Equivalence of Round-Based and Non-Round-Based Message-Passing Models

Dhrubajyoti Ghosh, supervised by Thomas Nowak, Laboratoire Méthodes Formelles (LMF)

August 20, 2025

General context

We study the relative power of message-passing models of computation in distributed computing systems with complete networks, i.e., all processes can send each other messages. Most distributed algorithms for asynchronous message-passing models construct a round structure. If f out of n processes can fail by crashing, then each process waits for messages from all but f other processes before advancing to the next round. Due to the ubiquity of round-based algorithms, several round-based models for analyzing asynchronous systems have been proposed. In these models, processes take steps synchronously, but up to f incoming messages can be dropped at each process in each round. The question of whether round-based message-passing models are equivalent to the original asynchronous models in terms of solving decision tasks has mostly remained open. This is because while the above method of implementing synchronous systems in asynchronous systems is commonly used, the converse direction has not received as much attention.

In [1], Afek and Gafni study round-based models that are equivalent to the read-write wait-free model, which is a shared memory model. In [10], Gafni and Losa consider the problem of comparing round-based and asynchronous message passing models for the case of at most one fault, restricting themselves to colorless decision tasks, and show that it is possible. We attempt to generalize this result.

Research problem

We study whether there is a round-based message-passing model that is equivalent to the Asynchronous Message Passing (AMP) model with f process failures, i.e., whether they solve the same set of decision tasks. One of our candidate round-based models is the Heard-Of (HO) model, which is the focus of this work.

This is equivalent to asking whether the above approach of constructing a virtual round-based system in AMP in order to solve tasks is *always* possible. Round-based models are often easier to analyze, for designing and analyzing algorithms as well as for proving lower bounds and impossibility results. Hence an affirmative answer would ease our work in dealing with the AMP model considerably.

Another motivation for this problem is the following. A question posed by our research group is whether it is possible for two models that are non-isomorphic in the longest-common-prefix topology [2] to solve the same set of decision tasks. As the AMP model with process failures is non-compact and the HO model is compact, these are reasonable candidates for our work.

Our problem is not new: the question of whether round-based models are equivalent to models where late messages are not discarded has for example been raised by Charron-Bost and Schiper in [5]. Gafni and Losa [10] have studied this problem for colorless tasks with at most one process failure.

I chose this problem because it appeared to be a fundamental unanswered question. It also constitutes an important first step towards the comparison of models that are non-isomorphic as topological spaces.

My contribution

I have comprehensively answered how the AMP model with f process failures and the HO model agree or differ on the classes of colorless and colored tasks (any task that is not colorless) for $f < n/2$. They solve the same set of colorless tasks for $f < n/2$, while only solving the same set of colored tasks for $f = 1$. If $1 < f < n/2$, they do not solve the same set of colored tasks.

In order to obtain the first two results, I first defined a variant of the HO model, called SFHO. Using existing techniques, I showed that the set of tasks solvable in AMP lies between that solvable in HO and that solvable in SFHO. The next step was to find the conditions under which HO and SFHO coincide. The difference between the HO and SFHO models is the existence of a so-called silenced process which is not obliged to produce an output in SFHO but is obliged to do so in HO. Depending on the value of f and whether the task is colored or colorless, one can guarantee that a silenced process can produce an output.

The last case with colored tasks and $1 < f < n/2$ presented a considerable challenge. I tried to extend our approach in the case of colored tasks with $f = 1$ but without any success. I then tried to search for colored tasks that would be solvable in one model but not the other. Eventually, I found the *renaming task* [3] as a possible candidate which was known to be solvable in AMP. I was then able to prove that it is not solvable in HO for $f > 1$. This was fortunate as not many tasks are known that are solvable in AMP.

Arguments supporting their validity

We establish the validity of our results through formal proofs.

Some assumptions had to be made on the protocols that are considered in the context of the renaming task (Section 8). These are reasonable assumptions, and the same as those used in the original paper [3] defining the task. Without these assumptions, the renaming task becomes trivial in any model.

Summary and future work

I was able to extend the result of Gafni and Losa [10] which only considered the case of colorless tasks for $f = 1$ to the case $f < n/2$. Moreover I was able to answer the question for colored tasks for all $f < n/2$. I showed that the AMP and HO models coincide in the case of colorless tasks for $f < n/2$ and in the case of colored tasks for $f = 1$. I also showed that this is not true in the case of colored tasks where $1 < f < n/2$. We believe this to be the first instance where a task solvable in the AMP model has been shown to be not solvable in its round-based counterpart. This implies that the approach of constructing a virtual round-based system in AMP cannot always be used to solve tasks in AMP. My result for the case $f = 1$ also implies that the solvability of decision tasks cannot be used to distinguish models that are non-isomorphic as topological spaces.

The next step is to consider the case where the number of failures is at least half the number of processes, i.e., $f \geq n/2$. We also introduce an intermediary round-based model called the Silenced-Faulty Heard-Of model (SFHO) that coincides with AMP in the same cases where HO does so with AMP. However, it is not clear whether SFHO and AMP agree or differ in the case of colored tasks for $1 < f < n/2$ and thus can be studied in future work.

1 Introduction

We consider message-passing distributed systems consisting of a complete network of $n \geq 1$ processes where each process can send a message to another process including itself.

The *asynchronous message-passing* model [8] with at most f faults (AMP_f) has no restrictions on message delays between processes, and at most f processes may stop functioning. One approach to solving problems is to have each process progress in rounds where in each round it sends a message and waits to receive at least $n - f$ messages before proceeding to the next round. It does not wait to receive messages from the remaining at most f processes for it cannot be sure if they have stopped.

This leads us to consider synchronous round-based models such as the *Heard-Of model* [5] with f faults (HO_f) where in each round, every process may fail to receive at most f messages from other processes. Instead of having processes that fail, we only allow the loss of messages. Round-based models are convenient for designing algorithms and proving impossibility results, in part due to the fact that processes do not have to wait for unreceived messages in any round.

The class of problems that we are interested in solving is that of *decision tasks*, or simply *tasks*. A protocol solving a task in a model must ensure that for every combination of inputs that processes can start their computation with, the combination of their outputs is valid according to the task specification. The class of decision tasks can be split into two subclasses: *colorless tasks* and *colored tasks*.

We ask whether the above approach of constructing a virtual round-based system in AMP_f can *always* be used to solve tasks in AMP_f, or equivalently, whether AMP_f and HO_f solve the same set of tasks.

The question of whether round-based message passing models are equivalent to the ones in which late messages are not discarded has previously been raised by Charron-Bost and Schiper [6]. To our knowledge, this problem has only been studied by Gafni and Losa [10], where it was shown for $n > 2$ and $f = 1$ that AMP_f and HO_f solve the same set of colorless tasks. In related work, the problem of finding a round-based message-passing model equivalent to the read-write wait-free model (a shared memory model) was studied by Afek and Gafni [1].

We generalize the result of [10] to show that it is in fact true for any $f < n/2$. Moreover, we answer the question for colored tasks in the case $f < n/2$, thus giving a comprehensive list of similarities and differences between the two models for $f < n/2$. Under a general definition of task solvability, we show that for $n > 2f$,

- AMP_f and HO_f solve the same set of colorless tasks (Theorem 7.4).
- AMP_f and HO_f solve the same set of colored tasks for $f = 0, 1$ (Theorem 7.12).

We finally show that the result deviates for the case of colored tasks with $1 < f < n/2$. In this case, we show that the *renaming* task [3], which is a colored task, is solvable in AMP_f but *not* in HO_f (Theorem 8.1); to our knowledge, this is the only task now known to be solvable in AMP_f but not in its round-based counterpart HO_f. It should be noted that the specification of the renaming task requires us to slightly restrict the definition of task solvability in order to rule out trivial algorithms.

The proof strategy for Theorems 7.4 and 7.12 is as follows. We use the method of constructing a virtual round-based structure in AMP_f to show that tasks solvable in HO_f are solvable in AMP_f. The converse direction is more involved. We introduce a variant of the HO_f model called SFHO_f, and show that tasks solvable in AMP_f are solvable in SFHO_f. Now it remains to find the conditions under which HO_f and SFHO_f coincide. The sole difference between the HO_f and SFHO_f models is the existence of a *silenced* process, which is not obliged to decide an output when solving a task in SFHO_f but is obliged to do so in HO_f. Depending on the value of f and whether the task is colored or colorless, we can guarantee that a silenced process will be able to decide an output.

1.1 Paper organisation

In Section 2, we describe the models of computation AMP_f and HO_f, and define decision tasks and task solvability. In Section 3, we briefly describe simulations. In Section 5, we introduce the SFHO_f

model. Sections 4 and 6 together establish an inclusion hierarchy among the sets of tasks solvable in models HO_f, AMP_f and SFHO_f. In Section 7, we study the conditions under which HO_f and SFHO_f, and consequently AMP_f, coincide. Finally in Section 8 we show when AMP_f and HO_f differ.

2 Models of computation

The definitions in this section have been taken from multiple sources [3, 4, 7, 8, 9].

In a message-passing model $\mathcal{M}$, each process follows a deterministic algorithm involving the sending and receiving of messages. Each process has a *message buffer* that holds the messages that have been sent to it but not yet received. Messages are sent to processes by adding them to their buffers. The exact specifications of the send and receive primitives depend on the model of computation.

A protocol $\mathcal{P}$ in a model $\mathcal{M}$ is a system of $n \geq 1$ processes $\mathcal{P} = \{p_1, \dots, p_n\}$ modeled as deterministic automata with infinite state spaces. We assume that each process has a unique identifier that is hard-coded within itself. A *configuration* of $\mathcal{P}$ consists of the state of each automaton and the contents of every message buffer. In an *initial configuration*, each automaton is in an initial state and each message buffer is empty. An *event* of $\mathcal{P}$ in $\mathcal{M}$ is of the form $op_i(v)$ where:

- p_i is the process associated with the event.
- op is the send or receive communication primitive in $\mathcal{M}$.
- If op is a send operation, then p_i sends value v and if it is a receive operation, then p_i reads value (or a set of values) v from the message buffer.

State transitions in automata are triggered by the occurrence of events at processes. An event is *enabled* in a configuration C if its process can do a state transition labeled with the event on its automaton based on its current state and message buffer. For a send event, the next state depends only on the current state, while for a receive event, the next state also depends on the value(s) read from the buffer (for some models, it is possible that no value is read).

Each model specifies a set of *allowable sequences* that constrains the possible inter-leavings of events of all the processes in a schedule. A *schedule* S of protocol $\mathcal{P}$ is a finite or infinite sequence of events of $\mathcal{P}$. A *run* of protocol $\mathcal{P}$ in model $\mathcal{M}$ is a tuple $R = (I, S)$ where I is an initial configuration of $\mathcal{P}$ and S is a schedule such that it is an allowable sequence in $\mathcal{M}$ and the events in it are enabled and applied in turn starting from I.

Every model designates a subset of processes in its allowable sequences as *faulty* processes. As will be seen later while defining decision task solvability, faulty processes are not required to decide an output during their computation.

It will sometimes be convenient to assume protocols to be *full-information*, where every process sends at first its initial state and in every subsequent send, the history of messages it sent and received until that point. We can assume moreover that the automaton of each process is hard-coded into every other process.

2.1 AMP_f model

In the *Asynchronous Message-Passing Model with f process failures* (AMP_f), there is no fixed upper bound on the time it takes for a message to be delivered nor on the relative speeds of the processors. The send and receive events in AMP_f have the following specifications:

(1) $\text{amp-send}_i(m)$: Process p_i broadcasts message m to all processes including itself by adding m to each of the message buffers.

(2) $\text{amp-recv}_i(m)$: Process p_i retrieves message m from the message buffer.

(3) $\text{amp-recv}_i(\perp)$: Process p_i tries to retrieve a message from its buffer but receives nothing. This corresponds to the situation where messages placed in its buffer are still “in transit” and have not yet arrived.

The allowable sequences in AMP_f satisfy:

(1) Every amp-recv event is followed immediately by *at most one* amp-send event.

A process which has infinitely many events in the sequence is said to be *non-faulty*, and *faulty* otherwise.

(2) There are at most f faulty processes.

(3) *Non-faulty Liveness:* Messages sent by a non-faulty process are received by all non-faulty processes.

(4) *Integrity:* Every message received by a process was previously sent to the process.

(5) *No duplicates:* No message is received more than once at any process.

2.2 HO_f model

The Heard-Of model was first formally defined in [5]. Here we consider one of its instances, the *Heard-Of model with f faults.* In this synchronous model, the execution proceeds in rounds. In each round, each process broadcasts a message, receives messages sent to it in that round and performs a local computation determining what message it will send in the next round. It is possible for some messages to get lost - if a process is not able to retrieve a message sent to it in some round, the message is lost forever. Each process can lose up to f messages in each round of HO_f. No process is defined to be faulty. We now describe the model formally.

The send and receive events in HO_f have the following specifications:

(1) $\text{ho-send}_i(m)$: Process p_i broadcasts message m to all processes including itself by adding m to each of the message buffers.

(2) $\text{ho-recv}_i(M)$: Process p_i retrieves a subset M of all the messages in its buffer received during a round.

Our notion of execution is a sequence of events, so although each round conceptually occurs concurrently at all processes, we can choose a total order on the events that is consistent with the order of events at each process. The allowable sequences in HO_f have the following structure and properties:

(1) *Round-robin property:* σ is infinite and consists of a series of subsequences, where the first subsequence has the form $\text{ho-send}_1, \ldots, \text{ho-send}_n$ and each latter subsequence has the form $\text{ho-recv}_1, \text{ho-send}_1, \ldots, \text{ho-recv}_n, \text{ho-send}_n$. The kth ho-send_i and kth ho-recv_i events form round k for process p_i.

(2) *Weak Liveness:* For every $\text{ho-recv}_i(M)$ event in σ, $|M| \geq n - f$ i.e. p_i misses at most f messages from other processes; it always receives its own message.

(3) *Integrity:* Every message received by process p_i from process p_j in round k was sent in round k by p_j.

(4) *No Duplicates:* Every receive event contains at most one message from each neighbor.

2.3 Decision tasks

In a run of a protocol, every process p_i has a non-$\perp$ *input* value as part of its initial state, and *decides* in special *output* registers that are part of their state, that are *write-once* and initialized with the value $\perp$ (meaning it has not decided yet). If a process does not decide, the value in its output register remains $\perp$.

A *task* is a tuple $(\mathcal{I}, \mathcal{O}, \Delta)$ where $\mathcal{I}$ is a set of input vectors (one input value for each process), $\mathcal{O}$ is a set of output vectors (one output value for each process), and $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$ is a total relation that associates each input vector with a set of possible output vectors. An output value of $\perp$ denotes an *undecided* process. We require that if $(I, O) \in \Delta$, then for each O' resulting after replacing some items in O with $\perp$, $(I, O') \in \Delta$.

In a *colorless* task, processes are free to copy the inputs and outputs of other processes, meaning we care only about relations between *sets* of input and output values instead of vectors. Formally, let $val(U)$ denote the *set* of non-$\perp$ values in a vector U. In a colorless task, if for all input vectors I, I' and all output vectors O, O' such that $(I, O) \in \Delta$, $val(I) \subseteq val(I')$ and $val(O') \subseteq val(O)$, we have $(I', O) \in \Delta$ and $(I, O') \in \Delta$.

A *colored task* is any task that is not colorless.

Example. An example of a colorless task is the *binary consensus* task. Every process has input value 0 or 1, and all processes that decide must have the same output value. Moreover, if all processes have the same input v, this output must be v.

If a process copies another process's output for its own, this does not violate the task conditions. We can define this task through relations on the sets of input and output values given to the processes:

$$\Delta = \{(\{0,1\}, \{0,\perp\}), (\{0,1\}, \{1,\perp\}), (\{0\}, \{0,\perp\}), (\{1\}, \{1,\perp\})\}.$$

Example. An example of a colored task is the *renaming* task. Every process has a distinct input value from an unbounded domain, and must decide an output from a smaller finite domain (of size at least n). The task requires that every process that decides must have a distinct output.

Contrary to the above example, if a process copies another process's output for its own, this would violate the task condition.

2.4 Solving a decision task

A protocol $\mathcal{P}$ *solves* T in model $\mathcal{M}$, if in every run R of $\mathcal{P}$ in $\mathcal{M}$ with input vector I and output vector O, we have (1) $(I, O) \in \Delta$, and (2) $O[i] = \perp$ only if p_i is faulty in $\mathcal{M}$ in R.

Task T is *solvable* in model $\mathcal{M}$ if there exists a protocol $\mathcal{P}$ that solves T in $\mathcal{M}$.

Remark 2.1. Since there are no process failures in HO_f, every process must decide in a run of a protocol in HO_f. Faulty processes in AMP_f on the other hand are not obliged to decide.

3 Simulations

A simulation system simulating model $\mathcal{M}_2$ in model $\mathcal{M}_1$ consists of three layers: (1) n *simulated processes* $p_1, \ldots, p_n$, (2) n *simulating machines* $P_1, \ldots, P_n$, with machine P_i assigned to simulated process p_i, and (3) the communication system of $\mathcal{M}_1$ (see Figure 1). Each simulated process p_i interacts with its simulating machine P_i via $\mathcal{M}_2$-communication primitives, as if the communication system is that of $\mathcal{M}_2$. The simulating machines interact with each other through $\mathcal{M}_1$ using $\mathcal{M}_1$-communication primitives. Transitions between states in simulated processes and simulating machines are triggered by the occurrence of events in $\mathcal{M}_1$ and $\mathcal{M}_2$. The occurrence of an event between p_i and P_i entails a transition in both p_i and P_i. A simulation is described by specifying the automaton for the simulating machines.

$$\begin{tikzcd}
\text{simulated process } p_i \arrow[d, shift right=6ex, "\mathcal{M}_2\text{-send}"'] \\
\text{simulating machine } P_i \arrow[u, shift right=6ex, "\mathcal{M}_2\text{-recv}"'] \arrow[d, shift right=6ex, "\mathcal{M}_1\text{-send}"'] \\
\text{communication system } \mathcal{M}_1 \arrow[u, shift right=6ex, "\mathcal{M}_1\text{-recv}"']
\end{tikzcd}$$

Figure 1: System simulating model $\mathcal{M}_2$ in $\mathcal{M}_1$

A *configuration* of the system consists of the states of all the simulated processes, simulating machines and their message buffers. In an *initial configuration*, every process and machine is in its initial state and every message buffer is empty. Enabled events and schedules are defined in the same manner as in the previous section. A *run* of a simulation consists of an initial configuration I and a schedule α such that the events in α are enabled in turn starting from I.

Given run α, we define $top(\alpha)$ by restricting the initial configuration of α to the simulated processes (and their buffers) and the schedule of α to the events of the top interface.

A detailed treatment of simulations can be found in Chapter 7 of [4].

4 Tasks solvable in HO_f are solvable in AMP_f

For this section, we assume that $0 \leq f \leq n$. We will use the method of constructing a virtual round-based structure in AMP_f to show that tasks solvable in HO_f are solvable in AMP_f. We first introduce an intermediary model called the *Crash-Faulty Heard-Of* model with f faults, or CFHO_f.

4.1 The CFHO_f model

While there is no notion of a faulty process in HO_f, we allow processes in CFHO_f to crash, i.e. eventually halt events, and define these processes to be faulty.

The communication primitives used in CFHO_f are the same as in HO_f, i.e. ho-send and ho-recv.

Allowable sequences in CFHO_f follow the same round-robin structure as in HO_f except that certain processes halt after a finite number of events. We say that these are *crashed* processes. If the last event of a crashed process corresponds to round r, we say that the process crashes in round r.

Definition 4.1. We say that a process is *faulty* in a sequence σ in CFHO_f if it crashes in σ and non-faulty otherwise.

The other property requirements, namely Integrity, No Duplicates and Weak Liveness remain the same as in HO_f.

4.2 The result

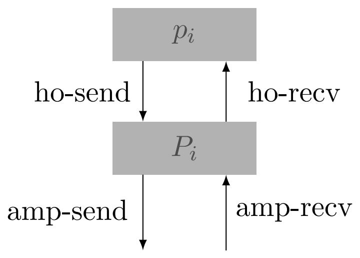
Let $0 \leq f \leq n$, and let $T = (\mathcal{I}, \mathcal{O}, \Delta)$ be a task that is solvable in CFHO_f by a protocol $\mathcal{P} = \{p_1, \ldots, p_n\}$. We assume without loss of generality that there is always a send event enabled after every receive event at each p_i; this is valid as $\mathcal{P}$ can be taken to be a full-information protocol for example.

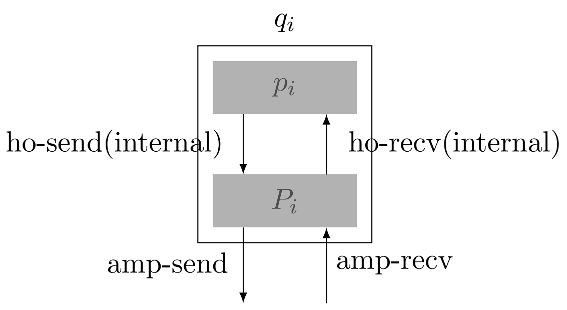
We can construct a simulation system that simulates CFHO_f in AMP_f so that for $1 \leq i \leq n$, (1) the i-th simulated process is p_i, and (2) the i-th simulation machine P_i has the following specification:

Algorithm 1 Pseudocode for machine S_i

```
1: Initialize counter $round \leftarrow 0$
2: Initialize empty lists $received[r]$ for all $r \geq 0$

3: When ho-send$_i(m)$ occurs:
4:     $round \leftarrow round + 1$
5:     Enable amp-send$_i(\langle m, round \rangle)$

6: When amp-recv$_i(\langle m, r \rangle)$ occurs:
7:     Add $m$ to $received[r]$

8: Enable ho-recv$_i(received[round])$ when:
9:     $|received[round]| \geq n - f$
```

To solve task T in AMP_f, a protocol $\mathcal{Q}$ can be constructed by converting the simulation system into a network of automata $\mathcal{Q} = \{q_1, \ldots, q_n\}$ such that q_i runs both p_i and the i-th simulation machine P_i





internally, and interacts with the communication system of AMP$_f$ via amp-send and amp-recv events. Events between p_i and P_i (namely ho-send and ho-recv) become part of q_i's internal computation.

The working of q_i can be seen as follows: The input of q_i is also used as the input for p_i. When p_i does an internal ho-send(m) for round r to P_i, q_i does an amp-send($\langle m, r\rangle$) and waits to receive at least $n - f$ messages of the form $\langle \cdot, r\rangle$. Once that happens, it performs an internal ho-recv(M) from P_i to p_i, where M consists of these messages with their round number tags removed, and changes p_i's state accordingly. If p_i internally decides an output o, then q_i decides o as well.

In order to prove that protocol $\mathcal{Q}$ solves T in AMP$_f$, consider any run γ of $\mathcal{Q}$ in AMP$_f$ with input vector I and output vector O. By considering the sequences of internal events during the local computations at each q_i as well as the events of γ, we can obtain the underlying simulation schedule for γ, which we call α.

Since each q_i in run γ uses the output of p_i in $top(\alpha)$ to decide, O is also the vector of outputs decided by the p_i in $top(\alpha)$ with input vector I. We will first show that $(I, O) \in \Delta$.

It is not necessary that $top(\alpha)$ has the round-robin pattern necessary for being allowable in CFHO$_f$. Due to the asynchronous nature of the actual communication system, machine P_i could receive $n - f$ round-r tagged messages much earlier than machine P_j does, meaning p_i could already do a round-$(r + 1)$ ho-send before p_j has a ho-recv for round r.

Consider σ obtained by rearranging the schedule of $top(\alpha)$ to have the round-robin pattern and such that the order of events at each p_i is preserved. Clearly O is still the vector of outputs decided by the p_i on σ with input vector I. We will show that σ is a run in CFHO$_f$, implying that $(I, O) \in \Delta$ as $\mathcal{P}$ solves T in CFHO$_f$.

Lemma 4.2. *σ satisfies Integrity, No Duplicates and Weak Liveness.*

Proof. Consider an event ho-recv$_i(M)$ in round r of σ. When simulating machine P_i enabled this event, its *round* variable had value r. So M is the set *received*$[r]$, which consists of messages in σ that were tagged with round number r, meaning that they must have been broadcast in round r in σ. This proves Integrity.

Since a process broadcasts at most one message every round and as γ satisfies Integrity in AMP$_f$, there cannot be any duplicated messages in ho-recv$_i$(*received*$[r]$). So σ satisfies the No Duplicates property.

Lastly, event ho-recv$_i$(*received*$[r]$) is enabled only when $|received[r]| \geq n - f$. Thus σ satisfies Weak Liveness since all messages in *received*$[r]$ are from distinct processes. □

Thus σ is a run in CFHO$_f$, implying that $(I, O) \in \Delta$ as $\mathcal{P}$ solves T in CFHO$_f$.

Lemma 4.3. *If q_i is non-faulty in γ then p_i does not crash in σ.*

Proof. Let $\mathcal{Q}_{NF}$ be the set of processes in $\mathcal{Q}$ that are non-faulty in γ.

Let $q_i \in \mathcal{Q}_{NF}$. In round 0, q_i does an internal ho-send from p_i and does a corresponding round-0 tagged amp-send. As $|\mathcal{Q}_{NF}| \geq n - f$, there are $n - f$ such amp-sends in γ. Thus q_i receives $n - f$ round-0 tagged messages in AMP$_f$ and enables a ho-recv from P_i to p_i; in turn, p_i enables a ho-send for the next round.

Thus for every $q_i \in \mathcal{Q}_{NF}$, process p_i does a ho-send for round 1. We can now continue our argument in an inductive fashion to conclude that p_i does not crash in σ. □

Since p_i decides an output if it does not crash in σ, it follows that q_i decides in γ if it is non-faulty. We conclude that protocol $\mathcal{Q}$ solves T in AMP_f, so:

Lemma 4.4. *Any task that is solvable in* CFHO_f *is also solvable in* AMP_f *for* $0 \leq f \leq n$.

Lemma 4.5. *Models* CFHO_f *and* HO_f *solve the same set of tasks for* $0 \leq f \leq n$.

Proof. Let $T = (\mathcal{I}, \mathcal{O}, \Delta)$ be any decision task.

As the set of allowable sequences of HO_f is a subset of that of CFHO_f, any protocol solving T in CFHO_f must also solve T in HO_f.

Conversely, suppose that there exists a protocol $\mathcal{P}$ that solves T in HO_f. We claim that $\mathcal{P}$ solves T in CFHO_f. Let α be a run of $\mathcal{P}$ with input vector $I \in \mathcal{I}$ in CFHO_f that yields output vector O. Note that by Weak Liveness, there can be at most f crashed processes in α. Construct α' from α such that:

(1) α' is identical to α for non-crashed processes of α.

(2) If p crashes in α and round r is the last time it sends a message, then it does not crash in α' and keeps following protocol $\mathcal{P}$, but no process hears from p after round r in α'. Also, p hears from all $\geq n - f$ non-crashed processes of α from round r onwards in α'.

Thus α' can be viewed as a run of $\mathcal{P}$ with input vector I in HO_f that yields some output vector O'. As every process decides in α', every non-crashed process in α must decide in α as well. Moreover, any process that decides in α must decide the same value it decided in α'. As $(I, O') \in \Delta$, it follows from the definition of a task that $(I, O) \in \Delta$. Thus $\mathcal{P}$ solves T in CFHO_f. □

Lemmas 4.4 and 4.5 combine to give:

Lemma 4.6. *Any task that is solvable in* HO_f *is also solvable in* AMP_f *for* $0 \leq f \leq n$.

5 The SFHO_f model

We now examine the conditions under which tasks solvable in AMP_f are solvable in HO_f. Instead of directly simulating AMP_f in HO_f, we again introduce an intermediary model that is a variant of HO_f, the *Silenced-Faulty Heard-Of* model or SFHO_f, as it allows us to have cleaner and more organized proofs. Moreover SFHO_f is an interesting candidate for future work.

The allowable sequences in SFHO_f must satisfy exactly the same conditions as they do in HO_f (Round-robin property, Integrity, No Duplicates, Weak Liveness). The difference between the models is that we can also declare a process to be faulty in SFHO_f. The concept of a faulty process in SFHO_f differs from that in the CFHO_f model. We first define the notion of a *silenced* process, which applies to both HO_f and SFHO_f.

The *reach* of a process p from round r can be defined as the union of the set of processes which hear from p in round r, the set of processes which hear from one of these processes in round $r+1$, the set of processes which hear from one of the previous processes (of rounds r and $r+1$) in round $r+2$ and so on. Formally,

Definition 5.1. Let $\text{Reach}_p(r)$ denote the set of processes which receive a message from process p in round r.

The reach of process p between rounds r and s (where $r \leq s$), denoted $\text{Reach}_p(r, s)$, is defined as

$$\text{Reach}_p(r, s) \overset{def}{=} \begin{cases} \text{Reach}_p(r), & \text{if } r = s \\ \text{Reach}_p(r, s-1) \cup \bigcup_{q \in \text{Reach}_p(r, s-1)} \text{Reach}_q(s), & \text{if } r < s \end{cases}$$

The reach of process p starting from round r, denoted $\text{Reach}_p(r, \infty)$, is defined as

$$\text{Reach}_p(r, \infty) \overset{def}{=} \bigcup_{s \geq r} \text{Reach}_p(r, s).$$

Definition 5.2. A process p is said to be *silenced* if there exists an $r > 0$ such that $|\text{Reach}_p(r, \infty)| \leq f$.

We say that it p is *silenced from round* r if r is the minimum for which this holds.

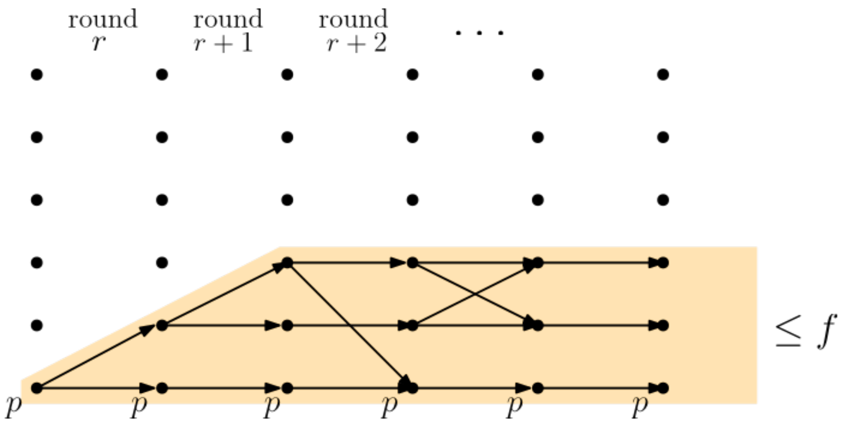


Figure 2: Process p silenced from round r

Definition 5.3. A process is faulty in a sequence σ in SFHO_f if it is silenced in σ.

We make a basic but crucial observation about non-silenced processes that also explains why we call a process "silenced".

Lemma 5.4. *Let α be an allowable sequence in HO_f or SFHO_f such that p is not silenced in α. Then for any $r > 0$, the reach of p from round r is the set of all processes, i.e. $|\text{Reach}_p(r, \infty)| = n$.*

Proof. As p is not silenced, $|\text{Reach}_p(r, \infty)| \geq f+1$ for all $r > 0$. As $\text{Reach}_p(r, r) \subseteq \text{Reach}_p(r, r+1) \subseteq \text{Reach}_p(r, r+2) \subseteq \dots$ and as $\text{Reach}_p(r, \infty)$ is finite, there exists $s \geq r$ such that $\text{Reach}_p(r, \infty) = \text{Reach}_p(r, s)$. So $\text{Reach}_p(r, s) \geq f+1$, which means that in round $s+1$, every process hears from some process in $\text{Reach}_p(r, s)$. Thus $|\text{Reach}_p(r, s+1)| = n$, and we conclude that $|\text{Reach}_p(r, \infty)| = n$. □

Another key property of HO_f and SFHO_f for $f < n/2$ which is not immediate is that the number of silenced processes is at most f. Thus in any allowable sequence of SFHO_f, there can be at most f faulty processes.

Lemma 5.5. *Suppose that $f < n/2$ and let α be an allowable sequence in HO_f or SFHO_f. There can be at most f silenced processes in α.*

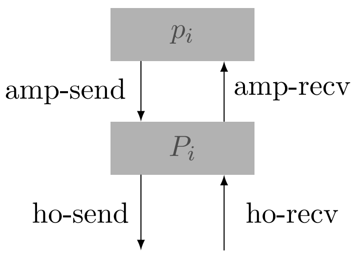
Proof. Let S be the set of silenced processes. Suppose that $p \in S$ is silenced from round r_p. We claim that all processes in $\text{Reach}_p(r_p)$ must be silenced. Indeed, if some process $q \in \text{Reach}_p(r_p)$ is not silenced, then by Lemma 5.4, $|\text{Reach}_q(r')| = n$ for some $r' > r_p$ and hence $|\text{Reach}_p(r_p)| = n$, contradicting our assumption that p is silenced.

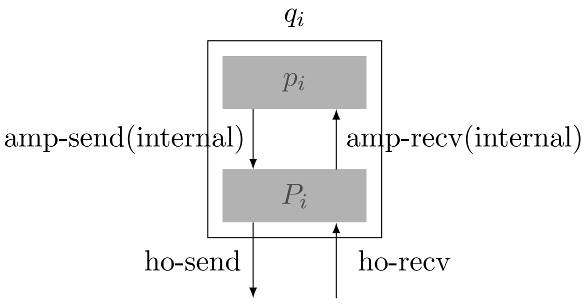
By the definition of reach, there is a round after which no message sent by a process in $\text{Reach}_p(r_p)$ is received by a process outside the set. As we have shown for any $p \in S$ that $\text{Reach}_p(r_p) \subseteq S$, it follows that there is a round after which no message sent by a process in S is received by a process outside S.

Claim 5.6. *There exists at least one process that is not silenced in α.*

Proof. Assume the contrary. Then there exists some round R before which every process becomes silenced. Thus in any round $r > R$, a message broadcast by any process must be received by at most f processes in round r. Thus the total number of successfully received messages in round r is at most nf. However, by Weak Liveness, this number is at least $n(n-f)$. Since $n - f > f$, we get a contradiction. □

Let p' be a process that is never silenced in α, so $p' \notin S$. As p' can miss at most f messages from other processes in any round, we must have $|S| \leq f$. Thus there can be a total of at most f silenced processes in α. □





6 Tasks solvable in AMP_f are solvable in SFHO_f for $f < n/2$

Let $0 \le f < n/2$, and let $T = (\mathcal{I}, \mathcal{O}, \Delta)$ be a task that is solvable in AMP_f by a protocol $\mathcal{P} = \{p_1, \ldots, p_n\}$. We assume without loss of generality that there is always a send event enabled after every receive event at each p_i; this is valid as $\mathcal{P}$ can be taken to be a full-information protocol for example. We also assume that each machine P_i has a hard-coded unique identifier id_i.

We construct a simulation system that simulates SFHO_f in AMP_f so that for $1 \le i \le n$, (1) the i-th simulated process is p_i, and (2) the i-th simulation machine P_i has the following specification:

Algorithm 2 Pseudocode for machine P_i

```
 1: Initialize list $seen := \emptyset$          ▷ History of messages received since the beginning
 2: Initialize list $old := \emptyset$           ▷ History of messages received before $p_i$'s last amp-send
 3: Initialize variable $latest :=$ null
 4: Initialize variable $T := 0$                  ▷ For timestamping

 5: When amp-send$_i(m)$ occurs:
 6:     $latest \leftarrow \langle id_i, m, T\rangle$
 7:     Add $latest$ to $seen$
 8:     Increment $T$ by 1
 9: In round $r \ge 1$:
10:     ho-send$_i(seen)$
11:     ho-recv$_i(M)$
12:     Add previously unseen messages from $M$ to $seen$ and acknowledge those of type $\langle s, m', t\rangle$ by
    adding $ack(id_i, \langle s, m', t\rangle)$ to $seen$
13:     If $latest \ne$ null and $ack(j, latest)$ occurs in $M$ for $\ge f$ distinct values of $j \ne id_i$:
14:         Initialize list $pending := \emptyset$
15:         Add messages of type $\langle s, m', t\rangle$ in $seen \backslash old$ to $pending$
16:         If $pending$ is empty:
17:             Enable amp-recv$_i(\perp)$
18:         Else:
19:             For each $\langle s, m', t\rangle \in pending$:        ▷ Release all newly received messages
20:                 Enable amp-recv$_i(m')$
21:         $old \leftarrow seen$
22:         $latest \leftarrow$ null                  ▷ Update $old$ and reset $latest$ for next AMP step
```

To solve task T in SFHO_f, a protocol $\mathcal{Q} = \{q_1, \ldots, q_n\}$ can again be constructed in the same manner as in Section 4, except that now the main events of q_i are ho-send and ho-recv, while its internal events are amp-send and amp-recv.

The working of q_i can be seen as follows: The input of q_i is also used as the input for p_i. When p_i does an internal amp-send$_i(m)$, P_i adds $\langle id_i, m, T\rangle$ to its $seen$ list which q_i then broadcasts in every subsequent round of SFHO_f. Every q_j that receives $\langle id_i, m, T\rangle$ rebroadcasts it along with an acknowledgement that it received $\langle id_i, m, T\rangle$. However it is possible that q_i is silenced, in which case $\langle id_i, m, T\rangle$ will not reach every machine, meaning not all of them will enable amp-recv$_i(m)$. As the simulated run has to satisfy Non-Faulty Liveness to be valid in AMP_f, q_i has to halt p_i, which it does by blocking all further amp-recv events from P_i to p_i until enough acknowledgements for $\langle id_i, m, T\rangle$ are received. If this is satisfied, q_i is certain that everyone will receive $\langle id_i, m, T\rangle$ and enables an internal amp-recv$_i(m')$ for every newly received message of the form $\langle \cdot, m', \cdot\rangle$. If p_i internally decides an output o, then q_i decides o as well.

Remark. Every P_i tags every message m from p_i with the identifier id_i and a timestamp T. This is so that the machines can distinguish a message that is sent multiple times by a process or by different processes.

In order to prove that protocol $\mathcal{Q}$ solves T in SFHO_f, consider any run γ of $\mathcal{Q}$ in SFHO_f with input vector I and output vector O. As before, we can obtain the underlying simulation schedule for γ, which we call α. Since each q_i in run γ uses the output of p_i in $top(\alpha)$ to decide, O is also the vector of outputs decided by the p_i in $top(\alpha)$. We will first show that $(I, O) \in \Delta$ by showing that $top(\alpha)$ is a run in AMP_f.

Lemma 6.1. *Simulated process p_i is faulty in $top(\alpha)$ if and only if q_i is faulty in γ.*

Proof. Suppose that p_i is faulty in $top(\alpha)$ but q_i is not faulty in γ. The last event at p_i in α is some amp-send$_i(m)$ event as we assumed that there is always an amp-send enabled after an amp-recv at p_i. Then P_i has the non-null *latest* value of $\langle id_i, m, T\rangle$ (for some T) for the remainder of α. As q_i is not silenced, $\langle id_i, m, T\rangle$ must reach all processes by Lemma 5.4. All processes must then add an acknowledgement for $\langle id_i, m, T\rangle$ to their *seen* lists, and by Weak Liveness of HO_f, q_i must eventually receive $n - f > f$ of these acknowledgements. But then P_i must satisfy line 13 and enable an amp-recv event in α, contradicting our assumption about p_i halting. Thus q_i must be silenced, hence faulty in γ.

Suppose that p_i is not faulty, meaning that it has infinitely many amp-recv events in $top(\alpha)$. For every amp-recv event at p_i, process q_i must have received at least f acknowledgements for a new *latest* message of q_i, meaning that its reach in any round has size more than f. Thus q_i is not silenced, hence not faulty in γ. □

By Lemma 5.5, there can be at most f faulty processes in γ. So we get the following corollary:

Corollary 6.2. There are at most f faulty simulated processes in $top(\alpha)$.

Lemma 6.3. *In $top(\alpha)$, let* amp-send$_i(m)$ *be an event at p_i that is followed by an amp-recv event at p_i. Then m is received by all non-faulty simulated processes in $top(\alpha)$.*

Proof. In α, once amp-send$_i(m)$ occurs at p_i, $\langle id_i, m, T\rangle$ (for some T) is included in subsequent ho-send events of q_i. By assumption, P_i later enables an amp-recv to p_i, so q_i must have received at least f acknowledgements for $\langle id_i, m, T\rangle$ from other q_j. So $\langle id_i, m, T\rangle$ has reached at least $f + 1$ processes of $\mathcal{Q}$, and thus eventually reaches all non-faulty processes q_j.

Any non-faulty simulated process p_j has infinitely many amp-recv events in $top(\alpha)$. Due to line 19 of Algorithm 2, it follows that amp-recv$_j(m)$ is one of those events. □

Corollary 6.4. $top(\alpha)$ satisfies Non-faulty Liveness.

Proof. If p_i is non-faulty in $top(\alpha)$, it always satisfies the condition of Lemma 6.3 and thus its messages are received by all non-faulty processes in $top(\alpha)$.

□

Lemma 6.5. *$top(\alpha)$ satisfies Integrity and No Duplicates.*

Proof. Integrity: If amp-recv$_j(m)$ occurs at p_j, then q_j must have received $\langle id_i, m, T\rangle$ (for some i and T), which must have been sent by q_i as γ satisfies Integrity in SFHO_f. Thus amp-send$_i(m)$ must have occurred at p_i.

No Duplicates: amp-recv$_j(m)$ occurs at p_j when P_j encounters the message $\langle id_i, m, T\rangle$ in $seen \setminus old$ and does line 19. Once P_j enables this event, $\langle id_i, m, T\rangle$ is moved to *old*, meaning the same event will not occur again unless q_j receives some message $\langle id_{i'}, m, T'\rangle$ where $id_{i'} \neq id_i$ or $T' \neq T$, in which case there is no problem. □

From corollaries 6.2 and 6.4 and Lemma 6.5, we get that $top(\alpha)$ is a run in AMP_f. Thus $(I, O) \in \Delta$ as $\mathcal{P}$ solves T in AMP_f. Since p_i decides an output if it is non-faulty in $top(\alpha)$, it follows from Lemma 6.1 that q_i decides in γ if it is non-faulty. We conclude that protocol $\mathcal{Q}$ solves T in SFHO_f, so:

Lemma 6.6. *Any task that is solvable in* AMP_f *is also solvable in* SFHO_f *for* $0 \leq f < n/2$.

7 Comparing HO_f and SFHO_f

In this section, we study the conditions under which a task solvable in SFHO_f is also solvable in HO_f and then merge our results with Lemmas 4.6 and 6.6.

7.1 HO_f and AMP_f solve the same colorless tasks for $f < n/2$

Suppose that $0 \leq f < n/2$ and let $T = (\mathcal{I}, \mathcal{O}, \Delta)$ be a colorless task that is solvable in SFHO_f by a full-information protocol $\mathcal{P} = \{p_1, \dots, p_n\}$. Thus in a run α of $\mathcal{P}$ in SFHO_f where process p is silenced, p's lack of an output value does not matter. We make the basic observation that if process p_j hears from process p_i after a full-information protocol assigns output o to p_i, then p_j can determine that p_i decided on output o.

One can easily construct a full-information protocol $\mathcal{Q} = \{q_1, \dots, q_n\}$ that solves T in HO_f: Every process q_i runs p_i, and if a process q_i hears from process q_j such that (1) q_j has already decided on output value o, and (2) process q_i does not have an output yet, then q_i decides output value o.

Let α be a run of $\mathcal{Q}$ with input vector I and output vector $O_{\mathcal{Q}}$ in HO_f.

We first prove that all processes of $\mathcal{Q}$ decide on a value.

Lemma 7.1. *All processes* q_i *have non-*$\perp$ *output values in* $O_{\mathcal{Q}}$.

Proof. We can view α as a run of $\mathcal{P}$ in SFHO_f, so if p_i is not silenced in α, it has an output. Thus if q_i is not silenced in α, it is certain to decide an output.

Since $f < n/2$, at least $n - f$ processes in $\mathcal{Q}$ are not silenced by Lemma 5.5 and decide by some round R. In round $R + 1$, any q_i that has not yet decided hears from a decided process by Weak Liveness and copies its output. □

It remains to show that $\mathcal{Q}$ gives a valid output vector for the input vector of α. Let $O_{\mathcal{P}}$ be the output vector of $\mathcal{P}$ on α. Then $(I, O_{\mathcal{P}}) \in \Delta$ as α can be viewed as a run of $\mathcal{P}$ in SFHO_f.

Lemma 7.2. $val(O_{\mathcal{Q}}) \subseteq val(O_{\mathcal{P}})$ *and thus* $(I, O_{\mathcal{Q}}) \in \Delta$.

Proof. For the purpose of induction, we assume there is a round 0, where processes have not started communicating. We prove the following statement by induction on the round number:

In round $r \geq 0$, *if process* q_i *decides on output* o, *then* $o \in O_{\mathcal{P}}$.

Base case: if process q_i decides output o in round 0, it must be because p_i decides o in α in round 0. Thus $o \in O_{\mathcal{P}}$.

Inductive step: Suppose that the statement is true for all rounds $r \leq R$. Suppose q_i decides output o in round $R + 1$. This is because either:

- p_i decides output o in α in round $R + 1$, meaning $o \in O_{\mathcal{P}}$, or
- q_i heard from process q_j which had decided output o before round $R + 1$. By the induction hypothesis, $o \in O_{\mathcal{P}}$.

This completes the inductive proof, and we conclude from the statement that $val(O_{\mathcal{Q}}) \subseteq val(O_{\mathcal{P}})$. To conclude, since $(I, O_{\mathcal{P}}) \in \Delta$, we have by the definition of colorless tasks that $(I, O_{\mathcal{Q}}) \in \Delta$. □

Hence protocol $\mathcal{Q}$ solves colorless task $T = (\mathcal{I}, \mathcal{O}, \Delta)$ in HO_f. This gives:

Lemma 7.3. *For* $f < n/2$, *any colorless task that is solvable in* SFHO_f *is solvable in* HO_f.

Combining with Lemmas 4.6 and 6.6, we conclude that:

Theorem 7.4. *For* $0 \leq f < n/2$, *models* HO_f *and* AMP_f *solve the same set of colorless tasks.*

7.2 HO_f and AMP_f solve the same tasks for $f \leq 1$

As there are no silenced processes in SFHO_0, it is the same model as HO_0 and thus AMP_0 (by Lemmas 4.6 and 6.6) for $n \geq 0$. Hence for the remainder of the section, we will assume that $f = 1$ and $n > 2f = 2$.

Let $T = (\mathcal{I}, \mathcal{O}, \Delta)$ be a task (colorless or colored) that is solvable in SFHO_1 by a protocol $\mathcal{P} = \{p_1, \ldots, p_n\}$. We assume without loss of generality that $\mathcal{P}$ is full-information. We shall show that the following full-information protocol $\mathcal{Q} = \{q_1, \ldots, q_n\}$ solves task T in HO_1.

Definition 7.5. In the run of a full-information protocol in HO_f or SFHO_f, the *initial view* of a process is defined to be its initial state and its *view in round* r is defined to be the history of messages sent and received by it till round r of the run.

In every round, process q_i runs p_i, but the decision rule is modified:

Algorithm 3 Process q_i

```
1: Round $r$: Send and receive messages and change the state of $p_i$ accordingly. If $q_i$ does not have
   an output yet, then
2:     If $p_i$ decides $o$ in round $r$, then $q_i$ decides $o$
3:     Else
4:         If $q_i$ determines from its round-$r$ view that every other process has decided, then
5:             $q_i$ considers an infinite run $\alpha'$ such that for each $q_j$, $\alpha'$ is identical to the current run
   till the round where $q_i$ last hears from $q_j$, and no process is silenced in $\alpha'$
6:             $q_i$ decides the output of $p_i$ in $\alpha'$
```

Let α be a run of $\mathcal{Q}$ in HO_1 with input vector I and output vector $O_{\mathcal{Q}}$.

Lemma 7.6. *Suppose q_i satisfies line 4 of Algorithm 3 in round r. There exists at least one run α' that q_i can consider in line 5. Moreover, q_i is able to decide in line 6.*

Proof. Process q_i starts constructing α' by using the view of each q_j for the round where q_i last hears from q_j. It can then fill in the missing events in α' till round r by going ever all possibilities (there is at least one, namely the current run). From round $r + 1$ onwards, it assumes that all processes in α' hear from everybody, so no process is silenced in α'.

Thus α' can be viewed as a run of $\mathcal{P}$ in SFHO_1 where p_i is not silenced, so p_i decides an output in α' and consequently q_i is able to decide in line 6. ☐

Remark 7.7. From a practical perspective, q_i cannot first construct an infinite run α' and then run p_i on it. Instead, it must construct α' round by round, running p_i on it at the end of each round.

Lemma 7.8. *If process q_i is silenced from round R in α, then for every $r \geq R + 2$, its round-r view contains the round-$(r-2)$ views of all processes.*

Proof. Let $r \geq R+2$. In round $r-1$, q_i must receive the round-$(r-2)$ views of at least $n-1$ processes including its own by Weak Liveness.

Suppose q_j is a process whose round-$(r-2)$ view q_i did not receive in round $r-1$. As q_i is silenced from round R, no other process receives messages from q_i in round $r-1$. So all the other processes must hear from q_j in round $r-1$. Thus the $n-2 \geq 1$ messages that q_i receives from other processes in round r contain q_j's round-$(r-2)$ view. ☐

We prove that all processes in $\mathcal{Q}$ decide on a value.

Lemma 7.9. *All processes q_i have non-$\perp$ output values in $O_{\mathcal{Q}}$.*

Proof. We can view α as a run of $\mathcal{P}$ in SFHO_1, so if p_i is not silenced in α, it has an output. Thus if q_i is not silenced in α, it is certain to decide an output.

Suppose that q_i is silenced from round R in α and it does not get to decide using line 2. By Lemma 7.8, in round $r \geq R+2$, q_i's round-r view contains the round-$(r-2)$ views of all other processes. As they are not silenced by Lemma 5.5, each of them decides on an output, say by round R'. By round $\max(R, R') + 2$, q_i satisfies the condition in line 4 and hence decides by Lemma 7.6. ☐

It remains to prove that $\mathcal{Q}$ gives a valid output vector for the input vector of α.

Lemma 7.10. $(I, O_{\mathcal{Q}}) \in \Delta$.

Proof. There can be at most one process q_i that decides using line 6. Indeed, if there were two such processes, both must have satisfied line 4, meaning each decided their outputs before the other; this is absurd.

This gives us two cases:

1. All processes in $\mathcal{Q}$ decide their outputs using line 2. Then $\mathcal{P}$ must yield output vector $O_{\mathcal{Q}}$ on α in SFHO_1. Thus $(I, O_{\mathcal{Q}}) \in \Delta$.

2. Some process q_i decides using line 6. Then all other processes q_j decide in line 2 using the output of their p_j. Let α' be the run considered by q_i in line 5, which by Lemma 7.6 exists and on which q_i decides the output of p_i. The output of q_j $(j \neq i)$ is the same in α' as in α since the sequence of events it uses to decide is the same. Since all processes in $\mathcal{Q}$ use their counterpart in $\mathcal{P}$ to decide in α', $\mathcal{P}$ must yield output vector $O_{\mathcal{Q}}$ on α' in SFHO_1. Thus $(I, O_{\mathcal{Q}}) \in \Delta$.

□

Hence protocol $\mathcal{Q}$ solves task $T = (\mathcal{I}, \mathcal{O}, \Delta)$ in HO_1. This gives:

Lemma 7.11. *For $f \leq 1$ and $n > 2f$, tasks that are solvable in SFHO_f are solvable in HO_f.*

Combined with Lemmas 4.6 and 6.6, we conclude that

Theorem 7.12. *For $f \leq 1$ and $n > 2f$, models HO_f and AMP_f solve the same set of tasks.*

8 AMP_f and HO_f do not solve the same colored tasks for $1 < f < n/2$

After proving that HO_1 and SFHO_1 solve the same set of tasks in the previous section, we naturally attempted to extend the same idea in Algorithm 3 to the case where $f > 1$. The idea for $f = 1$ was that if a process was silenced in HO_1, it would eventually be able to determine the outputs of the other processes and decide a compatible output for itself as well. However for $f > 1$, it is possible to have a run in HO_f with two silenced processes such that no process hears from them except themselves. Since the two silenced processes will never have any information about each other, getting them to decide compatible outputs is difficult. Gradually we became convinced that this was not possible; and indeed we were able to show that this is the case.

The *renaming task* [3] with initial name space of potentially unbounded size M and new name space of size N and $M > N$ is defined as follows. Every process initially has as input a distinct identifier from the initial name space. Every process with a non-$\perp$ output must return a distinct output name from the new name space.

Without any restriction on the definition of task solvability, there exist trivial protocols for this task, e.g., for $N = n$, the protocol where process p_i always chooses i as output. Thus the processes' indexing must be external, i.e., the renaming protocol must only depend on the initial names given to the processes. We use the same assumption on renaming protocols as in [3]:

Anonymity assumption: For any initial name, any two processes must have the same corresponding initial states. Moreover, let π be a permutation of $\{1, \ldots, n\}$, let J and J' be two initial configurations, such that for any $1 \leq i \leq n$, process p_i in J and process $p_{\pi(i)}$ in J' are in the same state. Let R be a run with initial configuration J and let $\pi(R)$ be the run with initial configuration J' and the permutation π applied to the schedule of R (i.e., the sequence of events associated with p_i is now associated to $p_{\pi(i)}$). Then p_i in R and $p_{\pi(i)}$ in $\pi(R)$ must have the same sequence of states.

Theorem 8.1. *For n, f such that $2 \leq f < n/2$, the renaming task with an initial name space of size $N + n - 1$ and new name space of size $N = n + f$ is solvable in AMP_f but is not solvable in HO_f.*

Proof. It has been shown in [3] that the renaming task with an unbounded initial name space and a new name space of size $N = n + f$ where $f < n/2$ is solvable in AMP_f. Trimming the size of the initial name space to $N + n - 1$ clearly preserves the result.

Assume to the contrary that there is a protocol $\mathcal{P}$ solving the task in HO_f. We can assume that $\mathcal{P}$ is a full-information protocol. We show how to obtain an input vector on which $\mathcal{P}$ fails.

Fix arbitrary distinct initial names $x_1, \ldots, x_{n-2}$ for processes $p_1, \ldots, p_{n-2}$. Let processes p_{n-1} and p_n have initial names a and b which shall be chosen later. Consider a run of $\mathcal{P}$ in HO_f with input vector $(x_1, \ldots, x_{n-2}, a, b)$ such that:

(1) $p_1, \ldots, p_{n-2}$ hear from each other in every round.

(2) No process hears from processes p_{n-1} and p_n except themselves in any round.

(3) Processes p_{n-1} and p_n get to hear from $p_1, \ldots, p_{n-2}$ in every round.

The output of a process can be seen as being determined by a *decision function* δ that maps every sequence of views (Definition 7.5) at the process to an output value. Due to the anonymity assumption, every process has the same δ function.

The sequence of views of p_{n-1} is the value of some function $f(a)$ that does not depend on b, and its new name is some value $\delta(f(a))$. The sequence of views of p_{n-1} and p_n are symmetric, so the sequence of views of p_n is $f(b)$, so its new name is $\delta(f(b))$. As there are $N + 1$ initial names apart from $x_1, \ldots, x_{n-2}$ and only N possible new names, there exist values for a, b such that such that $x_1, \ldots, x_{n-2}, a, b$ are distinct and $\delta(f(a)) = \delta(f(b))$. Thus $\mathcal{P}$ fails on the input vector $(x_1, \ldots, x_{n-2}, a, b)$; a contradiction. □

9 Conclusion and future work

We have shown that in the context of solvability of colorless tasks, AMP_f and HO_f coincide for $f < n/2$. For colored tasks, this is true if $f = 1$ (and $n > 2$), and false if $1 < f < n/2$. The reason for this split in the case of colored tasks is explained by the presence of silenced processes in HO_f. For $f = 1$, there is at most one silenced process; it will eventually learn the outputs of the other processes and accordingly decide an output that is compatible. For $f > 1$, there can be two such processes who are never heard of by any other process; this makes it difficult for them to decide compatible outputs.

We have not yet addressed the question for the case where $f \geq n/2$. We can also ask if there is a class of tasks larger than that of colorless tasks for which the two models coincide. A comparison of AMP_f and SFHO_f would also be interesting. Indeed, our results show that like HO_f, the SFHO_f model coincides with AMP_f for colorless tasks where $f < n/2$ and colored tasks where $f = 1$. However it is not clear whether they agree or differ for the case of colored tasks where $1 < f < n/2$.

There is also a need to modify the definition of task solvability. The general definition requires the existence of *any* collection of processes solving the task; this would trivialize the renaming task in any model. Rather, the definition of solvability should depend on the task and should specify what protocols can be used to solve it.

References

[1] Yehuda Afek and Eli Gafni. Asynchrony from synchrony. In *Distributed Computing and Networking*, volume 7730 of *Lecture Notes in Computer Science*, pages 225–239. Springer Berlin / Heidelberg, Germany, 2012.

[2] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

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

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

[5] Bernadette Charron-Bost and André Schiper. The heard-of model: Computing in distributed systems with benign failures. 2007. Replaces TR-2006: The Heard-Of Model: Unifying all Benign Failures. URL: http://infoscience.epfl.ch/record/109375.

[6] Bernadette Charron-Bost and André Schiper. The Heard-Of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, April 2009. doi:10.1007/s00446-009-0084-6.

[7] Carole Delporte-Gallet, Hugues Fauconnier, Eli Gafni, and Petr Kuznetsov. Wait-freedom with advice. *Distributed Comput.*, 28(1):3–19, 2015. URL: https://doi.org/10.1007/s00446-014-0231-6, doi:10.1007/S00446-014-0231-6.

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

[9] Eli Gafni and Petr Kuznetsov. On set consensus numbers. *Distributed Comput.*, 24(3-4):149–163, 2011. URL: https://doi.org/10.1007/s00446-011-0142-8, doi:10.1007/S00446-011-0142-8.

[10] Eli Gafni and Giuliano Losa. Invited Paper: Time Is Not a Healer, but It Sure Makes Hindsight 20:20. In Shlomi Dolev and Baruch Schieber, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 62–74. Springer Nature Switzerland, 2023. doi:10.1007/978-3-031-44274-2_6.