Tight Lower Bounds on Early Local Decisions in Uniform Consensus
[Extended Abstract]

Partha Dutta    Rachid Guerraoui    Bastian Pochon

Distributed Programming Laboratory
Swiss Federal Institute of Technology in Lausanne (EPFL)

Abstract

When devising a (uniform) consensus algorithm, it is common to minimize the time complexity of global decisions, which is typically measured as the number of communication rounds needed for all correct processes to decide. In practice, what we might want to minimize is the time complexity of local decisions, which we define as the number of communication rounds needed for at least one correct process to decide. We investigate tight lower bounds on consensus local decisions in crash-stop message-passing model.

In the synchronous model where $t$ processes may fail, we show that in runs with at most $f \leq t-1$ failures, there is a run in which no correct process decides before round $f+1$, and there is a run in which at most one correct process decides before round $f+2$. This result generalizes the well-known $f+2$ round global decision lower bound. Moreover, we point out a simple consensus algorithm which achieves these lower bounds.

In the eventually synchronous model, we show that there is a synchronous run with $f$ failures in which no correct process decides before round $f+2$; i.e., the local and the global decision lower bounds are identical for synchronous runs. We describe a new algorithm which matches the $f+2$ round lower bound for global decision (and hence, for local decision as well) in synchronous runs, closing a challenging open question.

Category: Regular and student paper (Partha Dutta and Bastian Pochon are full-time PhD students)
Contact Author: Partha.Dutta@epfl.ch, EPFL-IC-LPD, CH-1015, Switzerland, tel: +41 21 693 8121
1 Introduction

Motivation. Determining how long it takes to reach consensus\textsuperscript{1} among a set of processes is an important question in distributed computing. For instance, the performance of a replicated system is impacted by the performance of the underlying consensus service used to ensure that the replica processes agree on the same order to deliver client requests [14]. Traditionally, lower bounds on the time complexity of consensus have been stated in terms of the number of communication rounds (also called steps) needed for all correct processes to decide [15] (i.e., global decision), or even halt [7], possibly as a function of the number of failures $f$ that actually occur, out of the total number $t$ of failures that are tolerated.

From a practical perspective, what we might sometimes want to measure and optimize, is the number of rounds needed for at least one correct process to decide, i.e., for a local decision. Indeed, a replicated service can respond to its clients as soon as a single replica decides on a reply and knows that other replicas will reach the same decision (even if they did not decide yet).

Background. Consider the synchronous crash-stop model where a set of $n$ processes proceed by exchanging message in a round by round manner [13]. In any run of the model, at most $t$ processes might fail, and they can only do so by crashing. For any consensus algorithm, let $R(f)$ denote the set of runs in which at most $f \leq t-1$ processes fail. The global decision lower bound on consensus states that there is a run in $R(f)$ in which some correct processes decides in round $f+2$ or in a higher round [2, 12]. In other words, for all correct processes to decide, we need at least $f+2$ rounds. However, a global decision lower bound does not say whether some correct process can decide before $f + 2$ rounds in every run in $R(f)$, and if yes, how many processes may actually do so.

In the eventually synchronous model, from [8] we know that for any consensus algorithm there is a run in $R(f)$ which may take an arbitrary number of rounds for any process to decide. However, if we define $SR(f)$ as the set of synchronous runs,\textsuperscript{2} then in $SR(f)$ we can bound the number of rounds needed for correct processes to decide. In fact, it is easy to see that the $f + 2$ round global decision lower bound in synchronous model immediately extends to $SR(f)$. However, unlike synchronous model, a matching algorithm for $f + 2$ round global decision lower bound has been an open problem [4, 11].

Contributions. This paper points out that, in the synchronous model, the local decision tight lower bound is $f + 1$. In other words, (1) for any consensus algorithm, there is a run in $R(f)$ in which no correct process decides in a round lower than $f + 1$, and (2) there is a consensus algorithm for which, in every run in $R(f)$ some correct process decides by round $f + 1$.

Moreover, we show that for every consensus algorithm, there is a run in $R(f)$ in which either none or exactly one process decides before round $f + 2$; this gives a bound on the number of correct processes which can decide before the global decision lower bound. This result generalizes the global decision lower bound of [2, 12] which states that there is a run in which at least one correct process decides in round $f + 2$ or in a higher round, whereas, our result implies that there is run in which at least $n-t-1$ correct processes decide in round $f + 2$ or in a higher round (because there are at least $n-t$ correct processes and at most one of them can decide before round $f + 2$).

In the eventually synchronous model, we show that, for the synchronous runs $SR(f)$, the local decision lower bound is $f + 2$, the same as the global decision lower bound. We give a matching

\textsuperscript{1}In this paper consensus always refers to the uniform variant of the problem. In the consensus problem [10, 16] processes start with a proposal value and is supposed to eventually decide on a final value such that the following properties are satisfied: (validity) if a process decides $v$, then some process has proposed $v$; (agreement) no two processes decide differently; and (termination) every correct process eventually decides. Binary consensus is a variant of consensus in which the proposal values are restricted to 0 and 1.

\textsuperscript{2}Synchronous runs may be the most frequent runs in practice if process failures and unpredictable communication delays are rare.
algorithm which globally decides (and hence, locally decides) by round \( f + 2 \) in every synchronous run with at most \( f \) failures, for every \( 0 \leq f \leq t \). The algorithm proceeds in a round-by-round manner. At the end of round \( f + 1 \) (for every \( 0 \leq f \leq t \)) the algorithm tries to detect whether the run is synchronous and there has been at most \( f \) failures, and if so, it tries to decide in the next round.

Section 2 presents the lower bound results in the synchronous model. We give our lower bound results for the eventually synchronous model in Section 3. Due to lack of space, the correctness proof of the matching algorithm for the eventually synchronous case is described in the optional appendix A.2. To strengthen our results, we provide our lower bound proofs for binary consensus and propose matching algorithm for the multivalued case.

## 2 Consensus in the Synchronous Model

**System Model.** We assume a distributed system model composed of \( n \geq 3 \) processes, \( \Pi = \{p_1, p_2, \ldots, p_n\} \). Processes communicate by message-passing and every pair of processes is connected by a bi-directional communication channel. Processes may fail by crashing and do not recover from a crash. Any process that does not crash in a run is said to be correct in that run; otherwise the process is faulty. In any given run, at most \( t < n \) processes can fail. Processes proceed in rounds [13]. Each round consists of two phases: (a) in the send phase, processes are supposed to send messages to all processes; (b) in the receive phase, processes receive messages sent in the send phase, update local states, and (possibly) decide. If some process \( p_i \) completes the send phase of the round, every process that completes the receive phase of the round, receives the message sent by \( p_i \) in the send phase. If \( p_i \) crashes during the send phase, then any subset of the messages \( p_i \) is supposed to send in that round may be lost. We denote the synchronous model by \( SCS \).

**Time complexity metrics.** Consider any consensus algorithm in the synchronous model. We say that a process decides in round \( k \geq 1 \) iff it decides in the receive phase of round \( k \). A run of an algorithm globally decides in round \( k \) if all correct processes decide in round \( k \) or in a lower round, and some correct process decides in round \( k \). For every \( 0 \leq f \leq t \), we define the global decision tight lower bound \( g_f \), as the round number such that, every algorithm has a run with at most \( f \) failures, which globally decides in round \( g_f \), or in a higher round, and there is an algorithm, which globally decides by round \( g_f \) in every run with at most \( f \) failures. A run of an algorithm locally decides in round \( k \) if all correct processes decide in round \( k \) or in a higher round and some correct process decides in round \( k \). For every \( 0 \leq f \leq t \), we define the local decision tight lower bound \( l_f \), as the round number such that, every algorithm has a run with at most \( f \) failures, which locally decides in round \( l_f \) or in a higher round, and there is an algorithm, which locally decides by round \( l_f \) in every run with at most \( f \) failures.

**Proof Technique.** Our lower bound proofs are devised following the layering technique of [17], also used in [12]. Similar to [12, 17], we consider only a subset of runs in the model for showing lower bounds. A subsystem is a subset of the set of all possible runs in the model. The first subsystem of \( SCS \) that we consider is the set of runs in which at most one process crashes in every round, and we denote it by \( sub_{scs} \). If \( p_i \) crashes at round \( k \), then any subset of the messages that \( p_i \) is supposed to send in that round may not be received.\(^3\)

A configuration at (the end of) round \( k \geq 1 \) in a run is the collection of the states of all processes at the end of round \( k \). The state of a process which has crashed in a configuration is a special symbol denoting that the process has crashed. We say that a process \( p_i \) is alive in a given configuration if \( p_i \)

\(^3\)It is important to notice that the subsystem in [12] contains runs with the additional restriction that, if a process \( p_i \) crashes in the send phase of round \( k \), the round \( k \) messages may not be received by a prefix of \( \Pi \). Thus the subsystem in [12] is a subset of \( sub_{scs} \).
has not crashed in that configuration. An initial configuration (or round 0 configuration) in a run is the collection of initial states of all processes in that run. We denote the set of all initial configurations as $\text{Init}$. A run of an algorithm is completely defined by its initial configuration and its failure pattern. (The failure pattern for a run, states for each round $k$, the process which crashes in round $k$ (if any), and the set of processes which did not receive round $k$ message from the crashed process.) Therefore, for any configuration $C$ at round $k$ (of a consensus algorithm), we can define $r(C)$ as the run in which (1) round $k$ configuration is $C$, and (2) no processes crashes after round $k$. We denote by $val(C)$ the decision value of the correct processes in $r(C)$. Note that a process $p_i$ is alive in $C$ iff $p_i$ is correct in $r(C)$.

All our lower bound proofs start from the following lemma in $\text{sub}_{scs}$ or one of its variants. (A proof of the lemma, slightly modified from [12], is presented in the optional appendix A.1.)

**Lemma 1** For every binary consensus algorithm in $\text{sub}_{scs}$ and $0 \leq k \leq t$, there are two configurations $y, y' \in I^k(\text{Init})$ such that (1) at most $k$ processes have crashed in each configuration, (2) the configurations differ at exactly one process, and (3) $val(y) = 0$ and $val(y') = 1$.

**Local decision lower bounds in synchronous model.** Given that the global decision lower bound is $f+2$, intuitively it is easy to see that the local decision lower bound is $f+1$: if some correct process can decide at round $f$ in every run in $R(f)$, then it can broadcast the decision value, and thus, enforce a global decision at round $f+1$ in every run in $R(f)$.

**Proposition 2** Let $1 \leq t \leq n-1$. For every consensus algorithm and every $0 \leq f \leq t-1$, there is a run with $f$ failures in which no correct process decides before round $f+1$.

From Lemma 1, deriving Proposition 2 is rather straightforward, and we present the proof in the optional appendix A.1. It is also easy to design a matching algorithm, and we refer our readers to the full-version of the paper [5] for a detailed description of such an algorithm.

From the above tight local decision lower bound we know that some correct process can decide in round $f+1$ in every run in $R(f)$. On the other hand, the global decision lower bound states that there is a run in $R(f)$ in which $f+2$ rounds are needed for all correct processes to decide. It is natural to ask whether it is possible for more than one process to decide before round $f+2$. In the following proposition we show that the answer is negative.

Before we present the proposition, we enlarge the subsystem used in the proof. The subsystem $\text{sub}_{sca1}$ consists of all runs in the synchronous crash-stop model, i.e., any number of processes can crash in a round. (We revisit few definitions which were presented in the context of the subsystem $\text{sub}_{scs}$ in which at most one process can crash in a round.) We define the following in $\text{sub}_{sca1}$. For any configuration $C$ at round $k$ of a consensus algorithm $A$, we define $R(C)$ as the run in which (1) the configuration at round $k$ is $C$ and (2) no process crashes after round $k$. We denote by $Val(C)$ the decision value of the correct processes in $R(C)$. Serial runs are those runs of $A$ which are in $\text{sub}_{scs}$ (i.e., runs in which at most one process crashes in every round). Similarly, serial configurations are the configurations of the serial runs of $A$. As every run in $\text{sub}_{scs}$ is a serial run $\text{sub}_{sca1}$, from Lemma 1 we immediately have:

**Claim SCS1.** For every binary consensus algorithm in $\text{sub}_{sca1}$ and $0 \leq k \leq t$, there are two serial configurations $y, y' \in I^k(\text{Init})$ such that (1) at most $k$ processes have crashed in each configuration, (2) the configurations differ at exactly one process, and (3) $Val(y) = 0$ and $Val(y') = 1$.

**Proposition 3** Let $3 \leq t \leq n-1$. For every consensus algorithm and every $0 \leq f \leq t-3$, there is a run with $f$ failures in which at most one correct process decides before round $f+2$. 

3
Proof: Suppose by contradiction that there is a binary consensus algorithm $A$ in $sub_{scs1}$ and a round number $f + 1$ such that $0 \leq f \leq t - 3$, and in every run of $A$ with at most $f$ failures, there are two correct processes which decide before round $f + 2$.

From Claim SCS1 that at the end of round $f$ there are two serial configurations of $A$, $y$ and $y'$, such that, (1) at most $f$ processes have crashed in each configuration, (2) the configurations differ at exactly one process, say $p_i$, and (3) $Val(y) = 0$ and $Val(y') = 1$. Let $z$ and $z'$ denote the configurations at the end of round $f + 1$ of $R(y)$ and $R(y')$, respectively. From our initial assumption about $A$, in $z$, there are two alive processes $q_1$ and $q_2$ which have decided 0. Similarly, in $z'$, there are two alive processes $q_3$ and $q_4$ which have decided 1. Since $q_1$ and $q_2$ are distinct, at least one of them is distinct from $p_i$, say $q_1$. Similarly, without loss of generality we can assume that $q_3$ is distinct from $p_i$.

Thus we have (1) a $f + 1$ round configuration $z$ with $f$ failures in which an alive process $q_1$ has decided 0, (2) a $f + 1$ round configuration $z'$ with $f$ failures in which an alive process $q_3$ has decided 1, and (3) process $p_i$ is distinct from both $q_1$ and $q_3$. (Processes $q_1$ and $q_3$ may or may not be distinct.) There are two cases to consider.

Case 1. Process $p_i$ is alive in $y$ and $y'$. Consider the following two non-serial runs:

**R1** is a run such that (1) the configuration at the end of round $f$ is $y$, (2) $p_i$ crashes in the send phase of round $f + 1$ such that only $q_1$ receives the message from $p_i$, (3) $q_1$ and $q_3$ crash before sending any message in round $f + 2$, and (3) no process distinct from $p_i$, $q_1$, and $q_3$ crashes after round $f$. Notice that $q_1$ cannot distinguish the configuration at the end of round $f + 1$ in $R1$ from $z$, and therefore, decides 0 at the end of round $f + 1$ in $R1$. By agreement, every correct process decides 0. Since $t \leq n - 1$, there is at least one correct process in $R1$, say $p_j$.

**R2** is a run such that (1) the configuration at the end of round $f$ is $y'$, (2) $p_i$ crashes in the send phase of round $f + 1$ such that only $q_3$ receives the message from $p_i$, (3) $q_1$ and $q_3$ crash before sending any message in round $f + 2$, and (3) no process distinct from $p_i$, $q_1$, and $q_3$ crashes after round $f$. Notice that $q_3$ cannot distinguish the configuration at the end of round $f + 1$ in $R2$ from $z'$, and therefore, decides 1 at the end of round $f + 1$ in $R2$. However, $p_i$ cannot distinguish $R1$ from $R2$: at the end of round $f + 1$, the two runs are different only at $p_i$, $q_1$, and $q_3$, and none of the three processes send messages after round $f + 1$ in both runs. Thus (as in $R1$) $p_i$ decides 0 in $R2$; a contradiction with agreement.

Case 2. Process $p_i$ has crashed in either $y$ or $y'$. Without loss of generality, we can assume that $p_i$ has crashed in $y$, and hence, $p_i$ is alive in $y'$. (Recall that $p_i$ has different states in both configurations.) Consider the following two non-serial runs:

**R12** is a run such that (1) the configuration at the end of round $f$ is $y$ (and hence, $p_i$ has crashed before round $f + 1$), (2) no process crashes in round $f + 1$, and (3) $q_1$ and $q_3$ crash before sending any message in round $f + 2$. No process distinct from $p_i$, $q_1$ and $q_3$ crashes after round $f$. Notice that $q_1$ cannot distinguish the configuration at the end of round $f + 1$ in $R12$ from $z$ because $q_1$ does not receive the round $f + 1$ message from $p_i$ in both runs. Thus (as in $z$) $q_1$ decides 0 at the end of round $f + 1$ in $R12$. Due to agreement, every correct process decides 0 in $R12$. Since $f \leq t - 3 \leq n - 4$, there is at least one correct process in $R12$, say $p_k$.

**R21** is a run such that (1) the configuration at the end of round $f$ is $y'$, (2) $p_i$ crashes in the send phase of round $f + 1$ such that only $q_3$ receives the message from $p_i$, and (3) $q_1$ and $q_3$ crash before sending any message in round $f + 2$. No process distinct from $p_i$, $q_1$ and $q_3$ crashes after round $f$. Notice that $q_3$ cannot distinguish the configuration at the end of round $f + 1$ in $R21$ from $z'$ because it receives the message from $p_i$ in both runs. Thus (as in $z'$) $q_3$ decides 1 at the end of round $f + 1$ in $R21$. However, $p_i$ cannot distinguish $R12$ from $R21$: at the end of round $f + 1$, the two configurations are different only at $p_i$, $q_1$ and $q_3$, and none of them send messages after round $f + 1$ in both runs. Thus (as in $R12$), $p_i$.

---

4 The runs are not serial because in round $f + 2$, two processes crash in each run.
decides 0 in R21; a contradiction with agreement.

As mentioned in the introduction, the above proposition can be seen as a generalization of the $f + 2$ round global decision lower bound. Interestingly, this proposition differs from most synchronous model consensus lower bounds in one respect. In most lower bound results on consensus in synchronous model \cite{11, 12, 13, 14}, the proofs are done in a restricted synchronous model where at most one process can crash in a round (precisely sub_scs) or where at most $k$ processes can crash in the first $k$ rounds, in order to simplify and strengthen the result. However, notice that in the proof of Proposition 3, runs $R1$ and $R2$ are not in sub_scs: two processes crash in round $f + 2$. In fact, unlike most lower bound results on consensus in synchronous model, Proposition 3 does not hold in sub_scs. For example, there is a binary consensus algorithm in sub_scs, in every failure-free run of which two processes decide at the end of round 1.

### 3 Consensus in the Eventually Synchronous Model

**System model.** Intuitively, the eventually synchronous model $ES$ is a model that is guaranteed to become synchronous, but only after an unbounded period of time. In $ES$, computation proceeds in rounds. We consider “communication-open” rounds: messages sent to correct processes are eventually received.\footnote{ES is one of the round based partial synchrony models in \cite{11}, and it can emulate an asynchronous round-based model augmented with an eventually perfect failure detector $\Phi P$ \cite{2}. In \cite{5} we specify the eventually synchronous model based on round-by-round fault detector framework \cite{9} and denote it by $RF_{\Phi P}$.}

In a round of $ES$, every process sends a message to all processes and waits for other messages sent in the round. The model notifies the processes when to stop waiting for the messages in each round. However, unlike the synchronous model, messages may be delayed (not received in the same round in which they were sent) by an arbitrary number of rounds, provided that the following conditions are met in every run: (1) messages sent to a correct process are eventually received, (2) in every round $k$, if some process $p_i$ completes the round, then $p_i$ has received at least $n - t$ messages of that round, and (3) there is an unknown round number $K$, such that, in every round $k \geq K$, for any process $p_i$, if some process completes round $k$ without receiving round $k$ message from $p_i$, then $p_i$ has crashed before completing round $k$. We say that a run in $ES$ is synchronous if in every round $k \geq 1$, for any process $p_i$, if some process completes round $k$ without receiving round $k$ message from $p_i$, then $p_i$ has crashed before completing round $k$.

From \cite{8}, it is easy to see that, for every consensus algorithm, and for every $0 \leq f \leq t$ ($t \geq 1$), there is a run of the algorithm with at most $f$ failures which takes an arbitrary number of rounds for a local decision. Hence, we define $l_f$ and $g_f$ in this model as bounds on the synchronous runs of the algorithm with at most $f$ failures.

**Local decision lower bound in eventually synchronous model.** In synchronous runs of any consensus algorithm in $ES$ we show that there is a run with at most $f$ failures in which no correct process decides before round $f + 2$; i.e., the local decision lower bound is identical to the global decision lower bound. This one round difference between local decisions in SCS and that of synchronous runs in ES, can be seen as a price paid by algorithms in ES to tolerate an “unreliable model” \cite{5}.

The subsystem $sub_{es}$ consists of all runs in $ES$. Let $A$ be any consensus algorithm in $sub_{es}$, Synchronous configurations are the configurations of synchronous runs of $A$. For any synchronous configuration $C$ at round $k$ of $A$, we define $R(C)$ as the synchronous run in which (1) the configuration at round $k$ is $C$ and (2) no process crashes after round $k$. We denote by $Val(C)$ the decision value of correct processes in $R(C)$. As every run in $sub_{scs}$\footnote{The subsystem used in the proof of Proposition 3.} is a synchronous run in $sub_{es}$, from Claim SCS1 we
immediately have:

Claim ESI. For any consensus algorithm in subes, there are two synchronous configurations, \( y \) and \( y' \), at the end of round \( f \) (\( 0 \leq f \leq t \)), such that, (1) at most \( f \) processes have crashed in each configuration, (2) the configurations differ at exactly one process, and (3) \( \text{Val}(y) = 0 \) and \( \text{Val}(y') = 1 \).

Proposition 4 Let \( 1 \leq t \leq n - 1 \). For every consensus algorithm in ES and every \( 0 \leq f \leq t - 3 \), there is a synchronous run with at most \( f \) failures where no correct process decides before round \( f + 2 \).

Proof: Suppose by contradiction that there is a binary consensus algorithm \( A \) in subes and an integer \( f \) such that \( 0 \leq f \leq t - 3 \) and in every synchronous run of \( A \) with at most \( f \) failures, some correct process decides by round \( f + 1 \). From Claim ESI, we know that at the end of round \( f \) there are two synchronous configurations of \( A \), \( y \) and \( y' \), such that (1) at most \( f \) processes have crashed in each configuration, (2) the configurations differ at exactly one process, say \( p_i \), and (3) \( \text{Val}(y) = 0 \) and \( \text{Val}(y') = 1 \). Let \( z \) and \( z' \) denote the configurations at the end of round \( f + 1 \) in synchronous runs \( R(y) \) and \( R(y') \), respectively.

From our initial assumption on \( A \), in \( z \), there is at least one alive process, say \( q_1 \), which has decided 0. Similarly, in \( z' \), there is at least one alive process, say \( q_3 \), which has decided 1. There are three cases to consider.

Case 1. \( p_i \notin \{q_1, q_3\} \). This case is exactly similar to the case in the proof of Proposition 3. We can derive a contradiction by constructing the same runs \( R_1, R_2, R_{12}, \) and \( R_{21} \).

Case 2. \( p_i \in \{q_1, q_3\} \) and \( p_i \) is alive in both \( y \) and \( y' \). Notice that if \( p_i = q_1 \) then \( R_1 \) is not in ES; \( p_i \) cannot crash in the send phase of round \( f + 1 \), and decide at the end of round \( f + 1 \). (Similarly, if \( p_i = q_3 \) then \( R_2 \) is not in ES.) Thus we construct non-synchronous runs of \( A \) to show the contradiction. Without loss of generality we can assume that \( p_i = q_1 \). (Note that the proof holds even if \( p_i = q_1 = q_3 \).)

Consider the following synchronous run \( R_3 \) and two non-synchronous runs, \( R_4 \) and \( R_5 \).

**R3** is a run such that (1) the configuration at the end of round \( f \) is \( y \), (2) \( p_i \) crashes in round \( f + 1 \) before sending any message, (3) if \( q_3 \neq p_i \) then \( q_3 \) crashes before sending any message in round \( f + 2 \) and every message sent by \( q_3 \) in round \( f + 1 \) is received in the same round, and (4) no process distinct from \( p_i \) and \( q_3 \) crashes after round \( f \). Since \( t \leq n - 1 \), there is at least one correct process in \( R_3 \), say \( p_i \). Suppose \( p_i \) decides \( v \in \{0, 1\} \) in some round \( K' \geq f + 1 \).

**R4** is a run such that (1) the configuration at the end of round \( f \) is \( y \), (2) \( p_i \) crashes before sending any message in round \( f + 2 \), such that, in round \( f + 1 \), every message from \( p_i \) to any process distinct from \( p_i \) and \( q_3 \) is delayed until round \( K' + 1 \), (3) if \( q_3 \neq p_i \) then \( q_3 \) crashes before sending any message in round \( f + 2 \) and every message sent by \( q_3 \) in round \( f + 1 \) is received in the same round, and (4) no process distinct from \( p_i \) and \( q_3 \) crashes after round \( f \). Notice that \( p_i \) cannot distinguish the configuration at the end of round \( f + 1 \) in \( R_4 \) from \( z \) (because \( p_i \) receives its own message in round \( f + 1 \)), and thus, \( p_i \) decides \( 0 \) at the end of round \( f + 1 \) in \( R_4 \). However, \( p_i \) cannot distinguish the configuration at the end of round \( K' \) in \( R_4 \) from that in \( R_3 \) because (1) at the end of round \( f \) the two runs are different only at \( p_i \), and every round \( f + 1 \) messages from \( p_i \) to processes distinct from \( p_i \) and \( q_3 \) are delayed until round \( K' + 1 \), and (2) \( p_i \) and \( q_3 \) do not send messages after round \( f + 1 \). Thus (as in \( R_3 \)) \( p_i \) decides \( v \) at the end of round \( K' \).

**R5** is a run such that (1) the configuration at the end of round \( f \) is \( y' \), (2) \( p_i \) crashes before sending any message in round \( f + 2 \), such that, in round \( f + 1 \), every message from \( p_i \) to any process distinct from \( p_i \)

---

7To see that \( p_i \) cannot decide before round \( f + 1 \) in \( R_3 \), notice that the state of \( p_i \) at the end of round \( f \) is the same in runs \( R(y) \), \( R(y') \) and \( R_3 \). If \( p_i \) decides \( v \) before round \( f + 1 \) in \( R_3 \) then it also decides \( v \) in \( R(y) \) and \( R(y') \). However, \( \text{Val}(y) \neq \text{Val}(y') \).
and \( q_3 \) is delayed until round \( K' + 1 \). (3) if \( q_3 \neq p_i \) then \( q_3 \) crashes before sending any message in round \( f + 2 \) and every message sent by \( q_3 \) in round \( f + 1 \) is received in the same round, and (4) no process distinct from \( p_i \) and \( q_3 \) crashes after round \( f \). Notice that \( q_3 \) cannot distinguish the configuration at the end of round \( f + 1 \) in \( R5 \) from \( z' \) (because \( q_3 \) receives the message from \( p_i \) in round \( f + 1 \)), and thus, \( q_3 \) decides 1 at the end of round \( f + 1 \) in \( R5 \). However, \( p_i \) cannot distinguish the configuration at the end of round \( K' \) in \( R5 \) from that in \( R3 \) because, (1) at the end of round \( f \) the two runs are different only at \( p_i \), and all round \( f + 1 \) message from \( p_i \) to processes distinct from \( p_i \) and \( q_3 \) are delayed until round \( K' + 1 \), and (2) \( p_i \) and \( q_3 \) do not send messages after round \( f + 1 \). Thus (as in \( R3 \)) \( p_i \) decides \( v \) at the end of round \( K' \).

It is easy to see that either \( R4 \) or \( R5 \) violates agreement: \( p_i \) decides \( v \) in both runs, however, \( p_i \) decides 0 in \( R4 \) and \( q_3 \) decides 1 in \( R5 \).

Case 3. \( p_i \in \{ q_1, q_3 \} \) and \( p_i \) has crashed in either \( y \) or \( y' \). Notice that the case \( p_i = q_1 = q_3 \) is not possible because, in that case, \( p_i \) is alive in \( z \) and \( z' \), and hence in \( y \) and \( y' \). We show the contradiction for the case when \( p_i = q_1 \neq q_3 \). (The contradiction for \( p_i = q_3 \neq q_1 \) is symmetric.)

Since, \( p_i = q_1 \), \( p_i \) is alive in \( z \), and hence, alive in \( y \). Thus \( p_i \) has crashed in \( y' \). Consider the following non-synchronous run.

**R6** is a run such that (1) the configuration at the end of round \( f \) is \( y \), (2) \( p_i \) crashes before sending any message in round \( f + 2 \), such that, in round \( f + 1 \), every message from \( p_i \) to a process distinct from \( p_i \), is delayed until round \( f + 2 \), and (3) no process distinct from \( p_i \) crashes after round \( f \). At the end of round \( f + 1 \) in \( R6 \), \( p_i = q_1 \) cannot distinguish the configuration from \( z \) (because \( p_i \) receives its own message in round \( f + 1 \)), and therefore, decides 0 at the end of round \( f + 1 \) in \( R6 \). However, \( q_3 \) does not receive the round \( f + 1 \) message from \( p_i \) in \( R6 \) (the message is delayed until the next round), and furthermore, even in \( z' \), \( q_3 \) does not receive the round \( f + 1 \) message from \( p_i \) (because \( p_i \) has crashed in \( y' \)). Thus \( q_3 \) cannot distinguish the configuration at the end of round \( f + 1 \) in \( R6 \) from \( z' \), and hence, decides 1 in \( R6 \); a contradiction with agreement.

A closer look at the proof of Proposition 4 reveals that the non-synchronous runs we construct (\( R4 \), \( R5 \), and \( R6 \)) have the following “weak synchrony” property: if a message from any process \( p_i \) delayed in round \( k \) then \( p_i \) crashes before sending any message in round \( k + 1 \). It is easy to see that such runs are also valid runs in synchronous send-omission model as well as in an asynchronous round-by-round model enriched with a Perfect failure detector. Thus the \( f + 2 \) local decision lower bound in synchronous runs also extend to these two models.

**A matching algorithm in the eventually synchronous model.** Figure 1 gives a consensus algorithm \( A_{f+2} \) in the eventually synchronous model which matches the \( f + 2 \) round global decision lower bound (and hence, matches the local decision bound) in synchronous runs. Namely, the algorithm satisfies the following property: (Fast Early Decision) For \( 0 \leq t < n/2 \), in every synchronous run of \( A_{f+2} \) with at most \( f \) failures \( (0 \leq f \leq t) \), every process which decides, decides by round \( f + 2 \).

For simplicity of presentation, \( A_{f+2} \) assumes an independent consensus algorithm \( C \),\(^8\) accessed by procedure \( \text{propose}_C(\ast) \). The fast decision property is achieved by \( A_{f+2} \) regardless of the time complexity of \( C \). More precisely, our algorithm assumes: (1) the model \( ES \) with \( 0 \leq t < n/2 \), (2) messages sent by a process to itself is received in the same round in which it is sent, (3) an independent consensus algorithm \( C \) in \( ES \), and (4) the set of proposal values in a run is a totally ordered set, e.g., every process \( p_i \) can tag its proposal value with its index \( i \) and then the values can be ordered based on this tag.

The processes invoke \( \text{propose}(\ast) \) with their respective proposal values, and the procedure progresses in round. Every process \( p_i \) maintains three primary variables:

---

\(^8\)This algorithm can be any \( \Diamond P \)-based or \( \Diamond S \)-based consensus algorithm (e.g., the one based on \( \Diamond S \) in [3]) transposed to \( ES \).
at process $p_i$

1: **procedure** propose($v_i$)  
2: start Task 1; start Task 2  
3: **Task 1**  
4: $state_i \leftarrow sync1$; $est_i \leftarrow v_i$; $halt_i \leftarrow \emptyset$  
5: **for** $1 \leq k_i \leq t + 2$  
6: **send**($k_i$, $est_i$, $state_i$, $halt_i$) to all  
7: **wait** **until** received messages in this round  
8: **if** received($k_i$, $est'$, **DECIDE**, $*$) **then**  
9: **send**($k_i + 1$, $est'$, **DECIDE**, $\emptyset$) to $\Pi \backslash p_i$; return($est'$)  
10: **if** $state_i \in \{sync1, sync2\}$ **then**  
11: **halt**$ \leftarrow \text{halt}_{i} \cup \{p_j \mid \{p_j \text{ received}(k_i, *, \text{NSync}, *) \text{ from } p_j\} \text{ or } (p_j \text{ received}(k_i, *, *, \text{halt}_{j}) \text{ from } p_j \text{ s.t. } p_i \in \text{halt}_{j}) \text{ or } (p_j \text{ did not receive round } k_i \text{ message from } p_j)\}$  
12: $msgSet_i \leftarrow \{m \mid m \text{ is a round } k_i \text{ message received from } p_j \notin \text{halt}_{i}\}$  
13: $est_i \leftarrow \text{Min} \{est \mid (k_i, est, *, *) \in msgSet_i\}$  
14: **if** ($state_i = \text{sync2}$) **and** ($|\text{halt}_{i}| \leq t$) **and** ($state = \text{sync2}$ for every message in $msgSet_i$) **then**  
15: **send**($k_i + 1$, $est_i$, **DECIDE**, $\emptyset$) to $\Pi \backslash p_i$; return($est_i$)  
16: **if** $|\text{halt}_{i}| \leq k_i - 1$ **then**  
17: $state_i \leftarrow \text{sync2}$  
18: **if** $k_i \leq |\text{halt}_{i}| \leq t$ **then**  
19: $state_i \leftarrow \text{sync1}$  
20: **if** $|\text{halt}_{i}| > t$ **then**  
21: $state_i \leftarrow \text{NSync}$  
22: **if** ($state = \text{NSync}$) **and** (received($k_i$, $est'$, $\text{sync2}$, $*$)) **then**  
23: $est_i \leftarrow est'$  
24: return(propose$_C$(est$_i$))  
25: **Task 2**  
26: **upon** receiving ($k'$, $est'$, **DECIDE**, $*$) **do**  
27: **when** $k_i = k' + 1$: **send**($k_i$, $est'$, **DECIDE**, $\emptyset$) to $\Pi \backslash p_i$; return($est'$)  

Figure 1: A Consensus algorithm $A_{f+2}$ in $ES$
• \(\text{STATE}_i\) at the end of a round denotes the fact that \(p_i\) considers (a) the run to be non-synchronous (\(\text{STATE} = \text{NSYNC}\)), (b) the run to be synchronous but \(p_i\) cannot decide at the next round (\(\text{STATE} = \text{SYNC1}\)), (c) the run to be synchronous with a possibility of deciding at the next round (\(\text{STATE} = \text{SYNC2}\)).

• \(\text{est}_i\) is the estimate of the possible decision value, and roughly speaking, the minimum value seen by \(p_i\).

• \(\text{Halt}_i\) is a set of processes. At the end of a round, \(\text{Halt}_i\) contains \(p_j\) if any of the following holds in the current round or in a lower round: (1) \(p_i\) did not receive a message from \(p_j\), (2) \(p_i\) receives a messages from \(p_j\) with \(\text{state} = \text{NSYNC}\), or (3) \(p_i\) receives a messages from \(p_j\) with \(p_i \in \text{Halt}_j\).

In the first \(t+2\) rounds, the processes exchange these three variables and then updates their variable depending on the messages received. We say that a message is a state \(S'\) message, if it is sent with \(\text{state} = S'\). Figure 2 (optional appendix A.2) shows the rules for updating \(\text{state}\) in each round \(k\). At the end of round \(t+2\), if a process has not yet decided, then it invokes the underlying consensus \(C\) with its \(\text{est}\) as the proposal value. The algorithm ensures the following elimination property: if a process completes some round \(k < t+2\) with \(\text{state} = \text{SYNC2}\) and \(\text{est} = \text{est}'\) and no process decides in round \(k\) or in a lower round, then every process which completes round \(k\) with \(\text{state} = \text{SYNC1}\) has \(\text{est} \geq \text{est}'\), and every process which completes round \(k\) with \(\text{state} = \text{SYNC2}\) has \(\text{est} = \text{est}'\). (Processes which complete round \(k\) with \(\text{state} = \text{NSYNC}\) may have \(\text{est} < \text{est}'\).)

We now briefly discuss the agreement property of our algorithm assuming the elimination property. (We give a detailed proof of correctness in optional appendix A.2.) If every process which decides, decides at a round higher than \(t+2\) then agreement follows from the corresponding property of algorithm \(C\). Consider the lowest round \(k' \leq t+2\) in which some process \(p_i\) decides, say \(d\). From line 14, at least \(n-t\) processes (a majority) completes round \(k'-1\) with \(\text{state} = \text{SYNC2}\), and hence, every process which completes round \(k'\) receives a message with \(\text{state} = \text{SYNC2}\) and \(\text{est} = d\). From the elimination property, processes which complete round \(k'-1\) with \(\text{est} < d\) have \(\text{state} = \text{NSYNC}\). Notice that while updating \(\text{est}\) for the next round, processes with \(\text{state} = \text{SYNC1}\) or \(\text{state} = \text{SYNC2}\), ignore messages from processes with \(\text{state} = \text{NSYNC}\) (line 11, line 12). Therefore, every process with \(\text{state} = \text{SYNC1}\) or \(\text{state} = \text{SYNC2}\), updates \(\text{est}\) to \(d\) in round \(k'\) (line 13). Since a majority of processes sends round \(k'\) messages with \(\text{est} = d\) and \(\text{state} = \text{SYNC2}\), every process which completes round \(k'\) with \(\text{state} = \text{NSYNC}\) receives such a message and updates \(\text{est}\) to \(d\) (line 22). Consequently, every process which completes round \(k'\), does so with \(\text{est} = d\), and no value distinct from \(d\) can be decided at round \(k'\) or at a higher round.

References


Keidar I. and Rajsbaum S., Open Questions on Consensus Performance in Well-Behaved Runs. *Future Directions in Distributed Computing (FuDiCo)*, Springer Verlag (LNCS 2584).


### A Optional Appendix

#### A.1 Proofs of Lemma 1 and Proposition 2

All discussions in this section are in the context of subseq: the model is synchronous and at most one process can crash in every round. We denote a one round extension of a round $k$ configuration $C$ as follows: for $1 \leq i \leq n$ and $S \subseteq \Pi$, $C(i,S)$ denotes the configuration reached by crashing $p_i$ in round $k+1$ such that any process $p_j$ does not receive a round $k+1$ message from $p_i$ if any of the following holds: (1) $p_j = p_i$, (2) $p_j$ is crashed in $C$, or (3) $p_j \in S$; $C(0,\emptyset)$ denotes the one round extension of $C$ in which no process crashes. Obviously, $(i,S)$ for $i > 0$ and $S \subseteq \Pi$ is an applicable extension to $C$ if at most $t-1$ processes have crashed in $C$ and $p_i$ is alive in $C$.

A layer $L(C)$ is defined as $\{C(i,S) | i \in \Pi, S \subseteq \Pi, (i,S)$ is applicable to $C\}$. For a set of configurations $SC$ at the same round, $L(SC)$ is another set of configurations defined as $\cup_{C \in SC} L(C)$. $L^k(SC)$ is recursively defined as follows: $L^0(SC) = SC$ and for $k > 0$, $L^k(SC) = L(L^{k-1}(SC))$. 

10
Two configurations $C$ and $D$ at the same round are similar, denoted $C \sim D$, if they are identical or there exists a process $p_j$ such that (1) $C$ and $D$ are identical except at $p_j$, and (2) there exists a process $p_i \neq p_j$ that is alive in both $C$ and $D$. A set of configurations $SC$ is similarly connected if, for every $C, D \in SC$ there are states $C = C_0, \ldots, C_m = D$ such that $C_i \sim C_{i+1}$ for all $0 \leq i < m$.

All our lower bound proofs start from the following lemma in $\text{sub}_\text{scs}$ or its variants. We present a proof of the lemma, slightly modified from [12].

**Lemma 1** For every binary consensus algorithm in $\text{sub}_\text{scs}$ and $0 \leq k \leq t$, there are two configurations $y, y' \in L^k(\text{Init})$ such that (1) at most $k$ processes have crashed in each configuration, (2) the configurations differ at exactly one process, and (3) $\text{val}(y) = 0$ and $\text{val}(y') = 1$.

**Proof:** The proof of the lemma proceeds through two claims:

**Claim 1(a):** Let $SC = L^0(SC)$ be a similarly connected set of configurations in which no process has crashed, then for all $k \leq t$, $L^k(SC)$ is a similarly connected set of states in which no more than $k$ processes are crashed in any configuration.

**Proof:** (A simple modification of the proof of [12,9]) The proof is by induction on round number $k$. The base case $k = 0$ is immediate. For the inductive step, assume that $L^{k-1}(SC)$ is similarly connected and in every configuration at most $k - 1$ processes have crashed. Notice that in every extension which is applicable to any configuration in $L^{k-1}(SC)$, at most one new process can crash. Therefore, in every configuration in $L^k(SC)$ at most $k$ processes have crashed.

We now show that for any configuration $C \in L^{k-1}(SC)$, $L(C)$ is similarly connected. Consider any two configurations in $L(C)$, $C1 = C.(i, S1)$ and $C2 = C.(j, S2)$, where $S1, S2 \subseteq T$, and $p_i$ and $p_j$ are alive in $C$. We will show that $C1$ and $C.(0, \emptyset)$ are similarly connected. Using the same procedure, we can show that $C2$ and $C.(0, \emptyset)$ are similarly connected, thus showing that $C1$ and $C2$ are similarly connected.

$C.(i, \emptyset) \sim C(0, \emptyset)$ since the configurations only differ at $p_i$. If $S1 = \emptyset$ then we are done. Otherwise, let $S1 = \{q_1, q_2, \ldots, q_m\}$. For $1 \leq l \leq m$, let $S1_l = \{q_1, \ldots, q_l\}$, and $S1_0 = \emptyset$. For $0 \leq l < m$, $C.(i, S1_l) \sim C.(i, S1_{l+1})$ because the two configurations differ only at $q_{l+1}$. Thus $C(i, \emptyset) = C.(i, S1_0)$ and $C1 = C.(i, S1_m)$ are similarly connected.

It remains to be shown that if $C \sim D$ and $C, D \in L^{k-1}(SC)$ then there are configurations $C' \in L(C)$ and $D' \in L(D)$ which are similar. Let $p_i$ be the process such that $C$ and $D$ are different only at $p_i$. Then, configurations $C.(i, \Pi)$ and $D.(i, \Pi)$ are identical because no process receives message from $p_i$ in round $k + 1$.

**Claim 1(b):** In a similarly connected set $SC$ of states, if there are states $C$ and $D$ such that $\text{val}(C) \neq \text{val}(D)$, then there are two states $C1, D1 \in SC$ such that (1) $C1 \sim D1$ and (2) $\text{val}(C1) \neq \text{val}(D1)$.

**Proof:** Suppose, by contradiction, in a similarly connected set $SC$ of states there are two states $C$ and $D$ such that $\text{val}(C) \neq \text{val}(D)$ and for every pair of similar states $C1, D1 \in SC$, $\text{val}(C1) = \text{val}(D1)$. Since $C$ and $D$ are similarly connected, there exist a set of states, $C = C_0, C_1, \ldots, C_m = D$, such that, for $0 \leq l < m$, $C_l \sim C_{l+1}$. From our initial assumption, and a simple induction, it follows that $\text{val}(C_0) = \text{val}(C_1) = \ldots = \text{val}(C_m)$; a contradiction.

**Proof of Lemma 1 continued.** We use the well-known lemma that $\text{Init}$ is similarly connected [8, 12]. Thus from Lemma 1, $L^k(\text{Init})$ is similarly connected. Consider the configuration $C$ at round $k$ of the
failure-free run in which all processes propose 1. Obviously, \( C \in L^k(\text{Init}) \), and from consensus validity, \( \text{val}(C) = 1 \). Similarly, consider the configuration \( D \) at round \( k \) in the failure-free run in which all processes propose 0. We have, \( D \in L^k(\text{Init}) \) and \( \text{val}(D) = 0 \). Thus from Claim 1(a) and Claim 1(b), at the end of round \( k \) there exists two configurations \( y \) and \( y' \) such that (1) at most \( k \) processes have crashed in each configuration, (2) the configurations are similar, and (c) \( \text{val}(y) = 0 \) and \( \text{val}(y') = 1 \). Since, \( \text{val}(y) \neq \text{val}(y') \), the configurations cannot be identical. Thus they differ at exactly one process.

\[ \square \]

**Proposition 2** Let \( 1 \leq t \leq n - 1 \). For every consensus algorithm and every \( 0 \leq f \leq t - 1 \), there is a run with \( f \) failures in which no correct process decides before round \( f + 1 \).

**Proof:** Suppose by contradiction that there is a binary consensus algorithm \( A \) in \( \text{sub}_{\text{scs}} \) and a round number \( f \) such that \( 0 \leq f \leq t - 1 \), and in every run with at most \( f \) failures, some correct process decides before round \( f + 1 \). Consider the set of configurations of \( A \) at the end of round \( f \): \( L_f^f(\text{Init}) \). From our assumption it follows that in every configuration \( x \in L_f^f(\text{Init}) \), there is an alive process \( p_j \) which has already decided. (Otherwise, since every correct process in \( r(x) \) is an alive process in \( x \), \( r(x) \) is a run with \( f \) crashes in which no correct process decides before round \( f + 1 \).) Furthermore, \( p_j \) decides \( \text{val}(x) \) in \( x \) because \( p_j \) is a correct process in \( r(x) \).

From Lemma 1 we know that there are two configurations \( y, y' \in L_f^f(\text{Init}) \) such that (1) at most \( f \) processes have crashed in each configuration, (2) the configurations differ at exactly one process, say \( p_i \), and (3) \( \text{val}(y) = 0 \) and \( \text{val}(y') = 1 \). From our assumption it follows that, in \( y \), there is an alive process \( q_1 \) which has decided 0, and, in \( y' \), there is an alive process \( q_2 \) which has decided 1. There are two cases to consider:

1. \( q_1 \neq p_i \): As \( y \) and \( y' \) are identical at all processes different from \( p_i \), in \( y' \), \( q_1 \) is alive and has decided 0. Thus in \( r(y') \), \( q_1 \) is a correct process and decides 0. However, in \( r(y') \) every correct process decides \( \text{val}(y') = 1 \); a contradiction.

2. \( q_1 = p_i \): We distinguish two subcases:

   - \( q_2 = p_i \): Thus \( p_i = q_1 = q_2 \), and hence, \( p_i \) is alive in \( y \) and \( y' \). Consider a run \( r1 \) which extends \( y \) and in which \( p_i \) crashes before sending any message in round \( f + 1 \); i.e., \( r1 = r(y, (i, II)) \). (Recall that \( f \leq t - 1 \)). As \( p_i \) has decided 0 in \( y \), from agreement, it follows that every correct process decides 0 in \( r1 \). Since \( t < n \), there is at least one correct process, say \( p_i \) in \( r1 \). Now consider a run \( r2 \) which extends \( y' \) and in which \( p_i \) crashes before sending any message in round \( f + 1 \); i.e., \( r2 = r(y', (i, II)) \). Notice that no correct process can distinguish between \( r1 \) and \( r2 \): no alive process which is distinct from \( p_i \) can distinguish \( y \) from \( y' \), and \( p_i \) crashes before sending any message in round \( f + 1 \). Thus every correct process decides the same value in \( r1 \) and \( r2 \), in particular \( p_i \) decides 0 in \( r2 \). However, \( p_i = q_2 \) decides 1 in \( r2 \); a contradiction with agreement.

   - \( q_2 \neq p_i \): Then, \( q_2 \) has the same state in \( y \) and \( y' \). Thus in \( y \), \( q_2 \) is alive and has decided 1. In any extension of \( y \), \( p_i = q_1 \) has decided 0 and \( q_2 \) has decided 1; a contradiction with agreement.

\[ \square \]

**A.2 Correctness of the consensus algorithm in Figure 1**

The validity and termination properties of \( A_{f+2} \) easily follow from the corresponding properties of the underlying consensus algorithm \( C \). We focus here on the agreement and the fast early decision properties. For presentation simplicity, we introduce the following notation. Given a variable \( \text{val}_i \) at process
$p_i$, we denote by $val_i[k]$ ($1 \leq k \leq t + 2$) the value of the variable $val_i$ immediately after the completion of round $k$; $val_i[0]$ denotes the value of $val_i$ immediately after completing line 4 (i.e., before sending any message in round 1). We assume that there is a symbol $\text{undefined}$ which is distinct from any possible value of the variables in the algorithm $A_{f+2}$. If $p_i$ crashes before completing round $k$, then $val_i[k] = \text{undefined}$; if $p_i$ crashes before completing line 4, then $val_i[0] = \text{undefined}$. In other words, if for any variable $val$, $val_i[k] \neq \text{undefined}$ then $p_i$ has completed round $k$.

**Lemma 5:** Consider a process $p_i$ and a round $1 \leq k \leq t + 2$, such that $\text{state}_i[k] \in \{\text{SYNC1, SYNCG2}\}$ ($p_i$ completes round $k$ with $\text{state} = \text{SYNC1}$ or $\text{state} = \text{SYNC2}$). Let $\text{sender}\text{MS}_i[k]$ be the set of processes which have sent the messages in $\text{msgSet}_i[k]$. Then, $\text{sender}\text{MS}_i[k] = \Pi - Halt_i[k]$.

**Proof:** Process $p_i$ completes round $k$ with $\text{state} = \text{SYNC1}$ or $\text{state} = \text{SYNC2}$, and hence, updates $\text{Halt}$ and $\text{msgSet}$ at line 11 and line 12 of round $k$, respectively. Consider any process $p_m \in \Pi$. There are two exhaustive and mutually exclusive cases regarding the message from $p_m$ to $p_i$ in round $k$ ($1 \leq k \leq t + 2$):
- If $p_i$ does not receive the messages from $p_m$ in round $k$, then from the third condition in line 11, $p_m \in Halt_i[k]$, and from line 12, $p_m \in \text{sender}\text{MS}_i[k]$.
- If $p_i$ receives the round the message from $p_m$ in round $k$, then from line 12, $p_m \in \text{sender}\text{MS}_i[k]$ iff $p_m \notin Halt_i[k]$.

**Lemma 6.** (Agreement) No two processes decide differently.

**Proof.** If no process ever decides then the lemma is trivially true. If every process which decides, decide in algorithm $C$, then the lemma follows from the agreement property of $C$. Thus we consider the case where some process decides within the first $t + 2$ rounds. Consider the lowest round number in which some process decides, say round $k' + 1$ ($\leq t + 2$). It is easy to see that, if some process decides $v$ in line 9 or line 27, then some other process has decided $v$ in a lower round. Thus some process decides at line 15 of round $k' + 1$. We claim the following:

**Claim 6.1:** (Elimination) If there are two processes $p_x$ and $p_y$ such that $\text{state}_x[k'] \in \{\text{SYNC1, SYNCG2}\}$ and $\text{state}_y[k'] = \text{SYNC2}$ then $\text{est}_x[k'] \geq \text{est}_y[k']$.

**[Proof of Lemma 6 cont.]** We now complete the proof of agreement assuming Claim 6.1. We later give the proof of Claim 6.1. Suppose that some process $p_w$ decides $d$ at line 15 of round $k' + 1$. From line 14 it follows that $p_w$ has completed round $k'$ with $\text{state} = \text{SYNC2}$ and $\text{est} = d$. Consider another process $p_u$ which completes round $k'$ with $\text{state} = \text{SYNC2}$ and $\text{est} = d'$. In Claim 6.1, if we substitute $p_x$ by $p_w$ and $p_y$ by $p_u$ then, $d \geq d'$. Similarly, if we substitute $p_x$ by $p_u$ and $p_y$ by $p_w$ then, $d' \geq d$. Thus $d = d'$, and any process which completes round $k'$ with $\text{state} = \text{SYNC2}$, does so with $\text{est} = d$. Notice that every process which decides at line 15 in round $k' + 1$, completes round $k'$ with $\text{state} = \text{SYNC2}$ and decides on its own $\text{est}$ (line 14, line 15). Thus every process which decides in round $k + 1$ decides $d$. It remains to be shown that no process decides a different value in a higher round.

From line 14 we have $|\text{Halt}_w[k' + 1]| \leq t$, and hence, Lemma 5 implies that $\text{msgSet}_w[k' + 1]$ contains at least $n - t$ messages, i.e., messages from a majority of processes. Furthermore, the last condition in line 14 requires that all messages in $\text{msgSet}_w[k' + 1]$ has $\text{state} = \text{SYNC2}$. Applying Claim 6.1, we have, in round $k' + 1$, messages from a majority of processes have $\text{state} = \text{SYNC2}$ and $\text{est} = d$, and every message with $\text{state} = \text{SYNC1}$ has $\text{est} \geq d$.

Now consider the $\text{est}$ value of any process $p_i$ at the end of round $k' + 1$. If $\text{state}_i[k' + 1] = \text{NSYNC}$, then $p_i$ has received at least one message with $\text{state} = \text{SYNC2}$ and $\text{est} = d$ (because a majority of processes send such messages, and in every round, $p_i$ receives messages from a majority of processes), and therefore, updates its $\text{est}$ to $d$ (line 22). If $\text{state}_i[k' + 1] \neq \text{NSYNC}$ then $\text{Halt}_i[k' + 1] \leq t$ (line 20). Therefore, $\text{msgSet}_i[k' + 1]$ contains at least $n - t$ messages (Lemma 5). Furthermore, $\text{msgSet}_i[k' + 1]$
Figure 2: Rules for updating state at round $k$ for process $p_i$ (algorithm $A_{f+2}$)

contains no message with $\text{state}_i[k'] + 1 = \text{NSYN}$ (line 11, line 12). Therefore, from Claim 6.1, every message in $\text{msgSet}_i[k' + 1]$ has $\text{est} \geq d$ and at least one message with $\text{state} = \text{SYNC2}$ and $\text{est} = d$ (because a majority of processes sent messages with $\text{state} = \text{SYNC2}$ and $\text{est} = d$ in round $k' + 1$). Therefore, at line 13, $p_i$ updates $\text{est}$ to $d$.

Thus every process which completes round $k' + 1$ updates its $\text{est}$ to $d$, and every process which decides at line 15 of round $k' + 1$, decides $d$. Now notice that the $\text{est}$ value of a process at the end of some round $k$ is $\text{est}$ value of some process at the end of round $k - 1$ ($1 < k \leq t + 2$). Therefore, for round $k$ such that $k' + 1 \leq k \leq t + 2$, no process completes round $k$ with $\text{est}$ different from $d$ (D1). Notice that if a process decides $d'$ at line 15 of round $k$ such that $k' + 1 \leq k \leq t + 2$, then its $\text{est}$ is $d'$ at the end of round $k - 1$. Therefore, from D1, $d' = d$. Furthermore, proposal value for the underlying consensus algorithm $C$ at a given process $p_i$ is the $\text{est}$ value of $p_i$ at the end of round $t + 2$. Hence, from D1, every proposal value for algorithm $C$ is $d$, and from validity property of $C$, every process which decides in algorithm $C$, decides $d$.

**Claim 6.1:** If $k' + 1 \leq t + 2$ is the lowest round in which some process decides then: if there are two processes $p_x$ and $p_y$ such that $\text{state}_x[k'] = \{\text{SYNC1,SYNC2}\}$ and $\text{state}_y[k'] = \text{SYNC2}$ then $\text{est}_x[k'] \geq \text{est}_y[k']$.

**Proof:** Suppose by contradiction that there are two processes $p_x$ and $p_y$ such that

**Assumption A1:** $\text{state}_x[k'] = \{\text{SYNC1,SYNC2}\}$, $\text{state}_y[k'] = \text{SYNC2}$, $\text{est}_x[k'] = c$, $\text{est}_y[k'] = d$, and
c < d.
We show Claims 6.1.1 to 6.1.7 based on the definition of $k'$ and the assumption A1. Claim 6.1.4 contradicts Claim 6.1.7, which completes the proof of Claim 6.1 by contradiction.

Let us define the following sets for $1 \leq k \leq k' + 1$:

- $C[k] = \{p_i|est_i[k] \leq c\}$ (Set of processes which complete round $k$ with $est \leq c$).
- $crashed[k]$ = set of processes which crashes before completing round $k$.
- $NSYN[k] = \{p_i|state_i[k] = NSYNC\}$.
- $Z[k] = C[k] \cup crashed[k] \cup NSYN[k]$.

Additionally, let us define, $C[0]$ to be the set of processes whose proposal value is less than or equal to $c$, $crashed[0]$ to be the set of processes which crash before sending any message in round 1, $NSYN[0] = \emptyset$, and $Z[0] = C[0] \cup crashed[0] \cup NSYN[0]$. We make the following observation:

Observation A2: $|C[0]| \geq 1$, and hence, $|Z[0]| \geq 1$. Otherwise, if every process proposed a value greater than $c$, then $est_x[k'] > c$ (contradicts A1).

Claim 6.1.1: (a) For $0 \leq k \leq k' - 1$, $(crashed[k] \cup NSYN[k]) \subseteq (crashed[k + 1] \cup NSYN[k + 1])$.
(b) For $0 \leq k \leq k' - 1$, if $p_i \notin (NSYN[k] \cup crashed[k])$ then $p_i$ sends messages with $state \in \{SYNC1, SYNC2\}$ in round $k$ and in the lower rounds.

Proof: (a) Suppose by contradiction that there is process $p_i$ such that $p_i \in crashed[k] \cup NSYN[k]$ and $p_i \notin crashed[k + 1] \cup NSYN[k + 1]$. Obviously, $crashed[k] \subseteq crashed[k + 1]$, and hence, $p_i \notin crashed[k + 1] \cup NSYN[k + 1]$ implies $p_i \notin crashed[k]$. Then $p_i \in crashed[k] \cup NSYN[k]$ implies $p_i \notin NSYN[k]$, i.e., $p_i$ completes round $k$ with $state = NSYNC$. Notice that by the definition of $k'$ (i.e., $k' + 1$ is the lowest round in which some process decides), $p_i$ does not decide in round $k + 1$. Thus the state of $p_i$ remains NSYNC at the end of round $k + 1$, i.e., $p_i \in NSYN[k + 1]$; a contradiction.

(b) If $p_i \notin (NSYN[k] \cup crashed[k])$, then from 6.1.1.a, it follows that, $p_i \notin (NSYN[k + 1] \cup crashed[k + 1])$ for $0 \leq k \leq k'$; i.e., $p_i$ completes every round lower than round $k$ with $state \neq NSYNC$. Thus $p_i$ cannot send message with $state \neq NSYNC$ in round $k$ or in a lower round.

Claim 6.1.2: $Z[k] \subseteq Z[k + 1]$ ($0 \leq k \leq k' - 1$).

Proof: Suppose by contradiction that there is a process $p_i$ and a round number $k$ such that $p_i \in Z[k]$ and $p_i \notin Z[k + 1]$. Since $p_i \notin Z[k + 1]$, then $p_i \notin crashed[k + 1] \cup NSYN[k + 1]$. Applying Claim 6.1.1.a, we get $p_i \notin crashed[k] \cup NSYN[k]$. However, $p_i \in Z[k] = C[k] \cup crashed[k] \cup NSYN[k]$, and hence, $p_i \in C[k]$.

Since $p_i \notin crashed[k], p_i \notin NSYN[k], and p_i \in C[k], p_i$ sends round $k + 1$ message $m'$ with $est \leq c$ and $state \neq NSYNC$. As $p_i \notin crashed[k + 1] \cup NSYN[k + 1]$, so $p_i$ evaluates $est$ in line 13 of round $k' + 1$. From Claim 6.1.1.b and $p_i \notin NSYN[k]$, it follows that $p_i$ never sends a message with $state = NSYNC$ at round $k$ or at a lower round. Since a process always receives the message sent to itself without a delay and $p_i$ never sends a message with $state = NSYNC$ at round $k$ or at a lower round, $p_i \notin Halt_i[k + 1]$. Applying Lemma 5 we have, $p_i \in senderMS_i[k + 1]$, and therefore, $m' \in msgSet_i[k + 1]$. Thus when $p_i$ evaluate $est$ in round $k + 1$, it consider message $m'$ with $est \leq c$, and hence, adopts a values less than equal to $c$ as the new $est$. Thus $p_i \in C[k + 1] \subseteq Z[k + 1]$; a contradiction.

Claim 6.1.3: $0 \leq k \leq k' - 1, \forall p_i \notin Z[k + 1], Z[k] \subseteq Halt_i[k + 1]$.

Proof: Consider a process $p_j \in Z[k]$ and a process $p_i \notin Z[k + 1]$. In round $k + 1$, $msgSet_i[k + 1]$ either contains a message from $p_j$ or does not contain any message from $p_j$. In the second case, Lemma 5
implies that \( p_j \in Halt_t[k+1] \). Consider the case when \( msgSet_t[k+1] \) contains a message \( m \) from \( p_j \). From line 11 and line 12, it follows that, \( m \) has state \( \neq \text{NSYNC} \), and hence, \( p_j \notin NSY[N[k] \). Furthermore, \( p_j \) sent a message in round \( k+1 \), and so, \( p_j \notin \text{crashed}[\text{k}] \). Thus \( p_j \notin \text{crashed}[\text{k}] \cup NSY[N[k] \) but \( p_j \in Z[k] \). So, \( p_j \in C[k] \). Thus \( m \) has est \( \leq c \), and hence, \( est_t[k+1] \leq c \). Thus \( p_t \in C[k+1] \subseteq Z[k+1] \); a contradiction. Thus \( msgSet_t[k+1] \) does not contain a message \( m \) from \( p_j \). \( \square \)

Claim 6.1.4: \(|Z[k'] - 1| \leq k' - 1 \).

**Proof:** Suppose by contradiction \(|Z[k' - 1]| > k' - 1 \). From A1, it follows that \( p_y \notin Z[k'] \). Therefore, from claim 6.1.3, \( Z[k' - 1] \subseteq Halt_y[k'] \). Hence, \( |Halt_y[k']| > k' - 1 \). However, \( state_y[k'] = \text{SYNC2} \) implies that \( |Halt_y[k']| \leq k' - 1 \) (line 16, line 17), a contradiction. \( \square \)

Claim 6.1.5: \( p_x \in Z[k'] \) and \( p_x \notin Z[k' - 2] \).

**Proof:** As \( est_x[k'] = c \), so \( p_x \in C[k'] \subseteq Z[k'] \).

For the second part of the claim, suppose by contradiction that \( p_x \in Z[k' - 2] \). From Claim 6.1.3, for every process \( p_i \notin Z[k' - 1] \), \( p_x \in Halt_x[k' - 1] \). Therefore, in round \( k' \), if any process in \( \Pi - Z[k' - 1] \) sends a message \( m \), then \( p_x \in m.Halt \) (where, \( m.Halt \) denotes the \( Halt \) field of \( m \)). If \( p_x \) receives \( m \) then it includes the sender of \( m \) in \( Halt_x \) (condition 2, line 11), and even if \( p_i \) does not receive \( m \) then it includes the sender of \( m \) in \( Halt_x \) (condition 3, line 11). Thus \( \Pi - Z[k' - 1] \subseteq Halt_x[k'] \). Using, Claim 6.1.4, \( |Halt_x[k']| \geq |\Pi - Z[k' - 1]| \geq n - (k' - 1) \). Since \( k' + 1 \leq t + 2 \) and \( t < n/2 \), we have \( |Halt_x[k']| \geq n - t > t \). However, \( |Halt_x[k']| > t \) implies that \( state_x[k'] = \text{NSYNC} \) (line 20, line 21); a contradiction. \( \square \)

Claim 6.1.6: (1) For every \( k \) such that \( 0 \leq k < k' - 3 \): \( Z[k] \subseteq Z[k+1] \). (2) \(|Z[k'] - 1| \leq k' - 1 \).

**Proof:** (1) Recall from Claim 6.1.2 that \( Z[k] \subseteq Z[k+1] \) (0 \( \leq k < k' - 1 \)). Suppose by contradiction that there is a round number \( s \) (0 \( \leq s < k' - 3 \)), such that \( Z[s] = Z[s+1] \).

We first show by induction on the round number \( k \) that, for \( s + 1 \leq k < k' - 1 \), \( C[k] - (NSY[N[k] \cup crashed[k]) \supseteq C[k+1] - (NSY[N[k+1] \cup crashed[k+1]) \).

**Base Case (k = s+1):** \( C[s+1] - (NSY[N[s+1] \cup crashed[s+1]) \subseteq C[s+2] - (NSY[N[s+2] \cup crashed[s+2]) \).

Suppose by contradiction that there is a process \( p_i \) such that \( p_i \in C[s+2] - (NSY[N[s+2] \cup crashed[s+2]) \) (A4) and \( p_i \notin C[s+1] - (NSY[N[s+1] \cup crashed[s+1]) \) (A5).

A4 implies that \( p_i \notin NSY[N[s+2] \cup crashed[s+2]) \). Applying Claim 6.1.1, we have \( p_i \notin NSY[N[s+1] \cup crashed[s+1]) \), and therefore, from A5 it follows that \( p_i \notin C[s+1] \). Thus \( p_i \) completes round \( s+1 \) with \( est > c \). Furthermore, A4 implies that \( p_i \in C[s+2] \), and hence, \( p_i \) completes round \( s + 2 \) with \( est \leq c \). So, \( msgSet_i[s+2] \) contains a message with \( est \leq c \) from some process \( p_j \) (i.e., \( p_j \in senderMS_i[s+2]) \).

From the definition of \( Z[s+1] \), it follows that \( p_j \in C[s+1] \subseteq Z[s+1] \). As \( p_i \notin NSY[N[s+1] \cup crashed[s+1]) \) and \( p_i \notin C[s+1] \), so from the definition of \( Z[s+1] \) we have \( p_i \notin Z[s+1] \). Claim 6.1.3 implies that \( Z[s] \subseteq Halt_t[s+1] \). Recall that we assumed \( Z[s] = Z[s+1] \) and, from line 11, \( Halt_t[s+1] \subseteq Halt_t[s+2] \). Therefore, \( Z[s+1] \subseteq Halt_t[s+2] \). Thus \( p_j \in C[s+1] \subseteq Z[s+1] \) implies that \( p_j \in Halt_t[s+2] \). Therefore, \( p_j \in senderMS_i[s+2] \cap Halt_t[s+2] \).

As \( p_i \notin NSY[N[s+2] \cup crashed[s+2]) \), then \( p_i \) completed round \( s + 2 \) with \( state = \text{SYNC1} \) or \( state = \text{SYNC2} \). From Lemma 5 it follows that \( senderMS_i[s+2] \cap Halt_t[s+2] = \emptyset \). However, \( p_j \in senderMS_i[s+2] \cap Halt_t[s+2] \); a contradiction.

**Induction Hypothesis (s + 1 \( \leq k < k' - 1 \)):** \( C[k] - (NSY[N[k] \cup crashed[k]) \supseteq C[k+1] - (NSY[N[k+1] \cup crashed[k+1]) \).

16
Induction Step \((k = r + 1)\): \(C[r + 1] - (NSY N[r + 1] \cup \text{crashed}[r + 1]) \supseteq C[r + 2] - (NSY N[r + 2] \cup \text{crashed}[r + 2])\). Suppose by contradiction that there is a process \(p_i\) such that \(p_i \in C[r + 2] - (NSY N[r + 2] \cup \text{crashed}[r + 2])\) (A6) and \(p_i \notin C[r + 1] - (NSY N[r + 1] \cup \text{crashed}[r + 1])\) (A7).

As in the base case, using A6, A7, and Claim 6.1.1, we can show \(p_i \notin NSY N[r + 2] \cup \text{crashed}[r + 2], p_i \notin NSY N[r + 1] \cup \text{crashed}[r + 1]\), and \(p_i \notin C[r + 1]\). Thus \(p_i \notin Z[r + 1]\). Since \(s + 1 < r + 1\), from Claim 6.1.2, we have \(Z[s + 1] \subseteq Z[r + 1]\), and therefore, \(p_i \notin Z[s + 1]\).

Applying Claim 6.1.3 on \(p_i \notin Z[s + 1]\) implies that \(Z[s] \subseteq \text{Halt}_i[s + 1]\). Recall that we assumed \(Z[s] = Z[s + 1]\), and from line 11, \(\text{Halt}_i[s + 1] \subseteq \text{Halt}_i[r + 2]\). Therefore, \(Z[s + 1] \subseteq \text{Halt}_i[r + 2]\) (A8).

From induction hypothesis, we have \((C[s + 1] - (NSY N[s + 1] \cup \text{crashed}[s + 1])) \supseteq (C[r + 1] - (NSY N[r + 1] \cup \text{crashed}[r + 1]))\). From the definition of \(Z[s + 1], C[s + 1] - (NSY N[s + 1] \cup \text{crashed}[s + 1]) \subseteq C[s + 1] \subseteq Z[s + 1]\), and therefore, \(C[r + 1] - (NSY N[r + 1] \cup \text{crashed}[r + 1]) \subseteq Z[s + 1]\). Applying A8, we have \((C[r + 1] - (NSY N[r + 1] \cup \text{crashed}[r + 1])) \subseteq \text{Halt}_i[r + 2]\) (A9).

As \(p_i \notin Z[r + 1]\), \(p_i\) completes round \(r + 1\) with \(est > c\). Furthermore, A6 implies that \(p_i \in C[r + 2]\), and hence, \(p_i\) completes round \(r + 2\) with \(est \leq c\). Therefore, \(\text{msgSet}_i[r + 2]\) contains a message with \(est \leq c\) from some process \(p_j\) (i.e., \(p_j \in \text{senderMS}_i[r + 2]\)). From the definition of \(Z[r + 1]\), it follows that \(p_j \in C[r + 1] \subseteq Z[r + 1]\).

As the round \(r + 2\) message of \(p_j\) is in \(\text{msgSet}_i[r + 2]\), so from line 11 it follows that the message sent by \(p_j\) had state \(\neq \text{NSYNC}\). Therefore, \(p_j \notin NSY N[r + 1]\) and \(p_j \notin \text{crashed}[r + 1]\). Therefore, \(p_j \in C[r + 1] - (NSY N[r + 1] \cup \text{crashed}[r + 1])\). From A9 it follows that \(p_j \in \text{Halt}_i[r + 2]\).

As \(p_i \notin NSY N[r + 2] \cup \text{crashed}[r + 2]\) (from A6), so \(p_i\) completed round \(r + 2\) with state \(= \text{SYNC1}\) or state \(= \text{SYNC2}\). Lemma 5 implies that \(\text{senderMS}_i[r + 2] \cap \text{Halt}_i[r + 2] = \emptyset\). However, \(p_j \in \text{senderMS}_i[r + 2] \cap \text{Halt}_i[r + 2]\); a contradiction.

From the above result, we have \((C[k' - 2] - (NSY N[k' - 2] \cup \text{crashed}[k' - 2])) \supseteq C[k' - 2] - (NSY N[k' - 2] \cup \text{crashed}[k' - 2])\). From A1, \(p_x \in C[k'] - (NSY N[k'] \cup \text{crashed}[k'])\). From Claim 6.1.5, we have \(p_x \notin Z[k' - 2] \supseteq (C[k' - 2] - (NSY N[k' - 2] \cup \text{crashed}[k' - 2])\). Therefore, there is process in \(C[k'] - (NSY N[k'] \cup \text{crashed}[k'])\) which is not in \(C[k' - 2] - (NSY N[k' - 2] \cup \text{crashed}[k' - 2])\); a contradiction.

(2) Part (1) of this lemma implies that for every \(k\) such that \(0 \leq k \leq k' - 3, |Z[k + 1]| - |Z[k]| \geq 1\). From A4 that \(|Z[0]| \geq 1\). Therefore, \(|Z[k' - 2]| \geq k' - 1\).

\[\]
following which immediately implies the lemma: Every process in $H[l]$ ($0 \leq l \leq t + 2$) crashes before completing round $l$.

We prove the claim by induction on round $l$. For $l = 0$, the lemma is trivially true, because $H[0] = \emptyset$ (base case). Suppose that the claim is true for $0 \leq l \leq l - 1 \leq t + 1$: every process in $H[l]$ crashes before completing round $l$ (induction hypothesis). Consider $H[l]$ (induction step). If $H[l] - H[l - 1] = \emptyset$ then the induction step is trivial. Suppose by contradiction that there is process $p_j \in H[l] - H[l - 1]$ such that $p_j$ completes round $l$. Thus there is a process $p_a$ such that $p_j \notin H_a[l - 1]$ and $p_j \in H_a[l]$. Since $p_j$ completes round $l$ and the run is synchronous, in that round, $p_a$ must have received the round $l$ message $m$ of $p_j$. Since, $p_j \in H_a[m]$, $m$ contains either (a) state $=$ NSYNC or (b) $halt_j$ such that $p_a \in Halt_j$. Now, we show both the cases to be impossible and thus prove the induction step by contradiction.

From our assumption, for every round lower than $l$, every process in $Halt_j$ has crashed. Since more than $l$ processes cannot crash in a run, in rounds lower than $l$, $|Halt_j|$ is never more than $l$. Thus $p_j$ cannot update its state to NSYNC in rounds lower than $l$ (line 20). Thus the round $l$ message from $p_j$ does not contain state $=$ NSYNC.

If the round $l$ message from $p_j$ contains $halt_j$ such that $p_a \in Halt_j$ then $p_a \in Halt_j[l - 1] \subseteq H[l - 1]$. However, from our assumption, every process in $H[l - 1]$ crashes before completing round $l - 1$, which implies that $p_a$ crashes before completing round $l - 1$; a contradiction.

Lemma 8. (Fast Early Decision) In every synchronous run of $A_{f+2}$ with at most $f$ failures ($0 \leq f \leq t < n/2$), every process which decides, decides by round $f + 2$.

Proof. Consider a synchronous run in which at most $f$ processes fail. From Lemma 7, $|Halt|$ at every process is less than or equal to $f$ in the first $t + 2$ rounds (A11). Suppose by contradiction that some process $p_i$ completes round $f + 2$ but does not decide in that round. Then, either (1) state$_i[f + 2] =$ NSYNC, or (2) some process $p_j$ sent a message in round $f + 2$ with state $=$ SYNC. For case 1 to hold, $|Halt| > t$ in round $f + 2$ or a lower round (line 20), which clearly violates Observation A11. For case 2 to be true, state$_j[f + 1] =$ SYNC, and therefore, $|Halt_j[f + 1]| \geq f + 1$ (line 18), which contradicts A11 as well.