Why Extension-Based Proofs Fail

Dan Alistarh
IST Austria
dan.alistarh@ist.ac.at

James Aspnes
Yale
james.aspnes@gmail.com

Faith Ellen
University of Toronto
faith@cs.toronto.edu

Rati Gelashvili
University of Toronto
gelash@gmail.com

Leqi Zhu
University of Michigan
zhu.jimmy@gmail.com

August 2, 2020

Abstract

We introduce *extension-based proofs*, a class of impossibility proofs that includes valency arguments. They are modelled as an interaction between a prover and a protocol. Using proofs based on combinatorial topology, it has been shown that it is impossible to deterministically solve k-set agreement among $n > k \geq 2$ processes in a wait-free manner in certain asynchronous models. However, it was unknown whether proofs based on simpler techniques were possible. We show that this impossibility result cannot be obtained for one of these models by an extension-based proof and, hence, extension-based proofs are limited in power.

1 Introduction

One of the most well-known results in the theory of distributed computing, due to Fischer, Lynch, and Paterson [FLP85], is that there is no deterministic, wait-free protocol solving consensus among $n \geq 2$ processes in an asynchronous message passing system, even if at most one process may crash. Their result has been extended to asynchronous shared memory systems where processes communicate by reading from and writing to shared registers [Abr88, CIL87, Her91, LAA87]. Moses and Rajsbaum [MR02] gave a unified framework for proving the impossibility of consensus in a number of different systems.

Chaudhuri [Cha93] conjectured that the impossibility of consensus could be generalized to the *k-set agreement problem*. In this problem, there are $n > k \geq 1$ processes, each starting with an input in $\{0, 1, \ldots, k\}$. Each process that does not crash must output a value that is the input of some process (*validity*) and, collectively, at most k different values may be output (*agreement*). In particular, *consensus* is just 1-set agreement.

Chaudhuri's conjecture was eventually proved in three concurrent papers by Borowsky and Gafni [BG93a], Herlihy and Shavit [HS99], and Saks and Zaharoglou [SZ00]. These proofs and a later proof by Attiya and Rajsbaum [AR02] all relied on sophisticated machinery from combinatorial topology, using a simplicial complex to model the set of all initial configurations of a wait-free protocol and a subdivision of it to model the set of all its final configurations. Then they used

Sperner's Lemma to show that there exists a final configuration in which $k+1$ different values have been output. This proves that the protocol does not correctly solve k-set agreement.

Later on, Attiya and Castañeda [AC11] and Attiya and Paz [AP12] showed how to obtain the same results using purely combinatorial techniques, without explicitly using topology. Like the topological proofs, these proofs also consider the set of final configurations of a supposedly wait-free k-set agreement protocol. However, by relating different final configurations to one another using indistinguishability and employing arguments similar to proofs of Sperner's Lemma, they proved the existence of a final configuration in which $k+1$ different values have been output.

A common feature of these impossibility proofs is that they are non-constructive. They prove that any deterministic protocol for k-set agreement among $n > k$ processes in an asynchronous system has an execution in which some process takes infinitely many steps without returning a value, but do not construct such an execution.

In contrast, impossibility proofs for deterministic, wait-free consensus in asynchronous systems explicitly construct an infinite execution by repeatedly extending a finite execution by the steps of some processes. Specifically, they define a *bivalent configuration* to be a configuration from which there is an execution in which some process outputs 0 and an execution in which some process outputs 1. Then they show that, from any bivalent configuration, there is a step of some process that results in another bivalent configuration. This allows them to explicitly construct an infinite execution in which no process has output a value. A natural question arises: is there a proof of the impossibility of k-set agreement that explicitly constructs an infinite execution by repeated extensions? This question is related to results in proof complexity that show certain theorems cannot be obtained in weak formal systems. For example, it is known that relativized bounded arithmetic cannot prove the pigeonhole principle [PBI93].

Our contributions. In this paper, we formally define the class of *extension-based proofs*, which model impossibility proofs that explicitly construct an infinite execution by repeated extensions. We also prove that there is no extension-based proof of the impossibility of a deterministic, wait-free protocol solving k-set agreement among $n > k \geq 2$ processes in asynchronous systems where processes communicate using an unbounded sequence of snapshot objects, to which each process can `update` and `scan` only once.

A *task* is a problem in which each process starts with a private input value and must output one value, such that the sequence of values produced by the processes satisfies certain specifications, which may depend on the input values of the processes. We view a proof of the impossibility of solving a task as an interaction between a *prover* and any protocol that claims to solve the task. The prover has to refute this claim. To do so, it can repeatedly query the protocol about the states of processes in configurations that can be reached in a small number of steps from configurations it already knows about. It can also ask the protocol to exhibit an execution by a set of processes from a configuration it knows about in which some process outputs a particular value, or to declare that no such execution exists. The goal of the prover is to construct a *bad* execution, i.e. an execution in which some processes take infinitely many steps without terminating or output values that do not satisfy the specifications of the task. The definition of extension-based proofs is presented in Section 3.

A key observation is that, from the results of its queries, many protocols are indistinguishable to the prover. It must construct a single execution that is bad for all these protocols. To prove that no prover can construct a bad execution, we show how an adversary can adaptively define a protocol in response to any specific prover's queries. In this *adversarial protocol,* all processes eventually terminate and output correct values in executions consistent with the results of the prover's queries. In Section 5, we argue that no such proof can refute the possibility of a deterministic, wait-free protocol solving k-set agreement among $n > k \geq 2$ processes in the *non-uniform iterated snapshot* (NIS) model. In the conference version of this paper [AAE⁺19], we made a very similar argument in the *non-uniform iterated immediate snapshot* (NIIS) model. The snapshot and NIS models are defined in Section 2.

From a computability standpoint, the snapshot, NIS, and NIIS models are equivalent in power to the basic asynchronous model in which processes communicate through shared registers: Any protocol in the basic asynchronous model can be easily adapted to run in the snapshot model by replacing each `read` by a `scan` and then throwing away the information it does not need. Afek, Attiya, Dolev, Gafni, Merritt, and Shavit [AAD⁺93] gave a wait-free implementation of a snapshot object using registers, so the converse is also true. Any execution of a protocol using an immediate snapshot object is also an execution of the protocol using a snapshot object, so protocols designed for the snapshot model also run in the immediate snapshot model. Borowsky and Gafni [BG93b] gave a wait-free implementation of an immediate snapshot object from a snapshot object, so protocols designed for the immediate snapshot model can be modified to run in the snapshot model. Likewise, the non-uniform iterated snapshot (NIS) model and the non-uniform iterated immediate snapshot (NIIS) model are computationally equivalent. The *iterated immediate snapshot* (IIS) model was introduced by Borowsky and Gafni [BG97]. The NIIS model is a slight generalization, introduced by Hoest and Shavit [HS06]. Any protocol in the NIS model can be easily adapted to run in the single-writer snapshot model by appending values rather than overwriting them when performing an `update` and throwing away the information in a `scan` about values that would have appeared in other snapshot objects. Similarly, any protocol in the NIIS model can be adapted to run in the immediate snapshot model. Borowsky and Gafni gave a nonblocking emulation in the NIIS model of any protocol in the immediate snapshot model [BG97]. The same emulation can be applied to a protocol in the snapshot model to obtain a protocol in the NIS model. Thus, to show there is no bounded wait-free protocol to solve a certain task in all of these models, it suffices to show that there is no bounded wait-free protocol to solve that task in any one of them.

The IIS and NIIS models are nice because the reachable configurations of a protocol have a natural representation using combinatorial topology. For the NIS model, there is a similar representation using graphs, which may be easier to understand. This representation is presented in Section 4, together with some properties that are needed for our proof. In this view, when an extension-based prover makes queries, it is essentially performing local search on the configuration space of the protocol. Because the prover obtains incomplete information about the protocol, the adversary has some flexibility when specifying the protocol's behaviour in configurations not yet queried by the prover.

There are a number of interesting directions for extending this work, which are discussed in Section 6.

2 Models

An execution is a sequence of steps. In each step of a shared memory model, a process performs an atomic operation on a shared object and then updates its local state. Communication among processes occurs through the atomic operations on shared objects. We use n to denote the number of processes, $p_1, \ldots, p_n$ to denote the processes, and x_i to denote the input to process p_i. When solving a task, we let y_i denote the output of process p_i. A value is assigned to y_i immediately before p_i terminates.

In the *single-writer register* model, there is one shared register R_i for each process p_i, to which only it can `write`, but which can be `read` by every process. The initial value of each register is $-$.

In the *single-writer snapshot* model, there is one shared single-writer snapshot object S with n components. The initial value of each component is $-$. The snapshot object supports two operations, `update`(v) and `scan`$()$. An `update`(v) operation by process p_i updates $S[i]$, the i'th component of S, to have value v, where v is an element of an arbitrarily large set that does not contain $-$. A `scan`$()$ operation returns the value of each component of S.

In the *non-uniform iterated snapshot* (NIS) model, there is an infinite sequence, $S_1, S_2, \dots,$ of shared *single-writer snapshot* objects, each with n components. The initial value of each component is $-$. The *initial state* of process p_i consists of its identifier, i, and its input, x_i. Each process accesses each snapshot object at most twice, starting with S_1. The first time p_i accesses a snapshot object S_r, it performs `update`(s_i) to set $S_r[i]$ to its current state, s_i. Its new state is the same as its previous state, except for an extra bit indicating that it has performed the `update`. At its next step, it performs a `scan` of S_r. Its new state, s'_i, is a pair consisting of i and the result of the `scan`. Process p_i remembers its entire history, because s_i is the i'th component of the result of the `scan`. Next, p_i consults a function, δ, from the set of possible states of processes to the set of possible output values and the special symbol $\perp$. This indicates whether p_i should output a value: If $\delta(s'_i) \neq \perp$, then p_i outputs $\delta(s'_i)$ and is *terminated.* If $\delta(s'_i) = \perp$, then p_i is poised to access the next snapshot object. A protocol in the NIS model is completely specified by the function δ.

In the snapshot and NIS models, we assume that a process is only terminated after performing a `scan`. This is without loss of generality, because a `scan` does not change the contents of shared memory and, so, does not affect any other process.

A *configuration* of a protocol consists of the contents of each shared object and the state of each process at some point during an execution of the protocol. An *initial* configuration is a configuration in which every process is in an initial state and every object has its initial value. A process is *active* in a configuration if it is not terminated. A configuration is *final* if it has no active processes. If C is a configuration and p_i is a process that is active in C, then Cp_i denotes the configuration that results when p_i takes the step from configuration C specified by the protocol. A *schedule* from C is a finite or infinite sequence of (not necessarily distinct) processes $\alpha = \alpha_1, \alpha_2, \dots$ such that there is a sequence of configurations $C = C_0, C_1, C_2 \dots$ where process α_i is active in C_{i-1} and $C_i = C_{i-1}\alpha_i$ for each process α_i in α. If α is a finite schedule from C, then $C\alpha$ denotes the configuration of the protocol that is reached by performing steps, with one step by a process for each occurrence of the process in α, in order, starting from configuration C. Each finite schedule from an initial configuration results in a *reachable* configuration. A protocol is *wait-free* if it does not have an infinite schedule.

Two configurations C and C' are *indistinguishable* to a set of processes P if every process in P has the same state in C and C'. Two finite schedules α and β from C are *indistinguishable* to the set of processes P if the resulting configurations $C\alpha$ and $C\beta$ are indistinguishable to P. A *P-only schedule* from C is a schedule in which only processes in P appear.

3 Extension-Based Proofs

An *extension-based* proof is an interaction between a prover and a (supposedly) wait-free protocol for solving a task, which the prover is trying to prove is incorrect. The prover starts with no knowledge about the protocol (except its initial configurations) and makes the protocol reveal information about various configurations by asking *queries*, which it chooses adaptively, based on the responses to its queries. The interaction proceeds in phases, beginning with phase 1.

In each phase $\varphi \geq 1$, the prover starts with a finite schedule, $\alpha(\varphi)$, from a set of initial configurations, $\mathcal{B}(\varphi)$, which only differ from one another in the input values of processes that do not occur in $\alpha(\varphi)$, and the set of resulting configurations $\mathcal{A}(\varphi) = \{C_0\alpha(\varphi) \mid C_0 \in \mathcal{B}(\varphi)\}$. At the start of phase 1, $\alpha(1)$ is the empty schedule and $\mathcal{A}(1) = \mathcal{B}(1)$ is the set of all initial configurations of the protocol. If every configuration in $\mathcal{A}(\varphi)$ is final and the values output by the processes before terminating satisfy the specifications of the task, then the prover *loses*. The prover also maintains a set, $\mathcal{A}'(\varphi)$, containing the configurations it reaches by taking non-empty sequences of steps from configurations in $\mathcal{A}(\varphi)$ during phase φ. This set is empty at the start of phase φ and it will be constructed so that, for every configuration $C' \in \mathcal{A}'(\varphi)$, there exists a configuration $C \in \mathcal{A}(\varphi)$ and a schedule β from C such that $C' = C\beta$ and $C\beta' \in \mathcal{A}'(\varphi)$ for every nonempty prefix β' of β.

A *query* (C, q) by the prover is specified by a configuration $C \in \mathcal{A}(\varphi) \cup \mathcal{A}'(\varphi)$ and a process q that is active in C. The protocol replies to this query with the configuration C' resulting from q taking the step from C specified by the protocol. Then the prover adds C' to $\mathcal{A}'(\varphi)$ and we say that the prover has *reached* C'. Since there exists a configuration $C_0 \in \mathcal{A}(\varphi)$ and a schedule β from C_0 such that $C = C_0\beta$ and $C_0\beta' \in \mathcal{A}'(\varphi)$ for every nonempty prefix β' of β, the same is true for C' with the schedule βq from C_0. If the prover reaches a configuration C' in which the outputs of the processes do not satisfy the specifications of the task, it has demonstrated that the protocol is incorrect. In this case, the prover *wins*.

A *chain of queries* is a (finite or infinite) sequence of queries $(C_1, q_1), (C_2, q_2), \ldots$ such that, for all consecutive queries (C_i, q_i) and (C_{i+1}, q_{i+1}) in the chain, C_{i+1} is the configuration resulting from q_i taking the step from C_i specified by the protocol.

An *output query* (C, Q, y) in phase φ is specified by a configuration $C \in \mathcal{A}(\varphi) \cup \mathcal{A}'(\varphi)$, a set of processes Q that are all active in C, and a possible output value y. If there is a Q-only schedule starting from C that results in a configuration in which some process in Q outputs y, then the protocol returns some such schedule. Otherwise, the protocol returns NONE. Note that the prover does not add the resulting configuration to $\mathcal{A}'(\varphi)$. However, it can do so by asking a chain of queries starting from C following the schedule returned by the protocol. If Q is the set of all processes, then the results of the output queries (C, Q, y), for every possible output value y, tells the prover which values can be output by the protocol starting from configuration C. For example, if 0 and 1 are the only possible output values, this enables the prover to determine whether C is *bivalent*.

After constructing finitely many output queries and chains of queries in phase φ without winning, the prover must end the phase by committing to a nonempty schedule α' from some configuration $C \in \mathcal{A}(\varphi)$ such that $C\alpha' \in \mathcal{A}'(\varphi)$. Since $\mathcal{A}(\varphi) = \{C_0\alpha(\varphi) \mid C_0 \in \mathcal{B}(\varphi)\}$, there is an initial configuration $C'_0 \in \mathcal{B}(\varphi)$ such that $C = C'_0\alpha(\varphi)$. Hence $C\alpha' = C'_0\alpha(\varphi)\alpha'$. Then $\alpha(\varphi+1) = \alpha(\varphi)\alpha'$, $\mathcal{B}(\varphi + 1)$ is the set of all initial configurations that only differ from C'_0 by the states of processes that do not appear in this schedule, and $\mathcal{A}(\varphi + 1) = \{C_0\alpha(\varphi + 1) \mid C_0 \in \mathcal{B}(\varphi + 1)\}$. Then the prover begins phase $\varphi + 1$.

If the interaction between the prover and the protocol is infinite, either because the prover is allowed to continue a chain of queries indefinitely or the number of phases is infinite, the prover *wins*. In this case, the prover has demonstrated that the protocol is not wait-free. For example, a valency proof for the impossibility of consensus shows how to construct an infinite schedule for any protocol that satisfies agreement and validity. More generally, if the protocol satisfies the specifications of the task in all of its final configurations, making the interaction go on forever is the only way that the prover can win. For the trivial protocol in which no process ever outputs a value, the prover can win by asking any infinite chain of queries.

To prove that a task is impossible using an extension-based proof, one must show there exists a prover that wins against *every* protocol.

If a deterministic protocol is wait-free and there are only a finite number of initial configurations, then, by König's lemma, there is a finite upper bound on the length of all schedules of the protocol. If the prover is given (or is able to ask for) such an upper bound, it can perform a finite number of chains of queries to examine all reachable configurations. In other words, this allows the prover to perform exhaustive search in the first phase to learn everything about the protocol. Likewise, if a prover does not have to eventually end phase 1, it can win against every wait-free protocol by performing exhaustive search. Such proofs violate the spirit of extension-based proofs.

For any protocol in the IIS model, there is a bound T such that every process terminates after taking exactly T steps. Although the prover is not given T, it is easy for a prover to determine T by performing one output query or one chain of queries. Thus, extension-based proofs are too powerful in the IIS model. If a task has a finite number of initial configurations and there is a protocol to solve this task in the NIIS model, then there is a protocol to solve this task in the IIS model. However, without knowledge of an upper bound on the length of all schedules, there is no general way to construct an IIS protocol from an NIIS protocol.

4 Properties of the NIS Model

The proof of our main result relies on properties of the non-uniform iterated snapshot model, including a simple graphical representation of protocols in this model. We begin with two simple observations.

Observation 4.1. *A reachable configuration in the NIS model is completely determined by the states of all processes in the configuration (including the processes that have terminated).*

This is true because only process p_i can update the i'th component of each snapshot object and each process remembers its entire history.

The second observation is a special case of a general, well-known result about indistinguishability. (For example, see Corollary 2.2. in [AE14].)

Observation 4.2. *Suppose C and C' are two reachable configurations in the NIS model and each snapshot object S_r has the same contents in C and C', for all $r \geq t$. If C and C' are indistinguishable to a set of processes P, each active process in P is poised in C to access a snapshot object S_r, for some $r \geq t$, and α is a finite, P-only schedule from C, then α is a schedule from C' and the configurations $C\alpha$ and $C'\alpha$ are indistinguishable to P.*

Suppose C is a reachable configuration in which all active processes are poised to `update` the same snapshot object. A *1-round schedule* from C is a schedule consisting of two occurrences of each process that is active in C. Each active process in the resulting configuration is poised to `update` the next snapshot object in the sequence. If none of the processes are active in C, then the empty schedule is the only 1-round schedule from C. The following observation is a corollary of Observation 4.2.

Observation 4.3. *Suppose β is a 1-round schedule from C, α is a prefix of β, and P is the set of those processes that occur twice in α,. Then α and β are indistinguishable to P and the terminated processes in configuration C.*

For $t > 1$, a *t-round schedule* from C is a schedule $\beta_1\beta_2\cdots\beta_t$ such that β_1 is a 1-round schedule from C and, for $1 < i \leq t$, β_i is a 1-round schedule from $C\beta_1\cdots\beta_{i-1}$. Notice that some processes may have terminated during $\beta_1\cdots\beta_{i-1}$. These processes are not included in β_i.

Every schedule from an initial configuration C that reaches a final configuration C' is indistinguishable (to all processes) to an r-round schedule, for some value of r. This is a special case of the following lemma, where $t = 1$ and P is the set of all processes.

Lemma 4.4. *Let C be a configuration in which every active process is poised to perform an `update` to S_t and let C' be a configuration reachable from C. Suppose that P is a set of processes that is each poised to perform an `update` to S_{t+r} in C' or has terminated prior to performing an `update` to S_{t+r}. Then there exists an r-round schedule β from C such that $C\beta$ and C' are indistinguishable to P, i.e. each process in P has the same state in $C\beta$ and C'.*

Proof. Since C' is reachable from C, there is a finite schedule α from C such that $C' = C\alpha$. Note that each process in P occurs at most $2r$ times in α. Let γ be the schedule from C obtained from α by removing all but the first $2r$ occurrences of every process. The steps that are performed in α, but not γ, are accesses to S_{t+r} or snapshot objects that follow S_{t+r}. Since no process in P accesses these objects when the protocol is performed from C according to α or γ, $C\alpha$ and $C\gamma$ are indistinguishable to P. So, it suffices to show that $C\gamma$ and $C\beta$ are indistinguishable to P for some r-round schedule β.

The proof proceeds by induction on r. First suppose that $r = 1$. Let β be any 1-round schedule that has γ as a prefix. In other words, append sufficiently many occurrences of each process that is active in C to the end of γ so that each occurs exactly 2 times in β. Note that each process in P that is active in C occurs exactly 2 times in α, so all occurrences of processes in P that occur in β occur in γ. By Observation 4.3, $C\gamma$ and $C\beta$ are indistinguishable to P.

Now suppose that $r > 1$. Let α' be the schedule from C obtained from γ by removing all but the first $2r-2$ occurrences of every process. This removes all accesses of S_{t+r-1}, but no other steps. Let P' be the set of all processes that are poised to perform an `update` to S_{t+r-1} in $C\alpha'$ or have terminated prior to performing an `update` to S_{t+r-1}. Then $P \subseteq P'$. By the induction hypothesis, there exists an $(r-1)$-round schedule β' from C such that $C\beta'$ and $C\alpha'$ are indistinguishable to P'.

Let α'' be the schedule from $C\alpha'$ obtained from γ by removing the first $2r-2$ occurrences of every process. Each process that is terminated in $C\alpha'$ does not occur in α''. Each process in P that is active in $C\alpha'$ occurs exactly twice in α''.

Let β'' be a 1-round schedule from $C\beta'$ obtained from α'' by appending sufficiently many occurrences of every process that is active in $C\beta'$ so that each occurs exactly twice in β''.

Note that if some process p_i performs its `update` to S_{t+r-1} in γ then $p_i \in P'$ and p_i occurs at least once in α'', so it is active in $C\alpha'$ and performs the same `update` to S_{t+r-1} in α''. Since $C\beta'$ and $C\alpha'$ are indistinguishable to p_i, it performs the same `update` to S_{t+r-1} in β''. By construction, the accesses of S_{t+r-1} in α'' occur in the same order as in γ. Moreover, because each process in P that is active in $C\alpha'$ occurs exactly twice in α'', its `scan` of S_{t+r-1} gets the same result in γ, α'', and β''. Hence $C\gamma$ and $C\beta$ are indistinguishable to P, where $\beta = \beta'\beta''$ is an r-round schedule. □

In particular, the state of each process in each reachable configuration in which the process is poised to perform an `update` to S_{1+r} or has terminated prior to performing an `update` to S_{1+r} is the state of that process in a configuration reachable by an r-round schedule from an initial configuration.

Consider a protocol in the NIS model (specified by a function δ from the set of possible states of processes to the set of possible output values and the special symbol $\perp$). We use an undirected graph $\mathbb{G}_t = (\mathbb{V}_t, \mathbb{E}_t)$ to represent the configurations of this protocol reachable from initial configurations by t-round schedules. Each vertex $v \in \mathbb{V}_t$ represents the state of one process in some such reachable configuration and $id(v)$ is the identifier of this process, which is the first part of the state. There is an edge in $\mathbb{E}_t$ between two vertices if there is some such reachable configuration that contains the states represented by both vertices. In each configuration, there are exactly n vertices, each with a different identifier. Therefore each edge in $\mathbb{E}_t$ belongs to an n-vertex clique in $\mathbb{G}_t$ consisting of vertices with distinct identifiers.

An n-vertex clique *represents* a configuration if the vertices of the clique represent the states of the processes in that configuration. In particular, the n-vertex cliques in $\mathbb{G}_0$ represent all initial configurations. For the k-set agreement problem, $\mathbb{V}_0 = \{(i,a) \mid i \in \{1,\ldots,n\} \text{ and } a \in \{0,\ldots,k\}\}$ and $\{(i,a),(j,b)\} \in \mathbb{E}_0$ if and only if $i \neq j$. A vertex v is *active* if the state it represents is active and we use $\delta(v) = \perp$ to denote this. A vertex v is *terminated* if the state it represents is terminated and we use $\delta(v) \neq \perp$ to denote the value the process outputs in this state. An n-vertex clique represents a final configuration if and only if all its vertices are terminated.

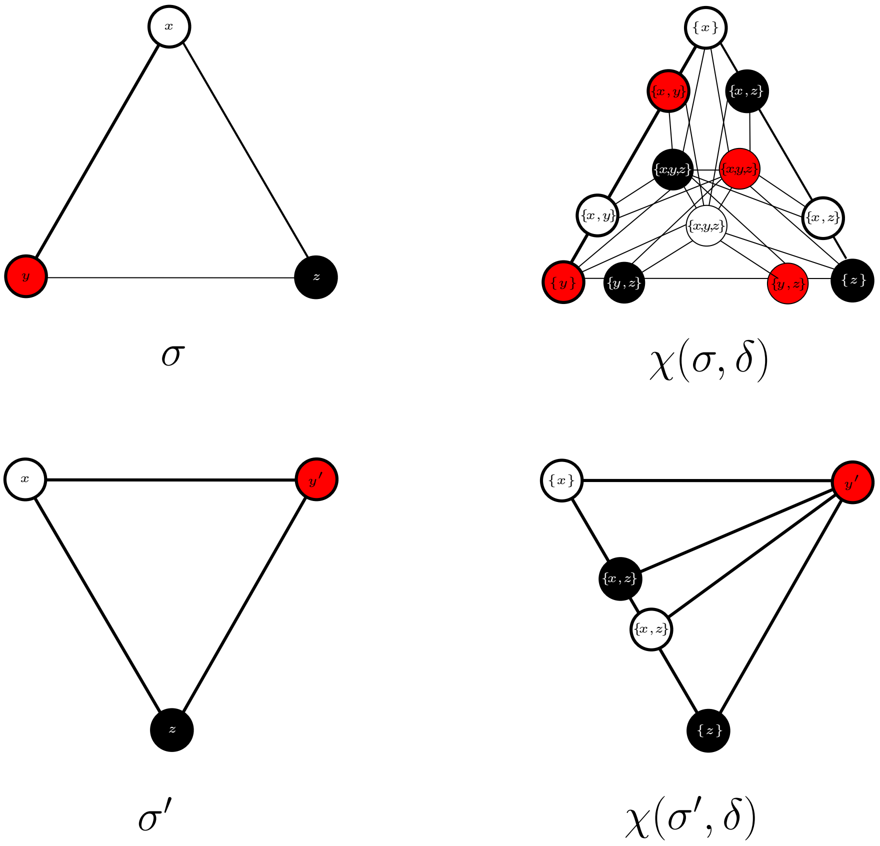
We show how to construct $\mathbb{G}_{t+1}$ from $\mathbb{G}_t$, given $\delta(v)$ for all $v \in \mathbb{V}_t$. We start with an n-vertex clique σ in $\mathbb{G}_t$, which represents some configuration C reachable from an initial configuration by a t-round schedule, and construct the n-vertex cliques of $\mathbb{G}_{t+1}$ representing configurations reachable from C by 1-round schedules.

Consider any subset τ of the active vertices in σ. Let $id(\tau) = \{id(v) \mid v \in \tau\}$ be the set of identifiers of processes whose states are represented by vertices in τ. Each process p_i, for $i \in id(\tau)$, is poised to perform an `update` to S_{t+1} in configuration C. Suppose process p_i performs its `update` to S_{t+1}, for each $i \in id(\tau)$, but no other process does so. Then, for each $v \in \tau$, $S_{t+1}[id(v)]$ is the state represented by v and, for each $j \notin id(\tau)$, $S_{t+1}[j] = -$. If some process p_i now performs a `scan` of S_{t+1}, the result is an n-component vector containing these values. Since there is a one-to-one correspondence between τ and this vector, we can represent the resulting state of process p_i by the pair (i,τ).

Given $\delta(v)$ for each $v \in \sigma$, we can define the graph $\chi(\sigma,\delta)$, representing the configurations reachable from C by 1-round schedules, as follows:

- v is a vertex in $\chi(\sigma,\delta)$ if and only if
  - v is a terminated vertex in σ or
  - $v = (i,\tau)$, where τ is a subset of the active vertices in σ and $i \in id(\tau)$.

  If $v = (i,\tau)$, then $id(v) = i$.
- $\{v, v'\}$ is an edge in $\chi(\sigma,\delta)$ if and only if $id(v) \neq id(v')$ and



σ $\chi(\sigma, \delta)$

σ' $\chi(\sigma', \delta)$

Figure 1: The subdivisions of two 3-vertex cliques.

- at least one of v and v' is a terminated vertex in σ or
- $v = (id(v), \tau)$ and $v' = (id(v'), \tau')$, where τ and τ' are subsets of the active vertices in σ such that $\tau \subseteq \tau'$ or $\tau' \subseteq \tau$.

A vertex is in both σ and $\chi(\sigma, \delta)$ if and only if it is terminated. If vertex (i, τ) is in $\chi(\sigma, \delta)$, but not in σ, then it represents the state of process p_i immediately after it has performed its `scan` of S_{t+1}, τ represents the result of the `scan`, and $id(\tau)$ is the set of identifiers of the processes that performed an `update` to S_{t+1} prior to this `scan`. Note that $i \in id(\tau)$, since process p_i performs its `update` to S_{t+1} before its `scan`.

Figure 1 illustrates the subdivisions of two different 3-vertex cliques. In σ, all three vertices are active, the state of process p_1 is represented by vertex x, the state of p_2 is represented by vertex y, and the state of p_3 is represented by vertex z. In σ', process p_1 and p_3 have the same states, but the state of p_2 is represented by vertex y', which is terminated. For readability, process identifiers are omitted from the representation of states in $\chi(\sigma, \delta)$ and $\chi(\sigma', \delta)$. Instead, white vertices indicate states of p_1, red vertices indicate states of p_2, and black vertices indicate states of p_3.

The next two results show that there is a correspondence between the n-vertex cliques in $\chi(\sigma, \delta)$ and the configurations reachable from C by 1-round schedules.

Lemma 4.5. *Let σ be an n-vertex clique that represents a configuration C in which all active processes are poised to* `update` *the same snapshot object. If β is a 1-round schedule from C, then* *the configuration $C\beta$ is represented by an n-vertex clique in $\chi(\sigma, \delta)$.*

Proof. Let A be the set of active processes in configuration C and let S_{t+1} be the snapshot object the processes in A are poised to `update`. Then a 1-round schedule β from C is a sequence consisting of two copies of each process in A. Consider the second occurrence in β of a process p_i. This corresponds to the step in the schedule β at which p_i performs its `scan` of S_{t+1}. Let α be the prefix of β prior to this step. For each $p_j \in A$, p_j occurs in α if and only if $S_{t+1}[j]$ in configuration $C\alpha$ contains the state of p_j in configuration C. Let τ be the set of vertices in σ that represent the states of processes appearing in S_{t+1} in configuration $C\alpha$. Then τ represents the result of the `scan` of S_{t+1} by process p_i. In particular, p_i occurs in α, since it performs its `update` to S_{t+1} before its `scan`. Hence $i \in id(\tau)$ and $(i, \tau) \in \chi(\sigma, \delta)$.

Suppose $j \neq i$, $p_j \in A$, and the second occurrence of p_j in β occurs after p_i. Let ρ be the subset of σ representing the result of p_j's `scan`, so $(j, \rho) \in \chi(\sigma, \delta)$. Then the prefix of β prior to the second occurrence of p_j begins with α. Hence $\tau \subseteq \rho$ and $\{(i, \tau), (j, \rho)\}$ is an edge in $\chi(\sigma, \delta)$.

For each process p_i that is terminated in C, the vertex in σ with identifier i is also in $\chi(\sigma, \delta)$ and this vertex is connected to every other vertex in $\chi(\sigma, \delta)$ with a different identifier. Thus the configuration $C\beta$ is represented by an n-vertex clique in $\chi(\sigma, \delta)$. □

Lemma 4.6. *Let σ be an n-vertex clique that represents a configuration C in which all active processes are poised to* `update` *the same snapshot object. Every n-vertex clique in $\chi(\sigma, \delta)$ represents a configuration reachable from C by a 1-round schedule.*

Proof. Let S_{t+1} be the snapshot object the active processes in C are poised to `update`. Let σ' be an n-vertex clique in $\chi(\sigma, \delta)$. Since $id(v) \neq id(v')$ for all edges $\{v, v'\}$ in $\chi(\sigma, \delta)$, there is vertex $v_i \in \sigma'$ with $id(v_i) = i$ for each $i \in \{1, \ldots, n\}$. Let I be the set of indices of active processes in configuration C. The definition of $\chi(\sigma, \delta)$ implies that $v_i \in \sigma$, for each $i \notin I$, and $v_i = (i, \tau_i)$, for each $i \in I$, where τ_i is a subset of the active vertices in σ and $i \in id(\tau_i)$. Furthermore, if $i, j \in I$ and $i \neq j$, then either $\tau_i \subseteq \tau_j$ or $\tau_j \subseteq \tau_i$, since $\{v_i, v_j\}$ is an edge of $\chi(\sigma, \delta)$. Since $\tau_i \subseteq \tau_j$ implies $id(\tau_i) \subseteq id(\tau_j)$, the sets $id(\tau_i)$ for $i \in I$ can be ordered by inclusion. Let $\prec$ be a total order on I such that if i occurs in more of these sets than j does, then $i \prec j$. In other words, for all $i \in I$, the elements of $id(\tau_i)$ occur before the elements of $I - id(\tau_i)$.

Let α' be a sequence containing one copy of each process whose identifier is in I such that p_i occurs before p_j if and only if $i \prec j$. If the schedule α' is performed starting from C, then, for each $i \in I$, τ_i represents the contents of S_{t+1} at some point during the execution. Note that, since $i \in id(\tau)$, this point occurs after p_i performs its `update`. For each $i \in I$, insert a second copy of p_i after the process in α' whose `update` causes the contents of S_{t+1} to be represented by τ_i and before the next process in α'. Let α be the resulting sequence. Then α is a 1-round schedule such that $v_i = (i, \tau_i)$ represents the state of p_i in configuration $C\alpha$, for each $i \in I$. For each $i \notin I$, $v_i \in \sigma$, so it represents the state of the terminated process p_i in C and, thus, the state of p_i in $C\alpha$, too. Hence σ' represents the configuration $C\alpha$. □

For any two graphs $G = (V, E)$ and $G' = (V', E')$, the *union* of G and G' is the graph $G \cup G' = (V \cup V', E \cup E')$. Then $\mathbb{G}_t$ is a union of n-vertex cliques. Consider any subgraph $\mathbb{A}$ of $\mathbb{G}_t$ that is the union of n-vertex cliques. We define $\chi(\mathbb{A}, \delta)$ to be the union of the graphs $\chi(\sigma, \delta)$ for all n-vertex cliques σ in $\mathbb{A}$. In particular, $\chi(\mathbb{G}_t, \delta)$ is the union of $\chi(\sigma, \delta)$ for all n-vertex cliques σ in $\mathbb{G}_t$. By Lemma 4.5 and Lemma 4.6, it follows that $\mathbb{G}_{t+1} = \chi(\mathbb{G}_t, \delta)$. This method for obtaining $\mathbb{G}_{t+1}$ from $\mathbb{G}_t$ is closely related to the non-uniform chromatic subdivision of a simplicial complex representing a protocol in the NIIS model, introduced by Hoest and Shavit [HS06]. Consequently, we will call the graph $\chi(\sigma, \delta)$ the *subdivision* of the clique σ and the graph $\mathbb{G}_{t+1}$ the *subdivision* of the graph $\mathbb{G}_t$. However, Herlihy and Shavit [HS99, page 884] mention that $\chi(\sigma, \delta)$ is not necessarily a topological subdivision of σ.

By definition, a clique is connected. We show that subdivision of a clique in $\mathbb{G}_t$ is still connected.

Lemma 4.7. *The subdivision $\chi(\sigma, \delta)$ of every n-vertex clique σ in $\mathbb{G}_t$ is connected.*

Proof. First suppose that σ contains some terminated vertex u. By definition, $u \in \chi(\sigma, \delta)$ and no other vertex of $\chi(\sigma, \delta)$ has the same *id*. Consider any other vertex $v \in \chi(\sigma, \delta)$. By definition, $\{u, v\}$ is an edge in $\chi(\sigma, \delta)$. Thus every vertex in $\chi(\sigma, \delta)$ is connected to u, so the graph $\chi(\sigma, \delta)$ is connected.

Now suppose that σ contains only active vertices. Then, for each $i \in \{1, \ldots, n\} = id(\sigma)$, $(i, \sigma) \in \chi(\sigma, \delta)$. Furthermore, if $i, j \in \{1, \ldots, n\}$ and $i \neq j$, then $\{(i, \sigma), (j, \sigma)\}$ is an edge in $\chi(\sigma, \delta)$, so the vertices (i, σ) for $i \in \{1, \ldots, n\}$ form an n-vertex clique. Now consider any vertex $(i, \tau) \in \chi(\sigma, \delta)$, where $\tau \subsetneq \sigma$. Then, by definition, $\{(i, \tau), (j, \sigma)\}$ is an edge in $\chi(\sigma, \delta)$ for any $j \in \{1, \ldots, n\} - \{i\}$. Since $n \geq 2$, such a j exists. Thus every vertex in $\chi(\sigma, \delta)$ is connected to this clique, so the graph $\chi(\sigma, \delta)$ is connected. □

More generally, connectivity is preserved by subdivision.

Lemma 4.8. *Let $\mathbb{A}$ be a connected subgraph of $\mathbb{G}_t$ that is the union of n-vertex cliques. Then $\chi(\mathbb{A}, \delta)$ is a connected subgraph of $\mathbb{G}_{t+1}$.*

Proof. Consider any two vertices $u', v' \in \chi(\mathbb{A}, \delta)$. Then $u' \in \chi(\sigma, \delta)$ and $v' \in \chi(\tau, \delta)$ for some n-vertex cliques $\sigma, \tau \subseteq \mathbb{A}$. Since $\mathbb{A} \subseteq \mathbb{G}_t$ is connected, there is a path $w_0, \ldots, w_\ell$ in $\mathbb{A}$ of length $\ell \geq 0$ in $\mathbb{A}$ such that $w_0 \in \sigma$ and $w_\ell \in \tau$. For $0 \leq i \leq \ell$, let $w'_i = w_i$ if $\delta(w_i) \neq \perp$, and let $w'_i = (id(w_i), \{w_i\})$ if $\delta(w_i) = \perp$.

Consider any i such that $1 \leq i \leq \ell$. Since $\{w_{i-1}, w_i\}$ is an edge of $\mathbb{A}$, there exists an n-vertex clique $\sigma_i \subseteq \mathbb{A}$ that contains this edge. Since $w_{i-1}, w_i \in \sigma_i$, it follows by construction that $w'_{i-1}, w'_i \in \chi(\sigma_i, \delta)$. By Lemma 4.7, the subdivision $\chi(\sigma_i, \delta)$ of σ_i is connected. Thus, there exists a path between w'_{i-1} and w'_i in $\chi(\sigma_i, \delta) \subseteq \chi(\mathbb{A}, \delta)$. By Lemma 4.7, $\chi(\sigma, \delta)$ and $\chi(\tau, \delta)$ are connected, so there exist a path between u' and w'_0 in $\chi(\sigma, \delta) \subseteq \chi(\mathbb{A}, \delta)$ and a path between w'_ℓ and v' in $\chi(\tau, \delta) \subseteq \chi(\mathbb{A}, \delta)$. Hence, there is a path between u' and v' in $\chi(\mathbb{A}, \delta)$.

Since u' and v' are arbitrary, $\chi(\mathbb{A}, \delta)$ is connected. □

The next result follows by induction, because $\mathbb{G}_{t+1} = \chi(\mathbb{G}_t, \delta)$.

Corollary 4.9. *If $\mathbb{G}_0$ is connected, then, for all $t \geq 1$, $\mathbb{G}_t$ is connected.*

If $\mathbb{T} \subseteq \mathbb{V}_t$ is a set of terminated vertices in $\mathbb{G}_t$, we define $\chi(\mathbb{T}, \delta) = \mathbb{T} \subseteq \mathbb{V}_{t+1}$. Let $\mathbb{A}$ and $\mathbb{B}$ each be either a nonempty set of terminated vertices in $\mathbb{G}_t$ or the nonempty union of n-vertex cliques in $\mathbb{G}_t$. Then the *distance between* $\mathbb{A}$ and $\mathbb{B}$ (in $\mathbb{G}_t$) is the minimum of the length of the paths between $u \in \mathbb{A}$ and $v \in \mathbb{B}$. If $\mathbb{G}_0$, is connected, then Corollary 4.9 implies that at least one such path exists. Now we show that if the distance between $\mathbb{A}$ and $\mathbb{B}$ is 0 (i.e. they intersect), then the same is true for $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ and, if the distance between $\mathbb{A}$ and $\mathbb{B}$ is greater than 0 (i.e., they are disjoint), then so are $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$.

Lemma 4.10. *Suppose $\mathbb{A}$ and $\mathbb{B}$ are each either a set of terminated vertices in $\mathbb{G}_t$ or the union of n-vertex cliques in $\mathbb{G}_t$. Then $\mathbb{A}$ and $\mathbb{B}$ are disjoint if and only if $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ are disjoint.*

Proof. When $\mathbb{A}$ or $\mathbb{B}$ is a set of terminated vertices in $\mathbb{G}_t$, any vertex $u \in \mathbb{A} \cap \mathbb{B}$ is terminated, so $u \in \mathbb{A} \cap \mathbb{B}$ if and only if $u \in \chi(\mathbb{A}, \delta) \cap \chi(\mathbb{B}, \delta)$. So, assume that $\mathbb{A}$ and $\mathbb{B}$ are the unions of n-vertex cliques.

Suppose that $\mathbb{A}$ and $\mathbb{B}$ share a common vertex u. Let σ be an n-vertex clique in $\mathbb{A}$ that contains u and let ρ be an n-vertex clique in $\mathbb{B}$ that contains u. If u is a terminated vertex in $\mathbb{G}_t$, then, by definition, u is a vertex in both $\chi(\sigma, \delta)$ and $\chi(\rho, \delta)$. Otherwise, u is active in $\mathbb{G}_t$. In this case, let $\tau = \{u\}$ and $i = id(u)$. Then $i \in id(\tau)$, $\tau \subseteq \sigma$, and $\tau \subseteq \rho$. By definition (i, τ) is a vertex in both $\chi(\sigma, \delta)$ and $\chi(\rho, \delta)$. Since $\chi(\sigma, \delta)$ is a subgraph of $\chi(\mathbb{A}, \delta)$ and $\chi(\rho, \delta)$ is a subgraph of $\chi(\mathbb{B}, \delta)$, in both cases it follows that $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ are not disjoint.

Conversely, suppose that $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ share a common vertex v. By definition, there exists an n-vertex clique $\sigma \subseteq \mathbb{A}$, such that $v \in \chi(\sigma, \delta)$. Similarly, there exists an n-vertex clique $\rho \subseteq \mathbb{B}$ such that $v \in \chi(\rho, \delta)$. If v is a terminated vertex in $\mathbb{G}_t$, then v is a vertex in both σ and ρ. Otherwise, $v = (i, \tau)$ where $i \in id(\tau)$, $\tau \subseteq \sigma$, and $\tau \subseteq \rho$. Hence, in both cases, $\mathbb{A}$ and $\mathbb{B}$ are not disjoint. □

Lemma 4.10 can be generalized to show that subdividing does not decrease distances.

Lemma 4.11. *Suppose $\mathbb{A}, \mathbb{B} \subseteq \mathbb{G}_t$ are nonempty and each is either a set of terminated vertices or the union of n-vertex cliques. Then the distance between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$ is at least as large as the distance between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$.*

Proof. Let d be the distance between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$. The proof is by induction on d. If $d = 0$, then the claim is true, since distances are always non-negative. If $d = 1$, then $\mathbb{A}$ and $\mathbb{B}$ are disjoint. By Lemma 4.10, $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ are also disjoint, so the distance between them is at least 1.

Now suppose that $d \geq 2$ and the claim is true for all nonempty $\mathbb{A}^*, \mathbb{B}^* \subseteq \mathbb{G}_t$ such that the distance between $\mathbb{A}^*$ and $\mathbb{B}^*$ is $d - 1$ and each is either a set of terminated vertices or the union of n-vertex cliques. Consider any vertex $v \in \mathbb{V}_t$ at distance 1 from $\mathbb{A}$. Then there exists a vertex $u \in \mathbb{A}$ such that $\{u, v\} \in \mathbb{E}_t$. Since $\mathbb{G}_t$ is a union of n-vertex cliques, there exist $n-2$ other vertices that form a clique with $\{u, v\}$. Since these vertices are adjacent to u, they are all at distance at most 1 from $\mathbb{A}$. Let $\mathbb{A}'$ denote the union of all n-vertex cliques in $\mathbb{G}_t$ that contain at least one vertex in $\mathbb{A}$.

Consider any path $u_0, u_1, \ldots, u_d$ of length d between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$. Note that $u_1 \notin \mathbb{A}$, since the distance between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ is d. Thus u_1 is a vertex at distance 1 from $\mathbb{A}$ and, hence, is in $\mathbb{A}'$. Therefore the distance between $\mathbb{A}'$ and $\mathbb{B}$ in $\mathbb{G}_t$ is at most $d - 1$. In fact, the distance between $\mathbb{A}'$ and $\mathbb{B}$ is exactly $d-1$. Suppose not. Then there exists a path $v_0, \ldots, v_\ell$ in $\mathbb{G}_t$ between $\mathbb{A}'$ and $\mathbb{B}$ where $\ell < d - 1$. If $v_0 \in \mathbb{A}$, then this path is between $\mathbb{A}$ and $\mathbb{B}$. If $v_0 \in \mathbb{A}' - \mathbb{A}$, then, by definition of $\mathbb{A}'$, there exists a vertex $u \in \mathbb{A}$ such that $\{u, v_0\} \in \mathbb{E}_t$. But then $u, v_0, \ldots, \ell$ is a path between $\mathbb{A}$ and $\mathbb{B}$. In both cases, this shows that the distance between $\mathbb{A}$ and $\mathbb{B}$ is less than d, which contradicts the definition of d.

Consider any shortest path $w_0, w_1, \ldots, w_\ell$ between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$. Note that $w_1 \notin \chi(\mathbb{A}, \delta)$, since this is a shortest path. By definition of $\mathbb{G}_{t+1}$, $\{w_0, w_1\}$ is an edge of $\chi(\sigma, \delta)$ for some n-vertex clique $\sigma \subseteq \mathbb{G}_t$. If w_0 is terminated in $\mathbb{G}_t$, then $w_0 \in \mathbb{A}$ and $w_0 \in \sigma$, so, by definition, σ is in $\mathbb{A}'$. Otherwise, since $w_0 \in \chi(\sigma, \delta)$, there is a process identifier i and a set of vertices $\tau_i \subseteq \sigma$ such that $w_0 = (i, \tau_i)$ and $i \in id(\tau_i)$. Moreover, since $w_0 \in \chi(\mathbb{A}, \delta)$, there exists an n-vertex clique $\rho \subseteq \mathbb{A}$ such that $w_0 \in \chi(\rho, \delta)$, so $\tau_i \subseteq \rho$. In this case, let $v_i \in \tau_i$ be such that $id(v_i) = i$. Since $\tau_i \subseteq \rho \subseteq \mathbb{A}$, we have $v_i \in \mathbb{A}$ and, since $\tau_i \subseteq \sigma$, we have $\sigma \in \mathbb{A}'$. Hence, in both cases, $w_1 \in \chi(\mathbb{A}', \delta)$.

By the induction hypothesis, the distance between $\chi(\mathbb{A}', \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$ is at least $d-1$. Thus, $\ell \geq d$. □

If $\mathbb{A}$ and $\mathbb{B}$ are disjoint unions of n-vertex cliques and $\mathbb{C}$ is a union of n-vertex cliques all of whose vertices are active, then there is no edge in the subdivision of $\mathbb{C}$ that connects the subdivisions of $\mathbb{A}$ and $\mathbb{B}$.

Lemma 4.12. *Suppose $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$ are nonempty unions of n-vertex cliques in $\mathbb{G}_t$, $\mathbb{A} \cap \mathbb{C}$ is nonempty, $\mathbb{B} \cap \mathbb{C}$ is nonempty, $\mathbb{A}$ and $\mathbb{B}$ are disjoint, and all vertices in $\mathbb{C}$ are active. Then the distance between $\chi(\mathbb{A}, \delta) \cap \chi(\mathbb{C}, \delta)$ and $\chi(\mathbb{B}, \delta) \cap \chi(\mathbb{C}, \delta)$ in $\mathbb{G}_{t+1}$ is at least 2.*

Proof. Since $\mathbb{A} \cap \mathbb{C}$ and $\mathbb{B} \cap \mathbb{C}$ are nonempty, Lemma 4.10 says that $\chi(\mathbb{A}, \delta) \cap \chi(\mathbb{C}, \delta)$ and $\chi(\mathbb{B}, \delta) \cap \chi(\mathbb{C}, \delta)$ are nonempty. Since $\mathbb{A}$ and $\mathbb{B}$ are disjoint, it also says that the distance between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$ is at least 1. Hence, the distance between $\chi(\mathbb{A}, \delta) \cap \chi(\mathbb{C}, \delta)$ and $\chi(\mathbb{B}, \delta) \cap \chi(\mathbb{C}, \delta)$ in $\mathbb{G}_{t+1}$ is at least 1.

To obtain a contradiction, suppose that the distance between $\chi(\mathbb{A}, \delta) \cap \chi(\mathbb{C}, \delta)$ and $\chi(\mathbb{B}, \delta) \cap \chi(\mathbb{C}, \delta)$ in $\mathbb{G}_{t+1}$ is 1. Then there exist vertices $u \in \chi(\mathbb{A}, \delta) \cap \chi(\mathbb{C}, \delta)$ and $v \in \chi(\mathbb{B}, \delta) \cap \chi(\mathbb{C}, \delta)$ such that $\{u, v\} \in \mathbb{E}_{t+1}$. Since $u, v \in \chi(\mathbb{C}, \delta)$ and all vertices in $\mathbb{C}$ are active, $u = (i, \tau_i)$ and $v = (j, \tau_j)$, where $i \in id(\tau_i)$, $j \in id(\tau_j)$, $\tau_i \subseteq \sigma_i$, and $\tau_j \subseteq \sigma_j$ for some n-vertex cliques $\sigma_i, \sigma_j \subseteq \mathbb{C}$. Since $u \in \chi(\mathbb{A}, \delta)$, it follows that $\tau_i \subseteq \rho_i$ for some n-vertex clique $\rho_i \subseteq \mathbb{A}$. Similarly, $\tau_j \subseteq \rho_j$ for some n-vertex clique $\rho_j \subseteq \mathbb{B}$. Since $\{u, v\} \in \mathbb{E}_{t+1}$, $i \neq j$ and either $\tau_i \subseteq \tau_j \subseteq \mathbb{B}$ or $\tau_j \subseteq \tau_i \subseteq \mathbb{A}$. In both cases, $\mathbb{A}$ and $\mathbb{B}$ are not disjoint, contrary to assumption. □

We now prove one of the main technical tools used in this paper. It shows that the distance between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ is less than the distance between their subdivisions in $\mathbb{G}_{t+1}$, provided that there is no path between $\mathbb{A}$ and $\mathbb{B}$ in which every edge contains at least one terminated vertex.

Figure 2 illustrates Lemma 4.13. In the top diagram, which is part of $\mathbb{G}_t$, the grey triangle represents $\mathbb{A}$, which consists of one 3-vertex clique and $\mathbb{B} = \{v_4\}$ is a set containing one terminated vertex. The blue path, which has length 4, is a shortest path between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$. Note that v_2 and v_3 are both active vertices. In the bottom diagram, which is part of $\mathbb{G}_{t+1}$, the grey triangle represents $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta) = \mathbb{B}$. The blue path, which now has length 5, is a shortest path between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$.

Lemma 4.13. *Suppose $\mathbb{A}, \mathbb{B} \subseteq \mathbb{G}_t$ are nonempty and each is either a set of terminated vertices or the union of n-vertex cliques. If every path between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ contains at least one edge between active vertices, then the distance between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$ is larger than the distance between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$.*

Proof. Assume that every path between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ contains at least one edge between active vertices. Consider any such path $v_0, \ldots, v_\ell$ between $\mathbb{A}$ and $\mathbb{B}$. Suppose that, for all $1 \leq i \leq \ell$, the edge $\{v_{i-1}, v_i\}$ is contained in an n-vertex clique that contains a terminated vertex u_i. Replace each edge $\{v_{i-1}, v_i\}$ that is between active vertices by the subpath v_{i-1}, u_i, v_i. The result is a path between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ that contains no edges between active vertices, contrary to our assumption. Therefore, every path between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ contains at least one edge such that every n-vertex clique which contains this edge is comprised of active vertices.

Let $\mathbb{C}$ be the union of a minimal set of n-vertex cliques in $\mathbb{G}_t$ comprised of active vertices such that every path between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ contains at least one edge in $\mathbb{C}$. Let $\mathbb{F}$ be the union of all other cliques in $\mathbb{G}_t$. Then there are no paths between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{F}$. Let $\mathbb{A}'$ be the union of all

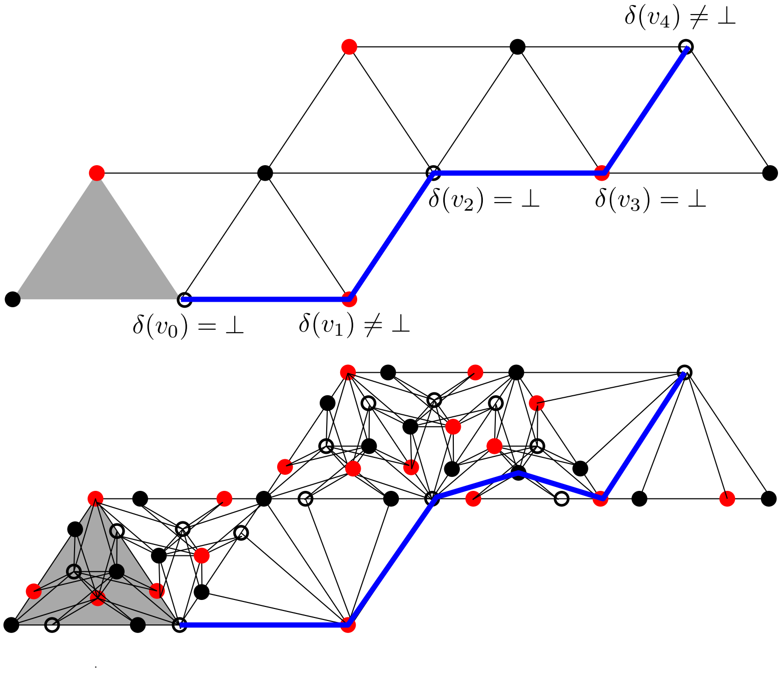


Figure 2: An illustration of Lemma 4.13.

cliques in $\mathbb{F}$ that are connected to $\mathbb{A}$ and let $\mathbb{B}'$ be the union of all cliques in $\mathbb{F}$ that are connected to $\mathbb{B}$. If $\mathbb{A}$ is the union of n-vertex cliques, then $\mathbb{A}'$ is nonempty, since $\mathbb{A} \subseteq \mathbb{A}'$. If $\mathbb{A}$ is a set of terminated vertices, consider the first vertex of some path from $\mathbb{A}$ to $\mathbb{B}$ in $\mathbb{G}_t$. By definition, it is contained in some n-vertex clique $\sigma \subseteq \mathbb{G}_t$. Since the first vertex of this path is in $\mathbb{A}$ and all vertices in $\mathbb{C}$ are active, $\sigma \not\subseteq \mathbb{C}$. Hence $\sigma \subseteq \mathbb{A}'$, so $\mathbb{A}'$ is nonempty. Similarly, $\mathbb{B}'$ is nonempty.

Every path between $\mathbb{A}'$ and $\mathbb{B}'$ in $\mathbb{G}_t$ contains at least one edge in $\mathbb{C}$. Consider any path $v_0, \dots, v_\ell$ between $\mathbb{A}'$ and $\mathbb{B}'$. Suppose $\{v_i, v_{i+1}\}$ is the first edge on this path that is contained in $\mathbb{C}$. Then $v_i \in \mathbb{A}'$. Hence every path between $\mathbb{A}'$ and $\mathbb{B}'$ in $\mathbb{G}_t$ and, hence, every path between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ contains an edge in $\mathbb{C}$ with one endpoint in $\mathbb{A}'$. By the minimality of $\mathbb{C}$, every clique in $\mathbb{C}$ intersects $\mathbb{A}'$. Similarly, every clique in $\mathbb{C}$ intersects $\mathbb{B}'$. Therefore, every shortest path between $\mathbb{A} \subseteq \mathbb{A}'$ and $\mathbb{B} \subseteq \mathbb{B}'$ in $\mathbb{G}_t$ consists of a path between $\mathbb{A}$ and $\mathbb{A}' \cap \mathbb{C}$, followed by an edge between $\mathbb{A}' \cap \mathbb{C}$ and $\mathbb{B}' \cap \mathbb{C}$, followed by a path between $\mathbb{B}' \cap \mathbb{C}$ and $\mathbb{B}$.

Since $\mathbb{G}_{t+1} = \chi(\mathbb{G}_t, \delta)$ is the union of $\chi(\sigma, \delta)$ for all n-vertex cliques σ in $\mathbb{G}_t$, it follows that $\mathbb{G}_{t+1}$ is the union of the n-vertex cliques in $\chi(\mathbb{A}', \delta)$, $\chi(\mathbb{B}', \delta)$, and $\chi(\mathbb{C}, \delta)$. Furthermore, since $\mathbb{A}'$ and $\mathbb{B}'$ are disjoint, Lemma 4.10 implies that $\chi(\mathbb{A}', \delta)$ and $\chi(\mathbb{B}', \delta)$ are disjoint. Thus, every path between $\chi(\mathbb{A}, \delta) \subseteq \chi(\mathbb{A}', \delta)$ and $\chi(\mathbb{B}, \delta) \subseteq \chi(\mathbb{B}', \delta)$ in $\mathbb{G}_{t+1}$ consists of a path between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{A}', \delta) \cap \chi(\mathbb{C}, \delta)$, followed by a path between $\chi(\mathbb{A}', \delta) \cap \chi(\mathbb{C}, \delta)$ and $\chi(\mathbb{B}', \delta) \cap \chi(\mathbb{C}, \delta)$, followed by a path between $\chi(\mathbb{B}', \delta) \cap \chi(\mathbb{C}, \delta)$ and $\chi(\mathbb{B}, \delta)$.

Since $\chi(\mathbb{A}', \delta) \cap \chi(\mathbb{C}, \delta) \subseteq \chi(\mathbb{C}, \delta)$, the distance between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{A}', \delta) \cap \chi(\mathbb{C}, \delta)$ in $\mathbb{G}_{t+1}$ is at least as large as the distance between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{C}, \delta)$ in $\mathbb{G}_{t+1}$. By Lemma 4.11, the distance between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{C}, \delta)$ in $\mathbb{G}_{t+1}$ is at least as large as the distance between $\mathbb{A}$ and $\mathbb{C}$ in $\mathbb{G}_t$. The distance between $\mathbb{A}$ and $\mathbb{C}$ in $\mathbb{G}_t$ is equal to the distance between $\mathbb{A}$ and $\mathbb{A}' \cap \mathbb{C}$ in $\mathbb{G}_t$, because $\mathbb{A}'$ is the union of all cliques in $\mathbb{F}$ that are connected to $\mathbb{A}$. Hence, the distance between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{A}', \delta) \cap \chi(\mathbb{C}, \delta)$ in $\mathbb{G}_{t+1}$ is at least as large as the distance between $\mathbb{A}$ and $\mathbb{A}' \cap \mathbb{C}$ in $\mathbb{G}_t$.

Similarly, the distance between $\chi(\mathbb{B}', \delta) \cap \chi(\mathbb{C}, \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$ is at least as large as the distance between $\mathbb{B}' \cap \mathbb{C}$ and $\mathbb{B}$ in $\mathbb{G}_t$. By Lemma 4.12, the distance between $\chi(\mathbb{A}', \delta) \cap \chi(\mathbb{C}, \delta)$ and $\chi(\mathbb{B}', \delta) \cap \chi(\mathbb{C}, \delta)$ in $\mathbb{G}_{t+1}$ is at least 2. Therefore,

the distance between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$
$\geq$ the distance between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{A}', \delta) \cap \chi(\mathbb{C}, \delta)$ in $\mathbb{G}_{t+1}$
$+$ the distance between $\chi(\mathbb{A}', \delta) \cap \chi(\mathbb{C}, \delta)$ and $\chi(\mathbb{B}', \delta) \cap \chi(\mathbb{C}, \delta)$ in $\mathbb{G}_{t+1}$
$+$ the distance between $\chi(\mathbb{B}', \delta) \cap \chi(\mathbb{C}, \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$
$\geq$ the distance between $\mathbb{A}$ and $\mathbb{A}' \cap \mathbb{C}$ in $\mathbb{G}_t$
$+ 2$
$+$ the distance between $\mathbb{B}' \cap \mathbb{C}$ and $\mathbb{B}$ in $\mathbb{G}_t$
$>$ the distance between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$.

□

5 Why Extension-Based Proofs Fail

In this section, we prove that no extension-based proof can show the impossibility of deterministically solving k-set agreement in a wait-free manner in the NIS model, for $n > k \geq 2$ processes. Specifically, we define an adversary that is able to win against every extension-based prover. The adversary maintains a *partial* specification of δ (the protocol it is adaptively constructing) and an integer $t \geq 0$. The integer t represents the number of times it has subdivided the input complex, $\mathbb{G}_0$. Once the adversary has defined δ for each vertex in $\mathbb{G}_t$, it may subdivide $\mathbb{G}_t$, construct $\mathbb{G}_{t+1} = \chi(\mathbb{G}_t, \delta)$, and increment t.

For each $0 \leq r \leq t$ and each input value a, let $\mathbb{T}_r(a)$ be the subset of terminated vertices in $\mathbb{V}_r$ that have output a. The following simple property is true because every terminated vertex remains unchanged when a subdivision is performed.

Proposition 5.1. *For all input values a and all $0 \leq r < t$, $\mathbb{T}_r(a) = \chi(\mathbb{T}_r(a), \delta) \subseteq \mathbb{T}_{r+1}(a)$. If the adversary defines $\delta(v) = \perp$ for each vertex $v \in \mathbb{G}_t$ where $\delta(v)$ is undefined, and subdivides $\mathbb{G}_t$ to construct $\mathbb{G}_{t+1}$, but does not terminate any additional vertices in $\mathbb{V}_{t+1}$, then $\mathbb{T}_t(a) = \mathbb{T}_{t+1}(a)$.*

We say that a vertex $v \in \mathbb{V}_0$ *has seen* input value a if it denotes the state of a process whose input has value a. Inductively, we say that $v \in \mathbb{V}_{r+1}$ *has seen* input value a if $v \in \mathbb{V}_r$ and v has seen a or $v = (i, \tau)$ for some subset τ of active vertices of an n-vertex clique in $\mathbb{G}_r$ such that $i \in id(\tau)$ and some vertex in τ has seen a. In other words, if v represents the state of a process p_i in some configuration reachable by an r-round schedule, then v has seen a if and only if p_i had input $x_i = a$ or, in some round $r' \leq r$ of this schedule, there was a process p_j that performed its `update` before p_i performed its `scan` and the vertex representing p_j has seen a in round $r' - 1$. For each $r \geq 0$ and each input value a, let $\mathbb{N}_r(a)$ be the the union of the n-vertex cliques in $\mathbb{G}_r$ none of whose vertices have seen a. To avoid violating validity, the adversary should not let any vertex in $\mathbb{N}_t(a)$ output the value a.

Proposition 5.2. *For $0 \leq r < t$ and for any input value a, $\mathbb{N}_{r+1}(a) = \chi(\mathbb{N}_r(a), \delta)$.*

Proof. Consider any n-vertex clique $\sigma \subseteq \mathbb{N}_r(a)$. Since no vertex in σ has seen a, it follows, by definition, that no vertex in $\chi(\sigma, \delta)$ has seen a. Thus $\chi(\sigma, \delta) \subseteq \mathbb{N}_{r+1}(a)$ and, hence, $\chi(\mathbb{N}_r(a), \delta) \subseteq \mathbb{N}_{r+1}(a)$.

Conversely, consider any n-vertex clique $\sigma' \subseteq \mathbb{N}_{r+1}(a)$. By definition of $\mathbb{G}_{r+1}$, $\sigma' = \chi(\sigma, \delta)$ for some n-vertex clique σ in $\mathbb{G}_r$. If some vertex in σ has seen a, then the process in σ' with the same *id* has seen a. But none of the vertices in σ' have seen a, so none of the vertices in σ have seen a. Hence $\sigma \subseteq \mathbb{N}_r(a)$ and $\sigma' \subseteq \chi(\mathbb{N}_r(a), \delta)$. Therefore $\mathbb{N}_{r+1}(a) \subseteq \chi(\mathbb{N}_r(a), \delta)$. □

In fact, every vertex in $\mathbb{G}_r$ that has not seen a is in $\mathbb{N}_r(a)$. This is the special case of the following lemma when τ contains only one vertex.

Lemma 5.3. *If τ is a subset of an n-vertex clique in $\mathbb{G}_r$ and no vertex in τ has seen the input value a, then τ is a subset of an n-vertex clique in $\mathbb{N}_r(a)$.*

Proof. The proof is by induction on r. Every two vertices in $\mathbb{G}_0$ that have not seen a are adjacent provided they represent the states of different processes, i.e. they have different *id*s. Thus, if τ is a subset of an n-vertex clique in $\mathbb{G}_0$, no vertex in τ has seen the input value a, and $b \neq a$, then $\tau \cup \{(j, b) \mid j \notin id(\tau)\}$ are the vertices of an n-vertex clique in $\mathbb{N}_0(a)$.

Let $r \geq 0$ and assume the claim is true for r. Consider any n-vertex clique σ' in $\mathbb{G}_{r+1}$. Let τ' be the subset of all vertices of σ' that have not seen a. Since $\mathbb{G}_{r+1} = \chi(\mathbb{G}_r, \delta)$, there exists an n-vertex clique σ in $\mathbb{G}_r$ such that σ' is in $\chi(\sigma, \delta)$. Let $\tau = \{v \in \sigma \mid id(v) \in id(\tau')\}$. Note that, by definition, if $u \in \sigma$ has seen a, then every vertex $u' \in \chi(\sigma, \delta)$ with $id(u') = id(u)$ has seen a. Hence, no vertex in τ has seen a. By the induction hypothesis, there exists an n-vertex clique ρ in $\mathbb{N}_r(a)$ that contains τ.

By definition of $\chi(\rho, \delta)$, it contains each vertex of τ'. Since $\tau' \subseteq \sigma'$, the vertices in τ' are adjacent to one another. Let $active(\rho)$ denote the set of active vertices in ρ and let $\rho' = \{(j, active(\rho)) \mid j \in id(active(\rho)) - id(\tau')\} \subseteq \chi(\rho, \delta)$. The vertices in ρ' are adjacent to one another and to each vertex in τ'. Furthermore each terminated vertex in ρ is adjacent to all the vertices in τ' and ρ'. Hence, these vertices form an n-vertex clique in $\chi(\rho, \delta) \subseteq \mathbb{N}_{r+1}(a)$. Thus the claim is true for $r + 1$. □

For each $0 \leq r \leq t$ and each input value a, let $\mathbb{X}_t(a)$ be the subset of vertices in $\mathbb{V}_r$ that represent the states of processes in P in configurations reachable reachable from C by P-only schedules, for all output queries (C, P, a) to which the adversary answered NONE. To avoid contradicting its responses, the adversary should not let any vertex in $\mathbb{X}_t(a)$ output the value a.

Throughout the first phase, the adversary ensures that the following invariants hold after its response to each query:

1. For each $0 \leq r < t$ and each vertex $v \in \mathbb{V}_r$, $\delta(v)$ is defined.

2. If $v \in \mathbb{V}_t$, then $\delta(v) \neq \perp$.

3. Suppose s is the state of a process in configuration $C \in \mathcal{A}(1) \cup \mathcal{A}'(1)$ and $C = C_0\beta$ for some initial configuration $C_0 \in \mathcal{A}(1)$. The process occurs at most $2t + 1$ times in β. If the process occurs $2r$ times in β, then $s \in \mathbb{V}_r$ and $\delta(s)$ is defined.

4. For any input value a, if $\mathbb{T}_t(a)$ is nonempty, then the distance between $\mathbb{T}_t(a)$ and $\mathbb{N}_t(a)$ in $\mathbb{G}_t$ is at least 2.

5. For any two input values $a \neq b$, if $\mathbb{T}_t(a)$ and $\mathbb{T}_t(b)$ are nonempty, then the distance between them in $\mathbb{G}_t$ is at least 3.

6. For every input a and every vertex $v \in \mathbb{X}_t(a)$, either $v \in \mathbb{N}_t(a)$ or there exists an input $b \neq a$ such that v is distance at most 1 from $\mathbb{T}_t(b)$.

There is nothing special about the values 2 and 3. They are simply the smallest values such that the invariants can be maintained and every chain of queries is finite. The following lemma is a consequence of the invariants.

Lemma 5.4. *For any two input values $a \neq b$, every path between $\mathbb{T}_t(a)$ and $\mathbb{N}_t(a) \cup \mathbb{T}_t(b)$ in $\mathbb{G}_t$ contains at least one edge between active vertices.*

Proof. Consider any path $v_0, v_1, \ldots, v_\ell$ between $\mathbb{T}_t(a)$ and $\mathbb{N}_t(a) \cup \mathbb{T}_t(b)$ in $\mathbb{G}_t$. Let v_j be the last vertex in $\mathbb{T}_t(a)$. Since the invariants hold after each query and $v_\ell \in \mathbb{T}_t(b) \cup \mathbb{N}_t(a)$, invariants 4 and 5 imply that the distance between v_j and v_ℓ is at least 2. Hence, $\ell \geq j+2$. Since v_j is the last vertex in $\mathbb{T}_t(a)$, $v_{j+1}, v_{j+2} \notin \mathbb{T}_t(a)$. Moreover, by invariant 5, $v_{j+1}, v_{j+2} \notin \mathbb{T}_t(c)$ for any input value $c \neq a$. Hence, $\{v_{j+1}, v_{j+2}\}$ is an edge between active vertices. □

Essentially, a subdivision maintains the invariants, but increases the distance increases between vertices that output different values and between vertices that output a and vertices that have not seen a.

Lemma 5.5. *Suppose all the invariants hold, the adversary defines $\delta(v) = \bot$ for each vertex $v \in \mathbb{V}_t$ such that $\delta(v)$ is undefined, and subdivides $\mathbb{G}_t$ to construct $\mathbb{G}_{t+1}$. If $\mathbb{T}_t(a)$ is nonempty, then the distance between $\mathbb{T}_{t+1}(a)$ and $\mathbb{N}_{t+1}(a)$ in $\mathbb{G}_{t+1}$ is greater than the distance between $\mathbb{T}_t(a)$ and $\mathbb{N}_t(a)$ in $\mathbb{G}_t$. If $b \neq a$ and $\mathbb{T}_t(b)$ is also nonempty, then the distance between $\mathbb{T}_{t+1}(a)$ and $\mathbb{T}_{t+1}(b)$ in $\mathbb{G}_{t+1}$ is greater than the distance between $\mathbb{T}_t(a)$ and $\mathbb{T}_t(b)$ in $\mathbb{G}_t$. Furthermore, if the adversary increments t, then all the invariants hold.*

Proof. By Proposition 5.1, $\mathbb{T}_{t+1}(a) = \mathbb{T}_t(a) = \chi(\mathbb{T}_t(a), \delta)$, for each input a. In addition, $\mathbb{N}_{t+1}(a) = \chi(\mathbb{N}_t(a), \Delta)$ by Proposition 5.2. Lemma 5.4 says that every path between $\mathbb{T}_t(a)$ and $\mathbb{N}_t(a) \cup \mathbb{T}_t(b)$ in $\mathbb{G}_t$ contains at least one edge between active vertices. Therefore, if $\mathbb{T}_t(a)$ is nonempty, Lemma 4.13 implies that the distance between $\chi(\mathbb{T}_t(a), \delta) = \mathbb{T}_{t+1}(a)$ and $\chi(\mathbb{N}_t(a), \delta) = \mathbb{N}_{t+1}(a)$ in $\mathbb{G}_{t+1}$ is greater than the distance between $\mathbb{T}_t(a)$ and $\mathbb{N}_t(a)$ in $\mathbb{G}_t$. Similarly, if both $\mathbb{T}_t(a)$ and $\mathbb{T}_t(b)$ are nonempty, Lemma 4.13 implies that the distance between $\chi(\mathbb{T}_t(t), \delta) = \mathbb{T}_{t+1}(a)$ and $\chi(\mathbb{T}_t(b), \delta) = \mathbb{T}_{t+1}(b)$ in $\mathbb{G}_{t+1}$ is greater than the distance between $\mathbb{T}_t(a)$ and $\mathbb{T}_t(b)$ in $\mathbb{G}_t$. Hence invariants 4 and 5 remain true after t is incremented.

Before incrementing t, the adversary defines $\delta(v) = \bot$ for each vertex $v \in \mathbb{G}_t$ where $\delta(v)$ was undefined. Since invariant 1 was true, it remains true. Invariant 2 remains true by construction and the definition of χ. Invariant 3 is not affected. Invariant 6 remains true by the definition of χ. □

The adversarial strategy for phase 1. Initially, the adversary sets $\delta(v) = \bot$ for each vertex $v \in \mathbb{G}_0$, it subdivides $\mathbb{G}_0$ to construct $\mathbb{G}_1$, and it sets $t = 1$. By construction, invariants 1 and 2 are true. Before the first query, $\mathcal{A}'(1)$ is empty. Since $\mathcal{A}(1)$ is the set of all initial configurations and $\mathbb{G}_0$ represents all initial configurations, invariant 3 is true.

No vertices in $\mathbb{G}_1$ have terminated, so $\mathbb{T}_1(a)$ is empty for all inputs a. Since there have been no output queries, $\mathbb{X}_1(a)$ is empty for all inputs a. Therefore invariants 4, 5, and 6 are vacuously true.

Now suppose that the invariants are true immediately prior to some query (C, q) in phase 1, where $C \in \mathcal{A}(1) \cup \mathcal{A}'(1)$ and q is a process that is active in C. Note that the prover already knows the state of every process in configuration C, including which of them have terminated. Let β be a schedule from an initial configuration $C_0 \in \mathcal{A}(1)$ such that $C = C_0\beta$ and $C_0\beta' \in \mathcal{A}'(1)$ for every nonempty prefix β' of β.

If q occurs $2r$ times in β, then, by invariant 3, $0 \leq r \leq t$ and the state s of q in configuration C is a vertex in $\mathbb{G}_r$. Since q is active in C, $\delta(s) = \bot$. Hence, by invariant 2, $r < t$. In this case, the adversary returns the configuration Cq, which is the same as C except that $S_{r+1}[id(q)] = (id(q), s)$ and the state of q has an extra bit indicating that it last performed an `update`. Note that q is active in this state. Invariant 3 is true for configuration $Cq = C_0\beta q$ since p_i occurs $2r+1$ times in βq and every other process is in the same state in configurations C and Cq. Since δ has not been changed by the adversary, $\mathbb{T}_t(a)$ is unchanged for all inputs a and invariants 1, 2 4, and 5 remain true. Since no vertices are added to $\mathbb{X}_t(a)$ for any input a, invariant 6 remains true.

So, suppose that q occurs $2r+1$ times in β. Let β' be the longest prefix of β in which q occurs $2r$ times. Then $0 \leq r \leq t$ and the state s of q in configuration $C_0\beta'$ is a vertex in $\mathbb{G}_r$, by invariant 3. Since q in active in C, it is active in configuration $C_0\beta'$, so $\delta(s) = \bot$. Hence, by invariant 2, $r < t$.

The state s' of q in configuration Cq is $(id(q), \sigma)$, where σ is the result of its `scan` of S_{r+1}. It is a vertex in $\mathbb{G}_{r+1}$. Note that, by Observation 4.1, the contents of S_{r+1} are determined by the states of all processes in C. If $r < t-1$, then $\delta(s')$ is defined, by invariant 1. It is also possible that $r = t-1$ and $\delta(s')$ is defined. In both these cases, the adversary returns configuration Cq, which is the same as C, except for the state of q and, if $\delta(s') \neq \bot$, the value it outputs. As above, all the invariants continue to hold.

Now, suppose that $r = t-1$ and $\delta(s')$ is not defined. If there exists an input a such that setting $\delta(s') = a$ maintains all the invariants, then the adversary defines $\delta(s') = a$ and returns configuration Cq, which is the same as C except for the state of q and the fact that q outputs a. In this case, the distance between s' and $\mathbb{N}_t(a)$ is at least 2 and, for all inputs $b \neq a$ such that $\mathbb{T}_t(b)$ is nonempty, the distance between s' and $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ is at least 3. The vertex s' is added to $\mathbb{T}_t(a)$. The sets $\mathbb{T}_t(b)$, for all inputs $b \neq a$, and the sets $\mathbb{N}_t(b)$ and $\mathbb{X}_t(b)$, for all inputs b, are unchanged. Hence, invariants 1, 2, 4, 5, and 6 continue to hold. By construction, $s' \in \mathbb{G}_t$ and $\delta(s')$ is defined. For every other process, its state in Cq is the same as its state in C. Thus, invariant 3 continues to hold. By invariant 6, each vertex $u \in \mathbb{X}_t(a)$ is either in $\mathbb{N}_t(a)$ or is at distance at most 1 from $\mathbb{T}_t(b)$ for some $b \neq a$. Since the distance between s' and $\mathbb{N}_t(a)$ is at least 2 and the distance between s' and $\mathbb{T}_t(b)$ is at least 3, the distance between s' and u is at least 2. Thus $s' \notin \mathbb{X}_t(a)$, so defining $\delta(s') = a$ does not contradict the result of any previous output query.

Otherwise, the adversary defines $\delta(v) = \bot$ for each vertex $v \in \mathbb{V}_t$ where $\delta(v)$ is undefined, including s', subdivides $\mathbb{G}_t$ to construct $\mathbb{G}_{t+1}$, and increments t. By Lemma 5.5, all the invariants continue to hold. The adversary returns configuration Cq, which is the same as C except for the state of q.

Finally, suppose that the invariants are true immediately prior to some output query (C, Q, y) in phase 1, where $C \in \mathcal{A}(1) \cup \mathcal{A}'(1)$, each process $q \in Q$ is active in C, and y is a possible output value. Let $\mathbb{Q}$ be the set of vertices in $\mathbb{G}_t$ that represent the states of processes in Q in configurations reachable from C via Q-only schedules.

If some vertex $v \in \mathbb{Q}$ has terminated with output y, then the adversary returns a Q-only schedule from C that leads to a configuration C' in which v represents the state of a process in C'. None of the invariants are affected.

If every vertex in $\mathbb{Q}$ is in $\mathbb{N}_t(y)$, $\mathbb{X}_t(y)$, or $\mathbb{T}_t(a)$, for some $a \neq y$, then it would be impossible for the adversary to return a Q-only schedule from C in which some vertex has terminated with output y without violating validity or contradicting one of its previous answers. In this case, the adversary adds $\mathbb{Q}$ to $\mathbb{X}_t(y)$ and returns NONE. Note that adding vertices in $\mathbb{N}_t(y)$ or $\mathbb{T}_t(a)$ for $a \neq y$ does not make invariant 6 false. The other invariants are not affected.

Otherwise, let $\mathbb{U} \neq \emptyset$ be the subset of vertices in $\mathbb{Q}$ that are not in $\mathbb{N}_t(y)$, $\mathbb{X}_t(y)$, or $\mathbb{T}_t(a)$, for some $a \neq y$. For each vertex $u \in \mathbb{U}$, let $\mathbb{A}_u$ be the union of all n-vertex cliques in $\mathbb{G}_t$ containing u. We consider three cases.

Case 1: *There is a vertex $u \in \mathbb{U}$ such that $\mathbb{A}_u \cap \mathbb{T}_t(y)$ is nonempty.* The adversary defines $\delta(v) = \perp$ for each vertex $v \in \mathbb{V}_t$ where $\delta(v)$ is undefined and subdivides $\mathbb{G}_t$ to construct $\mathbb{G}_{t+1}$. By invariant 4 and Lemma 5.5, the distance between $\mathbb{T}_{t+1}(y)$ and $\mathbb{N}_{t+1}(y)$ in $\mathbb{G}_{t+1}$ is at least 3. If $a \neq y$ and $\mathbb{T}_t(a)$ is nonempty, then Proposition 5.1 says that $\mathbb{T}_{t+1}(a)$ is nonempty and, by invariant 5 and Lemma 5.5, the distance between $\mathbb{T}_{t+1}(y)$ and $\mathbb{T}_{t+1}(a)$ is at least 4.

Let $i = id(u)$ and $v = (i, \{u\})$. Since $u \in \mathbb{U} \subseteq \mathbb{Q}$, process $p_i \in Q$. Let $w \in \mathbb{A}_u \cap \mathbb{T}_t(y)$, let σ be an n-vertex clique in $\mathbb{A}_u$ that contains w, and let C' be the configuration represented by σ. Then v is the state of process p_i in configuration $C'p_ip_i$.

Next, the adversary increments t, so all the invariants continue to hold by Lemma 5.5. Finally, the adversary defines $\delta(v) = y$, returns a Q-only schedule from C that results in process p_i being in state v. This adds vertex v to $\mathbb{T}_t(y)$. Invariants 1, 2, 3, and 6 continue to hold.

Since w is terminated, it is adjacent to every other vertex in $\chi(\sigma, \delta) \subseteq \mathbb{G}_t$, including v. It follows that the distance between v and $\mathbb{N}_t(y)$ is at least 2 and, if $a \neq y$ and $\mathbb{T}_t(a)$ is nonempty, then the distance between v and $\mathbb{T}_t(a)$ is at least 3. Thus, invariants 4 and 5 hold.

By invariant 6, each vertex in $\mathbb{X}_t(y)$ is adjacent to a vertex in $\mathbb{T}_t(b)$ for some $b \neq y$. Since the distance between v and $\mathbb{T}_t(b)$ is at least 3, the distance between v and $\mathbb{X}_t(y)$ is at least 2. Thus $v \notin \mathbb{X}_t(y)$. Hence, defining $\delta(v) = y$ does not contradict the result of any previous output query.

Case 2: *There is a vertex $u \in \mathbb{U}$ such that every vertex in $\mathbb{A}_u$ is active.* The adversary defines $\delta(v) = \perp$ for each vertex $v \in \mathbb{G}_t$ where $\delta(v)$ is undefined and subdivides $\mathbb{G}_t$ to construct $\mathbb{G}_{t+1}$.

Since no vertex in $\mathbb{A}_u$ has terminated and $\mathbb{A}_u$ contains all vertices at distance at most 1 from u in $\mathbb{G}_t$, it follows that the distance from u to $\mathbb{T}_t(a)$ in $\mathbb{G}_t$ is at least 2, for all inputs a. Moreover, since $u \notin \mathbb{N}_t(y)$, the distance from u to $\mathbb{N}_t(y)$ in $\mathbb{G}_t$ is at least 1.

Let $i = id(u)$ and let $v = (i, \{u\}) \in \mathbb{G}_{t+1}$. Since $u \in \mathbb{U} \subseteq \mathbb{Q}$, process $p_i \in Q$. Consider any vertex v' adjacent to v in $\mathbb{G}_{t+1}$. Then there exists an n-vertex clique $\sigma \subseteq \mathbb{G}_t$ such that $v, v' \in \chi(\sigma, \delta)$. Since v is not a terminated vertex in σ, $\{u\} \subseteq \sigma$, so $\sigma \subseteq \mathbb{A}_u$. All vertices in $\mathbb{A}_u$ are active, so $v' = (j, \rho)$ where $j \neq i$, $j \in id(\rho)$, $\rho \subseteq \sigma$, and $\{u\} \subseteq \rho$. Note that $u \notin \mathbb{N}_t(y)$ implies that $v, v' \notin \mathbb{N}_{t+1}(y)$. Therefore, the distance from v to $\mathbb{N}_{t+1}(y)$ in $\mathbb{G}_{t+1}$ is at least 2.

Next, we show that, for all inputs a, the distance from v to $\mathbb{T}_{t+1}(a)$ in $\mathbb{G}_{t+1}$ is at least 3. By Proposition 5.1, $\mathbb{T}_{t+1}(a) = \mathbb{T}_t(a) = \chi(\mathbb{T}_t(a), \delta)$, so no vertex v' adjacent to v in $\mathbb{G}_{t+1}$ is in $\mathbb{T}_{t+1}(a)$. To obtain a contradiction, suppose there is a path v, v', w of length 2 in $\mathbb{G}_{t+1}$ from v to $\mathbb{T}_{t+1}(a)$. Then $v' = (j, \rho)$, where $\{u\} \subseteq \rho$ and there exists an n-vertex clique $\sigma' \subseteq \mathbb{G}_t$ such that $\{v', w\}$ is an edge in $\chi(\sigma', \delta)$. Because $w \in \mathbb{T}_{t+1}(a) = \mathbb{T}_t(a)$, $w \in \sigma'$. By definition, $\rho \subseteq \sigma'$. This implies that $u \in \sigma'$ and, hence, $\sigma' \subseteq \mathbb{A}_u$. However, this contradicts the assumption that all vertices in $\mathbb{A}_u$ are active. Therefore, the distance from v to $\mathbb{T}_{t+1}(a)$ in $\mathbb{G}_{t+1}$ is at least 3 for all inputs a.

Now the adversary increments t, so all the invariants continue to hold, by Lemma 5.5. Finally, the adversary defines $\delta(v) = y$ and returns a Q-only schedule from C that results in process p_i being in state v. This adds vertex v to $\mathbb{T}_t(y)$. Invariants 1, 2, 3, and 6 continue to hold. Since the distance from v to $\mathbb{N}_t(y)$ in $\mathbb{G}_t$ is at least 2 and the distance from v to $\mathbb{T}_{t+1}(a)$ in $\mathbb{G}_{t+1}$ is at least 3 for all inputs a, invariants 4 and 5 hold. As in the previous case, defining $\delta(v) = y$ does not contradict the result of any previous output query.

Case 3. *For every vertex $u \in \mathbb{U}$, $\mathbb{A}_u \cap \mathbb{T}_t(y)$ is empty, but some vertex in $\mathbb{A}_u$ has terminated.* In this case, the adversary returns NONE and adds $\mathbb{U}$ to $\mathbb{X}_t(y)$. Since each vertex $u \in \mathbb{U}$ is adjacent to some vertex in $\mathbb{A}_u$ that has terminated with an output other than y, invariant 6 holds. Invariants 1, 2, and 3 still hold, since t and δ are not changed, and invariants 4 and 5 still hold, since $\mathbb{T}_t(a)$ and $\mathbb{N}_t(a)$ are not changed for any input a.

The prover does not win in phase 1. Suppose that the invariants all hold before and after each query made by the prover in phase 1. By invariant 5, at most one value is output in any configuration reached by the prover. Moreover, by invariant 4, if a process outputs value a, then it has seen a. Hence, the prover cannot win in phase 1 by showing that the protocol violates agreement or validity. It remains to show that the prover cannot win by constructing an infinite chain of queries in phase 1.

Lemma 5.6. *Every chain of queries in phase 1 is finite.*

Proof. Assume, for a contradiction, that there is an infinite chain of queries, $(C_1, q_1), (C_2, q_2), \ldots$ Let β be a schedule from an initial configuration C_0 to C_1 followed by the steps of the schedule $q_1, q_2, \ldots$ and, for each $j \geq 1$, let β_j be the prefix of β such that $C_j = C_0\beta_j$. Let P be the set of processes that occur infinitely often in β. Let $j' \geq 1$ be the first index such that $q_j \in P$ for all $j \geq j'$, so, from j' onwards, only processes in P appear in queries. Let $t' \geq 1$ be the value of t held by the adversary immediately prior to query $(C_{j'}, q_{j'})$. By invariant 3, each process occurs at most $2t+1$ times in $\beta_{j'}$. Hence, during the schedule $\beta_{j'}$ from C_0, no process performed an `update` to S_r for $r > t'$ or a `scan` of S_r for $r \geq t'$. Since each process in P eventually accesses every snapshot object, the adversary eventually defines $\delta(v) = \bot$ for each vertex $v \in \mathbb{G}_r$ where $\delta(v)$ is undefined and subdivides $\mathbb{G}_r$ to construct $\mathbb{G}_{r+1}$, for all $r \geq t'$. Since no process is terminated, $\mathbb{T}_r(a) = \mathbb{T}_{t'}(a)$, for all inputs a and all $r > t'$. By Lemma 5.4 and Lemma 5.5, if $\mathbb{T}_{t'}(a)$ is nonempty, the distance between $\mathbb{T}_{t'+2}(a)$ and $\mathbb{N}_{t'+2}(a)$ is at least 4 and, if $b \neq a$ and $\mathbb{T}_{t'}(b)$ is nonempty, the distance between $\mathbb{T}_{t'+2}(a)$ and $\mathbb{T}_{t'+2}(b)$ is at least 5.

Consider the first index $j'' > j'$ such that process $q_{j''}$ is poised to `scan` the snapshot object $S_{t'+2}$ in $C_{j''}$. By invariant 3, the state of process in $q_{j''}$ in configuration $C_{j''+1} = Cj''q_{j''}$ is a vertex $v \in \mathbb{V}_{t'+2}$. If there is some input a such that the distance from v to $\mathbb{T}_{t'+2}(a)$ in $\mathbb{G}_{t'+2}$ is at most 2, then the distance from v to $\mathbb{N}_{t'+2}(a)$ in $\mathbb{G}_{t'+2}$ is at least 2 and the distance from v to $\mathbb{T}_{t'+2}(b)$ in $\mathbb{G}_{t'+2}$ is at least 3. According to its strategy for phase 1, the adversary defines $\delta(v) = a$ after query $(C_{j''}, q_{j''})$. This contradicts the definition of P. Thus, the distance from v to $\mathbb{T}_{t+2}(a)$ in $\mathbb{G}_{t'+2}$ is at least 3, for all inputs a such that $\mathbb{T}_a^{t'+2}$ is nonempty.

Let a be the input of process $q_{j''}$ in configuration C_0. Consider any $(t'+2)$-round schedule β'' obtained from β by removing all but the first $2(t'+2)$ occurrences of processes in P and then appending sufficiently many occurrences of the processes not in P. Note that configurations $C_0\beta''$ and $C_0\beta_{j''+1}$ are indistinguishable to process $q_{j''}$, so v is in the n-vertex clique σ in $\mathbb{G}_{t'+2}$ representing the configuration $C_0\beta''$. Thus the distance in $\mathbb{G}_{t'+2}$ between σ and any terminated vertex is at least 2. During schedule β'' from C_0, $q_{j''}$ performs its `update` to $S_{t'+2}$ before any process performs its `scan` of $S_{t'+2}$, so all vertices in σ have seen a. Thus the distance in $\mathbb{G}_{t'+2}$ between σ and $\mathbb{N}_{t'+2}(a)$ is at least 1. The first edge on every path from σ to $\mathbb{N}_{t'+2}(a)$ or to $\mathbb{T}_{t'+2}(b)$, for any input b, is between active vertices. Therefore, by Proposition 5.2, Proposition 5.1, and Lemma 4.13, the distance in $\mathbb{G}_{t'+3}$ between $\chi(\sigma, \delta)$ and $\chi(\mathbb{N}_{t'+2}(a), \delta) = \mathbb{N}_{t'+3}(a)$ is at least 2 and the distance in $\mathbb{G}_{t'+3}$ between $\chi(\sigma, \delta)$ and $\chi(\mathbb{T}_{t'+2}(b), \delta) = \mathbb{T}_{t'+3}(b)$ is at least 3, for any input b.

Consider the first index $j''' > j''$ such that process $q_{j'''}$ is poised to `scan` the snapshot object $S_{t'+3}$ in $C_{q'''}$. The state of process $q_{j'''}$ in configuration $C_{j'''}q_{j'''}$ is a vertex in $\chi(\sigma, \delta)$. According to its strategy for phase 1, the adversary terminates this vertex after query $(C_{j'''}, q_{j'''})$. This contradicts the definition of P. □

Since the prover does not win in phase 1, it must eventually commit to a nonempty schedule $\alpha(2)$ from an initial configuration $C \in \mathcal{A}(1)$ such that $C\alpha(2) \in \mathcal{A}'(1)$, set $\mathcal{B}(2)$ to consist of all initial configurations that only differ from C by the states of processes that do not occur in $\alpha(2)$, set $\mathcal{A}(2) = \{C_0\alpha(2) \mid C_0 \in \mathcal{B}(2)\}$, and then start phase 2.

The adversarial strategy for later phases. At the beginning of phase 2, the adversary updates δ. Afterwards, it can answer all future queries by the prover without making any further changes to δ. Eventually, at the end of some future phase φ, the prover will commit to a schedule $\alpha(\varphi+1)$ such that all configurations in $\mathcal{A}(\varphi+1)$ are final. Consequently, the prover will lose at the beginning of phase $\varphi + 1$.

Let p be the first process in $\alpha(2)$ and let a be the input of p in the initial configuration C. Note that p has the same state in every configuration in $\mathcal{B}(2)$, so it has input a in all of them. Let $\mathbb{F}$ denote the union of all n-vertex cliques in $\mathbb{G}_1$ that represent a configuration reachable by a 1-round schedule beginning with p from a configuration in $\mathcal{B}(2)$. Since p performs its `update` to S_1 before any process performs its `scan` of S_1 in all such schedules, every vertex in $\mathbb{F}$ has seen a. Thus the distance between $\mathbb{F}$ and $\mathbb{N}_1(a)$ in $\mathbb{G}_1$ is at least 1.

The adversary defines $\delta(v) = \bot$ for each vertex $v \in \mathbb{V}_t$ where $\delta(v)$ is undefined, subdivides $\mathbb{G}_t$ to construct $\mathbb{G}_{t+1}$, and increments t. Since all the invariants hold at the end of phase 1, Lemma 5.5 says that they still hold and, for any two inputs $b \neq b'$ such that $\mathbb{T}_t(b)$ and $\mathbb{T}_t(b')$ are non-empty, the distance between $\mathbb{T}_t(b)$ and $\mathbb{T}_t(b')$ in $\mathbb{G}_t$ is at least 4. In particular, a vertex $v \in \mathbb{G}_t$ is adjacent to a vertex $w \in \mathbb{T}_t(b)$ for at most one input b. Let $\mathbb{F}' = \chi^{t-1}(\mathbb{F}, \delta) \subseteq \mathbb{G}_t$. Applying Lemma 5.5 $t-1$ times, it follows that the distance between $\mathbb{F}'$ and $\mathbb{N}_t(a)$ in $\mathbb{G}_t$ is at least 1.

Invariant 2 says that no vertex in $\mathbb{G}_t$ has $\delta(v) = \bot$. The adversary has not yet terminated any additional vertices in $\mathbb{G}_t$, so, by Proposition 5.1, $\mathbb{T}_t(b) = \mathbb{T}_{t-1}(b)$ for all input values b. For every vertex $v \in \mathbb{F}'$ for which $\delta(v)$ is undefined, the adversary defines $\delta(v)$ as follows. First, for each input value b and each vertex $v \in \mathbb{F}'$ that is distance 1 from $\mathbb{T}_{t-1}(b)$ in $\mathbb{G}_t$ and such that $\delta(v)$ is undefined, the adversary sets $\delta(v) = b$. This does not violate validity, since the distance between $\mathbb{T}_{t-1}(b)$ and $\mathbb{N}_t(b)$ in $\mathbb{G}_t$ is at least 2. Since each vertex in $\mathbb{X}_t(b)$ is adjacent to a vertex in $\mathbb{T}_{t-1}(b')$ for some $b' \neq b$, this assignment defines $\delta(v)$ for each vertex in $\mathbb{X}_t(b)$ for which it was undefined. Since each vertex in $\mathbb{X}_t(b)$ is at least distance 3 from any vertex in $\mathbb{T}_{t-1}(b)$, this assignment does not contradict any output query that returned NONE. Moreover, the distance between any two vertices in $\mathbb{F}'$ that have output different values is still at least 2. Thus, in each n-vertex simplex in $\mathbb{G}_t$, all the terminated vertices have output the same value.

Finally, for each vertex $v \in \mathbb{F}'$ where $\delta(v)$ is still undefined, the adversary sets $\delta(v) = a$. Validity is not violated, since no vertex in $\mathbb{F}'$ is in $\mathbb{N}_t(a)$. Agreement is not violated, since at most two different values are output by the vertices in each n-vertex simplex in $\mathbb{G}_t$.

In phases $\varphi \geq 2$, the prover can only query configurations reachable from some configuration in $\mathcal{A}(2)$. By definition, $\mathcal{A}(2)$ is the set of all configurations that are reached by performing $\alpha(2)$ from initial configurations in $\mathcal{B}(2)$. It follows that, for any process q and any extension α' of $\alpha(2)$ from $C' \in \mathcal{A}(2)$, q appears at most $2t$ times in $\alpha(2)\alpha'$ before its state is represented by a vertex in $\mathbb{F}'$. By construction, every vertex in $\mathbb{F}'$ has terminated. Thus, eventually, the prover chooses a configuration at the end of some phase in which every process has terminated. The prover loses in the next phase.

Thus, we have proved the following result:

Theorem 5.7. *No extension-based proof can show the impossibility of deterministically solving k-set agreement in a wait-free manner in the NIS model, for $n > k \geq 2$ processes.*

6 Conclusions

We have shown the limitation of extension-based proofs, including valency arguments, for proving the impossibility of deterministic, wait-free solutions to set-agreement in the NIS model. In the conference version of this paper [AAE+19], we obtained the same result in the NIIS model. Although we have restricted attention to the proof of impossibility of one problem in two closely related models, our approach should be applicable to other problems and other models. For example, we believe that there is no extension-based proof of the lower bound on the number of rounds to solve set agreement in synchronous message passing systems.

Recently, Alistarh, Ellen, and Rybicki [AER20] proved that there is no extension-based proof of the impossibility of deterministic, wait-free solutions to 4-cycle agreement for $n \geq 3$ processes in the NIIS model. This result helped lead to their impossibility proof for this problem, which turned out to be similar to the impossibility proof for set agreement.

There are two other results in distributed computing that have a similar flavour. Rincon Galeana, Winkler, Schmid and Rajsbaum [GWSR19] showed that partitioning arguments are insufficient to prove the impossibility of $(n-1)$-set agreement in the iterated immediate snapshot (IIS) model. For the CONGEST model, Bachrach, Censor-Hillel, Dory, Efron, Leitersdorf and Paz [BCD+19] showed that reductions from two party communication complexity with a static cut cannot be used to prove non-constant lower bounds on the number of rounds needed to solve maximum matching or maximum flow.

Combinatorial topology has been used to prove the impossibility of wait-free solutions to problems other than set agreement, such as weak symmetry breaking and renaming [CR10]. There are no extension-based proofs of these results and we conjecture that they cannot be proved using extension-based proofs.

The definition of an extension-based proof can be modified to handle other termination conditions, such as obstruction-freedom [HLM03]. It suffices for the prover to construct a schedule that violates this condition.

The NIS and NIIS models are computationally equivalent to an asynchronous shared memory model in which processes communicate by reading from and writing to shared registers. However, these models are not equivalent in terms of space and step complexities. A covering argument [BL93] is a standard approach for proving a lower bound on the number of registers needed to solve a problem in an asynchronous system. We have a definition for extension-based proofs that includes covering arguments.

Ellen, Gelashvili and Zhu [EGZ18] proved that any obstruction-free protocol for k-set agreement among $n > k \geq 2$ processes requires $\lceil n/k \rceil$ registers, but their proof is not extension-based. In fact, some of the early work about extension-based proofs motivated the approach in [EGZ18]. We conjecture that it is impossible to prove a non-constant lower bound on the number of registers needed by any obstruction-free protocol for k-set agreement using an extension-based proof.

We have considered allowing the prover to perform a number of other types of queries and can extend our adversarial protocol so that it can answer them. For example, if a prover asks the same output query (C, P, y) multiple times, the protocol could be required to return different schedules each time, until it has returned all possible P-only schedules from C that output y.

We cannot allow certain queries, such as asking for an upper bound on the length of any schedule. If the prover is given such an upper bound, then it can perform a finite number of chains of queries to examine all reachable configurations, thereby fixing the protocol. However, we can allow the prover to use this information in a restricted way and still construct an adversarial set agreement protocol. For example, we might require that the prover does not use this information to decide which queries to perform or what extensions to construct, but can use this information to win when it has constructed a schedule that is longer than this upper bound.

7 Acknowledgments

Support is gratefully acknowledged from the Natural Science and Engineering Research Council of Canada under grants RGPIN-2015-05080 and RGPIN-2020-04178, a University of Toronto post-doctoral fellowship, National Science Foundation under grants CCF-1217921, CCF-1301926, CCF-1637385, CCF-1650596, and IIS-1447786, the Department of Energy under grant ER26116/DE-SC0008923, and the Oracle and Intel corporations. We would also like to thank Toniann Pitassi for helpful discussions and Shi Hao Liu for his useful feedback on an earlier draft of this paper.

References

[AAD+93] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *jacm*, 40(4):873–890, 1993.

[AAE+19] Dan Alistarh, James Aspnes, Faith Ellen, Rati Gelashvili, and Leqi Zhu. Why extension-based proofs fail. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 986–996, 2019.

[Abr88] Karl Abrahamson. On achieving consensus using a shared memory. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 291–302, 1988.

[AC11] Hagit Attiya and Armando Castañeda. A non-topological proof for the impossibility of k-set agreement. In *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 108–119, 2011.

[AE14] Hagit Attiya and Faith Ellen. *Impossibility Results for Distributed Computing.* Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2014.

[AER20] Dan Alistarh, Faith Ellen, and Joel Rybicki. Approximate agreement is hard on cycles. manuscript, 2020.

[AP12] Hagit Attiya and Ami Paz. Counting-based impossibility proofs for renaming and set agreement. In *Proceedings of the 26th International Symposium on Distributed Computing (DISC)*, pages 356–370, 2012.

[AR02] Hagit Attiya and Sergio Rajsbaum. The combinatorial structure of wait-free solvable tasks. *SIAM J. Comput.*, 31(4):1286–1313, 2002.

[BCD$^+$19] Nir Bachrach, Keren Censor-Hillel, Michal Dory, Yuval Efron, Dean Leitersdorf, and Ami Paz. Hardness of distributed optimization. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 238–247, 2019.

[BG93a] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 91–100, 1993.

[BG93b] Elizabeth Borowsky and Eli Gafni. Immediate atomic snapshots and fast renaming (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 41–51, 1993.

[BG97] Elizabeth Borowsky and Eli Gafni. A simple algorithmically reasoned characterization of wait-free computation. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 189–198, 1997.

[BL93] James E. Burns and Nancy A. Lynch. Bounds on shared memory for mutual exclusion. *Information and Computation*, 107(2):171–184, 1993.

[Cha93] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, 1993.

[CIL87] Benny Chor, Amos Israeli, and Ming Li. On processor coordination using asynchronous hardware. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 86–97, 1987.

[CR10] Armando Castañeda and Sergio Rajsbaum. New combinatorial topology bounds for renaming: the lower bound. *Distributed Computing*, 22(5-6):287–301, 2010.

[EGZ18] 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)*, pages 61–70, 2018.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[GWSR19] Hugo Rincon Galeana, Kyrill Winkler, Ulrich Schmid, and Sergio Rajsbaum. A topological view of partitioning arguments: Reducing k-set agreement to consensus. In *Proceedings of the 21st International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 307–322, 2019.

[Her91] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(1):124–149, 1991.

[HLM03] Maurice Herlihy, Victor Luchangco, and Mark Moir. Obstruction-free synchronization: Double-ended queues as an example. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS)*, pages 522–529, 2003.

[HS99] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *jacm*, 46(6):858–923, 1999.

[HS06] Gunnar Hoest and Nir Shavit. Toward a topological characterization of asynchronous complexity. *SIAM J. Comput.*, 36(2):457–497, 2006.

[LAA87] M. C. Loui and H. H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. In *Advances in Computing Research*, volume 4, pages 163–183. JAI Press, 1987.

[MR02] Yoram Moses and Sergio Rajsbaum. A layered analysis of consensus. *SIAM J. Comput.*, 31(4):989–1021, 2002.

[PBI93] Toniann Pitassi, Paul Beame, and Russell Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational Complexity*, 3:97–140, 1993.

[SZ00] Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000.