


# A General Class of Reductions and Extension-Based Proofs

Yusong Shi ✉ 

Department of Computer Science and Technology, Tsinghua University, Beijing, China

Weidong Liu ✉ 

Department of Computer Science and Technology, Tsinghua University, Beijing, China  
Zhongguancun Laboratory, Beijing, China

---

## Abstract

The concept of extension-based proofs models the idea of a valency argument which is widely used in distributed computing. Extension-based proofs have been shown to be limited in power: there is no extension-based proof of the impossibility of a wait-free protocol for  $(n, k)$ -set agreement among  $n > k \geq 2$  processes.

Previous work used a restricted class of reductions to show that there are no extension-based proofs of the impossibility of wait-free protocols for some other distributed computing problems. It is known that for a restricted class of reductions, if a task  $\mathcal{T}$  reduces to  $\mathcal{S}$  and  $\mathcal{T}$  has an augmented extension-based proof that it is impossible to solve in the NIS model, then so does  $\mathcal{S}$ . We introduce multiple-instance extension-based proofs and show that, if  $\mathcal{T}$  reduces to multiple instances of  $\mathcal{S}$ , instead of just one instance and  $\mathcal{T}$  has an augmented extension-based proof, then  $\mathcal{S}$  has a multiple-instance extension-based proof that it is impossible to solve in the NIIS model. We introduce a new version of extension-based proofs that can further our understanding of extension-based proofs and their limitations.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Interactive proof systems; Theory of computation  $\rightarrow$  Distributed algorithms; Theory of computation  $\rightarrow$  Distributed computing models; Theory of computation  $\rightarrow$  Problems, reductions and completeness

**Keywords and phrases** Reductions, Impossibility proofs, Extension-based proof

**Digital Object Identifier** 10.4230/LIPIcs.OPODIS.2024.19

**Acknowledgements** We would like to thank Faith Ellen and Shihao Liu for helpful discussions and the anonymous reviewers for their comments.

## 1 Introduction

In 1985, Fischer, Lynch, and Paterson [12] proved one of the most important results in distributed computing: There is no deterministic wait-free protocol for the consensus task in the asynchronous message passing system. The key idea of their proof is called a valency argument, which proves the existence of an infinite execution when the algorithm never outputs an incorrect output. The  $(n, k)$ -set agreement task, which is a generalization of the consensus task where  $n$  processes can output at most  $k$  different values, first proposed by Chaudhuri [11], was independently shown to have no wait-free protocol by Borowsky and Gafni [7], Herlihy and Shavit [14], and Saks and Zaharoglou [17]. These papers used topological structures to model tasks and protocols.

In [2, 3], Alistarh, Aspnes, Ellen, Gelashvili and Zhu pointed out the differences between valency arguments and combinatorial or topological techniques. In those proofs using combinatorial or topological techniques, the existence of a bad execution is proved, but not explicitly constructed. In the proof by Fischer, Lynch and Paterson, an infinite execution can be obtained by extending an initial execution infinitely often. Alistarh et al. generalized this type of proof and called it an extension-based proof. An extension-based proof is defined



© Yusong Shi and Weidong Liu;

licensed under Creative Commons License CC-BY 4.0

28th International Conference on Principles of Distributed Systems (OPODIS 2024).

Editors: Silvia Bonomi, Letterio Galletta, Etienne Rivière, and Valerio Schiavoni; Article No. 19; pp. 19:1–19:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

as an interaction between a prover and a protocol that claims to solve a task. Initially, the prover only knows the initial configurations of the protocol. To obtain information about the protocol, the prover proceeds in phases. In each phase, the prover submits queries to the protocol to learn information about this protocol. The prover wins if it discovers any error of the protocol. Otherwise, the protocol wins. If there exists a prover that can win against any protocol that claims to solve a task, we say that this task has an extension-based impossibility proof. The proof of the impossibility of consensus [12] is an example of an extension-based proof. It was shown that there are no extension-based proofs for the impossibility of a wait-free protocol for  $(n, k)$ -set agreement in the non-uniform iterated immediate snapshot (NIIS) model [2] and in the non-uniform iterated snapshot (NIS) model [3].

Do other tasks also have no extension-based impossibility proofs? One possible way to extend existing results is via reductions, which is a widely used technique in the field of theoretical computer science. The *composition*  $\mathcal{R} \circ \mathcal{S}$  of two tasks  $\mathcal{S}$  and  $\mathcal{R}$  is a task that can be solved by a protocol that first solves  $\mathcal{S}$  and then solves  $\mathcal{R}$ , where outputs from  $\mathcal{S}$  serve as inputs to  $\mathcal{R}$ . A *reduction* from task  $\mathcal{T}$  to task  $\mathcal{S}$  is a protocol that solves  $\mathcal{T}$  by using a protocol that solves  $\mathcal{S}$ . Processes may use many instances of  $\mathcal{S}$  and communicate with one another to decide the input values for each instance.

Brusse and Ellen [10] introduced augmented extension-based proofs, which provide an additional type of query, and gave the first result about reductions: if  $\mathcal{T}$  reduces to  $\mathcal{S}$ , and  $\mathcal{T}$  has an augmented extension-based proof that it is impossible to solve in the NIS model, then so does  $\mathcal{S}$ , when the reduction is limited to the use of only one instance of  $\mathcal{S}$ . Such a reduction can be represented by  $\mathcal{R}_2 \circ \mathcal{S} \circ \mathcal{R}_1$  where  $\mathcal{R}_1, \mathcal{R}_2$  are solvable tasks in the NIS model. Our goal is to answer the open question proposed in their paper, i.e., whether the extension to reductions that allow multiple instances of  $\mathcal{S}$  is possible. In other words, we will consider the reductions that have the form  $\mathcal{R}_{l+1} \circ \mathcal{S} \circ \mathcal{R}_l \cdots \mathcal{S} \circ \mathcal{R}_1$  where  $\mathcal{R}_1, \mathcal{R}_2 \dots \mathcal{R}_{l+1}$  are solvable tasks.

Suppose that there is a protocol that claims to solve the task  $\mathcal{S}$ . To make use of the reduction from  $\mathcal{T}$  to  $\mathcal{S}$ , we need to compose multiple copies of this protocol and the protocols that solve  $\mathcal{R}_1, \mathcal{R}_2 \dots \mathcal{R}_{l+1}$  to generate a protocol that solves the task  $\mathcal{T}$ . This is not straightforward as in uniform models, such as the IIS model, in which all processes terminate after the same number of rounds during all executions of a protocol.

This paper mainly concerns two topics addressed in [10]: a way to compose non-uniform protocols and the construction of an augmented extension-based prover  $\mathcal{P}_{\mathcal{S}}$  for  $\mathcal{S}$  from an augmented extension-based prover  $\mathcal{P}_{\mathcal{T}}$  for  $\mathcal{T}$ . All the discussions in this paper are about the non-uniform iterated immediate snapshot (NIIS) model rather than the non-uniform iterated snapshot (NIS) model. The techniques in the two models are quite similar.

## 1.1 Our contribution

The way of composing multiple protocols in [10] can be generalized to a larger class of reductions. Note that these protocols are assumed to be transparent which means that the entity that tries to compose the protocols knows everything about these protocols. We present some properties of this composition that are necessary for discussions of the construction of an extension-based prover in both this paper and in [10], but are missing in the previous work. For example, a protocol in an extension-based proof is not a transparent protocol as assumed in the composition of multiple protocols, as the prover can only learn information about it by submitting queries to it. The usage of this composition in the discussions about extension-based proofs must be inspected in a more rigid way.

Next, we prove that if the task  $\mathcal{T}$  reduces to the task  $\mathcal{S}$  using multiple instances of  $\mathcal{S}$  and  $\mathcal{T}$  has an augmented extension-based proof that it is impossible to solve in the NIIS model, then  $\mathcal{S}$  has a multiple-instance extension-based proof that it is impossible to solve in the NIIS model. Our proof is not a trivial generalization of the proof in [10]. A technical difference here is that when the reduction from  $\mathcal{T}$  to  $\mathcal{S}$  involves more than one instance of  $\mathcal{S}$ , we have to discuss a new version of extension-based proofs, called multiple-instance extension-based proofs, where the prover interacts with multiple instances of the protocol (instead of only one instance of the protocol in the original definition). Then, similar to the approach in [10], we describe how to construct a multiple-instance extension-based prover  $\mathcal{P}_S$  from an extension-based prover  $\mathcal{P}_T$ . Given a protocol that claims to solve the task  $\mathcal{S}$ , the prover  $\mathcal{P}_S$  simulates an interaction between the composed protocol and  $\mathcal{P}_T$ .  $\mathcal{P}_S$  decides which queries it will submit in its interaction with multiple instances of the protocol that claims to solve  $\mathcal{S}$ , based on the queries it sees in this simulation. We will show that  $\mathcal{P}_S$  wins in the interaction since  $\mathcal{P}_T$  wins in its interaction with  $\mathcal{T}$  by assumption.

## 2 Preliminaries

### 2.1 NIIS model

A *snapshot* object consists of an array of elements and supports two types of operation: write and read. In a write operation, a process with id  $i$  writes a value to the  $i$ -th cell of the array. In a snapshot operation, a process atomically reads the contents of all the cells of the array. Similarly, the *immediate snapshot (IS)* object was introduced by Borowsky and Gafni in [8]. An IS object consists of an array of elements and provides writeread operations, where a process with id  $i$  writes a value to the  $i$ -th element of the array and returns a snapshot of the array immediately after this write. The writeread operations on a single IS object by multiple processes are said to be concurrent if all their snapshot operations happen after all their writes to the array are finished.

IS objects are the basic building blocks of the iterated immediate snapshot (IIS) model. The IIS model consists of  $n$  processes  $\Pi = \{u_1, u_2 \dots u_n\}$  and a bounded sequence of IS objects  $\{IS_1, IS_2 \dots IS_e\}$ . In the execution of a protocol in the IIS model, each process will terminate after performing a writeread operation on each IS object in ascending order in the sequence. The IIS model has been shown to have computational power equivalent to that of the standard shared memory model in [1, 8].

Hoest and Shavit [15] introduced a new model called the *non-uniform iterated snapshot (NIIS)* model, a variant of the iterated immediate snapshot (IIS) model. The NIIS model consists of  $n$  processes  $\Pi = \{u_1, u_2 \dots u_n\}$  and an unbounded sequence of IS objects  $\{IS_1, IS_2 \dots\}$ . Each process starts with a process id  $i$  and its input value  $v_i$ , and performs a writeread operation on each IS object sequentially. By definition, each NIIS protocol is determined by a decision map  $\delta$  from a local state to output values (or a special value  $\perp$ ). In the execution of an NIIS protocol  $\delta$ , each time a process completes a writeread operation, it checks whether it has reached a final state by applying  $\delta$  to its local state variable. If  $\delta$  returns  $\perp$ , the process continues to access the next IS object. Otherwise, the process uses the return value of  $\delta$  as its output and terminates. An NIIS protocol is said to be *transparent* to some entity if this entity knows everything about the function  $\delta$ . In this paper, we consider *full-information NIIS protocols* in which each process uses its current state as an argument to perform a writeread operation. After a writeread operation, its new state consists of its process id  $i$  and the result of the writeread operation. Note that unlike the IIS model, the NIIS model allows different processes to terminate after accessing a different number of IS objects.

A *configuration* of a protocol consists of the state of each process and the contents of each IS object. Since each process remembers its entire history and only process  $p_i$  can write to the  $i$ -th component of each IS object, the state of each IS object is included in states of the processes, which means that a configuration is fully determined by the states of processes. A process is *active* in a configuration if it is not terminated. A configuration is *final* if it has no active processes. If  $C$  is a configuration and  $U$  is a set of processes that are poised to perform writeread operations on the same IS object, then  $C\{U\}$  is the configuration that results from  $C$  when the processes in  $U$  concurrently take a step in the protocol. A *schedule* of a protocol from  $C_0$  is a finite or infinite sequence of sets of processes  $\gamma = U_1, U_2, \dots$  such that there is a sequence of configurations  $C_0, C_1, \dots$  where processes in  $U_i$  is active in  $C_{i-1}$  and  $C_i = C_{i-1}U_i$  for all  $U_i$  in  $\gamma$ . A  *$U$ -only schedule* from  $C$  is a schedule in which only processes in  $U$  appear. Each finite schedule from an initial configuration results in a *reachable configuration*. A protocol is *wait-free* if each process is terminated after a finite number of steps.

A *task*  $\mathcal{T}$  consists of a finite set of input vectors,  $I$ , a finite set of output vectors,  $O$ , and a map  $T : I \rightarrow 2^O$  that represents the task specification. For each input vector  $x \in I$ ,  $T$  maps  $x$  to a non-empty set  $T(x) \subseteq O$  of output vectors. Suppose that  $A$  is a wait-free protocol that has an initial configuration with input vector  $x$  for each  $x \in I$ . We say that a protocol  $A$  solves  $\mathcal{T}$  if, for each  $x \in I$ , every final configuration that is reached from the initial configuration of  $A$  that has input vector  $x$  has output vector  $y$  and  $y \in T(x)$ . If  $\mathcal{T}$  and  $\mathcal{T}'$  are tasks with maps  $T$  and  $T'$ , respectively, their composition  $\mathcal{T} \circ \mathcal{T}'$  is a task that has the same set of input vectors as  $\mathcal{T}$  and the same set of output vectors as  $\mathcal{T}'$ . The map of the composed protocol is defined by  $T' \circ T(x) = \cup\{T'(y) | y \in T(x)\}$ .

## 2.2 Extension-based proofs

In [3], Alistarh, Aspnes, Ellen, Gelashvili and Zhu formally defined the class of extension-based proofs that generalizes the impossibility proof technique used in the FLP impossibility result. The proof of impossibility is modeled as an interaction between a prover and a wait-free NIIS protocol that claims to solve the task. The protocol is defined by a map  $\delta$  from the process states to the output values or a special value  $\perp$ . The prover asks queries to learn information about the protocol, such as the  $\delta$  values of states of processes in some reached configuration, in an effort to find a violation of the task specification, or construct an infinite execution.

If there exists a prover that can defeat any protocol that claims to solve the task, we say that the task has an extension-based impossibility proof. But if for each prover, an adversary can adaptively design a protocol based on the queries made by the prover such that the prover cannot win in the interaction, there is no extension-based proof for the impossibility of the task. The adaptive protocol constructed by the adversary will be referred to as an adversarial protocol as in [3].

Three classes of extension-based proofs are defined in [3, 10]. We will first introduce restricted extension-based proofs. The interaction between the prover and a protocol proceeds in phases. In each phase  $\varphi$  the prover starts with a finite schedule  $\alpha(\varphi)$  and a set of configurations  $\mathcal{A}(\varphi)$  reached from some initial configurations of the task by  $\alpha(\varphi)$  which only differ in the input values of processes that are not in the schedule  $\alpha$ . For  $\varphi = 1$ , the schedule  $\alpha(1)$  is the empty schedule and  $\mathcal{A}(1)$  contains all initial configurations. The set of configurations reached in phase  $\varphi$  is denoted by  $\mathcal{A}'(\varphi)$  which is empty at the beginning of phase  $\varphi$ . The prover queries the protocol by choosing a configuration  $C \in \mathcal{A}(\varphi) \cup \mathcal{A}'(\varphi)$  and a set of processes  $U$  that are poised to perform writeread operations on the same IS object in  $C$ . The protocol replies with the value of  $\delta$  of each process in  $U$  in the configuration  $C'$

resulting from the processes in  $U$  concurrently performing writeread operations, and the prover adds the configuration  $C'$  to  $\mathcal{A}'(\varphi)$ . A chain of queries is a (finite or infinite) sequence of queries such that if  $(C_i, U_i)$  and  $(C_{i+1}, U_{i+1})$  are consecutive queries in the chain, then  $C_{i+1}$  is the configuration resulting from scheduling  $U_i$  from  $C_i$ . The prover is allowed to construct finitely many chains of queries in a phase.

When the output values from the response of a query are not allowed output values for the task, we say that the prover finds a violation of the task specification. Similarly, if a prover can construct an infinite chain of queries, the protocol must admit that it is not wait-free. If the prover does not find a violation or construct an infinite execution after a finite number of queries or chains of queries, it must end the current phase and choose some configuration  $C' \in \mathcal{A}'(\varphi)$ . Let  $\alpha'$  be the schedule such that  $C'$  is reached via  $\alpha'$  from some configuration in  $\mathcal{A}(\varphi)$ . The schedule  $\alpha(\varphi)\alpha'$  will be used as the initial schedule  $\alpha(\varphi + 1)$  in the next phase. Suppose that the configuration  $C'$  is reached from an initial configuration  $C_{ini}$ . Then  $\mathcal{A}(\varphi + 1)$  consists of all configurations reached by the schedule  $\alpha(\varphi + 1)$  from initial configurations that only differ from  $C_{ini}$  by the input values of the processes that do not appear in  $\alpha(\varphi + 1)$ .

Extension-based proofs allow for an additional type of query. The prover performs an output query by choosing a configuration  $C \in \mathcal{A}(\varphi) \cup \mathcal{A}'(\varphi)$ , a set of processes  $U$  that are poised to access the same IS object, and a value  $y \in \{0, 1, \dots, k\}$ . If there is a  $U$ -only schedule from  $C$  that results in a configuration in which a process in  $U$  outputs  $y$ , the protocol will return some such schedule. Otherwise, the protocol returns *NULL*. Augmented extension-based proofs provide a more general query, called an assignment query. An assignment query consists of a configuration  $C \in \mathcal{A}(\varphi) \cup \mathcal{A}'(\varphi)$ , a set of processes  $U$ , and an assignment function  $f$  from a subset  $U'$  of  $U$  to the output values. If there is a  $U$ -only schedule from  $C$  that results in a configuration in which the output value of each process  $u$  in  $U'$  is  $f(u)$ , the protocol will return some such schedule. Otherwise, the protocol will return *NULL*. It is shown that an output query  $(C, U, y)$  can be simulated by a sequence of assignment queries  $(C, U, f_u)$  for each process  $u$  in [10]. Therefore, we will discuss only augmented extension-based proofs in subsequent sections.

The tasks which have been proved to have no extension-based proofs of the impossibility are still quite limited. It is proved in [3, 2] that the  $(n, k)$ -set agreement task does not have extension-based proofs of the impossibility in the NIIS and NIS models. Some similar results on the approximate agreement task are proved in [4, 16]. Shi and Liu [18] gave a topological characterization of a colorless task to have no extension-based proofs of impossibility.

Attiya, Castañeda, and Rajsbaum [5] studied the limitations of valency arguments in the IIS model. Note that in the IS and IIS models, extension-based proofs are too powerful. They define the class of local valency impossibility proofs, that cannot show there is no wait-free protocol for  $(n, n - 1)$ -set agreement, weak symmetry breaking or  $(2n - 2)$ -renaming in the IIS model. Attiya, Fraigniaud, Paz and Rajsbaum [6] introduced an FLP-style impossibility proof which is a simple case of extension-based proofs and local valency impossibility proofs, and showed that there are no FLP-style proofs of the impossibility of any 1-dimensional colorless task if it is not wait-free solvable.

### 2.3 A class of reductions in the NIIS model

If a task  $\mathcal{T}$  reduces to a task  $\mathcal{S}$ , then there exists a protocol  $A'_{l+1} \circ A_l \circ A'_l \cdots A_1 \circ A'_1$  that solves the task  $\mathcal{T}$  where each  $A_i$  is a protocol to solve the task  $\mathcal{S}$  and each  $A'_i$  is an NIIS protocol for some task  $\mathcal{R}_i$ . We assume that the number of instances of the protocol to solve  $\mathcal{S}$  is a constant. We say that  $(\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_l, \mathcal{R}_{l+1})$  is a reduction from a task  $\mathcal{T}$  to a task  $\mathcal{S}$ .

Previous work[10] used a restricted version of this definition in which only one instance in this composition is a protocol that solves the task  $\mathcal{S}$ . The restricted version of reductions in their paper is enough to prove the non-existence of extension-based proofs for many tasks. But some reductions use multiple instances of  $\mathcal{S}$ , such as the reduction given by [13] from weak symmetry breaking to  $(n - 1)$ -set agreement, which uses two instances of  $(n - 1)$ -set agreement.

### 3 Composing NIIS protocols

Let  $(\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_l, \mathcal{R}_{l+1})$  be a reduction from a task  $\mathcal{T}$  to a task  $\mathcal{S}$ . Suppose that some protocol, denoted as  $A$ , claims that it solve the task  $\mathcal{T}$ . We denote the NIIS protocols that solve  $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_l, \mathcal{R}_{l+1}$  by  $A'_1, \dots, A'_l, A'_{l+1}$ . A difference between  $A$  and  $A'_i$ , for each  $1 \leq i \leq l + 1$ , is that everything about  $A'_i$  is known, which is not true for  $A$ . We can assume without loss of generality that all processes will terminate after accessing the same number of IS objects in the execution of  $A'_i$ . The composed protocol  $A'_{l+1} \circ A_l \circ A'_l \cdots A_1 \circ A'_1$  should solve  $\mathcal{T}$  where each  $A_i$  is an instance of  $A$ . Note that each  $A_i$  is an NIIS protocol, and we *cannot* assume that all processes will terminate after accessing the same number of IS objects in the execution of  $A_i$ .

We consider the case where there are two NIIS protocols, since a more general composition can be easily obtained if we can compose two NIIS protocols.

#### 3.1 Composing two NIIS protocols

The composition of NIIS protocols is much more complicated than the composition of IIS protocols, as processes can terminate after different rounds during the execution of some NIIS protocol. To compose IIS protocols, we can simply have each process execute IIS protocols sequentially. Since each process ends the execution of a protocol after the same round, the information used in the execution of different IIS protocols will not be mixed together. But this is *not* true for NIIS protocols, since processes accessing the same IS object in the composed protocol may be running different NIIS protocols. We need a mechanism to separate information from different NIIS protocols.

Let  $A_1$  and  $A_2$  be NIIS protocols such that each output from  $A_1$  is a possible input for  $A_2$ . We temporarily assume that everything about  $A_1$  and  $A_2$  is known (this is not true in the construction of  $\mathcal{P}_S$ ). In [10], Brusse and Ellen provided an implementation of the composed protocol  $A_2 \circ A_1$  in the NIS model using estimate vectors introduced by Borowsky and Gafni [9]. The composition of two NIIS protocols in our paper can use almost identical techniques. But since this is quite important in this paper, we have to explain it in detail. Let  $IS'_h$ ,  $IS''_h$  and  $IS_h$  denote the  $h$ -th IS object accessed by processes in the schedules of  $A_1$ ,  $A_2$ , and  $A_2 \circ A_1$ , respectively. Each process  $u_i$  simulates the steps of  $u_i$  in protocol  $A_1$  followed by the steps of  $u_i$  in protocol  $A_2$ . The result of a writeread operation in simulation of  $A_1$  can be obtained from some writeread operation of the composed protocol. Since different processes may terminate after different rounds in  $A_1$ , the result of a writeread operation in the simulation of  $A_2$  cannot be easily obtained. Each process locally maintains an infinite sequence of estimate vectors, each with  $n$  components. The  $h$ -th estimate vector of process  $u_i$  is a potential output of the writeread operation on  $IS''_h$  in the simulation of  $A_2$ .

Let  $\gamma$  be a schedule of the composed protocol  $A_2 \circ A_1$  from some initial configuration  $C_{ini}$ . Each process updates its estimate vectors during this schedule of the composed protocol  $A_2 \circ A_1$ , and finalizes its  $i$ -th estimate vector after collecting enough information. This estimate vector can be proved to be equivalent to the return value of the writeread operation

to  $IS_i''$  by the same process in some schedule of the second NIIS protocol. We can call this schedule the restriction of  $\gamma$  to  $A_2$ . A process will apply the function  $\delta$  that specifies the second protocol to the finalized estimate vector to decide whether it should end the simulation of  $A_2$  (and which value to output if so).

We give some pseudocode in Algorithm 1. The process  $u_i$  performs writeread operations to  $IS_1, IS_2 \dots$  sequentially, with  $h_i$  indicating the next IS object to be accessed. The value written to  $IS_i$  by a process consists of three parts:  $r$  (round number),  $state1$  (state in the simulation of  $A_1$ ) and  $state2$  (state in the simulation of  $A_2$ ). The round number is 0 when the process is simulating  $A_1$ , is  $r$  when simulating the  $r$ -round of  $A_2$ . The attribute  $state2$  is a sequence of estimate vectors.

When a process simulates  $A_1$ , it computes the result of the writeread operation on  $IS_i'$  from the result of the writeread operation on  $IS_i$  by ignoring the attribute  $state2$ . However, a process  $u_i$  will update its estimate vectors using the result as described in Algorithm 2.  $u_i$  updates the  $k'$ -th component of its  $k$ -th estimate vector when this component is blank and the result of a writeread operation on  $IS_{h_i}$  contains some estimate vectors from some other process, where the  $k'$ -th component of its  $k$ -th estimate vector is not blank. After a process  $u_i$  has finished simulating  $A_1$ , it modifies its first estimate vector to include its output of  $A_1$  as the input for  $A_2$  and then tries to finalize it. To finalize its  $r_i$ -th estimate vector, for any  $r_i \geq 1$ , the process repeatedly performs a writeread operation on the next IS object  $IS_{h_i}$  (to contain its first  $r_i$  estimate vectors). The process updates its estimate vectors just as it does when simulating  $A_1$ . The process does not finalize its  $r_i$ -th estimate vector in two cases. The first case is that it sees some other process  $u_i'$  accessing  $IS_{h_i}$  when that process was simulating  $A_1$  or an earlier round of  $A_2$ . This is to avoid process  $u_i'$ , which is simulating an earlier round, from missing the finalized  $r_i$ -th estimate vector of process  $u_i$ . The other case is that it changes its  $r_i$ -th estimate vector using information from  $IS_{h_i}$ .

Otherwise, the process finalizes its  $r_i$ -th estimate vector. When a process finalizes its  $r_i$ -th estimate vector, this vector will be used as the output of its writeread operation on  $IS_{r_i}''$  in the simulation of  $A_2$ . Note that it may take many rounds of the composed protocol for a process to finalize its current estimate vector, i.e. to take a step in the simulated execution. Based on the return values of its  $r_i$ -th estimate vector, the process terminates its simulation of  $A_2$  or modifies its  $(r_i + 1)$ -th estimate vector to include its new state in the simulation of  $A_2$  and continues to finalize its  $(r_i + 1)$ -th estimate vector.

### 3.2 Some properties of the composition

Now we present some properties that are necessary in Section 4. The first property was shown in [10].

► **Lemma 1.** *The composed protocol  $A_2 \circ A_1$  is wait-free if both  $A_1$  and  $A_2$  are wait-free.*

The second property is about restrictions. Given an initial configuration  $C_{ini}$  of  $A_2 \circ A_1$ , the composed protocol simulates a schedule of  $A_1$  from  $C_{ini}$ , followed by a schedule of  $A_2$ , where the output of the first schedule serves as input to the second schedule. Let  $\gamma$  be a schedule of  $A_2 \circ A_1$  from  $C_{ini}$ . The restriction of  $\gamma$  to  $A_1$  is defined as the schedule  $\gamma_1$  obtained from  $\gamma$  by removing all the occurrences of each process after simulating  $A_1$ . Each process  $u_i$  finishes its simulation of  $A_1$  with output  $y_i$  during the schedule  $\gamma$  of  $A_2 \circ A_1$  from  $C_{ini}$  if and only if  $u_i$  terminates with output  $y_i$  during the schedule  $\gamma_1$  of  $A_1$  from  $C_{ini}$ .

Let  $U$  be the set of processes that complete their simulation of  $A_1$  during  $\gamma$  and, for each  $u_i \in U$ , let  $y_i$  be its simulated output from  $A_1$ . Let  $C_2$  be any initial configuration of  $A_2$  in which each process  $u_i \in U$  has input  $y_i$ . We have to prove that, for each  $l$ , the  $l$ -th estimate

■ **Algorithm 1** The composition of two NIIS protocols  $A_1$  and  $A_2$ .

---

```

Input : Decision functions  $\delta_1$  and  $\delta_2$  representing  $A_1$  and  $A_2$ , input value  $I_i$ 
Output : Output value  $O_i$ 
1 Function Main( $i, n$ ):
2    $IS_1, IS_2 \dots \leftarrow$  an infinite sequence of IS objects /* shared by processes */
3    $E_i \leftarrow$  an infinite sequence of estimate vectors, each with  $n$  components
4    $r_i \leftarrow 0$  /* current round, which is 0 when simulating  $A_1$ , and is the round number in  $A_2$  when
   simulating  $A_2$  */
5    $s_i \leftarrow (r_i, (i, I_i), \perp)$  /* current state of simulation, a class with three attributes: r(round
   number), state1(state in the simulation of  $A_1$ ) and state2 (state in the simulation of  $A_2$ )
   */
6    $h_i \leftarrow 1$  /* next IS object to access */
7   while  $\delta_1(s_i.state1) \neq \perp$  do
8      $states\_of\_processes \leftarrow IS_{h_i}.writeread(s_i)$ 
9      $h_i \leftarrow h_i + 1$ 
10     $s_i.state1 \leftarrow \{states\_of\_processes[k].state1 | k \in [1, n]\}$ 
11    for  $i' \leftarrow 1$  to  $n$  do
12       $update\_ev(1, E_i, states\_of\_processes[i'])$ 
13    end
14  end
15  /* End of the simulation of  $A_1$  */
16   $r_i \leftarrow 1$ 
17   $s_i \leftarrow (r_i, s_i.state1, \delta_1(s_i.state1))$ 
18   $E_i[r_i][i] \leftarrow \delta_1(s_i.state1)$ 
19  while  $\delta_2(E_i[r_i][i]) \neq \perp$  do
20    while True do
21       $can\_finalize \leftarrow true$ 
22       $states\_of\_processes \leftarrow IS_{h_i}.writeread((r_i, s_i.state1, [E_i[1], \dots, E_i[r_i]]))$ 
23       $h_i \leftarrow h_i + 1$ 
24      for  $i' \leftarrow 1$  to  $n$  do
25        if  $states\_of\_processes[i'] \neq \perp$  then
26          if  $states\_of\_processes[i'].r < r_i$  then
27             $can\_finalize \leftarrow false$ 
28          else
29            if  $update\_ev(r_i, E_i, states\_of\_processes[i'])$  then
30               $can\_finalize \leftarrow false$ 
31            end
32          end
33        end
34      end
35      if  $can\_finalize$  then
36        break
37      end
38    end
39     $r_i \leftarrow r_i + 1$ 
40     $E_i[r_i][i] \leftarrow E_i[r_i - 1]$ 
41  end
42  return  $\delta_2(E_i[r_i][i])$ 

```

---

■ **Algorithm 2** update estimate vectors.

---

```

1 Function update_ev( $r_{min}, E_i, state\_of\_process$ ):
2   if  $state\_of\_process.r < r_{min}$  then
3     return false
4   end
5    $update \leftarrow false$ 
6   for  $k \leftarrow r_{min}$  to  $state\_of\_process.r$  do
7     for  $k' \leftarrow 1$  to  $n$  do
8       if  $E_i[k][k'] \neq \perp$  &  $state\_of\_process.state2[k][k'] \neq \perp$  then
9          $E_i[k][k'] \leftarrow state\_of\_process.state2[k][k']$ 
10         $update \leftarrow true$ 
11      end
12    end
13  end
14  return  $update$ 
15

```

---



vectors that have been finalized by the processes in  $U$  correspond to the responses of the writeread operations in some  $U$ -only schedule  $\gamma_2$  of  $A_2$  from  $C_2$ . Each process  $u_i$  finishes its simulation of  $A_2$  with output  $z_i$  during the schedule  $\gamma$  of  $A_2 \circ A_1$  from  $C_{ini}$  if and only if  $u_i$  terminates with output  $z_i$  during the schedule  $\gamma_2$  of  $A_2$  from  $C_2$ . The *restriction of  $\gamma$  to  $A_2$*  is defined as the schedule  $\gamma_2$ . Note that we are using the same definition as in [10] except that we are using the NIIS model rather than the NIS model.

► **Lemma 2.** *The estimate vectors finalized by the processes during the schedule  $\gamma$  of  $A_2 \circ A_1$  from  $C_{ini}$  correspond to the responses of the writeread operations during the schedule  $\gamma_2$ , which is the restriction of  $\gamma$  to  $A_2$ , starting from  $C_2$ .*

**Proof.** For any simulated IS object  $IS_r''$  of  $A_2$ , let  $Q_h$  denote the set of processes that have finalized their  $r$ -th estimate vectors after performing writeread operations to  $IS_h$  of the composed protocol  $A_2 \circ A_1$ . For any indices  $j, j'$  and  $k$ , if  $E_j[r][k] \neq \perp$  and  $E_{j'}[r][k] \neq \perp$ , then  $E_j[r][k] = E_{j'}[r][k]$ . This is because the value  $E_j[r][k]$  can only be defined on line 40 by the process with id  $k$ , and other processes are only allowed to copy this value into their estimate vectors on line 9 of Algorithm 2. We can define a partial order on the  $r$ -th estimate vectors. Given the indices  $j$  and  $j'$ , if for any  $k$ ,  $E_j[r][k] = \perp$  or  $E_j[r][k] = E_{j'}[r][k] \neq \perp$ , then  $E_j[r]$  is ordered before  $E_{j'}[r]$  (i.e.  $E_{j'}[r]$  contains each non-blank value in  $E_j[r]$ ). If there is a partial order defined on the set of finalized  $r$ -th estimate vectors, then these finalized  $r$ -th estimate vectors can be seen as the result of writeread operations on  $IS_r''$ .

Suppose that the process  $u_i$  is in  $Q_{h+1}$  but not in  $Q_h$ , i.e.  $u_i$  finalizes its  $r$ -th estimate vector after accessing  $Q_h$ . Consider the response of the writeread operation by a process  $u_i$  on  $IS_h$ . According to our implementation (checks on *can\_finalize* in Algorithm 1), the  $r$ -th estimate vector of the process  $u_i$  is ordered after or concurrently with the  $r$ -th estimate vector of  $u_{i'}$  if *states\_of\_processes*[ $i'$ ]  $\neq \perp$  (Line 25 in algorithm 1). For other index  $i'$  where *states\_of\_processes*[ $i'$ ]  $= \perp$ , the process  $u_{i'}$  will see the  $r$ -th finalized estimate vector of  $u_i$  before finalizing its  $r$ -th estimate vector, since  $u_i$  always includes  $E_i[r]$  in its written value to  $IS_{h+1}, IS_{h+2} \dots$ . This means that the  $r$ -th finalized estimate vector of  $u_i$  is ordered before that of  $u_{i'}$ . There exists a total order on the finalized  $r$ -th estimate vectors.

For each index  $r$ , we divide the processes that have finalized its  $r$ -th estimate vector into a sequence  $SEQ_r$  of sets, where processes having the same  $r$ -th estimate vector are in the same set. These sets are ordered using the partial order on the finalized  $r$ -th estimate vectors. The restriction of  $\gamma$  to  $A_2$  is defined as the concatenation  $SEQ_1 \circ SEQ_2 \dots$ . ◀

The last property is related to extension-based proofs. An assumption of the algorithm to compose two NIIS protocols is that the two NIIS protocols are transparent i.e. the entity that tries to compose them knows everything about the functions  $\delta_1, \delta_2$ . In the definition of extension-based proofs, the protocol that claims to solve  $\mathcal{S}$  is not transparent. We show that if an entity is allowed to interact with NIIS protocols that are not transparent just as a prover does, this entity can still compose two NIIS protocols.

► **Lemma 3.** *When two NIIS protocols are not transparent protocols, if an entity are allowed to submit queries to the protocols that are not transparent, the information obtained through queries is sufficient for the composition of NIIS protocols.*

**Proof.** When composing two NIIS protocols  $A_1 \circ A_2$ , updates and finalization of estimate vectors are independent of the protocols  $A_1, A_2$ . The functions  $\delta_1, \delta_2$  representing protocols  $A_1, A_2$  are only used when some estimate vector is finalized to determine whether to terminate the simulation of  $A_1, A_2$  on line 7, 19 of Algorithm 1. By assumption, this entity can submit queries to the protocol to learn about this information. ◀

### 3.3 Composing multiple NIIS protocols

The composition of multiple NIIS protocols  $A_m \cdots A_2 \circ A_1$  can be derived from the composition of two NIIS protocols. Inductively,  $A_m \cdots A_2 \circ A_1$  is the composition of  $A_1$  and  $A_m \cdots A_3 \circ A_2$ . Let  $\gamma$  be a schedule of  $A_m \cdots A_2 \circ A_1$  from some initial configuration  $C$ . Let  $\gamma(1)$  denote the restriction of  $\gamma$  to  $A_1$ ,  $\gamma(2)$  denote the restriction of  $\gamma$  to  $A_m \cdots A_3 \circ A_2$ . The restriction of  $\gamma$  to  $A_j$ , where  $2 \leq j \leq m$ , can be defined inductively as the restriction of  $\gamma(2)$  to  $A_j$ . The properties proved in Section 3.2 also apply to the composition of multiple NIIS protocols.

## 4 Reductions and extension-based proofs

### 4.1 Simulation

In the following sections, we extend the proof in [10] to more general reductions. Suppose that  $\mathcal{T}$  reduces to  $\mathcal{S}$  and  $\mathcal{P}_T$  is an augmented extension-based prover that can win against any protocol that claims to solve  $\mathcal{T}$ . Our goal is to construct an augmented extension-based prover  $\mathcal{P}_S$  for task  $\mathcal{S}$  that can also win against any protocol  $A$  that claims to solve  $\mathcal{S}$ .

$\mathcal{P}_S$  will simulate an interaction between  $\mathcal{P}_T$  and the composed protocol  $A'_{l+1} \circ A_l \circ A'_l \cdots A_1 \circ A'_1$ , and will use restrictions to interact with multiple copies of  $A$ , as shown in Figure 1. There is an interaction between  $\mathcal{P}_T$  and the composed protocol in Figure 1. But at the same time, if we regard the elements within the black frame as a prover  $\mathcal{P}_S$ , the interaction can be seen to happen between  $\mathcal{P}_S$  and multiple instances of  $A$ .

If the prover  $\mathcal{P}_T$  submits a query in the simulation, the prover  $\mathcal{P}_S$  will reinterpret this query as queries to some instances of  $A$  and will use the responses of the instances of the protocol  $A$  and its knowledge about the protocols  $A'_1, \dots, A'_l, A'_{l+1}$  to generate a response to the original query. The most important idea of this section is that during the interaction between  $\mathcal{P}_T$  and the composed protocol,  $\mathcal{P}_S$  interacts with each instance  $A_i$  to resume the logical executions of the NIIS protocols. Since the prover  $\mathcal{P}_T$  is going to win against the composed protocol, there must be something wrong within the workflow. This can only happen in interactions between  $\mathcal{P}_S$  and some copy of  $A$ , which means that  $\mathcal{P}_S$  wins in the interaction with some copy.

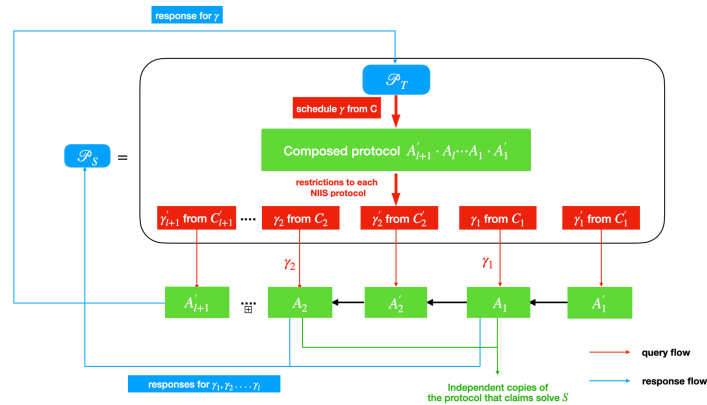


Figure 1 Simulation workflow.

## 4.2 Multiple-instance extension-based proofs

The prover  $\mathcal{P}_S$  will independently interact with a constant number of instances of the protocol  $A$ . Note that we are using a different definition of prover here. In the standard definition, an extension-based prover interacts with a single instance of the protocol. A multiple-instance extension-based prover interacts with multiple instances of the same protocol  $A$  and maintains a separate set of arguments (i.e. all arguments an extension-based prover will maintain during its interaction with a protocol) recording its interaction with each instance of  $A$ . For example,  $\phi_i$  and  $\phi_j$ , representing the current phases of interactions with  $A_i$  and  $A_j$  can be different.

A multiple-instances prover submits queries to each instance individually. It can use the response from any query to decide what to do next (such as how to submit a query in the interaction with another instance of  $A$ ). But we do not allow a multiple-instance prover to use responses from different instances to produce a conflict even when the protocol  $A$  responds with different output values for a single schedule in its interaction with different instances. A multiple-instance prover wins if it can win in the interaction with some instance using only the information obtained from this instance. Otherwise, the protocol wins. It is easy to see that a multiple-instance extension-based prover is at least as powerful as an extension-based prover since a multiple-instance extension-based prover can choose to interact with one instance only.

## 4.3 Construction of $\mathcal{P}_S$

Let  $\gamma$  be a schedule of the composed protocol  $A'_{l+1} \circ A_l \circ A'_l \cdots A_1 \circ A'_1$  from some initial configuration  $C_{ini}$  to some configuration  $C$ . The restriction of  $\gamma$  to a protocol  $A'_i$  or  $A_i$  is defined in Section 3.3, denoted by  $\gamma'_i$  and  $\gamma_i$ . By Lemma 3, given a schedule  $\gamma$  of the composed protocol, the prover  $\mathcal{P}_S$  can compute the restriction of  $\gamma$  to any  $A'_i$  or  $A_i$ .

An initial configuration  $C_i$  of  $A_i$  is defined as compatible with some configuration  $C'_i \gamma'_i$  of  $A'_i$  if the output value of every process terminated in  $C'_i \gamma'_i$  is the input value of the same process in  $C_i$ . An initial configuration  $C'_{i+1}$  of  $A'_{i+1}$  is defined to be compatible with some configuration  $C_i \gamma_i$  of  $A_i$  if the output value of every process terminated in  $C_i \gamma_i$  is the input value of the same process in  $C'_{i+1}$ .

In the simulation,  $\mathcal{P}_S$  maintains the following invariant. Suppose that  $\mathcal{P}_T$  has reached the configuration  $C = C_{ini} \gamma$  of the composed protocol  $A'_{l+1} \circ A_l \circ A'_l \cdots A_1 \circ A'_1$ . Then there exist initial configurations  $C'_1, C_1, C'_2, \dots, C_l, C'_{l+1}$  of the protocols  $A'_1, A_1, A'_2, \dots, A_l, A'_{l+1}$  such that, for all  $1 \leq i \leq l$ ,  $C_i$  is compatible with  $C'_i \gamma'_i$ ,  $C'_{i+1}$  is compatible with  $C_i \gamma_i$ , and  $\mathcal{P}_S$  has reached configuration  $C_i \gamma_i$  in its interaction with  $A_i$ . Let  $\psi$  denote the current phase of the simulated interaction, and let  $\phi_i$  denote the current phase of  $A_i$  for each  $1 \leq i \leq l$ .  $\alpha_i(\phi_i)$  is a prefix of  $\gamma_i$  for each  $1 \leq i \leq l$ .

At the beginning of the simulation,  $\mathcal{P}_T$  starts with an empty schedule and has reached the initial configurations of the composed protocol  $A'_{l+1} \circ A_l \circ A'_l \cdots A_1 \circ A'_1$ . For each  $1 \leq i \leq l$ , the interaction between  $\mathcal{P}_S$  and  $A_i$  also starts with an empty schedule and has reached the initial configurations of  $A_i$ . Since every initial configuration of  $A'_i$  is compatible with every initial configuration of  $A_i$ , the invariant holds at the start of phase 1 of the simulated interaction. We will discuss the strategy that the prover  $\mathcal{P}_S$  will use in interactions with these instances of  $A$  when  $\mathcal{P}_T$  submits a query, submits an assignment query, or ends a phase in the simulation.

### 4.3.1 Responding to queries

Suppose that the prover  $\mathcal{P}_T$  submits a query  $(C, U)$  in phase  $\psi$  of the simulated interaction, where  $C$  is some configuration and  $U$  is a set of processes that are poised to access the same IS object. The prover  $\mathcal{P}_S$  must respond with the  $\delta$  values of the processes in  $U$  in the configuration  $C\{U\}$  resulting from scheduling  $U$  from  $C$  in the composed protocol. We can express the configuration  $C$  as  $C_{ini}\alpha(\psi)\beta$  where  $C_{ini}$  is an initial configuration since the configuration  $C$  has been reached in phase  $\psi$ . Let  $\gamma = \alpha(\psi)\beta$ . The invariant holds before the query: Each interaction instance has reached some configuration  $C_i\gamma_i$  where  $C_i$  is an initial configuration of  $A_i$  compatible with  $C'_i\gamma'_i$ . And  $C'_{i+1}$  is compatible with  $C_i\gamma_i$ .

The processes in  $U$  in the configuration  $C$  of the composed protocol  $A'_{l+1} \circ A_l \circ A'_l \cdots A_1 \circ A'_1$  can be simulating different protocols as described in Section 3.1. For each process  $u$  in  $U$ , the NIIS protocol that  $u$  is simulating can be calculated. Therefore, we can separate the processes in  $U$  into groups  $\{U'_1, U_1, \dots, U'_l, U_l, U_{l+1}\}$ . For example,  $U'_1$  consists of the processes that simulate  $A'_1$ . If some process  $u$  in some group  $U_j$  or  $U'_j$  finalizes its current estimate vector (or is simulating the first protocol  $A'_1$ ) after accessing the new IS object,  $u$  takes a step in its current simulated NIIS protocol. Otherwise, it will not take a step in its current simulation. Let  $V'_i$  and  $V_i$  be the set of processes that take a step in the simulated protocol  $A'_i$  and  $A_i$ , respectively. It is easy to see that  $V_i \subseteq U_i$  and  $V'_i \subseteq U'_i$ . Let  $V'_{i,k}$  be the processes in  $V'_i$  that are poised to access the  $k$ -th simulated IS object. And with the same definition, we have  $V_{i,k}$ . There are several cases, depending on the steps that the processes in  $V'_{i,k}$  or  $V_{i,k}$  take in the simulation.

If processes in  $V'_{i,k}$  simulate some  $A'_i$  and do not execute the last step, they still simulate  $A'_i$  after taking a step. Let  $C'_{res}$  be the configuration resulting from scheduling  $V'_{i,k}$  from  $C'_i\gamma'_i$ . No more processes terminate in the simulated execution of  $A'_i$ , and  $C_i$  is compatible with  $C'_{res}$ . The invariant continues to hold.  $\mathcal{P}_S$  returns the states of  $V'_{i,k}$  in  $C'_{res}$  to  $\mathcal{P}_T$ .

If processes in  $V'_{i,k}$  simulate some  $A'_i$  and execute the last step, each process will output a value and terminate in the simulation of  $A'_i$  since processes terminate after accessing the same number of IS objects in any execution of  $A'_i$  (argued in the beginning of Section 3). Let  $C'_{res}$  be the configuration resulting from scheduling  $V'_{i,k}$  from  $C'_i\gamma'_i$ .  $\mathcal{P}_S$  returns the states of  $V'_{i,k}$  in  $C'_{res}$  to  $\mathcal{P}_T$ .  $C_i$  has to be changed since more processes have been terminated in the simulation of  $A'_i$ . We can choose a new initial configuration  $C_i$  of  $A_i$  in which each newly terminated process  $p$  will use the output of  $A'_i$  as input and every other process has the same input as the original  $C_i$ . The new initial configuration  $C_i$  is compatible with  $C'_{res}$ . The newly terminated processes do not appear in  $\gamma_i$ , so  $\gamma_i$  is a valid schedule from the new initial configuration  $C_i$ . Invariant holds in this case.

Suppose that the processes in  $V_{i,k}$  simulate some  $A_i$ . The prover  $\mathcal{P}_S$  can submit the query  $(C_i\gamma_i, V_{i,k})$  to the protocol  $A_i$  and return a response to  $\mathcal{P}_T$ . If no process terminates in the simulation, the invariant holds, since  $\gamma_i\{V_{i,k}\}$  is the new restriction to  $A_i$  and will not change the initial values of active processes in the simulation of  $A_i$  or  $A'_{i+1}$ . Otherwise, some processes in  $V_{i,k}$  terminate in the simulation of  $A_i$ , and the initial configuration  $C'_{i+1}$  of  $A'_{i+1}$  must be changed. For each terminated process  $u$  in  $V_{i,k}$ , the input of  $A'_{i+1}$  is the output of  $A_i$ . Let  $C_{res}$  be the configuration resulting from scheduling  $V_{i,k}$  from  $C_i\gamma_i$ . The new initial configuration  $C'_{i+1}$  of  $A'_{i+1}$  is compatible with  $C_{res}$ . The invariant still holds.

The prover  $\mathcal{P}_S$  may submit multiple queries  $(C_i\gamma_i, V_{i,k})$  for different  $k$  to some protocol  $A_i$ . Operations on different simulated IS objects will not affect each other. Therefore, submitting these queries in different orders will not change the response of each query and the internal variables (such as the set of configurations reached in phase  $\phi_i$ ) of the interaction between  $\mathcal{P}_S$  and  $A_i$ . Therefore, the invariant holds after all these queries.

### 4.3.2 Responding to assignment queries

Suppose that the prover  $\mathcal{P}_T$  submits an assignment query  $(C, U, f)$  in phase  $\psi$  of the simulated interaction, where  $U$  is a set of processes and  $f$  is a function from a subset  $U'$  of  $U$  to the output values. The prover  $\mathcal{P}_S$  has to reply to  $\mathcal{P}_T$  whether there exists a  $U$ -only schedule from  $C$  to a configuration in which the output value of each process  $u$  in  $U'$  is  $f(u)$ . We will say that this resulting configuration *satisfies the requirement of  $f$* . Since  $\mathcal{P}_T$  will not reach any new configuration to respond to assignment queries, the invariant will continue to hold.

Let  $\gamma$  be the schedule from some initial configuration to  $C$ . By the invariant, for  $1 \leq i \leq l$ ,  $\mathcal{P}_S$  has reached the configuration  $C_i\gamma_i$  in the  $i$ -th interaction with  $A$ , for some initial configuration  $C_i$  of protocol  $A_i$  compatible with  $C'_i\gamma'_i$ , where  $\gamma_i$  and  $\gamma'_i$  are restrictions to  $A_i$  and  $A'_i$ . And  $C'_{i+1}$  is compatible with  $C_i\gamma_i$ .

The prover  $\mathcal{P}_S$  will treat all protocols  $A'_i$  and  $A_i$  as black boxes that allow assignment queries. This is possible for each  $A_i$  since the prover  $\mathcal{P}_S$  can submit assignment queries to protocol  $A_i$ . The prover  $\mathcal{P}_S$  knows everything about the protocol  $A'_i$ , so assignment queries are also possible for  $A'_i$ . We regard protocols of both types as black boxes to simplify our proof. At first, the prover  $\mathcal{P}_S$  computes a set of assignment queries to  $A_l \circ A'_l \cdots A_1 \circ A'_1$  (note that  $A'_{l+1}$  does not appear), denoted by  $F_{2l}$ . The prover  $\mathcal{P}_S$  repeats this computation by removing the last black box until only one black box remains. We define  $F_{2l+1}$  as the original query  $(C, U, f)$ .

First,  $\mathcal{P}_S$  computes  $F_{2l}$ , which is the set of assignment queries to  $A_l \circ A'_l \cdots A_1 \circ A'_1$ . In the following discussions, we emphasize that schedules for different protocols are used. Whenever we talk about a schedule, we explicitly mention the protocol. Let  $C_{2l+1}^c$  be some initial configuration of the protocol  $A'_{l+1}$  compatible with  $C_l\gamma_l$ , where there is a  $U$ -only schedule  $\beta$  of  $A'_{l+1}$  from  $C_{2l+1}^c\gamma'_{l+1}$  to a configuration that satisfies the requirement of  $f$ . Let  $U'$  denote the set of processes obtained from  $U$  by removing the processes that appear in  $\gamma'_{l+1}$ .

Let  $C'$  be the configuration of the composed protocol  $A_l \circ A'_l \cdots A_1 \circ A'_1$  (note that  $A'_{l+1}$  is removed here), where the processes in  $\Pi - (U - U')$  are in the same state as the configuration  $C$  and the processes in  $U - U'$  have terminated with their input values in  $C'_{l+1}$ . Intuitively, suppose that the configuration  $C$  is reached from some initial configuration  $C_{ini}$  by a schedule  $\gamma$ , then  $C'$  is also reached from  $C_{ini}$  by  $\gamma$ , where terminated processes omit the additional steps assigned by the schedule. Now we will prove that there exists a  $U'$ -only schedule of the composed protocol  $A_l \circ A'_l \cdots A_1 \circ A'_1$  from  $C'$  to some configuration where the output values of  $U'$  are the input values of  $C_{2l+1}^c$  if and only if there exists a  $U$ -only schedule of  $A'_{l+1} \circ A_l \circ A'_l \cdots A_1 \circ A'_1$  from  $C$  to some configuration whose output values satisfy  $f$ .

Suppose that there is a  $U'$ -only schedule  $\beta'$  of  $A_l \cdots A'_1$  from  $C'$  to a configuration whose output values are the input values of  $C_{2l+1}^c$ .  $\beta'$  can be used directly as a schedule of  $A'_{l+1} \circ A_l \cdots A'_1$ . We cannot concatenate the schedule  $\beta'$  and  $\beta$ , since  $\beta$  (a schedule of the protocol  $A'_{l+1}$ ) cannot be used directly as a schedule of  $A'_{l+1} \circ A_l \cdots A'_1$ . However, we can construct a  $U$ -only schedule  $\beta_c$  of  $A'_{l+1} \cdots A_l \circ A'_l \cdots A_1 \circ A'_1$  whose restriction to  $A'_{l+1}$  is  $\gamma'_{l+1}\beta$ . Let  $\beta = \{U_1\}\{U_2\} \cdots \{U_e\}$ , then  $\beta_c$  is defined as  $\beta' \{U_1\} \cdots \{U_1\}\{U_2\} \cdots \{U_2\}\{U_e\} \cdots \{U_e\}$  such that for each set of processes  $U_i$  in  $\beta$ , processes in  $U_i$  concurrently access IS objects until they all finalize their  $i$ -th estimate vectors when they are poised to access the  $i$ -th IS object in the simulation of  $A'_{l+1}$ . Their estimate vectors are the same according to the checks performed when an estimate vector is finalized.  $\beta'\beta_c$  is a  $U$ -only schedule from  $C$  to a configuration that satisfies the requirement of the function  $f$ .

If there is a  $U$ -only schedule  $\beta''$  from  $C$  to some configuration  $C_d$  that satisfies the requirement of  $f$ ,  $C_d$  can be seen as an output configuration of the last protocol  $A'_{l+1}$ . Therefore, there exists an initial configuration  $C_{2l+1}^c$  of the protocol  $A'_{l+1}$  compatible with  $C_l\gamma_l$ , and there is a  $U$ -only schedule  $\beta$  of  $A'_{l+1}$  from  $C'_{l+1}\gamma'_{l+1}$  to  $C_d$ . Consider the  $U$ -only schedule  $\beta'' = c_1 \circ c_2 \cdots c_m$  from  $C$  to  $C_d$  of the composed protocol, where each  $c_i$  is a set of processes. We will construct a  $U'$ -only schedule based on  $\beta''$  by stopping the processes that simulate  $A'_{l+1}$  from taking actions. Let  $E = e_1 \circ e_2 \cdots e_m$  be a sequence, where  $e_i$  is the set of processes that simulate  $A'_{l+1}$  in the configuration reached from  $C$  by the schedule  $c_1 \circ c_2 \cdots c_i$ . The processes in  $U - U'$  already simulate the protocol  $A'_{l+1}$ , so  $U - U' \subseteq e_1 \subseteq e_2 \cdots e_m$ . The schedule  $c_1 - e_1, c_2 - e_2 \cdots c_m - e_m$  is a  $U'$ -only schedule of the composed protocol. Deleting the processes in  $e_i$  will not change the estimate vectors finalized by processes that do not simulate  $A'_{l+1}$ , since the processes in  $e_i$  write only information about the simulation of  $A'_{l+1}$ , which will not affect the simulation of previous NIIS protocols. Therefore,  $c_1 - e_1, c_2 - e_2 \cdots c_m - e_m$  is a  $U'$ -only schedule from  $C'$  to a configuration where the output values are the input values of  $C_{2l+1}^c$ .

For each such initial configuration  $C_{2l+1}^c$ , we can submit an assignment query  $(C', U', f')$  to the composed protocol  $A_l \circ A'_l \cdots A_1 \circ A'_1$  in which  $f'$  is defined as a function from the processes in  $U'$  to their output values in configuration  $C_{2l+1}^c$ . If a  $U'$ -only schedule is returned, there is a  $U$ -only schedule for the assignment query  $(C, U, f)$  to  $A'_{l+1} \circ A_l \circ A'_l \cdots A_1 \circ A'_1$ . Otherwise, the prover  $\mathcal{P}_T$  receives the response  $NULL$ .  $F_{2l}$  consists of all these assignment queries  $(C', U', f')$ .

We continue to trace back: Suppose that  $F_{i+1}$  is already calculated. For each assignment query  $(C', U', f')$  in  $F_{i+1}$ , the prover  $\mathcal{P}_S$  computes each initial configuration  $C_{i+1}^c$ .  $C_{i+1}^c$  has two requirements. It must be compatible with  $C_e\gamma_e$  where  $C_e$  is the initial configuration (some  $C_j$  or  $C'_j$  in the invariant) of the  $i$ -th black box and  $\gamma_e$  is the restriction (some  $\gamma_j$  or  $\gamma'_j$  in the invariant) to this black box, and there must exist a  $U'$ -only schedule  $\beta$  of the  $(i+1)$ -th black box from  $C_{i+1}^c\gamma_{e+1}$  to a configuration that satisfies the function  $f'$ , where  $\gamma_{e+1}$  is the restriction of  $\gamma$  to the  $(i+1)$ -th black box. To decide whether such a  $U'$ -only schedule  $\beta$  exists, the prover  $\mathcal{P}_S$  can submit assignment queries to the  $(i+1)$ -th black box. Let  $U''$  denote the set of processes obtained from  $U'$  by removing the processes in  $\gamma_{e+1}$ . For each initial configuration  $C_{i+1}^c$ , there is an assignment query  $(C'', U'', f'')$  in which  $f''$  is defined as a function from the processes in  $U''$  to the values in the configuration  $C_{i+1}^c$ . As proved in the above example, there is a  $U''$ -only schedule from  $C''$  to a configuration whose output values are the input values of  $C_{i+1}^c$  if and only if there is a  $U'$ -only schedule that can serve as a response to the assignment query  $(C', U', f')$ .  $F_i$  consists of all these assignment queries  $(C'', U'', f'')$ .

This procedure ends when  $F_0$  is computed. If  $F_0$  is not empty, there exists a  $U$ -only schedule from  $C$  to some configuration that satisfies the requirement of  $f$ . Otherwise, the prover  $\mathcal{P}_T$  gets a response of  $NULL$ .

### 4.3.3 Ending phases

Suppose that the prover  $\mathcal{P}_T$  decides to end its current phase  $\psi$  by choosing a configuration  $C = C_{ini}\gamma$  of the composed protocol  $A'_{l+1} \circ A_l \circ A'_l \cdots A_1 \circ A'_1$ . By the invariant,  $\mathcal{P}_S$  has reached configuration  $C_i\gamma_i$  in the  $i$ -th interaction with protocol  $A$ , for some initial configuration  $C_i$  of  $A_i$ .

For  $A_i$  that does not have an interaction with  $\mathcal{P}_S$  during the phase  $\psi$  of the simulation, the prover  $\mathcal{P}_S$  does nothing. For a protocol instance  $A_i$  in which at least one process has taken a simulation step during the schedule  $\gamma$  of the composed protocol, the prover  $\mathcal{P}_T$  has reached the configuration  $C_i\gamma_i$ . The prover  $\mathcal{P}_S$  ends phase  $\phi_i$  by choosing configuration  $C_i\gamma_i$ , sets  $\alpha_i(\phi_i + 1) = \gamma_i$  and proceeds to the next phase in the interaction with  $A_i$ .

#### 4.3.4 The prover $\mathcal{P}_S$ will win

The prover  $\mathcal{P}_T$  will finally win in the simulated interaction with the composed protocol  $A'_{l+1} \circ A_l \circ A'_l \cdots A_1 \circ A'_1$ . We will analyze all possible circumstances and show that  $\mathcal{P}_S$  will win against one of the instances of the protocol  $A$ .

The first case is that the prover  $\mathcal{P}_T$  finds a configuration,  $C$ , in which the output values violate the specification of the task  $\mathcal{T}$ . For example, the processes in  $C$  output different values when  $\mathcal{T}$  is the consensus task. By the invariant, the prover  $\mathcal{P}_S$  has reached configuration  $C_i\gamma_i$  in the interaction with  $A_i$ . Suppose that for each interaction instance, the output values of the processes in  $C_i\gamma_i$  satisfy the task specification of  $\mathcal{S}$ . The output values in configuration  $C$  will satisfy the task specification of  $\mathcal{T}$  as  $\mathcal{T}$  reduces to  $\mathcal{S}$ , which is a contradiction. Therefore, there exists an interaction with some  $A_i$  that reaches a configuration with incorrect output values. The prover  $\mathcal{P}_S$  finds a violation of the task specification and wins.

The second case is that the prover  $\mathcal{P}_T$  constructs an infinite chain of queries in the interaction. Each protocol  $A'_i$  is wait-free, so an infinite schedule is not possible in it. Estimation vectors will not cause an infinite schedule of the composed protocol, as proved in [10]. Therefore, there must be an interaction with some  $A$  instance in which the prover  $\mathcal{P}_S$  also constructs an infinite chain of queries. In this situation, the prover  $\mathcal{P}_S$  wins.

The last case is that the prover  $\mathcal{P}_T$  constructs infinite phases in the simulation. By definition,  $\mathcal{P}_T$  can submit finitely many queries or assignment queries in one phase. Each time the prover  $\mathcal{P}_T$  ends its phase by choosing a schedule, at least one step is taken by some set of processes. Since each protocol  $A'_i$  is wait-free, the prover  $\mathcal{P}_T$  can only end a finite number of phases by simulating only these protocols. For other phases, there must be some protocol  $A_i$  that will make progress and, therefore, end its current phase after the prover  $\mathcal{P}_T$  ends a finite number of phases (because an estimate vector can take several rounds to finalize). The number of  $A$  protocol instances is finite, so there must be an interaction with some  $A$  instance in which the prover  $\mathcal{P}_S$  also constructs infinite phases. The prover  $\mathcal{P}_S$  still wins.

► **Theorem 4.** *If a task  $\mathcal{T}$  reduces to a task  $\mathcal{S}$  using multiple instances of task  $\mathcal{S}$  and there is an augmented extension-based proof of the impossibility of solving the task  $\mathcal{T}$  in a wait-free manner in the NIIS model, then there is an multiple-instance extension-based proof of the impossibility of solving the task  $\mathcal{S}$  in a wait-free manner in the NIIS model.*

## 5 Conclusions

In this paper, we partially solve an open question proposed in [10]. If there is a reduction using multiple instances of  $\mathcal{S}$  from  $\mathcal{T}$  to  $\mathcal{S}$  and there is no multiple-instance extension-based proof for  $\mathcal{S}$ , then there is no augmented extension-based proof for  $\mathcal{T}$ . Although many reductions that appear in the literature reduce to a single instance of a problem, our result applies to reductions that reduce to a constant number of different instances of that problem. An open problem that remains is to extend our results to more general reductions (such as Turing reductions). Another open problem is whether multiple-instance extension-based provers are strictly more powerful than extension-based provers.

## References

- 1 Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4):873–890, September 1993. doi:10.1145/153724.153741.
- 2 Dan Alistarh, James Aspnes, Faith Ellen, Rati Gelashvili, and Leqi Zhu. Why extension-based proofs fail. *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 986–996, 2019. doi:10.1145/3313276.3316407.
- 3 Dan Alistarh, James Aspnes, Faith Ellen, Rati Gelashvili, and Leqi Zhu. Why extension-based proofs fail. *SIAM Journal on Computing*, 52(4):913–944, 2023. doi:10.1137/20M1375851.
- 4 Dan Alistarh, Faith Ellen, and Joel Rybicki. Wait-free approximate agreement on graphs. In *Structural Information and Communication Complexity: 28th International Colloquium, SIROCCO 2021, Wrocław, Poland, June 28 – July 1, 2021, Proceedings*, pages 87–105, Berlin, Heidelberg, 2021. Springer-Verlag. doi:10.1007/978-3-030-79527-6\_6.
- 5 Hagit Attiya, Armando Castañeda, and Sergio Rajsbaum. Locally solvable tasks and the limitations of valency arguments. *Journal of Parallel and Distributed Computing*, 176:28–40, 2023. doi:10.1016/j.jpdc.2023.02.002.
- 6 Hagit Attiya, Pierre Fraigniaud, Ami Paz, and Sergio Rajsbaum. One Step Forward, One Step Back: FLP-Style Proofs and the Round-Reduction Technique for Colorless Tasks. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing (DISC 2023)*, volume 281 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:23, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2023.4.
- 7 Elizabeth Borowsky and Eli Gafni. Generalized flip impossibility result for t-resilient asynchronous computations. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 91–100, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/167088.167119.
- 8 Elizabeth Borowsky and Eli Gafni. Immediate atomic snapshots and fast renaming. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, PODC '93, pages 41–51, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/164051.164056.
- 9 Elizabeth Borowsky and Eli Gafni. A simple algorithmically reasoned characterization of wait-free computation (extended abstract). In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '97, pages 189–198, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/259380.259439.
- 10 Kayman Brusse and Faith Ellen. Reductions and extension-based proofs. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, pages 497–507, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3465084.3467906.
- 11 Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Inf. Comput.*, 105(1):132–158, July 1993. doi:10.1006/inco.1993.1043.
- 12 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985. doi:10.1145/3149.214121.
- 13 Eli Gafni, Sergio Rajsbaum, and Maurice Herlihy. Subconsensus tasks: Renaming is weaker than set agreement. In *Proceedings of the 20th International Conference on Distributed Computing*, DISC'06, pages 329–338, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11864219\_23.
- 14 Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, November 1999. doi:10.1145/331524.331529.
- 15 Gunnar Hoest and Nir Shavit. Towards a topological characterization of asynchronous complexity. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '97, pages 199–208, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/259380.259440.



- 16 Shihao Liu. The Impossibility of Approximate Agreement on a Larger Class of Graphs. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems (OPODIS 2022)*, volume 253 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.OPODIS.2022.22.
- 17 Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, March 2000. doi:10.1137/S0097539796307698.
- 18 Yusong Shi and Weidong Liu. Colorless tasks and extension-based proofs, 2024. arXiv:2303.14769.