WHY EXTENSION-BASED PROOFS FAIL *

DAN ALISTARH†, JAMES ASPNES‡, FAITH ELLEN§, RATI GELASHVILI¶, AND LEQI ZHU‖

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 or approximate agreement on a cycle of length 4 among $n > 2$ processes in a wait-free manner in asynchronous models where processes communicate using objects that can be constructed from shared registers. However, it was unknown whether proofs based on simpler techniques were possible. We show that these impossibility results cannot be obtained by extension-based proofs in the iterated snapshot model and, hence, extension-based proofs are limited in power.

Key words. set agreement, impossibility proofs, valency arguments

AMS subject classifications. 68Q17, 68Q85

1. Introduction.

One of the most well-known results in the theory of distributed computing, due to Fischer, Lynch, and Paterson [22], is that there is no deterministic protocol solving consensus among $n \geq 2$ processes in an asynchronous message passing system that tolerates even one process crash.

Their result has been extended to asynchronous shared memory systems where processes communicate by reading from and writing to shared registers [1, 20, 24, 31]. Moses and Rajsbaum [32] gave a unified framework for proving the impossibility of consensus in a number of different systems.

Chaudhuri [18] 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 [12], Herlihy and Shavit [28], and Saks and Zaharoglou [34]. These proofs and a later proof by Attiya and Rajsbaum [10] all relied on sophisticated machinery from topology. They use a vertex to model the view of a process in an execution, a simplex or a clique to model a reachable configuration, and a simplicial complex or graph to model the set of all final configurations of a protocol. Then

*Submitted to the editors on October 25, 2020. A conference version of this paper [3] appeared at STOC 2019 and a brief announcement [4] appeared at PODC 2020.

Funding: Support is gratefully acknowledged from the Natural Science and Engineering Research Council of Canada under grants RGPIN-2015-05080 and RGPIN-2020-04178, a postgraduate scholarship, and a postdoctoral fellowship, a University of Toronto postdoctoral fellowship, the 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, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme grant agreement No. 805223 ScaleML, and the Oracle and Intel corporations. Some of the work on this paper was done while Faith Ellen was visiting IST Austria.

†IST Austria, Austria (dan.alistarh@ist.ac.at).

‡Yale University, USA (james.aspnes@gmail.com).

§University of Toronto, Canada (faith@cs.toronto.edu).

¶Aptos, USA (gelash@aptoslabs.com).

‖University of Michigan, USA (lezhu@umich.edu).

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

Later on, Attiya and Castañeda [6] and Attiya and Paz [9] 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 $(n-1)$-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 n different values have been output.

A common feature of these impossibility proofs is that they are non-constructive: They assume there is a bound on the maximum length of any execution and prove, using properties of the entire set of final configurations, that there exists a final configuration in which n different values have been output. However, they do not construct an execution leading to such a final configuration.

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 $(n-1)$-set agreement (or, more generally, 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 [33].

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. Then, we 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 the iterated snapshot model. We also prove that there is no extension-based proof of the impossibility of a deterministic, wait-free protocol solving approximate agreement among $n > 2$ processes on a cycle of length 4.

A *task* is a problem in which each process starts with a private input value and must output one value such that these values satisfy certain specifications. At a high level, an extension-based proof of the impossibility of solving a task is 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 a configuration that can be reached in one step from a known configuration. It can also ask the protocol to exhibit an execution by a set of processes from a known configuration 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 in which some processes take infinitely many steps without terminating or output values that do not satisfy the specifications of the task. The formal 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. The prover 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 extension-based proof can refute the possibility of a deterministic, wait-free protocol solving k-set agreement among $n > k \geq 2$ processes in the iterated snapshot model. This model is defined in Section 2. In the conference version of this paper [3], we made a very similar argument in the iterated immediate snapshot model. We also argue that there is no extension-based proof of the impossibility of a deterministic, wait-free protocol solving approximate agreement among $n > 2$ processes on a cycle of length 4 in the iterated snapshot model. Although the outline of the proof is the same as for k-set agreement, there are some important differences, which we present in Section 6.

From a computability standpoint, the snapshot model, the immediate snapshot model, the iterated snapshot model, and the iterated immediate snapshot model are equivalent in power to the single-writer register model in which processes communicate through registers. Any protocol in the single-writer register model can be easily adapted to run in the snapshot model by replacing each `read` by a `scan` and then throwing away the information that is not needed. Afek, Attiya, Dolev, Gafni, Merritt, and Shavit [2] gave a wait-free implementation of a snapshot object using registers, so any protocol in the snapshot model can be simulated in the single-writer register model. 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 [13] 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. The iterated immediate snapshot model was introduced by Borowsky and Gafni [14]. It consists of an unbounded sequence of immediate snapshot objects that are accessed by each process in order. Any protocol in this model can be easily adapted to run in the 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 immediate snapshot objects. Borowsky and Gafni gave a nonblocking simulation of a snapshot object in the iterated immediate snapshot model [14] and, hence, in the iterated snapshot model. Thus, to show there is no wait-free protocol to solve a certain task in all these models, it suffices to show that there is no wait-free protocol to solve that task in any one of them.

Hoest and Shavit [29] showed that the collection of reachable configurations of a full-information protocol in the iterated immediate snapshot model has a natural representation using combinatorial topology. For the iterated snapshot 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 used in our proofs. Both these representations extend the combinatorial topology representation of uniform full-information protocols in the iterated immediate snapshot model [14, 28]. A protocol is *uniform* if all processes in all executions terminate after taking the same number of steps.

A number of papers have appeared that extend our work. These are discussed in Section 7, together with some interesting directions for future work.

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 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. The *initial state* of process p_i consists of its identifier, i, and its input, x_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 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 *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 *iterated snapshot* model, there is an infinite sequence, $S_1, S_2, \ldots$, of shared *single-writer snapshot* objects, each with n components. The initial value of each component is $-$. 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 an `update`, which changes the value of $S_r[i]$ from $-$. At its next step, it performs a `scan` of S_r. If it has not terminated, then p_i is poised to access the next snapshot object, S_{r+1}. In both the snapshot and iterated snapshot models, we assume that a process only terminates immediately 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.

In a *full-information protocol*, processes share everything they know and do not forget anything they have learned. Formally, in the snapshot and iterated snapshot models, whenever a process performs an `update` during a full-information protocol, it changes the value of its component of the snapshot object to its current state and its new state stays the same, except for an extra bit indicating that it has performed the `update`. Whenever a process performs a `scan` during a full-information protocol, its new state is a pair consisting of its process identifier and the result of the `scan`. Note that each process, p_i, remembers its entire history, because its previous state is component i of the result of the `scan`. After performing a `scan`, a process consults a decision function, δ, from the set of possible process states to the set of possible output values and the special symbol $\perp$. If s_i is the state of process p_i, then $\delta(s_i) \neq \perp$ indicates that p_i should output $\delta(s_i)$ and terminate. When $\delta(s_i) = \perp$, process p_i is poised to access the (next) snapshot object. A full-information protocol in the snapshot or iterated snapshot model is completely specified by its decision function. Any protocol can be easily transformed into a full-information protocol by defining the decision function so that it ignores the information not used by the original protocol.

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 has not terminated. In every initial configuration, every process is active. A configuration is *final* if it has no active processes, i.e., all its processes have terminated. 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 $\alpha = \alpha_1, \alpha_2, \ldots$ such that there is a sequence of configurations $C_0, C_1, C_2, \ldots$ where $C = C_0$, process α_j is active in C_{j-1} and $C_j = C_{j-1}\alpha_j$ for every position $j \geq 1$ in the sequence. If α is a finite schedule from C, then $C\alpha$ denotes the configuration of the protocol that is reached by performing one step by a process for each occurrence of the process in α, in order, starting from configuration C and we say that $C\alpha$ is *reachable from* C. We say that a configuration is *reachable* if it is reachable from an initial configuration. A protocol is *wait-free* if it does not have an infinite schedule from an initial configuration.

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 any protocol that claims to solve a task in a wait-free manner. The prover is trying to prove that the protocol is incorrect by constructing a bad execution (that is either infinite or produces incorrect outputs). 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.

At the beginning of each phase, the prover starts with a prefix of the sequence of steps in the bad execution, together with the input values of all processes that have taken steps in this prefix. More formally, in each phase $\varphi \geq 1$, the prover starts with a finite schedule, $\alpha(\varphi)$, and a set of initial configurations, $\mathcal{B}(\varphi)$. The configurations in $\mathcal{B}(\varphi)$ only differ from one another in the input values of processes that do not occur in $\alpha(\varphi)$. If every process appears in $\alpha(\varphi)$, then $\mathcal{B}(\varphi)$ consists of exactly one initial configuration. In particular, if $C_0\alpha(\varphi)$ is a final configuration for some initial configuration $C_0 \in \mathcal{B}(\varphi)$, then every process appears in $\alpha(\varphi)$ and $\mathcal{B}(\varphi) = \{C_0\}$. We let $\mathcal{A}(\varphi) = \{C_0\alpha(\varphi) \mid C_0 \in \mathcal{B}(\varphi)\}$ denote the set of configurations reached by $\alpha(\varphi)$ from configurations in $\mathcal{B}(\varphi)$. At the beginning 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 $\mathcal{A}(\varphi)$ consists of a single final configuration and the values output by the processes in this configuration 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 *single-step query* (C, q) in phase φ 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'. Note that there is 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. Hence, $C' = C''\beta q$ and $C''\beta' \in \mathcal{A}'(\varphi)$ for every nonempty prefix β' of βq. 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 *query chain* in phase φ is a (finite or infinite) sequence of consecutive single-step queries (C_1, q_1), $(C_2, q_2), \ldots$ such that $C_1 \in \mathcal{A}(\varphi) \cup \mathcal{A}'(\varphi)$ and $C_i = C_{i-1}q_{i-1}$ for every position $i \geq 2$ in the sequence. The choice of process involved in each single-step query in the sequence can depend on the responses to the previous single-step queries in the sequence. Note that a single-step query is a query chain of length one. If a final configuration is reached, the prover cannot continue the query chain, since the configuration contains no active processes. The prover wins if it asks an infinite query chain, since it has demonstrated that the protocol is not wait-free.

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 query chain starting from C following the schedule returned by the protocol. If P is the set of all processes, then the results of the output queries (C, P, y), for every possible output value y, tell 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, the results of $(C, P, 0)$ and $(C, P, 1)$ enable the prover to determine whether C is bivalent.

To end phase φ, the prover commits 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 $C\alpha' = C_0'\alpha(\varphi)\alpha'$. Let $\alpha(\varphi+1) = \alpha(\varphi)\alpha'$ be the schedule obtained by extending $\alpha(\varphi)$ by α', let $\mathcal{B}(\varphi+1)$ be the set of all initial configurations that only differ from C_0' by the states of processes that do not appear in the schedule $\alpha(\varphi+1)$, and let $\mathcal{A}(\varphi+1) = \{C_0\alpha(\varphi+1) \mid C_0 \in \mathcal{B}(\varphi+1)\}$ be the set of configurations reached by $\alpha(\varphi+1)$ starting from configurations in $\mathcal{B}(\varphi+1)$. Then the prover begins phase $\varphi+1$.

The only way that a phase can be infinite is if it contains an infinite query chain. If the interaction between the prover and the protocol is infinite, either because it contains an infinite query chain or the number of phases is infinite, the prover wins. In this case, the prover has demonstrated that the protocol is not wait-free. Against the trivial protocol in which no process ever outputs a value, the prover can win by asking any infinite query chain. The prover also wins if the protocol ever gives contradictory answers, for example, if the protocol says that some process outputs outputs different values in two different configurations that are indistinguishable to that process. Another example of a contradictory answer is when the protocol says that process $q \in Q$ outputs the value y in some configuration C', the protocol returns NONE to the output query (C, Q, y), and there is a Q-only schedule starting from C that results in configuration C'.

An extension-based proof that a task is unsolvable is a prover which wins against *every* protocol (that claims to solve the task). In other words, it is a method for constructing a schedule for each protocol that starts from an initial configuration and is either infinite (demonstrating that the protocol is not wait-free) or ends in a final configuration whose outputs do not satisfy the specifications of the task.

A valency argument [22] proves a task is unsolvable by constructing an infinite schedule from an initial configuration (by repeatedly extending a finite schedule) of any protocol whose final configurations satisfy the specifications of the task. It is easy to express the valency argument that proves the impossibility of wait-free consensus as an extension-based proof:

THEOREM 3.1. *There is no wait-free protocol for binary consensus among $n \geq 2$ processes in the iterated snapshot model.*

Proof. Given any protocol for n processes, the prover first asks the output queries (C, P, y), for all initial configurations C and all $y \in \{0, 1\}$, where P is the set of all n processes. Suppose no initial configuration is bivalent. Then the prover can find two initial configurations C_0 and C_0' such that these configurations only differ from one another by the input value of one process q and the protocol answered NONE to the output queries $(C_0, P, 0)$ and $(C_0', P, 1)$. Next, the prover asks a query chain starting from C_0 involving only some process $q' \in P - \{q\}$. If, at some point, the protocol says that process q' outputs 0, the protocol loses, since it has contradicted its answer of NONE to the output query $(C_0, P, 0)$. If, at some point, the protocol says that process q' outputs 1, the prover asks a query chain starting from C_0' involving only process q'. Note that configuration C_0' is indistinguishable from configuration C_0 to process q' and the contents of shared memory is the same in both configurations. Thus, the protocol must return the same sequence of answers in both query chains or it loses because it has given inconsistent answers. However, if the protocol says that process q' outputs 1 at the end of this query chain, it loses, since it has contradicted its answer of NONE to the output query $(C_0', P, 1)$. The only other possibility is that the query chain is infinite and the prover wins. Therefore, assume that the prover finds a bivalent initial configuration C_1.

Next, for each process q, the prover asks the single-step query (C_1, q) followed by the output queries $(C_1 q, P, y)$ for each $y \in \{0, 1\}$. Suppose none of these n configurations is bivalent. Then the prover can find two processes q' and q'' such that the protocol answered NONE to the output queries $(C_1 q', P, 0)$ and $(C_1 q'', P, 1)$. Since the first step of every process is an `update`, the configurations $C_1 q' q''$ and $C_1 q'' q'$ are identical. The prover then asks a query chain starting from $C_1 q'$ involving only process q''. If, at some point, the protocol says that process q'' outputs 0, the protocol loses, since it has contradicted its answer of NONE to the output query $(C_1 q', P, 0)$. If, at some point, the protocol says that process q'' outputs 1, the protocol loses, since it has contradicted its answer of NONE to the output query $(C_1 q'', P, 1)$. The only other possibility is that the query chain is infinite and the prover wins. Therefore, assume that the prover finds a process q_1 such that configuration $C_2 = C_1 q_1$ is bivalent. The prover ends phase 1 by committing to the schedule q_1.

In each subsequent phase, φ, the prover first asks the single-step query (C_φ, q) followed by the output queries $(C_\varphi q, P, y)$, for each process q and each $y \in \{0, 1\}$, to find a bivalent configuration $C_{\varphi+1} = C_\varphi q_\varphi$. Suppose none of these configurations is bivalent. Then the prover can find two processes q' and q'' such that the protocol answered NONE to the output queries $(C_\varphi q', P, 0)$ and $(C_\varphi q'', P, 1)$. If q' and q'' are both poised to perform an `update` in C_φ, the prover wins, by the same argument as in phase 1. Otherwise, one of these processes, say q'', is poised to perform a `scan` in C_φ. Then the prover asks a query chain starting from $C_\varphi q''$ involving only process q'. If, at some point, the protocol says that process q' outputs 1, it loses, since it has contradicted its answer of NONE to the output query $(C_\varphi q'', P, 1)$. If, at some point, the protocol says that process q' outputs 0, the prover asks a query chain starting from C_φ involving only process q'. Note that configuration C_φ is indistinguishable from configuration $C_\varphi q''$ to process q' and the contents of shared memory is the same in both configurations. Thus, the protocol must return the same sequence of answers in both query chains or it loses because it has given inconsistent answers. However, if the protocol says that process q' outputs 0 at the end of this query chain, it loses,

since it has contradicted its answer of NONE to the output query $(C_\varphi q', P, 0)$. The only other possibility is that the query chain starting from $C_\varphi q''$ is infinite and the prover wins. Therefore, assume that the prover finds a process q_φ such that configuration $C_{\varphi+1} = C_\varphi q_\varphi$ is bivalent. In this case, the prover ends phase φ and commits to extending the schedule by q_φ.

Since the prover either wins in some phase or the number of phases is infinite, the prover wins. □

In the conference version of this paper [3], there is an extension-based proof of this result that does not use output queries.

If a prover knows that it is interacting with a wait-free protocol that has a finite number of input configurations, it can win by asking single-step queries to examine all reachable configurations in a breadth-first manner, violating the spirit of valency arguments. However, the definition of extension-based proofs says that the prover is given no knowledge about the protocol with which it is interacting (except its initial configurations). In particular, it does not know whether the protocol is wait-free. Such a prover would, in fact, lose against all protocols which are not wait-free, but whose final configurations satisfy the specifications of the task .

An alternative definition is to require that the prover announce, at the beginning of each phase, an upper bound on the number of output queries and (maximal) query chains it will ask during the phase. This upper bound would not limit the total number of single-step queries the prover could ask during the phase, since a query chain can contain arbitrarily many single-step queries. The prover's choice of this bound could depend on the number of processes in the system, the task the protocol is claiming to solve, and the information the prover has learned about the protocol in earlier phases. With this definition, even if the prover knows that the protocol with which it is interacting is wait-free and has a finite number of input configurations, it cannot perform exhaustive search when the number of reachable final configurations of the protocol is larger than its announced bound. However, if the prover also knows an upper bound on the step complexity of the protocol, it could announce a sufficiently large upper bound on the number of output queries and query chains in phase 1 to enable it to perform exhaustive search. If, instead, the prover knows that the protocol is uniform (i.e., every process terminates in the same number of steps in every execution), but does not know an upper bound on the step complexity, it could learn the step complexity of the protocol by performing a query chain in phase 1 and perform exhaustive search in phase 2. For the tasks considered in Section 5 and Section 6, this does not help the prover, but it might help the prover against other tasks.

4. Properties of the Iterated Snapshot Model. The proof of our main result relies on properties of full-information protocols in the iterated snapshot model. We begin with two simple observations. The first is true because each process remembers its entire history and only process p_i can `update` component i of a snapshot object.

OBSERVATION 4.1. *The contents of shared memory in a reachable configuration of a full-information protocol in the iterated snapshot model is completely determined by the states of all processes in the configuration (including the states of the processes that have terminated).*

The second observation is a special case of a general, well-known result about indistinguishability. (For example, see Corollary 2.2. in [8].)

OBSERVATION 4.2. *Suppose C and C' are two reachable configurations of a protocol in the iterated snapshot model that are indistinguishable to a set of processes P. Furthermore, suppose that, except for the snapshot objects prior to S_t, each snapshot object has the same contents in C and C'. If each active process in P is poised to access a snapshot object S_r in C, 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 $C\beta_1 \cdots \beta_{i-1}$. Note 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 each process in some set P is 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 a 1-round schedule from C obtained from γ by appending sufficiently many occurrences of every process that is active in C so that each occurs exactly twice in β. 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 terminated process 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, consider any reachable configuration C' in which some process p is poised to perform an `update` to S_{r+1}. Then Lemma 4.4 with $t = 1$ and $P = \{p\}$ says that there is a configuration reachable by an r-round schedule from an initial configuration in which the state of p is the same as its state in configuration C'.

Next, we present a simple graphical representation of the configurations of a full-information protocol for $n \geq 2$ processes in the iterated snapshot model. Recall that such a protocol is 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 s of one process in some such reachable configuration. The identifier of the process, $id(v)$, is the first part of this state s and we define $\delta(v) = \delta(s)$. 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, all initial configurations are represented by n-vertex cliques in $\mathbb{G}_0$. 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 $i \neq j$. Note that, for other tasks, there may be an n-vertex clique in $\mathbb{G}_0$ that does not represent an initial configuration.

A vertex v is *active* if it represents the state of an active process, i.e., $\delta(v) = \perp$. A vertex v is *terminated* if it represents the state of a process that has terminated, i.e., $\delta(v) \neq \perp$ is the value that the process has output. If an n-vertex clique represents a configuration, that configuration is final 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 $\mathbb{G}_t$ that 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 process p_i now performs a `scan` of S_{t+1}, the result is an n-component vector containing these values. Since this vector is uniquely

determined by τ, 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 $\chi(\sigma, \delta)$ 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
  - 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 $\tau' \subseteq \tau$.

We will show that the n-vertex cliques in $\chi(\sigma, \delta)$ represent the configurations reachable from C by 1-round schedules.

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`.

This method for obtaining $\chi(\sigma, \delta)$ from the n-vertex clique σ is closely related to the non-uniform chromatic subdivision of a simplex representing a configuration in the iterated immediate snapshot model. Consequently, we call the graph $\chi(\sigma, \delta)$ the *subdivision* of σ. However, as mentioned by Herlihy and Shavit [28, page 884], $\chi(\sigma, \delta)$ is not necessarily a topological subdivision of σ.

Figure 4.1 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', processes p_1 and p_3 have the same states as in σ, 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 is after the second occurrence of p_i in β. 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 $\chi(\sigma, \delta)$.

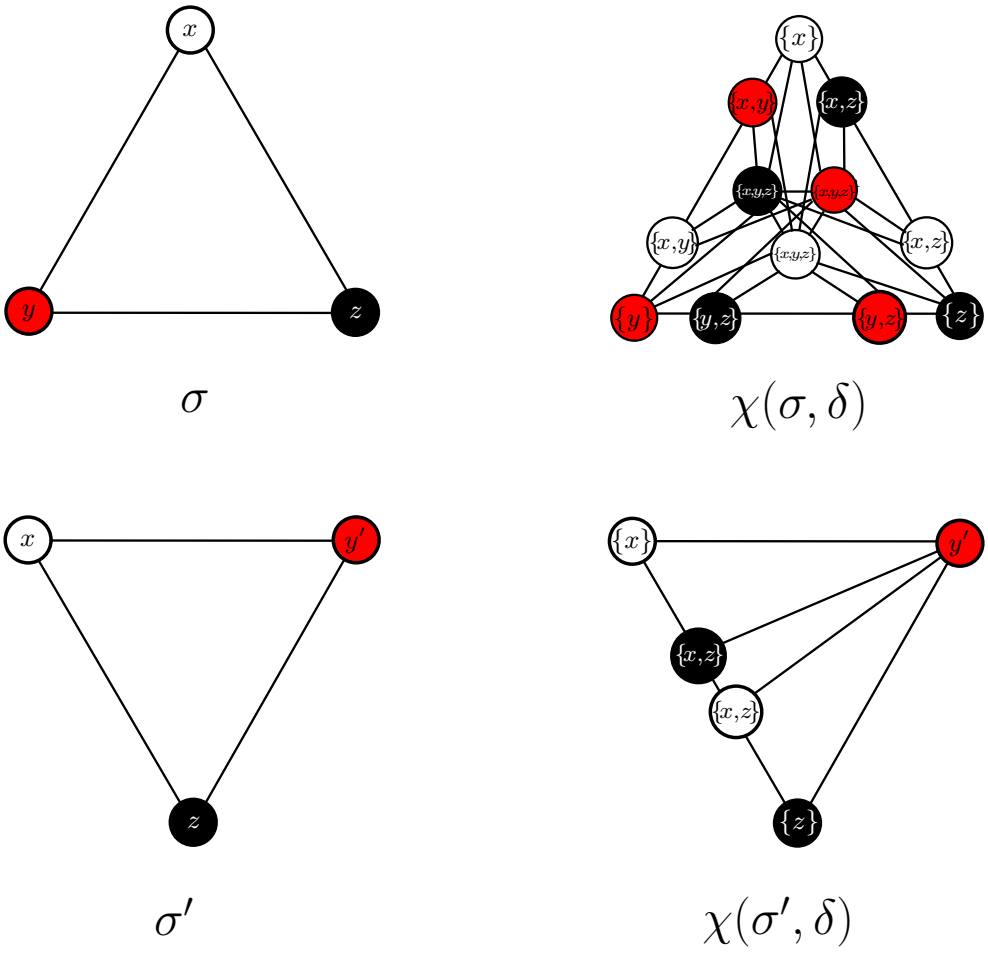

FIG. 4.1. *The subdivisions of two 3-vertex cliques.*

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 $I - id(\tau_i)$ in this order.

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 (not necessarily disjoint) 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, $\mathbb{G}_{t+1} = \chi(\mathbb{G}_t, \delta)$ is the union of $\chi(\sigma, \delta)$ for all n-vertex cliques σ in $\mathbb{G}_t$.

It follows from Lemma 4.5 that every configuration reachable from an initial configuration by a t-round schedule is represented by an n-vertex clique in $\mathbb{G}_t$. The converse is true, provided it is true for $t = 0$.

LEMMA 4.7. *If every n-vertex clique in $\mathbb{G}_0$ represents an initial configuration of the protocol, then, for all $t \geq 0$, every n-vertex clique in $\mathbb{G}_t$ represents a configuration reachable from an initial configuration by a t-round schedule.*

Proof. Let $t \geq 0$ and suppose that every n-vertex clique in $\mathbb{G}_t$ represents a configuration reachable from an initial configuration by a t-round schedule. Consider any n-vertex clique $\{v_1, \ldots, v_n\}$ in $\mathbb{G}_{t+1}$. If v_i is a terminated vertex in $\mathbb{G}_t$, let $u_i = v_i$. Otherwise, $v_i = (i, \tau_i)$ where $\tau_i \subseteq \sigma_i$ for some n-vertex clique σ_i in $\mathbb{G}_t$ such that $i \in id(\tau_i)$. In this case, let u_i be the vertex in τ_i such that $id(u_i) = i$. For all $1 \leq i < j \leq n$, it follows from the definition of $\mathbb{E}_{t+1}$ that there is an n-vertex clique in $\mathbb{G}_t$ that contains both u_i and u_j and, hence, there is an edge between u_i and u_j in $\mathbb{G}_t$. Thus $\sigma = \{u_1, \ldots, u_n\}$ is a clique in $\mathbb{G}_t$ and $\{v_1, \ldots, v_n\}$ is an n-vertex clique in $\chi(\sigma, \delta)$. By assumption, σ represents a configuration reachable from an initial configuration by a t-round schedule. Therefore, by Lemma 4.6, $\{v_1, \ldots, v_n\}$ represents a configuration reachable from an initial configuration by a $(t+1)$-round schedule. It follows by induction that the claim is true for all $t \geq 0$. ∎

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

LEMMA 4.8. *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 $(i, \sigma) \in \chi(\sigma, \delta)$, for each $i \in \{1, \ldots, n\} = id(\sigma)$. 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 4.9. *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 4.8, 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 4.8, $\chi(\sigma, \delta)$ 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 4.10. *If $\mathbb{G}_0$ is connected, then $\mathbb{G}_t$ is connected, for all $t \geq 1$.*

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 lengths of all paths in $\mathbb{G}_t$ between vertices in $\mathbb{A}$ and vertices in $\mathbb{B}$. If $\mathbb{G}_0$ is connected, then Corollary 4.10 implies that at least one such path exists. Now we show that, if $\mathbb{A}$ and $\mathbb{B}$ intersect (i.e., the distance between them in $\mathbb{G}_t$ is 0), then the same is true for $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ and, if $\mathbb{A}$ and $\mathbb{B}$ are disjoint (i.e., the distance between them in $\mathbb{G}_t$ is greater than 0), then so are $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$.

LEMMA 4.11. *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. ∎

We now prove one of the main technical tools used in this paper. Intuitively, it shows that subdividing does not decrease distances. For example, the distance

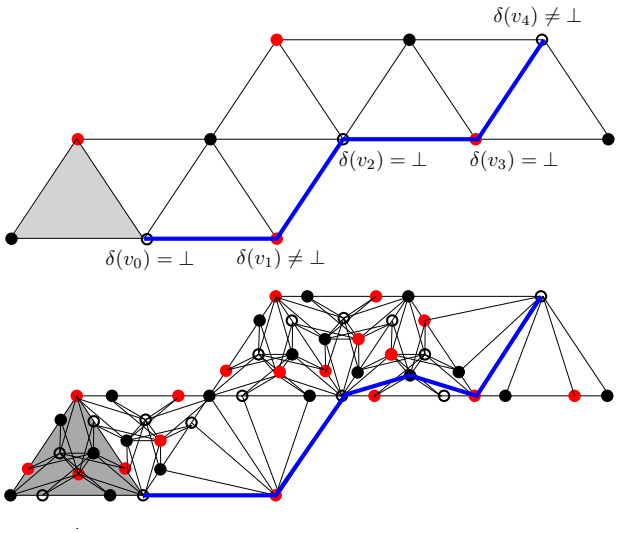

FIG. 4.2. *An illustration of Lemma 4.12.*

between two terminated vertices in $\mathbb{G}_t$ does not decrease in $\mathbb{G}_{t+1}$ and the distance between two n-vertex cliques in $\mathbb{G}_t$ is no larger than the distance between their subdivisions in $\mathbb{G}_{t+1}$. It also 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 4.2 illustrates Lemma 4.12. 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 4.12 (Distance Lemma). *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 $\mathbb{G}_t$. Moreover, 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. Let d be the distance between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$. If $d = 0$, then, by Lemma 4.11, the distance between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ is 0. Moreover, there is an empty path between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$, which does not contain an edge between active vertices. Therefore, assume that $d \geq 1$.

Consider any shortest path $w_0, w_1, \ldots, w_d$ between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$. Since $\mathbb{G}_{t+1} = \chi(\mathbb{G}_t, \delta)$, it follows that, for $1 \leq i \leq d$, there exists an n-vertex clique $\sigma_i \in \mathbb{G}_t$ such that $\{w_{i-1}, w_i\} \in \chi(\sigma_i, \delta)$.

We will construct a path $u_0, u_1, \ldots, u_d$ of length d between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$. Let $0 \leq i \leq d$. If w_i is a terminated vertex in $\mathbb{G}_t$, let $u_i = w_i$. If w_i is not a terminated vertex in $\mathbb{G}_t$, then $w_i = (id(w_i), \tau_i)$, where $id(w_i) \in id(\tau_i)$ and τ_i is a set of active vertices. In this case, let $u_i \in \tau_i$ be such that $id(u_i) = id(w_i)$. Note that, if $i \geq 1$,

then $u_i \in \tau_i \subseteq \sigma_i$ and, if $i < d$, then $u_i \in \tau_i \subseteq \sigma_{i+1}$.

Since $w_0 \in \chi(\mathbb{A}, \delta)$ and $w_d \in \chi(\mathbb{B}, \delta)$, it follows that $u_0 \in \mathbb{A}$ and $u_d \in \mathbb{B}$. Note that $id(u_{i-1}) = id(w_{i-1}) \neq id(w_i) = id(u_i)$, for $1 \leq i \leq d$. Since $u_{i-1}, u_i \in \sigma_i$ and σ_i is an n-vertex clique, it follows that $\{u_{i-1}, u_i\}$ is an edge in $\mathbb{G}_t$. Hence $u_0, u_1, \ldots, u_d$ is a path between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$. This implies that the distance between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ is at most d.

Now suppose that every path between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ contains at least one edge between active vertices. Then there exists $1 \leq i \leq d$ such that u_{i-1} and u_i are both active vertices. Since $\{w_{i-1}, w_i\}$ is an edge in $\chi(\sigma_i, \delta)$, either $\tau_{i-1} \subseteq \tau_i$ or $\tau_i \subseteq \tau_{i-1}$. Without loss of generality, suppose that $\tau_{i-1} \subseteq \tau_i$.

First, consider the case when $i = d$. Because $u_d \in \mathbb{B}$ is an active vertex, $\mathbb{B}$ is not a set of terminated vertices. Hence $\mathbb{B}$ is a union of n-vertex cliques. Since $w_d \in \chi(\mathbb{B}, \delta)$, there exists an n-vertex clique $\beta \subseteq \mathbb{B}$ such that $w_d \in \chi(\beta, \delta)$. This implies that τ_d is a subset of the active vertices in β and, hence, so is τ_{d-1}. But then $w_{d-1} \in \chi(\beta, \delta) \subseteq \chi(\mathbb{B}, \delta)$, which contradicts the assumption that $w_0, w_1, \ldots, w_{d-1}, w_d$ is a shortest path from $\chi(\mathbb{A}, \delta)$ to $\chi(\mathbb{B}, \delta)$.

Therefore $i < d$. Since $\tau_{i-1} \subseteq \tau_i \subseteq \sigma_{i+1}$, it follows that $u_{i-1} \in \sigma_{i+1}$. But σ_{i+1} is a clique, so either $u_{i-1} = u_{i+1}$, in which case $u_0, \ldots, u_{i-1}, u_{i+2} \ldots u_d$ is a path from $\mathbb{A}$ to $\mathbb{B}$ in $\mathbb{G}_t$ of length $d-2$, or $\{u_{i-1}, u_{i+1}\}$ is an edge of σ_{i+1}, in which case $u_0, \ldots, u_{i-1}, u_{i+1} \ldots u_d$ is a path from $\mathbb{A}$ to $\mathbb{B}$ in $\mathbb{G}_t$ of length $d-1$. In either case, the distance between $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{G}_t$ is less than the distance between $\chi(\mathbb{A}, \delta)$ and $\chi(\mathbb{B}, \delta)$ in $\mathbb{G}_{t+1}$. ∎

5. Why Extension-Based Proofs Fail for Set Agreement. 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 iterated snapshot model, for $n > k \geq 2$ processes. Specifically, we define an adversary that is able to win against every extension-based prover.

During phase 1, 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 $\mathbb{G}_0$, the graph representing the initial configurations. Recall that $\mathbb{G}_t$ is the graph representing configurations of the protocol reachable from initial configurations by t-round schedules. Once the adversary has defined $\delta(v)$ to be an element of $\{\perp\} \cup \{0, 1, \ldots, k\}$ for each vertex $v \in \mathbb{V}_t$ of $\mathbb{G}_t$, it may subdivide $\mathbb{G}_t$, construct $\mathbb{G}_{t+1} = \chi(\mathbb{G}_t, \delta)$, and increment t.

The adversary maintains a number of invariants about δ as it responds to queries. To ensure that these invariants are maintained, it may have to increment t multiple times (each time subdividing the graph $\mathbb{G}_t$). One invariant it maintains is that vertices terminated with different output values are far away from one another in $\mathbb{G}_t$. This will help the adversary satisfy agreement in its final configurations. Another invariant it maintains is that vertices terminated with output value y are far away from configurations reachable from initial configurations that do not have y as an input value. This will allow the adversary to satisfy validity in its final configurations. Some vertices cannot output value y because doing so would contradict the adversary's NONE response to an output query (C, Q, y). A third invariant the adversary maintains is that such vertices are far away from vertices with output value y. To do this, it ensures that each such vertex either is close to a vertex that has terminated with an output value other than y or is in a configuration reachable from an initial configuration that does not have y as an input. In addition, the adversary ensures that every query chain is finite.

At the beginning of phase 2, the adversary completes the specification of δ on all vertices in configurations reachable from configurations in $\mathcal{A}(2)$. It does so in such a way that at most two different values are output in any of these configurations, validity is not violated, no output query that returned NONE is contradicted, and there is no infinite schedule from any of these configurations. Eventually, at the end of some phase, the prover chooses a configuration reachable from a configuration in $\mathcal{A}(2)$ in which every processes has terminated, so the adversary wins.

Fix an extension-based prover. As its interaction with the prover proceeds, the adversary uses the integer t, the graphs $\mathbb{G}_r$ for $0 \le r \le t$, and the values of δ where it has been defined to determine how to respond to each of the prover's queries.

DEFINITION 5.1. *For each $0 \le r \le t$ and each input value a, let $\mathbb{T}_r(a) = \{v \in \mathbb{V}_r \mid \delta(v) = a\}$ be the subset of terminated vertices in $\mathbb{V}_r$ that have output a.*

The following simple properties are true because every terminated vertex remains unchanged when a subdivision is performed.

PROPOSITION 5.2. *For all $0 \le r < t$ and all input values a, $\mathbb{T}_r(a) = \chi(\mathbb{T}_r(a), \delta) \subseteq \mathbb{T}_{r+1}(a)$. If $\delta(v)$ is defined for every vertex $v \in \mathbb{V}_t$ and the adversary subdivides $\mathbb{G}_t$ to construct $\mathbb{G}_{t+1}$, then $\mathbb{T}_t(a) = \mathbb{T}_{t+1}(a)$ for all input values 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 $r' \le r$ 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 $r' - 1$.

DEFINITION 5.3. *For each $0 \le r \le t$ and each input value a, let $\mathbb{N}_r(a)$ be the union of the n-vertex cliques in $\mathbb{G}_r$ consisting of vertices that have not seen a.*

To avoid violating validity, the adversary should not let any vertex in $\mathbb{N}_t(a)$ output the value a.

PROPOSITION 5.4. *For all $0 \le r < t$ and all input values 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 $\mathbb{G}_{r+1}$, $\sigma' \subseteq \chi(\sigma, \delta)$ 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 5.5. *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. For k-set agreement, every two vertices in $\mathbb{G}_0$ are adjacent provided they represent the states of different processes, i.e., they have different *ids*. 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 is an input value different from a, then $\tau \cup \{(j, b) \mid j \notin id(\tau)\}$ is 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 $\sigma' \subseteq \chi(\sigma, \delta)$. 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, $\chi(\rho, \delta)$ 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 output query (C, P, a) to which the adversary answered NONE, the adversary cannot let a vertex output a if the vertex represents the state of a process in P in a configuration reachable from C by a P-only schedule. Otherwise, its definition of δ contradicts its response to (C, P, a).

DEFINITION 5.6. *For each $0 \leq r \leq t$ and each input value a, let $\mathbb{X}_r(a)$ be the subset of vertices in $\mathbb{V}_r$ that represent the states of processes in P in configurations reachable from C by P-only schedules, for all output queries (C, P, a) to which the adversary answered* NONE.

The adversary should not let any vertex in $\mathbb{X}_t(a)$ output the value a.

Throughout the first phase, the adversary will respond to each query so that the following invariants hold:

1. For each $0 \leq r < t$ and each vertex $v \in \mathbb{V}_r$, $\delta(v)$ is defined.
2. If $v \in \mathbb{V}_t$ and $\delta(v)$ is defined, then $\delta(v) \neq \perp$.
3. For every configuration $C \in \mathcal{A}'(1)$, there exists a schedule β from an initial configuration $C_0 \in \mathcal{A}(1)$ such that $C = C_0\beta$, no process occurs more than $2t$ times in β, and, for every nonempty prefix β' of β, configuration $C_0\beta' \in \mathcal{A}'(1)$ has been reached during phase 1. If a process occurs $2t$ times in β and $v \in \mathbb{V}_t$ represents the state of that process in C, then $\delta(v)$ is defined.
4. For every input value a, if $\mathbb{T}_t(a)$ is nonempty, then the distance in $\mathbb{G}_t$ between $\mathbb{T}_t(a)$, the set of vertices that have terminated with value a, and $\mathbb{N}_t(a)$, the set of vertices that have not seen value a, is at least 2.
5. For every 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 value a and every vertex $v \in \mathbb{X}_t(a)$, either $v \in \mathbb{N}_t(a)$ or there exists an input value $b \neq a$ such that the distance between v and $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ is at most 1.

There is nothing special about the values 2 and 3 in Invariants 4 and 5. They are simply the smallest values such that the invariants can be maintained and every query chain is finite.

LEMMA 5.7. *Suppose the invariants hold. If $a \neq b$, then every path between $\mathbb{T}_t(a)$ and $\mathbb{T}_t(b) \cup \mathbb{N}_t(a)$ in $\mathbb{G}_t$ contains at least one edge between vertices that are not terminated.*

Proof. Consider any path $v_0, v_1, \ldots, v_\ell$ between $\mathbb{T}_t(a)$ and $\mathbb{T}_t(b) \cup \mathbb{N}_t(a)$ in $\mathbb{G}_t$. Let v_j be the last vertex on this path that is in $\mathbb{T}_t(a)$. If $v_\ell \in \mathbb{T}_t(b)$, then Invariant 5 implies that the distance between v_j and v_ℓ in $\mathbb{G}_t$ is at least 3. If $v_\ell \in \mathbb{N}_t(a)$, then Invariant 4 implies that the distance between v_j and v_ℓ in $\mathbb{G}_t$ is at least 2. Hence, $\ell \geq j+2$ and $v_{j+1}, v_{j+2} \notin \mathbb{T}_t(a)$. 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 vertices that are not terminated. ∎

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

LEMMA 5.8. *Suppose the invariants hold, the adversary defines $\delta(v) = \perp$ for each vertex $v \in \mathbb{V}_t$ where $\delta(v)$ is undefined, and it 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 $a \neq b$ and both $\mathbb{T}_t(a)$ and $\mathbb{T}_t(b)$ are 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 then increments t, the invariants continue to hold.*

Proof. Since Invariant 1 was true and the adversary defines $\delta(v) = \perp$ for each vertex $v \in \mathbb{V}_t$ where $\delta(v)$ was undefined, Invariant 1 remains true after t is incremented. Since $\mathbb{G}_{t+1}$ has just been constructed, each vertex $v' \in \mathbb{V}_{t+1}$ is either a terminated vertex in $\mathbb{V}_t$ or $\delta(v')$ is undefined. Thus, Invariant 2 is true after t is incremented. Since no configurations are added to $\mathcal{A}'(1)$, Invariant 3 remains true.

By Proposition 5.2, $\chi(\mathbb{T}_t(a), \delta) = \mathbb{T}_t(a) = \mathbb{T}_{t+1}(a)$, for each input a. By Proposition 5.4, $\mathbb{N}_{t+1}(a) = \chi(\mathbb{N}_t(a), \delta)$. Lemma 5.7 says that, for $b \neq a$, 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 vertices that are not terminated. By construction, $\delta(v)$ is defined for all vertices $v \in \mathbb{V}_t$, so vertices that are not terminated are active. Therefore, if $\mathbb{T}_t(a)$ is nonempty, Lemma 4.12 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 4.12 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.

Finally, consider any vertex $v' \in \mathbb{X}_{t+1}(a)$. If v' is a terminated vertex in $\mathbb{G}_t$, let $v = v' \in \mathbb{X}_t(a)$. Otherwise, there exists an output query (C, P, a) to which the adversary answered NONE and a P-only schedule γ' from $C \in \mathcal{A}(1) \cup \mathcal{A}'(1)$ such that v' is the state of some process $p_i \in P$ in configuration $C\gamma'$. Recall that, for each configuration $C \in \mathcal{A}(1) \cup \mathcal{A}'(1)$, there is an initial configuration C_0 and a schedule β from C_0 such that $C = C_0\beta$. By Invariant 3, process p_i occurs at most $2t$ times in β. Hence, there is a prefix γ of γ' such that p_i occurs exactly $2t$ times in $\beta\gamma$. Let $v \in \mathbb{V}_t$ denote the state of p_i in $C_0\beta\gamma = C\gamma$. Since γ is a P-only schedule, $v \in \mathbb{X}_t(a)$. Furthermore, $v' \in \chi(\sigma, \delta)$ for some n-vertex clique σ that contains v.

In both cases, by Invariant 6, either $v \in \mathbb{N}_t(a)$ or the distance between v and $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ is at most 1 for some input $b \neq a$. By the definition of subdivision, if $v \in \mathbb{N}_t(a)$, then $v' \in \mathbb{N}_{t+1}(a)$. Similarly, if the distance between v and $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ is at most 1, then the distance between v' and $\mathbb{T}_{t+1}(b)$ in $\mathbb{G}_{t+1}$ is at most 1. Thus, Invariant 6 remains true after t is incremented. ∎

The adversarial strategy for phase 1. Initially, the adversary defines $\delta(v) = \perp$ for each vertex $v \in \mathbb{V}_0$, 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, so 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.

Suppose that the invariants are true immediately prior to some query in phase 1. We will show that the adversary can answer this query (and possibly define δ on more vertices) so that the invariants remain true afterwards.

First, consider a single-step query (C, q), 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 $C = C_0\beta$, no process occurs more than $2t$ times in β, and $C_0\beta' \in \mathcal{A}'(1)$ for every nonempty prefix β' of β. As a result of this query, the prover will add configuration Cq to $\mathcal{A}'(1)$. Note that βq is a schedule from C_0 such that $Cq = C_0\beta q$. Since every proper prefix of βq is a prefix of β, configuration $C_0\beta' \in \mathcal{A}'(1)$ for every nonempty prefix β' of βq.

If q occurs $2r$ times in β, then, by Invariant 3, $0 \leq r \leq t$. Let $v \in \mathbb{V}_r$ be the vertex that represents the state of q in configuration C. Since q is active in C, $\delta(v) = \perp$. Hence, by Invariant 2, $r < t$. In this case, the adversary returns the configuration Cq, which is the same as C except that $S_{r+1}[id(q)] = (id(q), v)$ and the state of q has an extra bit indicating that it last performed an `update`. Process q occurs $2r + 1 < 2t$ times in βq and all other processes occur the same number of times in βq and β. Thus Invariant 3 remains true. 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.

If q occurs $2r + 1$ times in β, then, by Invariant 3, $0 \leq r < t$. The state of q in configuration Cq is $(id(q), \tau)$, where τ is the result of its `scan` of S_{r+1}. It is represented by a vertex $v' \in \mathbb{V}_{r+1}$. 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 $\delta(v')$ is defined, by Invariant 1. It is also possible that $r = t - 1$ and $\delta(v')$ is already defined. In both these cases, the adversary returns configuration Cq, which is the same as C, except for the state of q and, if $\delta(v') \neq \perp$, the value q outputs. As above, the invariants continue to hold. So, suppose that $r = t - 1$ and $\delta(v')$ is not defined.

If there exists an input a such that setting $\delta(v') = a$ maintains the invariants, then the adversary defines $\delta(v') = a$ 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 v' and $\mathbb{N}_t(a)$ in $\mathbb{G}_t$ is at least 2 and, for all inputs $b \neq a$ such that $\mathbb{T}_t(b)$ is nonempty, the distance between v' and $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ is at least 3. The vertex v' 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. Process q occurs $2t$ times in βq and, by construction, $\delta(v')$ 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 $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ for some $b \neq a$. Since the distance between v' and $\mathbb{N}_t(a)$ in $\mathbb{G}_t$ is at least 2 and the distance between v' and $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ is at least 3, the distance between v' and u in $\mathbb{G}_t$ is at least 2. Thus $v' \notin \mathbb{X}_t(a)$, so defining $\delta(v') = a$ does not contradict the result of any previous output query.

Otherwise, the adversary defines $\delta(v) = \perp$ for each vertex $v \in \mathbb{V}_t$ where $\delta(v)$

is undefined (including v'), subdivides $\mathbb{G}_t$ to construct $\mathbb{G}_{t+1}$, and increments t. By Lemma 5.8, the invariants continue to hold. The adversary returns configuration Cq, which is the same as C except for the state of q.

Second, consider an output query (C, Q, y), 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 $v \in \mathbb{Q}$ is 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. So, suppose that no vertex in $\mathbb{Q}$ is terminated with output y.

Let $\mathbb{U}$ 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 any $a \neq y$. If $\mathbb{U} = \emptyset$, then it would be impossible for the adversary to return a Q-only schedule from C in which some vertex is 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 $\cup\{\mathbb{T}_t(a) \mid a \neq y\}$ to $\mathbb{X}_t(y)$ does not invalidate Invariant 6. The other invariants are not affected. So, suppose $\mathbb{U} \neq \emptyset$.

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. In the first case, there is a vertex $u \in \mathbb{U}$ such that some vertex in $\mathbb{A}_u$ is terminated and outputs y. Then the adversary can define $\delta(v) = y$ for some vertex in v in $\mathbb{G}_{t+1}$ and return a Q-only schedule from C that results in a configuration in which the state of some process in Q is represented by v. Moreover, we show that the adversary is able to maintain the invariants and does not contradict its answer to any previous output query. The same is true if there is a vertex $u \in \mathbb{U}$ such that no vertex in $\mathbb{A}_u$ is terminated. In the remaining case, the adversary can answer NONE without violating any invariant.

Case 1: *There is a vertex $u \in \mathbb{U}$ such that some vertex in $\mathbb{A}_u$ is terminated and outputs value y.* 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 5.8, 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 5.2 implies that $\mathbb{T}_{t+1}(a)$ is nonempty and, by Invariant 5 and Lemma 5.8, the distance between $\mathbb{T}_{t+1}(y)$ and $\mathbb{T}_{t+1}(a)$ in $\mathbb{G}_{t+1}$ 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. All invariants continue to hold, by Lemma 5.8. 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 $w \in \mathbb{T}_{t-1}(y)$, 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 $\mathbb{N}_t(y)$ in $\mathbb{G}_t$ is at least 2. Likewise, if $a \neq y$ and $\mathbb{T}_t(a)$ is nonempty, then the distance between v and $\mathbb{T}_t(a)$ in $\mathbb{G}_t$ is at least 3. Thus, Invariants 4 and 5 hold.

By Invariant 6, each vertex in $\mathbb{X}_t(y)$ is either in $\mathbb{N}_t(y)$ or is at distance at most 1 from $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ for some $b \neq y$. Since the distance between v and $\mathbb{N}_t(y)$ in $\mathbb{G}_t$ is at least 2 and the distance between v and $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ is at least 3, the distance between v and $\mathbb{X}_t(y)$ in $\mathbb{G}_t$ 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 no vertex in $\mathbb{A}_u$ is terminated.* 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 $\mathbb{G}_{t+1}$.

No vertex in $\mathbb{A}_u$ is terminated, so the distance between $\mathbb{A}_u$ and $\mathbb{T}_t(a)$ in $\mathbb{G}_t$ is at least 1, for all inputs a. Since $\mathbb{A}_u$ 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. 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 $v = (i, \{u\}) \in \mathbb{V}_{t+1}$. Since $u \in \mathbb{U} \subseteq \mathbb{Q}$, process $p_i \in Q$. Consider any vertex v' adjacent to v in $\mathbb{G}_{t+1}$. There exists an n-vertex clique $\sigma \subseteq \mathbb{G}_t$ such that $v, v' \in \chi(\sigma, \delta)$ and $\{u\} \subseteq \sigma$. Hence $\sigma \subseteq \mathbb{A}_u$. 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 $v, v' \notin \mathbb{N}_{t+1}(y)$. Therefore, the distance from v to $\mathbb{N}_{t+1}(y)$ in $\mathbb{G}_{t+1}$ is at least 2.

Let v'' be any vertex adjacent to v' in $\mathbb{G}_{t+1}$. There exists an n-vertex clique $\sigma' \subseteq \mathbb{G}_t$ such that $\{v', v''\}$ is an edge in $\chi(\sigma', \delta)$. This implies that $\rho \subseteq \sigma'$ and, hence $u \in \sigma'$. Therefore $\sigma' \subseteq \mathbb{A}_u$ and $v', v'' \in \chi(\mathbb{A}_u, \delta)$. Since the distance between $\mathbb{A}_u$ and $\mathbb{T}_t(a)$ in $\mathbb{G}_t$ is at least 1, Lemma 4.12 implies that the distance between $\chi(\mathbb{A}_u, \delta)$ and $\chi(\mathbb{T}_t(a), \delta)$ in $\mathbb{G}_{t+1}$ is at least 1. By Proposition 5.2, $\chi(\mathbb{T}_t(a), \delta) = \mathbb{T}_{t+1}(a)$. Hence $v', v'' \notin \mathbb{T}_{t+1}(a)$. Thus, 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 invariants continue to hold, by Lemma 5.8. 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 $\mathbb{T}_t(a)$ in $\mathbb{G}_t$ is at least 3 for all inputs $a \neq y$, 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 $u \in \mathbb{U}$, there is a vertex in $\mathbb{A}_u$ that is terminated, but there is no vertex in $\mathbb{A}_u$ that is terminated and outputs value y.* 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 is 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 $\mathbb{N}_t(a)$ and $\mathbb{T}_t(a)$ are not changed for any input a.

The prover does not win in phase 1. The invariants hold 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 and Lemma 5.5, 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 query chain in phase 1.

LEMMA 5.9. *Every query chain in phase 1 is finite.*

Proof. Assume, for a contradiction, there is an infinite query chain, (C_1, q_1), $(C_2, q_2), \ldots$ in phase 1. If C_1 is an initial configuration, let β_1 be the empty schedule and let $C_0 = C_1$. Otherwise, $C_1 \in \mathcal{A}'(1)$ and, by Invariant 3, there is a schedule β_1 from an initial configuration $C_0 \in \mathcal{A}(1)$ to C_1. For each $j \geq 1$, let $\beta_{j+1} = \beta_j q_j$, so $C_{j+1} = C_0\beta_{j+1}$. Consider the infinite schedule $\beta = \beta_1 q_1 q_2 \cdots$ from C_0.

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 the query $(C_{j'}, q_{j'})$. By Invariant 3, each process occurs at most $2t'$ times

in $\beta_{j'}$. Hence, during the schedule $\beta_{j'}$ from C_0, no process performed an `update` to S_r for $r > t'$. Each process in P eventually performs an `update` to every snapshot object during the schedule β from C_0. Therefore, while responding to this query chain, the adversary eventually defines $\delta(v) = \perp$ for each vertex $v \in \mathbb{V}_r$ where $\delta(v)$ is undefined and subdivides $\mathbb{G}_r$ to construct $\mathbb{G}_{r+1}$, for all $r \geq t'$. Since no process in P is terminated, $\mathbb{T}_r(a) = \mathbb{T}_{t'}(a)$, for all inputs a and all $r > t'$. If $\mathbb{T}_{t'}(a)$ is nonempty, then, by Invariant 4, the distance between $\mathbb{T}_{t'}(a)$ and $\mathbb{N}_{t'}(a)$ in $\mathbb{G}_{t'}$ is at least 2 and, by Lemma 5.8, the distance between $\mathbb{T}_{t'+2}(a)$ and $\mathbb{N}_{t'+2}(a)$ in $\mathbb{G}_{t'+2}$ is at least 4. If $b \neq a$ and $\mathbb{T}_{t'}(b)$ is also nonempty, then, by Invariant 5, the distance between $\mathbb{T}_{t'}(a)$ and $\mathbb{T}_{t'}(b)$ in $\mathbb{G}_{t'}$ is at least 3 and, by Lemma 5.8, the distance between $\mathbb{T}_{t'+2}(a)$ and $\mathbb{T}_{t'+2}(b)$ in $\mathbb{G}_{t'+2}$ 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 $C_{j''}$. Then process $q_{j''}$ occurs exactly $2t'+3$ times in $\beta_{j''}$ and $2(t'+2)$ times in $\beta_{j''+1}$. Let $v \in \mathbb{V}_{t'+2}$ denote the state of process $q_{j''}$ in configuration $C_{j''+1}$.

Suppose there is some input a such that $\mathbb{T}_{t'+2}(a)$ is nonempty and the distance from v to $\mathbb{T}_{t'+2}(a)$ in $\mathbb{G}_{t'+2}$ is at most 2. Since the distance between $\mathbb{T}_{t'+2}(a)$ and $\mathbb{N}_{t'+2}(a)$ in $\mathbb{G}_{t'+2}$ is at least 4, the distance from v to $\mathbb{N}_{t'+2}(a)$ in $\mathbb{G}_{t'+2}$ is at least 2. Likewise, for each $b \neq a$ such that $\mathbb{T}_{t'+2}(b)$ is nonempty, the distance between $\mathbb{T}_{t'+2}(a)$ and $\mathbb{T}_{t'+2}(b)$ in $\mathbb{G}_{t'+2}$ is at least 5, so 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 $\mathbb{T}_{t'+2}(a)$ 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, appending sufficiently many occurrences of the processes not in P, and then applying Lemma 4.4. 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 Lemma 4.12 and Propositions 5.2 and 5.4, 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 and 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.

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

Since the prover does not win in phase 1, it must eventually end phase 1. At the end of phase 1, the prover commits to a nonempty schedule $\alpha(2)$ from an initial configuration $C \in \mathcal{A}(1)$ such that $C\alpha(2) \in \mathcal{A}'(1)$, sets $\mathcal{B}(2)$ to consist of all initial configurations that only differ from C by the states of processes that do not occur in $\alpha(2)$, sets $\mathcal{A}(2) = \{C_0\alpha(2) \mid C_0 \in \mathcal{B}(2)\}$, and then starts 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. We will show that, 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. First, the adversary subdivides the graph $\mathbb{G}_t$, so that vertices which are terminated with different values are further apart. Then, for each vertex on which δ is undefined, but which is adjacent to a vertex on which δ is defined, the adversary defines δ to have the same value. This ensures that, in the future, the adversary will not contradict its NONE response to any output query. Finally, it defines δ to have value a on all remaining vertices representing the states of processes in configurations reachable from configurations in $\mathcal{B}(2)$.

More formally, 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.

The adversary defines $\delta(v) = \perp$ 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 the invariants hold at the end of phase 1, Lemma 5.8 says that they still hold. Furthermore, for any input b such that $\mathbb{T}_t(b)$ is non-empty, the distance between $\mathbb{T}_t(b)$ and $\mathbb{N}_t(b)$ in $\mathbb{G}_t$ is at least 3 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 $v \in \mathbb{V}_t$ is adjacent to a vertex $w \in \mathbb{T}_t(b)$ for at most one input b.

Invariant 2 says that no vertex $v \in \mathbb{V}_t$ has $\delta(v) = \perp$. The adversary has not yet terminated any additional vertices in $\mathbb{G}_t$, so, by Proposition 5.2, $\mathbb{T}_t(b) = \mathbb{T}_{t-1}(b)$ for all input values b.

Let $\mathbb{F}' = \chi^{t-1}(\mathbb{F}, \delta) \subseteq \mathbb{G}_t$. Since every vertex in $\mathbb{F}$ has seen a, it follows that every vertex in $\mathbb{F}'$ has seen a. Thus $\mathbb{F}'$ and $\mathbb{N}_t(a)$ are disjoint, i.e., the distance between $\mathbb{F}'$ and $\mathbb{N}_t(a)$ in $\mathbb{G}_t$ is at least 1.

For each input value b and each vertex $v \in \mathbb{F}'$ that is at distance 1 from $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ and on which δ is undefined, the adversary defines $\delta(v) = b$. This does not violate validity, since the distance between $\mathbb{T}_t(b)$ and $\mathbb{N}_t(b)$ in $\mathbb{G}_t$ is at least 3. By Invariant 6, each vertex in $\mathbb{X}_t(b)$ is either in $\mathbb{N}_t(b)$ or is at distance at most 1 from $\mathbb{T}_t(b')$ in $\mathbb{G}_t$ for some $b' \neq b$. If $\mathbb{T}_t(b)$ is nonempty, this implies that that the distance in $\mathbb{G}_t$ between $\mathbb{T}_t(b)$ and each vertex in $\mathbb{X}_t(b)$ was at least 3. Hence, this assignment does not contradict any output query that returned NONE. Moreover, the distance in $\mathbb{G}_t$ between any two vertices in $\mathbb{F}'$ that have output different values is still at least 2. Thus, in each n-vertex clique in $\mathbb{G}_t$, all 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 $\mathbb{N}_t(a)$. Since each vertex in $\mathbb{X}_t(a) \cap \mathbb{F}'$ is at distance at most 1 from $\mathbb{T}_t(b')$ in $\mathbb{G}_t$ for some $b' \neq a$, this does not define $\delta(v)$ for any $v \in \mathbb{X}_t(a)$. Hence, it does not contradict any output query that returned NONE. Agreement is not violated, since at most two different values are output by the vertices in each n-vertex clique in $\mathbb{G}_t$. Since two different values can be output by the vertices in some n-vertex clique, this construction does not work when $k = 1$.

In phases $\varphi \geq 2$, the prover can only query configurations reachable from configurations 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 $\mathbb{F}'$ is

terminated. Thus, eventually, the prover chooses a configuration at the end of some phase in which every process is terminated. The prover loses in the next phase.

Thus, we have proved the following result:

THEOREM 5.10. *There is no extension-based proof of the impossibility of a wait-free protocol solving k-set agreement for $n > k \geq 2$ processes in the iterated snapshot model.*

6. Why Extension-Based Proofs Fail for Approximate Agreement on a Cycle of Length 4. Next, we consider approximate agreement on a cycle of length 4. This is a special case of graphical approximate agreement [5], where the graph has node set $\{0, 1, 2, 3\}$ and four edges, $\{0, 1\}$, $\{1, 2\}$, $\{2, 3\}$, and $\{3, 0\}$. Each process p_i is given a node $x_i \in \{0, 1, 2, 3\}$ as input and, if it does not crash, it must output a node $y_i \in \{0, 1, 2, 3\}$. The outputs have to satisfy two properties: different output values are adjacent to one another on the cycle (*agreement*) and each output value is a node on a shortest path between two input values (*validity*). In particular, if all input values are the same node, then each output value is this node and, if all input values are endpoints of the same edge of the cycle, then each output value is an endpoint of this edge.

The rest of this section is devoted to proving the following result. It shows that the technique used for proving Theorem 5.10 can also be applied to another problem.

THEOREM 6.1. *There is no extension-based proof of the impossibility of a wait-free protocol solving approximate agreement on a cycle of length 4 for $n > 2$ processes in the iterated snapshot model.*

This result inspired an argument by Alistarh, Ellen, and Rybicki [5] showing, via a generalization of Sperner's Lemma, that there is no wait-free protocol solving approximate agreement on a cycle of length 4 for $n > 2$ processes in the iterated immediate snapshot model.

The proof of this theorem is similar to the argument in Section 5: We define an adversary that adaptively constructs a partial specification of a protocol δ for approximate agreement on a cycle of length 4, which wins against every extension-based prover. There are two main differences. One difference is the adversary's strategy at the beginning of phase 2: First, it subdivides the graph twice, so vertices that are terminated with different values are even further apart. Then, as in Section 5, for each possible output value $b \in \{0, 1, 2, 3\}$ and for each vertex on which δ is undefined, but which is adjacent to a vertex on which δ has value b, the adversary defines δ to have value b. The adversary would also like to define δ on the remaining vertices to be the input value a of the first process to take a step in the schedule $\alpha(2)$. However, agreement would be violated if there were remaining vertices adjacent to vertices on which δ has value $(a + 2) \bmod 4$. Instead, δ is defined to have value $(a + 1) \bmod 4$ on those remaining vertices and value a on the rest.

The other difference is the set $\mathbb{R}_t(a, a + 1)$, which is used in place of $\mathbb{N}_r(a)$. To simplify notation, we use $a + 1$ to denote $(a + 1) \bmod 4$ for any node a.

DEFINITION 6.2. *For each $0 \leq r \leq t$ and each input value $a \in \{0, 1, 2, 3\}$, let $\mathbb{R}_r(a, a + 1)$ be the union of the n-vertex cliques in $\mathbb{G}_r$ consisting of vertices that have only seen a or $a + 1$.*

In particular, the clique representing any initial configuration in which each process has input a or $a + 1$ is contained in $\mathbb{R}_0(a, a + 1)$ and the clique representing any configuration reachable by a t-round schedule from such an initial configuration is

contained in $\mathbb{R}_t(a, a+1)$. To avoid violating validity, the adversary should restrict $\delta(v)$ to be in $\{a, a+1, \perp\}$ for each vertex in $v \in \mathbb{R}_t(a, a+1)$.

The first result is the analogue of Proposition 5.4 and has a similar proof.

PROPOSITION 6.3. *For all $0 \leq r < t$ and all input values a, $\mathbb{R}_{r+1}(a, a+1) = \chi(\mathbb{R}_r(a, a+1), \delta)$.*

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

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

Similarly, the next result is the analogue of Lemma 5.5 and its proof is almost the same.

LEMMA 6.4. *If τ is a subset of an n-vertex clique in $\mathbb{G}_r$ and every vertex in τ has only seen a or $a+1$, then τ is a subset of an n-vertex clique in $\mathbb{R}_r(a, a+1)$.*

Proof. The proof is by induction on r. For approximate agreement on a cycle of length 4, every two vertices in $\mathbb{G}_0$ are adjacent provided they represent the states of different processes, i.e., they have different *ids*. Thus, if τ is a subset of an n-vertex clique in $\mathbb{G}_0$ and every vertex in τ has only seen a or $a+1$, then $\tau \cup \{(j, a) \mid j \notin id(\tau)\}$ is an n-vertex clique in $\mathbb{R}_0(a, a+1)$.

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 only seen a or $a+1$. Since $\mathbb{G}_{r+1} = \chi(\mathbb{G}_r, \delta)$, there exists an n-vertex clique σ in $\mathbb{G}_r$ such that $\sigma' \subseteq \chi(\sigma, \delta)$. Let $\tau = \{v \in \sigma \mid id(v) \in id(\tau')\}$. Note that, by definition, if $u \in \sigma$ has seen $b \neq a, a+1$, then every vertex $u' \in \chi(\sigma, \delta)$ with $id(u') = id(u)$ has seen b. Hence, no vertex in τ has seen a value other than a or $a+1$. By the induction hypothesis, there exists an n-vertex clique ρ in $\mathbb{R}_r(a, a+1)$ that contains τ.

By definition, $\chi(\rho, \delta)$ 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 $\chi(\rho, \delta) \subseteq \mathbb{R}_{r+1}(a, a+1)$. Thus the claim is true for $r+1$. □

Throughout the first phase, the adversary will respond to each query so that Invariants 1, 2, 3, 5, and following two additional invariants hold:

- 4′. For every input value a, if $\mathbb{T}_t(a)$ is non-empty and $a \neq b, b+1$, then the distance between $\mathbb{T}_t(a)$ and $\mathbb{R}_t(b, b+1)$ is at least 2.
- 6′. For every input value a and every vertex $v \in \mathbb{X}_t(a)$, there exists an input value $b \neq a$ such that either $v \in \mathbb{R}_t(b, b+1)$ and $a \neq b+1$, or the distance between v and $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ is at most 1.

These additional invariants are used in place of Invariants 4 and 6.

The following result is analogous to Lemma 5.7 and its proof is the same except that Invariant 4′ is used instead of Invariant 4.

LEMMA 6.5. *Suppose the invariants hold. If $a \neq b$, then every path between $\mathbb{T}_t(a)$*

and $\mathbb{T}_t(b)$ in $\mathbb{G}_t$ contains at least one edge between vertices that are not terminated. If $a \neq b, b+1$, then every path between $\mathbb{T}_t(a)$ and $\mathbb{R}_t(b, b+1)$ in $\mathbb{G}_t$ contains at least one edge between vertices that are not terminated.

Likewise, the next result is analogous to Lemma 5.8. It has the same proof, with Invariants 4′ and 6′ replacing Invariants 4 and 6, Proposition 6.3 replacing Proposition 5.4, and Lemma 6.5 replacing Lemma 5.7.

LEMMA 6.6. *Suppose the invariants hold, 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 $\mathbb{G}_{t+1}$. If $\mathbb{T}_t(a)$ is nonempty and $a \neq b, b+1$, then the distance between $\mathbb{T}_{t+1}(a)$ and $\mathbb{R}_{t+1}(b, b+1)$ in $\mathbb{G}_{t+1}$ is greater than the distance between $\mathbb{T}_t(a)$ and $\mathbb{R}_t(b, b+1)$ in $\mathbb{G}_t$. If $a \neq b$ and both $\mathbb{T}_t(a)$ and $\mathbb{T}_t(b)$ are 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 then increments t, the invariants continue to hold.*

The adversarial strategy for phase 1. As in Section 5, the adversary initially defines $\delta(v) = \perp$ for each vertex $v \in \mathbb{V}_0$, it subdivides $\mathbb{G}_0$ to construct $\mathbb{G}_1$, it sets $t = 1$, and the invariants hold.

Suppose that the invariants are true immediately prior to some query in phase 1. For a single-step query (C, q), where $C \in \mathcal{A}(1) \cup \mathcal{A}'(1)$ and q is an active process in C, the adversary behaves as in Section 5 and the proof that the invariants continue to hold is the same.

Now consider an output query (C, Q, y), 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. As in Section 5, let $\mathbb{Q}$ be the set of vertices in $\mathbb{G}_t$ representing the states of processes in Q in configurations reachable from C by 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 in which v represents the state of some process. None of the invariants are affected. So, suppose that no vertex in $\mathbb{Q}$ has terminated with output y.

Let $\mathbb{U}$ be the subset of vertices in $\mathbb{Q}$ that are not in $\mathbb{X}_t(y)$, $\mathbb{R}_t(b, b+1)$ for $y \neq b, b+1$, or $\mathbb{T}_t(a)$ for $y \neq a$. If $\mathbb{U} = \emptyset$, then, as in Section 5, the adversary adds $\mathbb{Q}$ to $\mathbb{X}_t(y)$ and returns NONE. Note that adding vertices in $\cup\{\mathbb{R}_t(b, b+1) \mid y \neq b, b+1\}$ or $\cup\{\mathbb{T}_t(a) \mid a \neq y\}$ to $\mathbb{X}_t(y)$ does not make Invariant 6′ false. The other invariants are not affected. So, suppose $\mathbb{U} \neq \emptyset$.

As in Section 5, 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. The same three cases are considered and, in each case, the proof is the same, using Invariants 4′ and 6′ instead of Invariants 4 and 6 and Lemma 6.6 instead of Lemma 5.8.

The prover does not win in phase 1. The invariants all hold 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′ and Lemma 6.4, if a process outputs value a in a configuration reached by the prover, then it is not the case that the process has only seen b or $b+1$, where $a \neq b, b+1$. Hence, the prover cannot win in phase 1 by showing that the protocol violates agreement or validity.

Using the same proof as in Section 5, with Invariant 4′ replacing Invariant 4, Proposition 6.3 replacing Proposition 5.4, and Lemma 6.6 replacing Lemma 5.8, it follows that the prover cannot win by constructing an infinite query chain in phase 1.

LEMMA 6.7. *Every query chain in phase 1 is finite.*

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

The adversarial strategy for later phases. Let p be the first process in $\alpha(2)$ and let a be the input of p in the initial configuration C. 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.

The adversary defines $\delta(v) = \perp$ 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. Then it repeats this a second time. Since the invariants hold at the end of phase 1, Lemma 6.6 says that they still hold. Furthermore, for any inputs b and b' such that $b \neq b', b'+1$ and $\mathbb{T}_t(b)$ is non-empty, the distance between $\mathbb{T}_t(b)$ and $\mathbb{R}_t(b', b'+1)$ in $\mathbb{G}_t$ is at least 4 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 5. In particular, a vertex $v \in \mathbb{V}_t$ is adjacent to a vertex $w \in \mathbb{T}_t(b)$ for at most one input b.

Invariant 2 says that no vertex in $\mathbb{G}_t$ has $\delta(v) = \perp$. The adversary has not yet terminated any additional vertices in $\mathbb{G}_t$, so, by Proposition 5.2, $\mathbb{T}_t(b) = \mathbb{T}_{t-2}(b)$ for all input values b.

Let $\mathbb{F}' = \chi^{t-1}(\mathbb{F}, \delta) \subseteq \mathbb{G}_t$. Since every vertex in $\mathbb{F}$ has seen a, it follows that every vertex in $\mathbb{F}'$ has seen a. If $a \neq b', b'+1$, then a vertex in $\mathbb{R}_t(b', b'+1)$ has not seen a and, hence, is not in $\mathbb{F}'$.

For each input value b and for each vertex $v \in \mathbb{F}'$ that is at distance 1 from $\mathbb{T}_{t-2}(b)$ in $\mathbb{G}_t$ and on which δ is undefined, the adversary defines $\delta(v) = b$. Next, for each vertex $v \in \mathbb{F}'$ that is at distance 2 from $\mathbb{T}_{t-2}(a+2)$ in $\mathbb{G}_t$ and on which δ is undefined, the adversary defines $\delta(v) = a+1$.

Consider any input value b and any vertex $v \in \mathbb{T}_t(b) - \mathbb{T}_{t-2}(b) \subseteq \mathbb{F}'$. The assignment $\delta(v) = b$ does not violate validity. This is because, for any input b' such that $b \neq b', b'+1$, the distance between $\mathbb{T}_{t-2}(b)$ and $\mathbb{R}_t(b', b'+1)$ in $\mathbb{G}_t$ is at least 4, so the distance between v and $\mathbb{R}_t(b', b'+1)$ is at least 2. By Invariant 6′, each vertex in $\mathbb{X}_t(b)$ is either in $\mathbb{R}_t(b', b'+1)$ for some value b' such that $b \neq b', b'+1$, or is at distance at most 1 from $\mathbb{T}_{t-2}(b')$ in $\mathbb{G}_t$ for some value $b' \neq b$. Since the distance between $\mathbb{T}_{t-2}(b)$ and $\mathbb{T}_{t-2}(b')$ in $\mathbb{G}_t$ is at least 5, it follows the distance between v and $\mathbb{T}_{t-2}(b')$ in $\mathbb{G}_t$ is at least 3. Hence $v \notin \mathbb{X}_t(b)$ and this assignment does not contradict any output query that returned NONE.

These assignments also do not violate agreement. To see why, consider any two vertices v and v' such that $\delta(v) = b \neq b' = \delta(v')$. If $\{b, b'\} = \{a+1, a+2\}$, it is possible that v and v' are adjacent, which is fine since $a+1$ and $a+2$ are adjacent nodes on the cycle. If $\{b, b'\} \neq \{a+1, a+2\}$, the distance between v and v' in $\mathbb{G}_t$ is at least 2, since the distance between $\mathbb{T}_{t-2}(b)$ and $\mathbb{T}_{t-2}(b')$ in $\mathbb{G}_t$ is at least 5. Hence, v and v' do not represent the states of two processes in one final configuration.

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 $\mathbb{R}_t(b, b+1)$, for $a \neq b, b+1$. In addition, v is at least distance 2 from any vertex in $\mathbb{T}_{t-2}(b)$ in $\mathbb{G}_t$ for any $b \neq a$. Hence, by Invariant 6′, $v \notin \mathbb{X}_t(a)$, so this assignment does not contradict any output query that returned NONE. By construction, v is not adjacent to any vertex on which

$a+2$ is output. Thus, at most two different values are output in any n-vertex clique in $\mathbb{G}_t$ and, if two different values are output, they are adjacent nodes on the cycle. Therefore, agreement is not violated.

In phases $\varphi \geq 2$, the prover can only query configurations reachable from configurations 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.

7. Conclusions. We have shown the limitation of extension-based proofs, including valency arguments, for proving the impossibility of deterministic, wait-free solutions to k-set agreement for $n > k \geq 2$ processes or approximate agreement on a cycle of length 4 for $n > 2$ processes in the iterated snapshot model. In the conference version of this paper [3], we showed the same limitation for k-set agreement in the iterated immediate snapshot model. We believe that these results also hold for any asynchronous model where processes communicate using objects that can be constructed from shared registers. More generally, we conjecture that if there is an extension-based proof that some task is impossible to solve in any model M and there is a non-blocking simulation of model M' in model M, then there is an extension-based proof that the task is impossible to solve in model M'.

Recently, Liu [30] gave an elegant generalization of Theorem 6.1. He showed that, for any connected graph G, there is no extension-based proof of the impossibility of approximate agreement on G among $n > 2$ processes.

In 2021, Brusse and Ellen [15] introduced *augmented extension-based proofs*, a generalization of extension-based proofs in which output queries are replaced by *assignment queries*. An assignment query (C, P, f) in phase φ is specified by a configuration $C \in \mathcal{A}(\varphi) \cup \mathcal{A}'(\varphi)$, a set of processes P that are all active in C, and an assignment f from a set of processes $Q \subseteq P$ to possible output values. If there is a P-only schedule from C that leads to a configuration in which each process $q \in Q$ is terminated with output $f(q)$, then the protocol returns some such schedule. Otherwise, the protocol returns NONE. They proved that, for a general class of reductions, if task $\mathcal{T}$ reduces to task $\mathcal{S}$ and there is an augmented extension-based proof that $\mathcal{T}$ is impossible to solve in the iterated snapshot model, then there is also an augmented extension-based proof that $\mathcal{S}$ is impossible to solve in the iterated snapshot model. Hence, if there is no augmented extension-based proof that $\mathcal{S}$ is impossible to solve in the iterated snapshot model, then there is no augmented extension-based proof that $\mathcal{T}$ is impossible to solve in the iterated snapshot model. They also extended the proof of Theorem 5.10 to show that there is no augmented extension-based proof of the impossibility of a wait-free protocol solving k-set agreement for $n > k \geq 2$ processes in the iterated snapshot model. Since there are reductions from k-leader election and k-test-and-set to k-set agreement [12] it follows that there are also no augmented extension-based proofs and, hence, no extension-based proofs of the impossibility of wait-free protocols solving these tasks. Although there is a reduction from 2-set agreement to approximate agreement on a cycle of length 4 [5, 17], there is no known reduction from approximate agreement on a cycle of length 4 to k-set agreement for $k \geq 2$. Thus, Theorem 6.1 does not follow from Brusse and Ellen's results.

A protocol is *anonymous* if the steps taken by a process do not depend on its identifier. Brusse and Ellen showed that if there is an anonymous reduction from $\mathcal{T}$ to $\mathcal{S}$ and there is an augmented extension-based proof that $\mathcal{T}$ is impossible to solve anonymously in the iterated snapshot model, then there is an augmented extension-based proof that $\mathcal{S}$ is impossible to solve anonymously in the iterated snapshot model. They also showed that there is no augmented extension-based proof of the impossibility of an anonymous wait-free protocol solving k-set agreement for $n > k \geq 2$ processes in the iterated snapshot model. There are anonymous reductions from weak symmetry breaking and $(2n-2)$-renaming to k-set agreement. Therefore, there are no augmented extension-based proofs that anonymous wait-free protocols cannot solve weak symmetry breaking or $(2n-2)$-renaming. However, techniques from combinatorial topology have been used to prove that no such protocols exist for infinitely many values of n [16].

In the synchronous message passing model, k-set agreement is solvable, but at least t rounds are needed when there are more than kt processes and at most k processes can crash each round [19, 26, 27]. Sheng and Ellen [35] defined a version of extension-based proofs for the synchronous message passing model. They showed that there is no extension-based proof of the impossibility of solving k-set agreement in fewer than t rounds when there are $kt+1$ processes and at most k processes can crash each round, for $k \geq 2$ and $t \geq 3$. An extension-based proof of a t-round lower bound for a task consists of $t+1$ phases. At the end of the first phase, the prover commits to an initial configuration and, at the end of each subsequent phase, the prover commits to a configuration reachable in 1 round from the configuration to which it last committed. At the end of the last phase, the prover wins if and only if the final configuration to which it committed violates the specifications of the task or the protocol has provided contradictory information. In the first t phases, a query asks the protocol to produce a schedule from a specified configuration, with a specified upper bound on the number of processes that can crash each round, in which a specified value is output by some process, or say that no such schedule exists. In the final phase, a query asks what value a specified process outputs in a specified final configuration. In the first phase, the specified configurations are initial configurations and, in subsequent phases, the specified configurations are restricted to be configurations reachable in one round from the configuration to which the prover last committed.

There are two other results in distributed computing that have a similar flavour. Rincon Galeana, Winkler, Schmid and Rajsbaum [23] showed that partitioning arguments are insufficient to prove the impossibility of a uniform wait-free protocol for $(n-1)$-set agreement in the iterated immediate snapshot model. For the CONGEST model, Bachrach, Censor-Hillel, Dory, Efron, Leitersdorf and Paz [11] 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.

We have considered allowing the prover to ask other types of queries. 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. This is left for future work.

We cannot allow certain queries, such as asking for an upper bound on the length of any schedule. If the number of input configurations is finite and the prover is given such an upper bound, then it can ask a finite number of query chains to exhaustively search all final configurations, thereby fixing the protocol.

It may be possible to allow the prover to use such information in a restricted way

and still construct an adversarial set agreement protocol. For example, the prover might not be allowed to use this information to decide which queries to ask or what extensions to construct, but could win when it has constructed a schedule that is longer than this upper bound.

Attiya, Castañeda, and Rajsbaum [7] studied the limitations of valency arguments for proving the impossibility of uniform protocols in the iterated immediate snapshot model. In their proofs, the prover learns the set of values that are output by processes in final configurations reachable from a given configuration, but it does not learn which value is output by any specific process. They showed their class does not contain proofs that $(n-1)$-set agreement has no uniform protocol and that weak symmetry breaking has no anonymous uniform protocol.

The definition of an extension-based proof can be modified to handle other termination conditions, such as obstruction-freedom [25]. It suffices for the prover to construct a schedule that violates this condition.

Ellen, Gelashvili and Zhu [21] 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. Specifically, the key technique in their proof is a reduction from the impossibility of set agreement via a simulation argument. Some of the early work about extension-based proofs motivated the approach used in that paper. 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.

Acknowledgments. We would like to thank Valerie King, Toniann Pitassi, and Michael Saks for helpful discussions and Shi Hao Liu for his useful feedback.

REFERENCES

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

[2] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J.ACM, 40 (1993), pp. 873–890.

[3] D. Alistarh, J. Aspnes, F. Ellen, R. Gelashvili, and L. Zhu, *Why extension-based proofs fail*, in Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC), 2019, pp. 986–996.

[4] D. Alistarh, J. Aspnes, F. Ellen, R. Gelashvili, and L. Zhu, *Brief announcement: Why extension-based proofs fail*, in Proceedings of the 39th Annual ACM Symposium on Principles of Distributed Computing (PODC), 2020, pp. 54–56.

[5] D. Alistarh, F. Ellen, and J. Rybicki, *Wait-free approximate agreement on graphs*, in Proceedings of the 28th International Colloquium on Structural Information and Communication Complexity (SIROCCO), 2021, pp. 87–105.

[6] H. Attiya and A. 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), 2011, pp. 108–119.

[7] H. Attiya, A. Castañeda, and S. Rajsbaum, *Locally solvable tasks and the limitations of valency arguments*, in Proceedings of the 24th International Conference on Principles of Distributed Systems (OPODIS), 2020, pp. 18:1–18:16.

[8] H. Attiya and F. Ellen, *Impossibility Results for Distributed Computing*, Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers, 2014.

[9] H. Attiya and A. Paz, *Counting-based impossibility proofs for renaming and set agreement*, in Proceedings of the 26th International Symposium on Distributed Computing (DISC), 2012, pp. 356–370.

[10] H. Attiya and S. Rajsbaum, *The combinatorial structure of wait-free solvable tasks*, SIAM J. Comput., 31 (2002), pp. 1286–1313.

[11] N. Bachrach, K. Censor-Hillel, M. Dory, Y. Efron, D. Leitersdorf, and A. Paz, *Hardness of distributed optimization*, in Proceedings of the 38th Annual ACM Symposium on Principles of Distributed Computing (PODC), 2019, pp. 238–247.

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

[13] E. Borowsky and E. Gafni, *Immediate atomic snapshots and fast renaming (extended abstract)*, in Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing (PODC), 1993, pp. 41–51.

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

[15] K. Brusse and F. Ellen, *Reductions and extension-based proofs*, in Proceedings of the 40th Annual ACM Symposium on Principles of Distributed Computing (PODC), 2021, pp. 497–507.

[16] A. Castañeda and S. Rajsbaum, *New combinatorial topology bounds for renaming: the lower bound*, Distributed Computing, 22 (2010), pp. 287–301.

[17] A. Castañeda, S. Rajsbaum, and M. Roy, *Convergence and covering on graphs for wait-free robots*, J. Braz. Comput. Soc., 24 (2018), p. 1.

[18] S. Chaudhuri, *More choices allow more faults: Set consensus problems in totally asynchronous systems*, Inform. and Comput., 105 (1993), pp. 132–158.

[19] S. Chaudhuri, M. Herlihy, N. A. Lynch, and M. R. Tuttle, *Tight bounds for k-set agreement*, J.ACM, 47 (2000), pp. 912–943.

[20] B. Chor, A. Israeli, and M. Li, *On processor coordination using asynchronous hardware*, in Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC), 1987, pp. 86–97.

[21] F. Ellen, R. Gelashvili, and L. Zhu, *Revisionist simulations: A new approach to proving space lower bounds*, in Proceedings of the 37th Annual ACM Symposium on Principles of Distributed Computing (PODC), 2018, pp. 61–70.

[22] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, J.ACM, 32 (1985), pp. 374–382.

[23] H. R. Galeana, K. Winkler, U. Schmid, and S. 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), 2019, pp. 307–322.

[24] M. Herlihy, *Wait-free synchronization*, ACM Trans. Programming Languages and Systems, 13 (1991), pp. 124–149.

[25] M. Herlihy, V. Luchangco, and M. Moir, *Obstruction-free synchronization: Double-ended queues as an example*, in Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS), 2003, pp. 522–529.

[26] M. Herlihy, S. Rajsbaum, and M. R. Tuttle, *An overview of synchronous message-passing and topology*, Electron. Notes Theor. Comput. Sci., 39 (2001), pp. 1–17.

[27] M. Herlihy, S. Rajsbaum, and M. R. Tuttle, *An axiomatic approach to computing the connectivity of synchronous and asynchronous systems*, Electron. Notes Theor. Comput. Sci., 230 (2009), pp. 79–102.

[28] M. Herlihy and N. Shavit, *The topological structure of asynchronous computability*, J.ACM, 46 (1999), pp. 858–923.

[29] G. Hoest and N. Shavit, *Toward a topological characterization of asynchronous complexity*, SIAM J. Comput., 36 (2006), pp. 457–497.

[30] S. Liu, *The impossibility of approximate agreement on alarger class of graphs*, in Proceedings of the 26th International Conference on Principles of Distributed Systems (OPODIS), 2022, pp. 22:1–22:20.

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

[32] Y. Moses and S. Rajsbaum, *A layered analysis of consensus*, SIAM J. Comput., 31 (2002), pp. 989–1021.

[33] T. Pitassi, P. Beame, and R. Impagliazzo, *Exponential lower bounds for the pigeonhole principle*, Comput. Complexity, 3 (1993), pp. 97–140.

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

[35] Y. Sheng and F. Ellen, *Extension-based proofs for synchronous message passing*, in Pro-

ceedings of the 35th International Symposium on Distributed Computing (DISC), 2021, pp. 36:1–36:17.