the triangle that represents a terminal configuration, then it is impossible for σ' to have three special edges. In the case that σ' has exactly one special edge, the set of labels on its vertices must be $\{0, 1, 2\}$. This is only possible if all three processes output 0 or if all three processes output 1, which violates the task specification, as it fails to break symmetry.

3.3 Boundary properties and extension-based proofs for general tasks

We now generalize the constructions described in Section 3.1 and Section 3.2 to arbitrary tasks with no deterministic wait-free solutions.

Given a task $\mathbb{T} = (\mathbb{I}, \mathbb{O}, \Delta)$, a *boundary property* of $\mathbb{T}$ is a predicate B that maps each boundary assignment $\partial(\sigma, \delta)$ to true or false, where δ is any protocol with input complex $\mathbb{I}$ and σ is any n-vertex simplex σ that represents a round-based configuration of the protocol δ. For proving impossibility results about tasks, we need a *bad* boundary property, B, which means that for any protocol δ with input complex $\mathbb{I}$, B is initially bad, extendable, and dangerous, where

- B is *initially bad* if there is an initial configuration that either has no boundary assignment or has a boundary assignment that satisfies B,
- B is *extendable* if every boundary assignment of a non-terminal round-based configuration that satisfies B has a one-round extension that either has no boundary assignment or has a boundary assignment that satisfies B, and
- B is *dangerous if, for every* boundary assignment of a terminal configuration that satisfies B, the outputs of its processes violate the specification of $\mathbb{T}$.

We prove that, for any finite task $\mathbb{T} = (\mathbb{I}, \mathbb{O}, \Delta)$ among $n \geq 2$ processes with no deterministic, wait-free solution, there is a bad boundary property $B_{\mathbb{T}}$ for $\mathbb{T}$. Then Theorem 4 follows from the proof strategy described at the beginning of Section 3.

Consider an arbitrary finite task $\mathbb{T} = (\mathbb{I}, \mathbb{O}, \Delta)$ among $n \geq 2$ processes. If δ is a protocol with input complex $\mathbb{I}$ and C is a round-based configuration reachable from some initial configuration of δ, then we say that δ *locally solves* $\mathbb{T}$ *from* C if δ has no infinite execution starting from C and the outputs produced in all executions starting from C satisfy the specification of $\mathbb{T}$. We define the *canonical bad boundary property* $B_{\mathbb{T}}$ of $\mathbb{T}$ to be the predicate such that, for each protocol δ with input complex $\mathbb{I}$ and for each n-vertex simplex σ that represents a round-based configuration C of δ that has a boundary assignment, $B_{\mathbb{T}}(\partial(\sigma, \delta))$ is true if and only if

- the outputs in $\partial(\sigma, \delta)$ do not satisfy the specification of $\mathbb{T}$ or
- the outputs in $\partial(\sigma, \delta)$ satisfy the specification of $\mathbb{T}$, but there is no protocol δ' that locally solves $\mathbb{T}$ from C for which $\partial(\sigma, \delta') = \partial(\sigma, \delta)$.

In particular, if σ represents a configuration consisting of only terminated processes, then $B_{\mathbb{T}}(\partial(\sigma, \delta))$ is true if and only if the outputs of the vertices of σ do not satisfy the specification of $\mathbb{T}$. Thus, $B_{\mathbb{T}}$ is dangerous, by definition.

The next lemma shows that, if the boundary assignments of a finite set of round-based configurations of δ do not satisfy $B_{\mathbb{T}}$, then there exists a protocol that locally solves $\mathbb{T}$ simultaneously from all these configurations.

▶ **Lemma 1.** *Consider a task $\mathbb{T} = (\mathbb{I}, \mathbb{O}, \Delta)$, a protocol δ with input complex $\mathbb{I}$, and a finite set D of configurations of δ in which each process has taken exactly t steps or has terminated before doing so. If each configuration in D has a boundary assignment under δ that does not satisfy $B_{\mathbb{T}}$, then there is a protocol δ' that locally solves $\mathbb{T}$ from every configuration in D such that the boundary assignment of every configuration in D is the same under both δ and δ'.*

In particular, if D is the set of all initial configurations and each of them has a boundary assignment that does not satisfy $B_{\mathbb{T}}$, then there is a wait-free protocol for $\mathbb{T}$.

▶ **Corollary 2.** *Let $\mathbb{T}$ be a finite task that has no wait-free protocol. Then every protocol for $\mathbb{T}$ has an initial configuration whose boundary assignment satisfies $B_{\mathbb{T}}$.*

The next lemma shows that $B_{\mathbb{T}}$ is extendable.

▶ **Lemma 3.** *Let $\mathbb{T} = (\mathbb{I}, \mathbb{O}, \Delta)$ be a task, let δ be a protocol with input complex $\mathbb{I}$, and let C be a configuration whose boundary assignment under δ satisfies $B_{\mathbb{T}}$. Then there is a one-round extension of C that either has no boundary assignment under δ or has a boundary assignment under δ that satisfies $B_{\mathbb{T}}$.*

Our main result in this section follows from Corollary 2, Lemma 3, and the fact that canonical bad boundary properties are dangerous.

▶ **Theorem 4.** *For any finite task among $n \geq 2$ processes with no deterministic, wait-free solution, there is an $(n-1)$-exhaustive extension-based proof that it has no deterministic, wait-free solution.*

The property B defined for 2-set agreement in Section 3.1 implies the canonical bad boundary property for 2-set agreement. To see why, let σ be a triangle representing a round-based configuration C in a protocol δ. If the boundary assignment of σ has an odd number of 0-1 edges, then, by a generalization of Sperner's lemma, any wait-free protocol δ' that agrees with δ on the boundary assignment of σ must have at least one execution starting from C that leads to a configuration in which three different values are output by the three processes. In other words, there is no protocol that locally solves 2-set agreement from C that also agrees with δ on the boundary assignment of C.

In general, to show that a finite task has no deterministic, wait-free solution, it suffices to define an explicit boundary property for the task and prove that this boundary property is initially bad, extendable, and dangerous. In the examples for set-agreement and zero-one exclusion tasks, the bad boundary properties we gave are computable. However, this is not the case for the canonical bad boundary properties defined in this section. Furthermore, a bad canonical boundary property is shown to be bad under the assumption that the task is impossible. Thus, the results we presented in this section do not immediately yield impossibility results for arbitrary tasks.

4 A Hierarchy of Extension-Based Proofs

We show that, for all $n \geq 3$ and for all $k \in \{2, \ldots, n-1\}$, $(k-1)$-exhaustive extension-based proofs are insufficient to reliably locate infinite executions in k-obstruction-free protocols that solve k-set agreement among n processes. In particular, for each $(k-1)$-exhaustive extension-based prover, we describe an adaptively built k-obstruction-free colorless protocol that defeats it.

▶ **Theorem 5.** *For any $n \geq 3$, any $k \in \{2, \ldots, n-1\}$, and any $(k-1)$-exhaustive extension-based prover, there is a correct k-obstruction-free colorless protocol for k-set agreement among $n > k$ processes for which the prover does not commit to a prefix of an infinite execution.*

This result shows that $(k-1)$-exhaustive extension-based proofs are also insufficient to establish the impossibility of a deterministic, wait-free protocol for k-set agreement among n processes.

▶ **Theorem 6.** *For all integers $n \geq 3$ and $k \in \{2, \ldots, n-1\}$, there is no $(k-1)$-exhaustive extension-based proof of the impossibility of a deterministic, wait-free protocol for k-set agreement among n processes.*

For $k = n-1$, Theorem 6 implies that there is no $(n-2)$-exhaustive extension-based proof for the impossibility of $(n-1)$-set agreement among n processes. Therefore, Theorem 4 is tight in terms of how exhaustive extension-based proofs need to be in order to prove the impossibility of an arbitrary task that has no deterministic, wait-free solution.

In the rest of this section, we give most of the construction used to prove Theorem 5. In Section 4.1, we explain how the adversarial protocol responds to queries in phase 1. Once the prover commits to a non-empty execution, we discuss how to define the protocol. This is done in Section 4.2. Complete proofs can be found in the full version of the paper.

4.1 The first phase

The argument by Alistarh, Aspnes, Ellen, Gelashvili and Zhu [1, 2] that there is no extension-based proof for $(n-1)$-set agreement among n processes constructed a partially defined protocol δ while maintaining the invariant that states which output different values are sufficiently far away from one another. This is not possible for ℓ-exhaustive extension-based proofs when $\ell \geq 2$. For example, the prover can ask length queries for all sets of 2 processes from all initial configurations and then perform all possible chains of queries involving at most 2 process from all initial configurations. The set of states reached form a single connected component of the protocol complex and include states in which each of the values in $\{0, 1, \ldots, n-1\}$ is output.

For simplicity, we will construct δ so that it is colorless. This is without loss of generality, as the prover must be able to handle arbitrary protocols. As we respond to the prover's queries during the first phase, we update δ while maintaining a number of invariants.

First, we ensure that δ is *defined* for every colorless state s in every colorless configuration that the prover has reached, i.e., $\delta(s)$ is either set to $\perp$ or an output value. If $\delta(s)$ is defined for some colorless state s reachable via an r-round execution, then $\delta(s')$ is defined for every colorless state s' reachable via an $(r-1)$-round execution. Moreover, if $\delta(s) = \perp$, then $\delta(s')$ is defined for every colorless state s' reachable via an r-round execution. Hence, there is an integer $t \geq 0$ such that δ is defined on all colorless states reachable via r-round executions, for all $0 \leq r < t$, and δ is only defined on terminated colorless states reachable via t-round executions.

More formally, let $\mathbb{B}^0$ be the simplicial complex representing the set of all initial colorless configurations of the k-set agreement task, i.e., $\mathbb{B}^0$ consists of an n-vertex simplex with one vertex corresponding to each of the possible inputs, as well as all faces of this simplex.

▶ **Invariant 1.** *There is an integer $t \geq 0$ such that δ is defined on every vertex in $\mathbb{B}^r = \beta^r(\mathbb{B}^0, \delta)$, for all $0 \leq r < t$, and $\delta(v)$ is either undefined or $\delta(v) \neq \perp$ for each vertex v in $\mathbb{B}^t = \beta^t(\mathbb{B}^0, \delta)$. If D is the colorless configuration of a configuration C reached by the prover, then δ is defined on every colorless state in D.*

In the first phase, we ensure that δ appears to be solving k-set agreement, from the perspective of the prover. In particular, we ensure that, in any colorless configuration reached by the prover, there are at most k different output values, each of which is valid. Additionally, if a colorless configuration reached by the prover has at least one non-terminated colorless state, then there are at most $k-1$ different output values. This is crucial when we fully define δ, at the end of the first phase.

Now consider colorless configurations reachable via t-round executions that each contains a colorless state on which δ is undefined and another colorless state on which δ outputs a value. We further ensure that it is possible to assign valid output values to the undefined colorless states in these configurations so that at most $k-1$ values are output in any of these configurations. Note that this will still leave many parts of the protocol undefined.

More formally, let $t \geq 0$ be the integer in Invariant 1, and let U be an arbitrary set of vertices in $\mathbb{B}^t$ on which δ is undefined. The *extension* of δ by a function $\gamma : U \to \{0, 1, \ldots, n-1\}$ is the protocol δ' such that $\delta'(u) = \gamma(u)$, for each vertex in $u \in U$, and $\delta'(v) = \delta(v)$, for each vertex $v \notin U$ on which $\delta(v)$ is defined. We say δ' is *good* if, in every colorless configuration represented by a simplex $\sigma \in \mathbb{B}^t$, the protocol outputs at most k different values, each of which is valid, and furthermore, if σ has at least one unterminated vertex, then the protocol outputs at most $k-1$ different values.

▶ **Invariant 2.** *There is a good extension δ' of δ by a function $\gamma : N \to \{0, 1, \ldots, n-1\}$, where N is the set of* undefined *neighbours of the terminated vertices in $\mathbb{B}^t$.*

At any point during the interaction, we can *prolong the protocol* by setting $\delta(v) = \perp$, for every undefined vertex $v \in \mathbb{B}^t$. It is one way of responding to a query and sometimes it is the only thing we can do. Intuitively, it provides us more room to assign outputs in future rounds, similar to how it was used in [2].

▶ **Lemma 7.** *Prolonging the protocol maintains the invariants.*

Finally, we describe our strategy for responding to queries so that the invariants are maintained. Initially, before any queries are made, $\delta(s) = \perp$ for every initial colorless state s and is undefined for every other colorless state. The invariants hold with $t = 1$. In particular, Invariant 2 is vacuously true.

Now suppose that the invariants hold immediately before a query by the prover. Recall that each query involves a previously reached configuration C and a set of (at most $k-1$) processes Q that are all poised to access the same snapshot object S_{r+1} in C, for some $r \geq 0$ (so all processes in Q have taken exactly r steps). Let σ be the set of colorless states of processes in Q in configuration C. Then σ is a simplex in $\mathbb{B}^r$. Let $t \geq 0$ be the integer in Invariant 1 immediately before the query. Since every process in Q is poised to access the same snapshot object, $\delta(v) = \perp$ for every $v \in V(\sigma)$, which implies that $r < t$. We consider step queries, length queries, and assignment queries separately.

Case 1: (C, Q) is a step query. Since the processes in Q perform their scans on S_{r+1} consecutively after they each perform their update on S_{r+1}, they all obtain the same result and, hence, have the same colorless state s in CQ. By definition, s is a vertex in $\mathbb{B}^{r+1}$. If $\delta(s)$ is defined, then the protocol replies with $\delta(s)$. Now suppose that $\delta(s)$ is undefined. By Invariant 1, $r + 1 = t$. If we can assign an output to s while maintaining Invariant 2, then we update $\delta(s)$ to any such value, which also maintains Invariant 1. Otherwise, we prolong the protocol. By doing so, $\delta(s)$ is set to $\perp$. In either case, $\delta(s)$ is now defined and the protocol replies with $\delta(s)$. We prove that every chain of step queries asked by the prover results in a finite execution of the protocol.

Case 2: (C, Q) is a length query. By definition, $V(\beta^{t-r}(\sigma, \delta))$ is the set of all colorless states of processes in Q in configurations reachable from C via $(t-r)$-round Q-only executions. If every vertex in $\beta^{t-r}(\sigma, \delta)$ has been assigned an output, then the protocol replies with $t-r$, which is an upper bound on the length of a longest Q-only execution from C. Otherwise, we prolong the protocol twice. We prove that it is possible to assign outputs to all vertices in $\beta^{t-r+2}(\sigma, \delta)$ while maintaining Invariant 2. Finally, the protocol replies with $t-r+2$.

Case 3: (C, Q, f) is an assignment query, for some output assignment $f : Q' \to \{0, ..., n-1\}$ to a subset of processes $Q' \subseteq Q$. Let $|f(Q')|$ denote the number of different values in $f(Q')$ and let $|V(\sigma)|$ denote the number of vertices in σ. Since the protocol is colorless and is correct for k-set agreement, it can output at most $\min\{|V(\sigma)|, k\}$ different values in any Q-only execution starting from C. Thus, if $|f(Q')| > \min\{|V(\sigma)|, k\}$, the protocol replies that no such execution exists. So suppose that $|f(Q')| \leq \min\{|V(\sigma)|, k\}$.

If there exists a value $y \in f(Q')$ that is not valid for any vertex in σ, then y is also not valid for any vertex obtained by subdividing σ any number of times. Since we maintain Invariant 2, y will never be assigned to any such vertex. It follows that the output assignment f will not be satisfied by any Q-only execution from C. In this case, the protocol replies that no such execution exists. So suppose that every $y \in f(Q')$ is valid for some vertex in σ.

Suppose that every vertex in $\beta^{t-r}(\sigma, \delta)$ is terminated or adjacent to some terminated vertex in $\mathbb{B}^t$. Then we prolong the protocol twice. We show that, after doing so, we can assign valid outputs to every vertex in $\beta^{t-r+2}(\sigma, \delta)$ while maintaining both invariants. This completely defines the outputs of the protocol in all Q-only executions starting from C. If the assignment f is satisfied by any such execution, then the protocol replies with this execution. Otherwise, the protocol replies that no such execution exists.

Now suppose that there is an unterminated vertex in $\beta^{t-r}(\sigma, \delta)$ whose neighbours in $\mathbb{B}^t$ are all unterminated. Then there is a simplex $\tau \subseteq \beta^{t-r}(\sigma, \delta)$ that contains this vertex, has the same number of vertices as σ, and all of its vertices are unterminated. In this case, we prolong the protocol four times. Let $v = \{\{\{\tau\}\}\}$. Pick any simplex $\rho \in \beta^4(\tau, \delta)$ with the same number of vertices as τ (and σ) that contains the vertex v. Intuitively, v is the vertex at the center of $\beta^4(\tau, \delta)$ and ρ is a simplex containing this vertex. We show that there is a face $\rho' \subseteq \rho$ and a configuration D reachable from C via a Q-only execution such that $|V(\rho')| = |f(Q')|$, ρ' is the set of colorless states of processes in Q' in configuration D, and for any two processes $p, q \in Q'$, $f(p) = f(q)$ if and only if p and q share the same colorless state in configuration D. Assign outputs to the vertices of ρ' so that f is satisfied in configuration D. Additionally, pick a value $y \in f(Q')$ arbitrarily and assign y to be the output of all neighbours of the vertices of ρ' in $\mathbb{B}^{t+4}$ that are not in ρ'. We show that this output assignment maintains both invariants. In this case, the protocol responds with the Q-only execution that starts from C and leads to D.

4.2 The prover eventually loses

Since the prover does not win in the first phase, it must eventually end the first phase and commit to a non-empty execution α.

By Invariant 1, there is an integer $t \geq 0$ such that δ is defined on every vertex in $\mathbb{B}^r$ for all $0 \leq r < t$ and $\delta(v) \neq \perp$ for any vertex v in $\mathbb{B}^t$. We first prolong the protocol twice to create $\mathbb{B}^{t+2}$, so that each terminated vertex in $\mathbb{B}^{t+2}$ is a terminated vertex in $\mathbb{B}^t$. Let N be the set of undefined neighbours of the terminated vertices in $\mathbb{B}^{t+2}$. By Invariant 2, there is a good extension δ' of δ by a function $\gamma : N \to \{0, 1, \ldots, n-1\}$. For each vertex $v \in N$, set $\delta(v) = \delta'(v)$. Since δ' is good, for every reachable configuration that has an unterminated process, the processes output at most $k-1$ different output values, each of which is valid.

Let x be the input value of any process that is scheduled in the first step of α. Notice that x is a valid output value for every process in any extension that extends α. Consider any colorless state s reachable by a t-round execution that begins with α. If $\delta(s)$ is undefined, then set $\delta(s) = x$. Then every extension of α is finite. This fully defines δ for every colorless configuration the prover can reach from the second phase onwards. Note that at most k different output values are produced in any such reachable configuration. In the full version of the paper, we extend this partially defined protocol, which only *appears* to be wait-free to the prover, into a correct k-obstruction-free protocol solving k-set agreement.

5 Conclusions and Future Work

We have shown that $(n-1)$-exhaustive extension-based proofs are sufficient to establish the impossibility of any task among n processes. An easy corollary is that k-exhaustive extension-based proofs are sufficient for locating a bad execution for k-set agreement among $n > k \geq 1$ processes. Furthermore, we showed that $(k-1)$-exhaustive extension-based proofs are insufficient to establish the impossibility of k-set agreement among $n > k \geq 2$ processes. We did this by constructing a k-obstruction-free adversarial protocol for which the prover cannot find an infinite execution. This suggests that the minimum k for which there is a k-exhaustive extension-based proof establishing the impossibility of a task is a natural metric.

It would be interesting to show further limitations on techniques for proving lower bounds on the complexity of problems. It is possible to solve k-set agreement among $n \geq k+1$ processes by an obstruction-free algorithm using $n-k+1$ registers [8] or by a randomized wait-free algorithm with $n/k+k$ expected step complexity [11]. However, there are no known lower bounds on expected step complexity for $k > 1$. Ellen, Gelashvili, and Zhu [12] showed that n/k registers are necessary for obstruction-free algorithms, but they use a complicated technique that involves considering many executions. When $k = 1$, their result is tight. Zhu [20] earlier proved a lower bound of $n-1$ registers when $k = 1$. His result is obtained via a combination of a covering argument and a valency argument. One of the key lemmas in his proof shows how an extension-based prover can construct an infinite execution. We would like to show that this type of argument cannot be generalized to a space lower bound for obstruction-free k-set agreement when $k > 1$.

References

1 Dan Alistarh, James Aspnes, Faith Ellen, Rati Gelashvili, and Leqi Zhu. Why extension-based proofs fail. In *Proceedings of the 51st ACM Symposium on Theory of Computing (STOC 2019)*, pages 986–996, 2019. doi:10.1145/3313276.3316407.

2 Dan Alistarh, James Aspnes, Faith Ellen, Rati Gelashvili, and Leqi Zhu. Why extension-based proofs fail. *SIAM J. Comput.*, 52(4):913–944, 2023. doi:10.1137/20m1375851.

3 Hagit Attiya and Armando Castañeda. A non-topological proof for the impossibility of k-set agreement. *Theor. Comput. Sci.*, 512:41–48, 2013. doi:10.1016/j.tcs.2012.09.012.

4 Hagit Attiya, Armando Castañeda, and Sergio Rajsbaum. Locally solvable tasks and the limitations of valency arguments. *Journal of Parallel and Distributed Computing*, 176:28–40, 2023. doi:10.1016/j.jpdc.2023.02.002.

5 Hagit Attiya and Ami Paz. Counting-based impossibility proofs for renaming and set agreement. In *Proceedings of the 26th International Conference on Distributed Computing (DISC 2012)*, pages 356–370, 2012. doi:10.1007/978-3-642-33651-5_25.

6 J. Adrian Bondy and Uppaluri S. R. Murty. *Graph Theory with Applications*. Macmillan Education UK, 1976. doi:10.1007/978-1-349-03521-2.

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

8 Zohir Bouzid, Michel Raynal, and Pierre Sutra. Anonymous obstruction-free (n, k)-set agreement with $n - k + 1$ atomic read/write registers. *Distributed Computing*, 31:99–117, 2018. doi:10.1007/s00446-017-0301-7.

9 Kayman Brusse and Faith Ellen. Reductions and extension-based proofs. In *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC 2021)*, pages 497–507, 2021. doi:10.1145/3465084.3467906.

10 Armando Castañeda and Sergio Rajsbaum. New combinatorial topology bounds for renaming: The upper bound. *Journal of the ACM*, 59(1):3:1–3:49, March 2012. doi:10.1145/2108242.2108245.

11 Keren Censor Hillel. Multi-sided shared coins and randomized set-agreement. In *Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2010)*, pages 60–68, 2010. doi:10.1145/1810479.1810490.

12 Faith Ellen, Rati Gelashvili, and Leqi Zhu. Revisionist simulations: A new approach to proving space lower bounds. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC 2018)*, pages 61–70, 2018. doi:10.1145/3212734.3212749.

13 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

14 Eli Gafni. The 0-1-exclusion families of tasks. In *Proceedings of the 12th International Conference on Principles of Distributed Systems (OPODIS 2008)*, volume 5401 of *Lecture Notes in Computer Science*, pages 246–258, 2008. doi:10.1007/978-3-540-92221-6_17.

15 Eli Gafni and Maurice Herlihy. Sporadic solutions to zero-one exclusion tasks. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, volume 8572 of *Lecture Notes in Computer Science*, pages 1–10, 2014. doi:10.1007/978-3-662-43948-7_1.

16 Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. *Distributed computing through combinatorial topology*. Morgan Kaufmann, 2013. URL: https://store.elsevier.com/product.jsp?isbn=9780124045781.

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

18 Gunnar Hoest and Nir Shavit. Toward a topological characterization of asynchronous complexity. *SIAM Journal on Computing*, 36(2):457–497, 2006. doi:10.1137/S0097539701397412.

19 Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM Journal on Computing*, 29(5):1449–1483, 2000. doi:10.1137/S0097539796307698.

20 Leqi Zhu. A tight space bound for consensus. *SIAM Journal on Computing*, 50(3):STOC16–18–STOC16–29, 2021. doi:10.1137/16M1096785.