Weak leader election for receive-omission process failures

Authors
Junqueira, Flavio
Marzullo, Keith

Publication Date
2005-01-26

Weak leader election for receive-omission process failures

Flavio P. Junqueira
flavio@cs.ucsd.edu

Keith Marzullo
marzullo@cs.ucsd.edu

University of California, San Diego
Department of Computer Science and Engineering
9500 Gilman Drive
La Jolla, CA – USA

1 Introduction

Leader election is an important primitive in fault-tolerant distributed computing because it enables the solution of problems broadly applicable in real systems such as Consensus, as illustrated by the Paxos algorithm [1], and Primary-Backup, as in [2].

The particular version of the Leader Election problem we develop upon first appeared in the context of Primary-Backup algorithms. In the Primary-Backup approach for fault-tolerant services, clients issue requests that the primary is responsible for handling and replying to. When the primary fails, one of the backup replicas emerges as the new primary. Thus, a Primary-Backup algorithm must embed a Leader Election algorithm to infinitely often select a primary.

In [2], Budhiraja and Marzullo show a lower bound of $n > 3t/2$ for such algorithms when processes can fail to receive messages. The basic idea of the lower bound proof is that multiple primaries can be elected if fewer than $(3t/2) + 1$ processes compose the system. In a later section, we repeat this result for exposition purposes.

Still on the early work by Budhiraja and Marzullo on Primary-Backup algorithms, the degree of replication necessary is higher because they assume that faulty processes cannot be elected [2]. According to their statement of the problem, if a process does not crash but it commits receive-omission failures, then it cannot be elected. This is due to the assumption that responses to client requests are bounded in time. Failure detection for receive-omission failures, however, requires at least twofold replication.

When implementing a system based on the Primary-Backup approach, servers should be connected by a local area network to bound response time to client requests, which implies bounded fail-over time. For such settings, partitions are unlikely to occur if processors operate at a reasonable speed. Messages, however, can be lost due to, for example, buffer overflows at the receiver. One can imagine using retransmissions to cope with such failures. A retransmission mechanism, however, only guarantees eventual delivery, bounded response is not possible with eventual delivery of messages. Due to the requirements on bounded response and failure-over time, Primary-Backup algorithms are usually synchronous.

In this paper, we describe a synchronous algorithm for leader election under receive-omission process failures and prove its correctness. The novelty in this algorithm is fourfold: 1) it proves tight a lower bound that has been known for over 10 years; 2) in permitting faulty (but not crashed) processes to be elected, it requires fewer replicas; 3) it is based on cores and survivor sets which are abstractions that enables one to more expressively represent failure patterns by considering failures that are not independent or not identically distributed; 4) although it allows for faulty processes to be elected, correct processes are able to detect it, enabling the use of alarms to indicate failures in the system. Relating to our discussion on Primary-Backup protocols, by assuming that faulty processes can be elected, we cannot bound response time for a Primary-Backup algorithm. We can guarantee, however, that there is a single primary at any time, and that response is bounded whenever a correct process emerges as the primary. We further discuss this and other issues with Primary-Backup protocols later in the paper.

The remainder of this paper is as follows. We detail the system model in Section 2. We then introduce the problem by stating the properties an algorithm should fulfill (Section 3). Still in Section 3, we repeat the lower bound proof for process replication, and generalize this bound to our model of dependent failures. Section 4 describes our WLE algorithm for Leader Election. As we shall see, the algorithm depends on a primitive that we call *RO Consensus*. The properties for RO Consensus resembles the ones for the traditional Consensus primitive. The differences, however, are significant enough for naming the problem differently. In Section 4, we also provide an algorithm for RO Consensus. Sections 5 and 6 provide proofs of correctness for ROC and WLE, respectively. In Section 7, we strengthen the definition of leader election to disable executions in which different leaders are elected infinitely often, and provide a simple modification of the algorithm that enables it. Before concluding, we provide a discussion on the implications of the properties of our algorithm in a Primary-Backup protocol in Section 8. We finally conclude in Section 9.

2 System model

A system is a collection of processes $\Pi = \{p_1, p_2, \ldots, p_n\}$ that communicate by messages.[1] For every pair of processes $p_i, p_j \in \Pi$, there is a channel that p_i uses to send messages to p_j.

In such a system, an algorithm $\mathcal{A}$ is a collection of state machines, one for each process. $\mathcal{A}$ then proceeds in steps of processes. In a step, a process p_i executes atomically the following:

$\bigwedge$ $\bigvee$ receives a message from a process p_j

$\bigvee$ sends a message to a process p_j

$\bigvee$ executes a local operation

$\bigwedge$ undergoes a state transition

We define an execution ϕ of $\mathcal{A}$ as a tuple $\langle F, I, S, T, \mathcal{T} \rangle$, where F is the set of processes that are faulty in ϕ; I is the set of initial values, one for each process; S is a set of steps; T is a set of real values; $\mathcal{T} : S \rightarrow T$ is a mapping from steps to real values. The real values in T correspond to the global time in which steps execute. $\mathcal{T}(s)$ therefore is the global time in which step $s \in S$ executes. We use global time in proofs, and we do not assume such a virtual clock that produces global time is available to processes. We also use $Correct(\phi)$ for the set of processes that are correct in $\phi = \langle F, I, S, T, \mathcal{T} \rangle$. That is, $Correct(\phi) = \Pi \setminus F$. Finally, Φ is the set of valid executions of $\mathcal{A}$. An execution is valid if it does not violate any assumption made regarding the system and is correct with respect to the algorithm.

We assume that processes can fail by crashing or by omitting to receive messages. If a process p_i crashes in an execution ϕ, then there is step s of p_i such that p_i executes no further steps after $\mathcal{T}(s)$. We call this step a *crash step*. On the other hand, if a process is receive-omission faulty, it can selectively fail in receiving messages. To describe valid failure patterns, we use our model of dependent process failure based on the abstractions of cores and survivor sets. We define cores and survivor sets as follows:

Definition 2.1 A subset $C \subseteq \Pi$ is a core if and only if: 1) $\forall \phi \in \Phi$, $Correct(\phi) \cap C \neq \emptyset$; 2) $\forall p_i \in C$, $\exists \phi \in \Phi$ such that $C \setminus \{p_i\} \cap Correct(\phi) = \emptyset$.

Definition 2.2 A subset $S \subseteq \Pi$ is a survivor set if and only if: 1) $\exists \phi \in \Phi$, $Correct(\phi) = S$; 2) $\forall \phi \in \Phi, p_i \in S$, $Correct(\phi) \not\subseteq S \setminus \{p_i\}$.

By the definition of a survivor set, every subset of processes $\Pi' \subset \Pi$ is a valid failure pattern if and only if exists $S \in S_\Pi$ such that $S \cap \Pi' = \emptyset$.

We use the term *system profile* to denote a description of the possible failure patterns. In the threshold model, a system profile is a pair $\langle \Pi, t \rangle$, which means that any subset

[1] We use p_i to denote a process and i to denote the identifier of this process.

of t processes in Π can be faulty. In our dependent failure model, the system profile is a triple $\langle \Pi, C_\Pi, S_\Pi \rangle$, where C_Π is the set of cores and S_Π is the set of survivor sets.

We assume that systems are synchronous: the steps of every execution of some algorithm $\mathcal{A}$ can be split into rounds. That is, there is a mapping $Round : S \rightarrow \mathcal{R}$ from steps of processes to round numbers, where $\mathcal{R} = \mathbb{Z}^*$ and round numbers monotonically increase with time. We then have the following properties for rounds:

P-Liveness : If a process executes all the steps of a round r, then every process that does not crash by r executes at least one step of r.

C-Liveness : If a process p_i sends a message m to a correct process p_j at round r and p_i does not crash by round r, then p_j receives m at round r.

Integrity : If p_i receives a message m from p_j, then p_j sent m to p_i.

No duplicates : No message m is received more than once.

3 Problem specification

For the following description of the problem, we assume that each process p_i in Π has a boolean variable $p_i.elected$ that is set to true if the process elects itself, and to false otherwise. We then define the *Weak Leader Election* problem with the following three properties:

Safety $\Box|\{p_i \in \Pi : p_i.elected\}| < 2$.

LE-Liveness $\Box\Diamond(|\{p_i \in \Pi : p_i.elected\}| > 0)$.

FF-Stability In a failure-free execution, only one process ever has *elected* set to true.

These properties basically state that infinitely often some process elects itself, and no more than one process elects itself at any time. The third property eliminates the possibility of an algorithm that, for example, elects processes in a round-robin fashion. It does not rule out, however, executions in which processes alternate as leaders forever. For this reason, we propose another property called *E-Stability* stated as follows:

E-Stability $\exists p_i \in \Pi : \Diamond\Box(\forall p_j \in \Pi : p_j.elected \Rightarrow (j = i))$

An algorithm satisfying this property eventually elects the same process forever in every execution.

In the following sections, we first derive an algorithm that satisfies the first three properties. Later we modify this algorithm to also satisfy E-Stability. First, we show a lower bound for this problem.

3.1 Lower bound on process replication

In [2], the following lower bound was shown. The proof was given in the context of showing a lower bound on replication for Primary-Backup protocols.

Lemma 3.1 *Weak leader election for receive-omission failures requires* $n > \lfloor 3t/2 \rfloor$.

Proof:
Assume that leader election for receive-omission failures can be solved with $n = \lfloor 3t/2 \rfloor$. Partition the processes into three blocks A, B and C where $|A| = \lfloor t/2 \rfloor$, $|B| = \lfloor t/2 \rfloor$, and $|C| = \lceil t/2 \rceil$. Consider an execution ϕ_A in which the processes in B and C initially crash. From LE-Liveness, eventually a process in A will be elected. Similarly, let ϕ_B be an execution in which the processes in A and C crash. From LE-Liveness, eventually a process in B will be elected.

Finally, consider an execution ϕ in which the processes in A fail to receive all messages except those sent by processes in A, and the processes in B fail to receive all messages except those sent by processes in B. This run is indistinguishable from ϕ_A to the processes in A and is indistinguishable from ϕ_B to the processes in B. Hence, there will eventually be a process in A elected and a process in B elected, violating Safety.
□

3.2 Replication predicate

To develop the protocol, we first generalize the replication predicate for this problem for the core/survivor set model, where a replication predicate is a predicate that establishes whether there is sufficient replication to enable the solution of a particular problem. From the lower bound proof of Lemma 3.1, we consider any partition of the processes into three blocks. Then, one constructs three executions, where in each execution all of the processes in two of the three subsets are faulty. If we can construct such a partition, then we cannot solve the problem. The conclusion of the lower bound proof is the following property for $k = 3$:

Property 3.2 : **(k,k-1)-Partition**, $k > 1$
For every partition $\mathcal{B} = \{B_1, \ldots, B_k\}$ of Π, there is a subset $\mathcal{B}' = \{B_{\ell_1}, \ldots, B_{\ell_{k-1}}\} \subset \mathcal{B}$ such that $\cup_i B_{\ell_i}$ contains a core. □

Let $G_b(A)$ be all subsets of A of size b. The equivalent intersection property is:

Property 3.3 : **(k,k-1)-Intersection**, $k > 1$
$\forall \mathcal{S} \in G_k(S_\Pi) : \exists P \in G_2(\mathcal{S}) : (\cap_{S \in P} S) \neq \emptyset$ □

Stated more simply, (k,k-1)-Intersection says that for any set of k survivor sets, at least two of them have a non-empty intersection.

Here is an example of a system that satisfies (3,2)-Intersection. It is based on a simple two-cluster system. Each cluster has the same number of processes. A process can fail by crashing, and there is a threshold t on the number of crash failures that can occur in a cluster. A cluster can also suffer a catastrophic failure, which causes all of the processes in that cluster to fail. Such a catastrophic failure can result from the failure of a cluster resource such as a disk array or a power supply, or from an administrative error. We assume that catastrophic failures are rare enough that the probability of both clusters suffering catastrophic failures is low. Processes can crash in one cluster at the same time that the other cluster suffers a catastrophic failure.

Assuming that each cluster has three processes and $t = 1$, we have the following system profile:

- $\Pi = \{p_{A_1}, p_{A_2}, p_{A_3}, p_{B_1}, p_{B_2}, p_{B_3}\}$;
- $C_\Pi = \{\{p_{i_1}, p_{i_2}, p_{i_3}, p_{i_4}\} | (i_1, i_2 \in \{A_1, A_2, A_3\}) \wedge (i_3, i_4 \in \{B_1, B_2, B_3\})\}$;
- $S_\Pi = \{\{p_{i_1}, p_{i_2}\} | (i_1, i_2 \in \{A_1, A_2, A_3\}) \vee (i_1, i_2 \in \{B_1, B_2, B_3\})\}$;

To see why this system satisfies $(3, 2)$, we just have to observe that for any three survivor sets, at least two of them intersect each other.

Theorem 3.4 *(k,k-1)-Partition ≡ (k,k-1)-Intersection*

Proof:
(k,k-1)-Partition → (k,k-1)-Intersection:
We show the contrapositive. Consider a system profile such that, for some subset $\mathcal{S} = \{S_1, S_2, \ldots, S_k\} \subset S_\Pi$, no pair of survivor sets S_i, S_j intersects. That is, $\cup_{P \in G_2(\mathcal{S})} \cap P = \emptyset$. We then build a partition $\mathcal{B} = \{B_1, B_2, \ldots, B_k\}$ as follows:

$$
\begin{aligned}
B_1 &= \Pi \setminus (S_2 \cup S_3 \cup \ldots \cup S_k) \\
B_2 &= \Pi \setminus (S_1 \cup S_3 \cup \ldots \cup S_k \cup B_1) \\
&\vdots \\
B_i &= \Pi \setminus (S_1 \cup S_2 \cup \ldots \cup S_{i-1} \cup S_{i+1} \cup \ldots \\
&\quad \ldots \cup S_k \cup B_1 \cup B_2 \ldots \cup B_{i-1}) \\
&\vdots \\
B_k &= \Pi \setminus (S_1 \cup S_2 \cup \ldots S_{k-1} \cup B_1 \ldots \cup B_{k-1})
\end{aligned}
$$

We have to show that: 1) $B_1, \ldots, B_k$ is a partition; 2) For every subset $\{B_{i_1}, B_{i_2}, \ldots, B_{i_{k-1}}\} \subset \{B_1, \ldots, B_k\}$, $\cup_j B_{i_j}$ does not contain a core. To show 1), let $\psi_i = \cup_{(S_j \in \mathcal{S} \setminus S_i)} S_j$, $i \in \{1, \ldots, k\}$. We then have the following derivation:

$$
\begin{aligned}
\cup B_i &= (\Pi \setminus \psi_1) \cup (\Pi \setminus \psi_2 \cup B_1) \cup \ldots \\
&\quad \ldots \cup (\Pi \setminus (\psi_k \cup B_1 \cup B_2 \ldots \cup B_{k-1})) \qquad (1) \\
&= \Pi \setminus ((\psi_1 \cap (\psi_2 \cup B_1)) \cap \ldots \\
&\quad \ldots \cap (\psi_k \cup B_1 \cup B_2 \ldots \cup B_{k-1})) \qquad (2) \\
&= \Pi \setminus (\psi_1 \cap \psi_2 \cap \ldots \\
&\quad \ldots \cap (\psi_k \cup B_1 \cup B_2 \ldots \cup B_{k-1})) \qquad (3) \\
&\vdots \\
&= \Pi \setminus (\cap_i \psi_i) \qquad (4) \\
&= \Pi \qquad (5)
\end{aligned}
$$

- Line 1 to Line 2 follows from the observation that for any subsets A, B of Π, we have that $(\Pi \setminus A) \cup (\Pi \setminus B) = \Pi \setminus (A \cap B)$;

- Line 2 to Line 3: the intersection between ψ_1 and B_1 has to be empty, since ψ_1 contains exactly the elements we removed from Π to form B_1.

- Line 3 to Line 4: by repeating inductively the process used to derive Line 3, we are able to remove every term B_i present in the equation.

- Line 4 to Line 5: Transforming from a conjunctive form to a disjunctive form, we have that $\cap_{P \in G_{k-1}(\mathcal{S})} \cup P = \cup_{P \in G_2(\mathcal{S})} \cap P$. To see why this is true, note that for every pair $S_i, S_j \in \mathcal{S}$, $i \neq j$, and $P \in G_{k-1}(\mathcal{S})$, we have that $(S_i \in P) \vee (S_j \in P)$. Finally, we have that $\cup_{P \in G_2(\mathcal{S})} (\cap_{S_i \in P} S_i) = \emptyset$ by assumption.

By the construction of the partition and from the assumption that for every $S_i, S_j \in \mathcal{S}$, $S_i \cap S_j = \emptyset$, we have that for every $i \in \{1, \ldots, k\}$, there is $S_i \in \mathcal{S}$ such that $S_i \subseteq B_i$. From this, we conclude that for any $\{B_{i_1}, B_{i_2}, \ldots, B_{i_{k-1}}\} \subset \mathcal{B}$, $\cup_j B_{i_j}$ does not contain elements from some survivor set, and consequently it does not contain a core.

(k,k-1)-Partition ← (k,k-1)-Intersection:

We also prove the contrapositive for this direction. Suppose a system profile $\langle \Pi, C_\Pi, S_\Pi \rangle$ such that there is a partition $\mathcal{B} = \{B_1, B_2, \ldots, B_k\}$ in which no union of $k - 1$ subsets of $\mathcal{B}$ contains a core. We then have that for B_i there is a $S_i \in S_\Pi$ such that $S_i \subseteq B_i$. Thus, for all $B_i, B_j \in \mathcal{B}$, $i \neq j$, we have by assumption that $B_i \cap B_j = \emptyset$, and consequently $S_i \cap S_j = \emptyset$. We conclude that no pair $S_i, S_j \in \{S_1, S_2, \ldots, S_k\}$ is such that $S_i \cap S_j \neq \emptyset$.
□

4 The algorithm

In this Section, we describe our algorithm WLE for Weak Leader Election (Figure 2). It assumes a system with a profile $\langle \Pi, C_\Pi, S_\Pi \rangle$ that satisfies (3,2)-Intersection and uses as a building block an algorithm ROC that implements a weak version of Uniform Consensus that we call *RO Consensus*. We call it RO Consensus because its definition resembles the one of Consensus. It is tailored, however, to fulfill the requirements of WLE.

Each process p_i has an initial value $p_i.a \in V \cup \{\perp\}$, where V is the set of initial values, and a decision value $p_i.d\,[1 \ldots n]$, where $p_i.d[j] \in V \cup \{\perp\}$. We use $v \in p_i.d$ to denote that there is some $p_\ell \in \Pi$ such that $p_i.d[\ell] = v$. If a process p_i crashes, then we assume that its decision value $p_i.d$ is $\mathcal{N}$. To avoid repetition throughout the discussion of our algorithm, we say that a process p decides in an execution ϕ if $p.d$ is different than $\mathcal{N}$.

As we descibe later, we execute ROC multiple times in electing a leader. We then have that processes may crash before starting an execution ϕ of ROC. Such processes consequently have initial value undefined in ϕ. We therefore use $\perp$ to denote the initial value of crashed processes. That is, if $p_i.a = \perp$, then p_i has crashed.

Let the relation $x \subseteq y$ for x and y lists of n elements be that, for all $i : 1 \leq i \leq n$, $(x[i] \neq \perp) \Rightarrow (x[i] = y[i])$. We use the symbol $\mathcal{N}$ to stand for the n element list $[\perp, \ldots, \perp]$.

The specification of RO Consensus is given by four properties as follows:

Termination: Every process that does not crash eventually decides on some value.

Agreement: If $p_j.d[\ell] \neq \perp$, then for every non-faulty p_i, $p_i.d[\ell] = p_j.d[\ell]$;

RO Uniformity: Let *vals* be $\{d : \exists p_i \in \Pi \; s.t. \; (p_i.d = d)\} \setminus \mathcal{N}$. Then,

$$\begin{array}{lll} \bigwedge & 1 \leq |vals| \leq 2 & \\ \bigwedge & \forall d, d' \in vals : d \subseteq d' \vee d' \subseteq d & \\ \bigwedge & \forall d_f, d_c \in vals, d_f \subseteq d_c : \exists S_f, S_c \in S_\Pi : & \\ & \quad \wedge \ \forall p \in S_f : & \vee \ p \text{ crashes} \\ & & \vee \ p.d = d_f \\ & \quad \wedge \ \forall p \in S_c : & \wedge \ p.d = d_c \\ & & \wedge \ p \text{ is not faulty} \end{array}$$

That is, there can be no more than two non-$\mathcal{N}$ decision values, and if there are two then one is a subset of the other. Furthermore, if there are two different decision values, then these are the values that processes in two disjoint survivor sets decide upon, one for the processes of each survivor set.

Validity:

$\bigwedge$ If p_j does not crash, then for all non-faulty p_i, $p_i.d[j] = p_j.a$

$\bigwedge$ If p_j does crash, then exists $v \in \{\bot, p_j.a\}$ such that for all non-faulty p_i, $p_i.d[j] = v$;
$\bigwedge$ If there are survivor sets $S_f, S_c \in S_\Pi$
and values $v_f, v_c \in V$, $v_f \neq v_c$, such that
$\wedge\ \forall p \in S_f : p.a \in \{v_f, \bot\}$
$\wedge\ \forall p \in S_c : \wedge\ p.a = v_c$
$\wedge\ p$ is not faulty
$\wedge\ \exists p_i, p_\ell \in \Pi : p_i.d[\ell] = v_f$
then for all p_j that does not crash, $v_f \in p_j.d$

That is, if a process p_i is not faulty and $p_i.d[j] \neq \bot$, then the value of $p_i.d[j]$ must be $p_j.a$. The value of $p_i.d[j]$, however, can be $\bot$ only if p_j crashes. The third case exists because we use the decision values of an execution as the initial values for another execution. From RO Uniformity, there can be two different non-$\mathcal{N}$ values d_f and d_c. If this is the case, then there is a survivor set S_c containing only correct processes such that all processes in S_c decide upon d_c, and another survivor set S_f containing only faulty processes such that all the processes in S_f either crash or decide upon d_f. Let v_f be d_f and v_c be d_c. By the third case, if some process that decides includes $v_f = d_f$ in its decision value, then every process that does not crash also includes $v_f = d_f$ in its decision value.

We now describe our algorithm ROC for RO Consensus called. Figure 1 shows the pseudocode for a single process. From the figure, the algorithm ROC executes exactly in $t+1$ rounds, where $t = \min_s\{s = |S_i| \wedge S_i \in S_\Pi\}$ or alternatively $t = \max_c\{c = |C_i| \wedge C_i \in C_\Pi\}$[2]. For the proof of correctness we present in the next section, we assume that the value of t is at least one ($t \geq 1$). Note that for $t = 0$ there is a trivial, much simpler algorithm.

In every round r of ROC, processes send their list of values to a subset of the processes Π' in Π. If process p_i does not crashes or stops[3] in round r, then $\Pi = \Pi'$. Otherwise, this subset is arbitrary. Before the end of round r, every process p_i that does not execute a crash step at r receives all the messages sent to it at round r. Note that if a process p_i crashes at round r, but sends a message m_i to process p_j, then p_j does not necessarily receives m_i by C-Liveness. We then use M_i to denote the set of messages p_i receives by the end of any round r, and $p.s(r)$ to denote the set of processes from which process p receives messages in round r, where $0 \leq r \leq t$. Processes send no messages at the last round. Note that, by the algorithm, messages received by the end of round r are available for processing at the beginning of round $r+1$.

If a process detects that it has failed to receive messages, then it stops by deciding on $\mathcal{N}$. In the discussion that follows, we treat processes that crash and processes that stop indistinctly. If a distinction is necessary, then we clearly state it. There are two ways a processes p_i can determine that it is faulty: 1) By receiving messages from a set of processes at round r such that $p_i.s(r) \not\subset p_i.s(r-1)$, $r > 1$; 2) By determining that in its set of messages of round r, there is no survivor set potentially containing only correct processes. The second form of detection relies on the set of values p_i receives from another process p_j. If p_i notices that p_j did not receive a previous message from p_i, then p_i declares p_j faulty. By removing the obviously faulty processes and looking at the remaining set, if there is no survivor set in the remaining set, then p_i must be faulty as well.

Figure 2 shows the pseudocode for an algorithm that solves the Weak Leader Election problem. It proceeds in iterations of an infinite repeat loop. In each iteration, a process executes ROC twice, and decide if it has to elect itself by the end of the second phase.

In the following sections, we prove the correctness of both ROC and WLE.

5 Correctness of ROC

We provide a proof of correctness for the ROC algorithm. We say that a process p_i is live at round r if either one of the following happens:

[2] The first and the last rounds in the algorithms are actually half rounds, and we then consider that both together constitute a single round. Put it another way, we can easily rearrange the order of sending and receiving messages to make it fit into $t+1$. We have chosen the former for expositional convenience.

[3] A *stop* instruction is equivalent to a crash in that a process does not execute any further steps after executing a stop instruction. A process, however, executes a stop instruction based on the code, and hence it is not arbitrary.

```
Algorithm ROC on input $p_i.a$
round 0:
    $p_i.s(0) \leftarrow \Pi$; $p_i.sr(0) \leftarrow p_i.s(0)$
    $p_i.A\,[i] \leftarrow p_i.a$
    for all $p_k \in \Pi$, $p_k \neq p_i$ : $p_i.A\,[i] \leftarrow \bot$
    $p_i.A' \leftarrow p_i.A$
    send $p_i.A$ to all

round 1:
    $p_i.sr(1) \leftarrow p_i.s(1)$
    if $\vee \nexists S \in S_\Pi : S \subseteq p_i.s(1)$
    then decide $[\bot, \ldots, \bot]$
    else
        for each message $m_j \in M_i$, $p_k \in \Pi$:
            if $(p_i.A\,[k] = \bot)$ $p_i.A\,[k] \leftarrow m_j.A\,[k]$
    send $p_i.A$ to all

round $r$: $2 \leq r \leq t$:
    $p_i.sr(r) \leftarrow p_i.s(r) \setminus \{p_j : \exists m_j \in M_i :$
                                    $p_i.A' \nsubseteq m.A\}$
    if $\vee p_i.s(r) \nsubseteq p_i.s(r-1)$
      $\vee \nexists S \in S_\Pi : S \subseteq p_i.sr(r)$
    then decide $[\bot, ..., \bot]$
    else
        $p_i.A' \leftarrow p_i.A$
        for each message $m \in M_i$, $p_k \in \Pi$:
            if $(p_i.A\,[k] = \bot)$ $p_i.A\,[k] \leftarrow m.A\,[k]$
    send $p_i.A$ to all

round $t+1$:
    $p_i.sr(t+1) \leftarrow p_i.s(t+1) \setminus \{p_j : \exists m_j \in M_i :$
                                         $p_i.A \nsubseteq m.A\}$
    if $\vee$ $p_i.s(t+1) \nsubseteq p_i.s(t)$
      $\vee \nexists S \in S_\Pi : S \subseteq p_i.sr(t+1)$
    then decide $[\bot, ..., \bot]$
    else for each message $m \in M_i$, $p_k \in \Pi$:
      if $(p_i.A\,[k] = \bot)$ $p_i.A[k] \leftarrow m.A[k]$
    decide $p_i.A$
```

Figure 1: Algorithm run by process p_i.

```
Algorithm WLE
repeat {
$p_i$.elected $\leftarrow$ FALSE
Phase 1:
   Run ROC with
        $p_i.a \leftarrow i$.
   if $(p_i.d = [\bot, \ldots, \bot])$ then stop
Phase 2:
   Run ROC with
        $p_i.a \leftarrow p_i.d$ from Phase 1.
   if $(p_i.d = [\bot, \ldots, \bot])$ then stop
   let $x$ be a value of $p_i.d\,[1 \ldots n]$
        such that $p_i.d\,[x] \neq [\bot, \ldots, \bot]$
        and it has the least number of non-$\bot$ values
   if ($p_i$ is the first index of $x$ such that $x[i] \neq \bot$)
        then $p_i$.elected $\leftarrow$ TRUE
}
```

Figure 2: Algorithm run by process p_i.

- if p_i sends at least one message m_i to some process p_j at round r, $0 \leq r \leq t$, and p_j receives m_i by the end of round r;

- if p_i decides at round r, $r = t+1$.

We use *Live*(r) to denote the processes that are live at round r. For an execution $\phi = \langle F, I, S, T, \mathcal{T} \rangle$ of ROC we define the following to use in the proofs of this section:

- $T^i_\phi(i,r) \in T$ denotes the value of $\mathcal{T}(s_f)$, where $p_i \in Live(r)$ and $s_f \in S$ is the first step p_i executes of round r, $r \in \{0, \ldots, t+1\}$. If p_i executes no steps in r, then $T^i_\phi(p_i, r)$ is undefined;

- $T^u_\phi(i,r) \in T$ denotes the value of $\mathcal{T}(s_m)$, where $p_i \in Live(r)$ and $s_m \in S$ is the first step of round r in which p_i sends a message, $0 \leq r \leq t$. If $p_i \notin Live(r)$, then $T^i_\phi(p_i, r)$ is undefined;

- $T^u_\phi(i, t+1) \in T$ denotes $\mathcal{T}(s_d)$, where $p_i \in Live(t+1)$ and $s_d \in S$ is the step in which p_i decides at round $t+1$. If $p_i \notin Live(r)$, then $T^i_\phi(p_i, t+1)$ is undefined;

- $A(i,r)$ denotes the value of $p_i.A$ at $T^u_\phi(i,r)$, if $p_i \in Live(r)$. Otherwise, $A(i,r)$ is undefined;

- $M(i, r)$ denotes the value of M_i at $T^i_\phi(i, r)$, if $p_i \in Live(r)$. Otherwise, $M(i, r)$ is undefined;

- M^r_i is a list of n values, one for each process in Π such that the following holds: If $\{v : v = m.A[j] \wedge m \in M(i, r) \wedge m.A[j] \neq\perp\}$ is not empty, then $M^r_i[j] = p_j.a$. Otherwise, $M^r_i[j] =\perp$;

Processes that are alive in a round r may send messages to a strict subset of Π. Thus, in executions in which processes fail, we have that processes may have a different knowledge of the initial values. For the purpose of analyzing these cases, we define a *process chain* (or simply a chain) $\omega_\ell = (i_0 \circ i_1 \circ \ldots \circ i_k)_\ell$, $k \leq t + 1$, to be a string over the set of process identifiers. Let $\omega_\ell[x]$ be the process identifier at position x of the chain ω_ℓ. The following holds for a process chain ω_ℓ:

1. $\omega_\ell[r] \neq \omega_\ell[r']$, if $r \neq r'$;

2. If $\omega_\ell[r] = i$, then $(A(i, r)[\ell] \neq\perp) \wedge (\forall r' \in \{x \in \mathbb{Z}^* : x \leq r - 1\} : A(i, r')[\ell] =\perp)$;

3. If $\omega_\ell[r] = i$, $r > 0$, then $\exists m_j \in M(i, r) : (m_j.A[\ell] \neq\perp) \wedge \omega_\ell[r - 1] = j$;

4. If $\omega_\ell[0] = i$, then $i = \ell$.

We say that a process p_i is in ω_ℓ ($p_i \in \omega_\ell$) if and only if there exists an index r such that $\omega_\ell[r] = i$. We use process chains in the proofs below to represent the propagation of knowledge in executions with failures.

Figure 5 shows the structure of the proof of proposition 5.1, which is stated as follows:

Proposition 5.1 ROC implements RO Consensus.

We prove Proposition 5.1 with the following lemmas.

Lemma 5.2 *Let ϕ be an execution of ROC, r be a round of ϕ, $0 \leq r \leq t + 1$, p_i be a process in* Live(r)*, and p_j be a process in Π. If $A(i, r)[j] \neq\perp$, then $A(i, r) = p_j.a$.*

Proof:
We show with an induction on the round numbers ρ, $0 \leq \ell \leq r'$, that for every round $r' \leq r$, if $p_\ell \in Live(r')$ and $A(\ell, r')[j] \neq\perp$, then $A(\ell, r')[j] = p_j.a$. The base case is $\rho = 0$. From the algorithm, at round 0 a process p_ℓ has $A(\ell, 0)[j] =\perp$, if $\ell \neq j$, and $A(\ell, 0)[j] = p_\ell.a$ otherwise. Thus, $A(\ell, 0)[j] \neq\perp$ only if $\ell = j$, and $A(\ell, 0)[\ell] = p_\ell.a$ by construction.

Proposition
5.1: {5.19, 5.20, 5.21, 5.22}

Theorems
5.20: {}
5.21: {5.15}
5.22: {5.15, 5.16, 5.18}
5.23: {5.2, 5.15, 5.17}

Lemmas
5.2: {}
5.3: {5.2}
5.4: {5.2, 5.3}
5.5: {5.3, 5.4}
5.6: {5.5}
5.7: {5.4, 5.6}
5.8: {5.6}
5.9: {}
5.10: {5.9}
5.11: {5.4, 5.5}
5.12: {5.6, 5.7, 5.10, 5.11}
5.13: {5.2, 5.6, 5.7, 5.10, 5.11, 5.12}
5.14: {5.2, 5.4, 5.12, 5.13}
5.15: {5.13}
5.16: {5.2, 5.4, 5.5, 5.11, 5.12}
5.17: {5.13, 5.16}
5.18: {5.5, 5.11, 5.13, 5.14, 5.15, 5.16, 5.17}
5.19: {5.3, 5.4, 5.13, 5.16}

Figure 3: ROC hierarchy.

Now suppose that for every $p_\ell \in Live(\rho)$ if $A(\ell, \rho)[j] \neq\perp$, then $A(\ell, \rho)[j] = p_j.a$. We show that for every $p_\ell \in Live(\rho + 1)$, if $A(\ell, \rho + 1)[j] \neq\perp$, then $A(\ell, \rho + 1)[j] = p_j.a$. By the algorithm, if $p_\ell \in Live(\rho)$ is such that $A(\ell, \rho)[j] = p_j.a$, then for every message m it sends at round ρ, $m.A[j] = p_j.a$. For $p_{\ell'} \in Live(\rho + 1)$, if $p_{\ell'}.A[j] \neq\perp$ at $T^i_\phi(\ell', \rho + 1)$, then $A(\ell', \rho)[j] \neq\perp$ and must be equal to $p_j.a$ by the induction hypothesis. Otherwise, for every message m it receives such that $m.A[j] \neq\perp$, $m.A[j] = p_j.a$. Thus, by the algorithm, if $p_{\ell'}$ receives at least one such a message, it sets $p_{\ell'}.A[j]$ to $p_j.a$, and we have that $A(\ell', \rho + 1)[j] = p_j.a$.

From the previous induction, we conclude that $A(i, r)[j] = p_j.a$.

□

Lemma 5.3 *Let ϕ be an execution of ROC, r be a round of ϕ, $0 < r \leq t+1$, and p_i be a process in* $\mathrm{Live}(r)$. *For every message $m \in M(i,r)$ such that $m.A[\ell] \neq \perp$, for some $p_\ell \in \Pi$, $m.A[\ell] = p_\ell.a$.*

Proof:
By Lemma 5.2, for every $p_j \in \mathit{Live}(r-1)$, if $A(j, r-1)[\ell] \neq \perp$, then $A(j, r-1)[\ell] = p_\ell.a$. If p_j sends a message m_j to p_i at round $r-1$, then $m.A[\ell] = p_\ell.a$. We conclude that for every $m \in M(i,r)$ such that $m.A[\ell] \neq \perp$, $m.A[\ell] = p_\ell.a$.

□

Lemma 5.4 *Let ϕ be an execution of ROC and r be a round of ϕ, $0 < r \leq t+1$. For every $p_i \in$* $\mathrm{Live}(r)$, *$M_i^r = A(i,r)$.*

Proof:
By the algorithm, if $p_i \in \mathit{Live}(r)$, then for every $j \in [1 \ldots n]$, such that $p_i.A[j] = \perp$ at $T^i_\phi(i,r)$, p_i sets $p_i.A[j]$ to a value $v \in V = \{v : v = m.A[j] \wedge m \in M(i,r) \wedge m.A[j] \neq \perp \}$, $j \in [1 \ldots n]$ at t if $V \neq \emptyset$, where $T^i_\phi(i,r) \leq t \leq T^u_\phi(i,r)$, $t = \mathcal{T}(s)$, and s is a step of p_i that updates $p_i.A[j]$ at round r. Otherwise, if $p_i.A[j] \neq \perp$ at $T^i_\phi(i,r)$, then no step of p_i in round r modifies the value of $p_i.A[j]$, and $A(i,r)[j] = A(i, r-1)[j]$.

Let p_j be a process of Π. By Lemma 5.3, we have that $V = \{v : v = m.A[j] \wedge m \in M(i,r) \wedge m.A[j] \neq \perp\}$, $j \in [1 \ldots n]$, is either empty or contains a single value. If $|V| = 1$, then $V = \{p_j.a\}$. There are two cases to consider: 1) $A(i, r-1)[j] \neq \perp$; 2) $A(i, r-1) = \perp$.

If $A(i, r-1)[j] \neq \perp$, then, by the algorithm, p_i does not modify the value of $p_i.A[j]$ in round r. By Lemma 5.2, $A(i, r-1)[j] = p_j.a$. By the algorithm, p_i sends $A(i, r-1)$ to itself. Finally, by Lemma 5.3, every message $m \in M(i,r)$ is such that $m.A[j] = p_j.a$. We the have that $A(i,r)[j] = M_i^r[j]$.

If $A(i, r-1) = \perp$ and $V = \{p_j.a\}$, then $A(i,r)[j] = p_j.a$. If $V = \emptyset$, then $A(i,r)[j] = \perp$. In both cases, $A(i,r)[j] = M_i^r[j]$.

We conclude that $A(i,r)$ must be equal to M_i^r.

□

Lemma 5.5 *Let ϕ be an execution of ROC. For every $r \in$* $\{z \in \mathcal{R} : z \leq t+1\}$, $\mathrm{Correct}(\phi) \subseteq \mathrm{Live}(r)$.

Proof:
By definition, a process is alive at round $t+1$ of ϕ if it neither crashes nor stops before deciding in this round. We show this claim by showing that for every $p_c \in \mathit{Correct}(\phi)$ and every $r \in \{x \in \mathcal{R} : x \leq t+1\}$, $p_c \in \mathit{Live}(r)$. Let p_c be a process in $\mathit{Correct}(\phi)$. By definition, p_c does not crash in any round. It remains to show that, for every $p_c \in \mathit{Correct}(\phi)$ and every $r \in \{z \in \mathcal{R} : z \leq t+1\}$, p_c does not stop in r. We show this with an induction on the round numbers ρ, $\rho \in \{z \in \mathcal{R} : z \leq t+1\}$.

By the algorithm, no process stops at round 0. At round 1, a process only stops if it does not receive messages from a survivor set. Let p_c be a process in $\mathit{Correct}(\phi)$. By definition, there is a survivor set S_c containing only correct processes, and every process $p_{c'} \in S_c$ sends a message to p_c at round 0. By Liveness, p_c must have messages in $M(c, 1)$ at least from the processes in S_c. That is, $S_c \subseteq p_c.s(1)$. Consequently, p_c does not stop at round 1.

Now suppose that the claim holds for every ρ, $\rho \in \{z \in \mathcal{R} : 1 \leq z \leq t\}$, and we show for $\rho + 1$. Let p_c be a process in $\mathit{Correct}(\phi)$. By assumption, if p_c does not receive a message from some process p_i in round ρ, then p_i must have crashed by round ρ. This implies that all processes in $\Pi \setminus p_c.s(\rho)$ crashed by round ρ, and $p_c.s(\rho+1)$ therefore cannot contain a process p_i that is not in $p_c.s(\rho)$. Consequently, $p_c.s(\rho+1) \subseteq p_c.s(\rho)$.

For the induction step, it remains to show that $p_c.sr(\rho+1)$ contains some survivor set. From the algorithm, a process p_i is in $p_c.sr(\rho+1)$ if there is a message $m_i \in M(c, \rho+1)$ and $p_c.A' \subseteq m_i.A$, where $p_c.A' = A(c, \rho-1)$. By assumption, no correct process stops by round ρ. Thus, for every $p_{c'} \in \mathit{Correct}(\phi)$, there is a message $m_c \in M(c', \rho)$ from p_c. By Lemma 5.3, for every $p_\ell \in \Pi$ such that $m_c.A[\ell] \neq \perp$, we have that $m_c.A[\ell] = A(c, \rho-1)[\ell] = p_\ell.a$ and $M_{c'}^\rho[\ell] = p_\ell.a$. By Lemma 5.4, we then have that for every $p_{c'} \in \mathit{Correct}(\phi)$, $A(c, \rho-1) \subseteq A(c', \rho)$. By the algorithm and by the assumption that no correct process stops at round ρ, for every $p_{c'} \in \mathit{Correct}(\phi)$, $p_{c'}$ sends a message to p_c. We then have that $\mathit{Correct}(\phi) \subseteq p_c.s(\rho+1)$. By the observation that for every $p_{c'}$, $A(c, rho-1) \subseteq A(c', \rho)$, we have that $\mathit{Correct}(\phi) \subseteq p_c.sr(\rho+1)$. By assumption, there is a survivor set $S_c \in S_\Pi$ such that $S_c \subseteq Correct(\phi)$. We conclude that $S_c \subseteq p_c.sr(\rho + 1)$.

This concludes the proof of the lemma.

□

Lemma 5.6 *Let ϕ be an execution of ROC such that ω_ℓ is a chain in ϕ, $1 \leq |\omega_\ell| \leq t+1$. If $\omega_\ell[r]$ is the identifier of a correct process for some r, then $M_j^{r+1}[\ell] \neq\perp$ for every process $p_j \in$* Correct(ϕ)*.*

Proof:
Let r be an index such that $\omega_\ell[r]$ is the identifier of a correct process in ϕ and i be the process identifier in $\omega_\ell[r]$. By the definition of a process chain, we have that $(A(i,r)[\ell] \neq\perp) \wedge (\forall r' \in \{x \in \mathcal{R} : x \leq r-1\} : A(i,r')[\ell] \neq\perp$). By the algorithm, process p_i sends $A(i,r)$ to all the processes in Π. By Lemma 5.5, every correct process decides in ϕ ($Correct(\phi) \subseteq Live(t+1)$). By C-Liveness, for every $p_j \in Correct(\phi)$, there is $m_i \in M(j, r+1)$ such that $m_i.A[\ell] \neq\perp$. Again by the algorithm, $M_j^{r+1}[\ell]$ must be different than $\perp$ for every correct process p_j in ϕ.

□

Lemma 5.7 *Let ϕ be an execution of ROC such that there is a chain ω_ℓ of length at least three in ϕ. There are no three correct processes $p_{c_1}, p_{c_2}, p_{c_3}$ such that $c_1, c_2, c_3 \in \omega_\ell$.*

Proof:
Proof by contradiction. Suppose that there are three processes p_{c_1}, p_{c_2}, $p_{c_3} \in Correct(\phi)$ such that $c_1, c_2, c_3 \in \omega_\ell$, and that r is the smallest index such that $\omega_\ell[r]$ is the identifier of a correct process. Observe that $|\omega_\ell|$ must be at least as large as $r+3$ ($|\omega_\ell| \geq r+3$), otherwise the assumption does not hold.

Without loss of generality, let $\omega_\ell[r] = i$. By Lemma 5.6, for every correct process p_c in $Correct(\phi)$ we have that $M_c^{r+1}[\ell]$ different than $\perp$ and by Lemma 5.4 $A(c, r+1)[\ell] = M_c^{r+1}[\ell]$. Consequently, we have that $A(c_2, r+1)[\ell] \neq\perp$ and $A(c_3, r+1)[\ell] \neq\perp$. By the definition of a process chain, c_2 and c_3 cannot be both in ω_ℓ, a contradiction.

□

Lemma 5.8 *Let ϕ be an execution of ROC such that there is a chain ω_ℓ of length at least three. There is no two correct processes p_i, p_j such that $i, j \in \omega_\ell$ and $\omega_\ell = (\omega' \circ i \circ \omega \circ j \circ \omega'')_\ell$, where $\omega, \omega', \omega''$ are substrings of ω_ℓ and ω is not the empty string.*

Proof:
Proof by contradiction. Suppose that there are two correct processes p_i and p_j in ϕ such that $\omega_\ell[r] = i$ and $\omega_\ell[r'] = j$, $r+1 < r'$. By Lemma 5.6 and by the definition of a process chain, for every correct process p_j in $Correct(\phi)$, $M_j^{r+1} \neq\perp$. We hence have that r' must be equal to $r+1$, and ω must be empty, contradicting out initial assumption that $r+1 < r'$.

□

Lemma 5.9 *Let ϕ be an execution of ROC and p_i be a process in* Live(r)*, $r \geq 2$, such that $A(i,r)[\ell] \neq\perp$ and for all $r' \in \{x \in \mathcal{R} : x \leq r-1\}$, $A(i,r')[\ell] =\perp$. For every round $\rho \in \{x \in \mathcal{R} : 1 \leq x \leq r\}$, there are processes $p_{j_1} \in$* Live(ρ) *and $p_{j_2} \in$* Live($\rho - 1$) *such that the following holds:*

1. *$A(j_1,\rho)[\ell] \neq\perp$, and for all $\rho' \in \{x \in \mathcal{R} : x \leq \rho - 1\}$, $A(j_1,\rho')[\ell] =\perp$;*
2. *$A(j_2,\rho-1)[\ell] \neq\perp$, and for all $\rho' \in \{x \in \mathcal{R} : x \leq \rho - 2\}$, $A(j_2,\rho')[\ell] =\perp$;*
3. *$\exists m_{j_2} \in M(j_1,\rho) : (m_{j_2}.A[\ell] \neq\perp)$.*

Proof:
We now show with an induction on the values of ψ, $0 \leq \psi \leq r-1$, that for every round $\rho = r - \psi$, the claim holds.

The base case is $\psi = 0$, $\rho = r$. By assumption, p_i is such that $A(i,r)[\ell] \neq\perp$ and for all $r' \in \{x \in \mathcal{R} : x \leq r-1\}$, $A(i,r')[\ell] =\perp$. This is implies that $M_i^r[\ell] \neq\perp$. From the algorithm, we have that $p_i.s(\rho) \subseteq p_i.s(\rho - 1) \subseteq \ldots \subseteq p_i.s(0)$. Consequently, there must be some process $p_j \in Live(\rho - 1)$ such that the following holds: A) $A(j, r-1) \neq\perp$; B) $A(j,\rho') =\perp$ for all $\rho' \in \{x \in \mathcal{R} : x \leq r-1\}$; C) $\exists m_j \in M(i,r) : (m_j.A[\ell] \neq\perp)$. If there is no such a process p_j that satisfies both A) and C), then $A(i,r) =\perp$, contradicting our initial assumption. By the algorithm, once p_j sets the value of $p_j.A[\ell]$ to a value different than $\perp$, then value of $p_j.A[\ell]$ does not change in subsequent rounds. This implies that for all ρ', $0 \leq \rho' < r-1$, $A(j,\rho')$ must be equal to $\perp$, because by the algorithm p_i receives a message from p_j at every round ($p_i.s(\varrho) \subseteq p_i.s(\varrho - 1)$, $r \geq \varrho > 0$) and $A(i, \rho'')$ is different than $\perp$ otherwise, for some $\rho'' < r$.

Suppose the claim is true for ψ. We show for $\psi + 1$. If it is true for ψ, then there is a process p_{j_1} live in $\rho = r - (\psi + 1)$ such that $A(j_1, \rho)[\ell] \neq\perp$, and for all $\rho' \in \{x \in \mathcal{R} : x \leq \rho - 1\}$, $A(j_1, \rho')[\ell] =\perp$. From the algorithm, we have that $p_{j_1}.s(\rho) \subseteq p_{j_1}.s(\rho - 1) \subseteq \ldots \subseteq p_{j_1}.s(0)$. Consequently, there must be some process $p_{j_2} \in Live(\rho - 1)$ such that the following holds: A) $A(j_2, \rho - 1) \neq\perp$; B) $A(j_2, \rho') =\perp$ for all $\rho' \in \{x \in \mathcal{R} : x \leq \rho - 1\}$; C) $\exists m_{j_2} \in M(j_1, \rho) : (m_{j_2}.A[\ell] \neq\perp)$. If there is no such a process p_{j_2} that satisfies both A) and C), then $A(j_1, \rho) =\perp$, contradicting our assumption that the hypothesis hold for ψ. By the algorithm, once p_{j_2} sets the value of $p_{j_2}.A[\ell]$ to a value different than $\perp$, then the value of $p_{j_2}.A[\ell]$ does not change in subsequent rounds. This implies that for all ρ', $0 \leq \rho' < \rho - 1$, $A(j_2, \rho')$ must be equal to $\perp$, because by the algorithm p_{j_1} receives a message from p_{j_2} at every round ($p_{j_1}.s(\varrho) \subseteq p_{j_1}.s(\varrho - 1)$, $\rho \geq \varrho > 0$) and $A(i, \rho'')$ is different than $\perp$ otherwise, for some $\rho'' < \rho$.

This concludes the proof of the lemma.

□

Lemma 5.10 *Let ϕ be an execution of ROC and p_i be a process that is live at round r of ϕ, $r \geq 0$, such that $A(i, r)[\ell] \neq\perp$ and for all $r' \in \{x \in \mathcal{R} : x \leq r - 1\}$, $A(i, r')[\ell] =\perp$. There is a chain ω_ℓ such that $|\omega_\ell| = r + 1$, and $\omega_\ell[r] = i$.*

Proof:

We have to build a chain ω_ℓ such that $|\omega_\ell| = r + 1$, and $\omega_\ell[r] = i$.

We build such a chain ω_ℓ as follows:

$$\begin{aligned}
&\omega_\ell[0] = \ell \\
&\omega_\ell[\rho] = j \quad , \quad \wedge(0 < \rho < r) \\
&\qquad\qquad\qquad \wedge(A(j, \rho) \neq\perp) \\
&\qquad\qquad\qquad \wedge(\forall \rho' \in \{x \in \mathcal{R} : x \leq \rho\} : A(j, \rho') =\perp) \\
&\qquad\qquad\qquad \wedge\exists m_j \in M(\omega_\ell[\rho + 1], \rho + 1) \text{ from } p_j \\
&\omega_\ell[r] = i
\end{aligned}$$

We can easily verify that ω_ℓ satisfies the properties of a process chain. It remains to show that it is a valid construction.

By definition, we have that $\omega_\ell[0] = \ell$. By Lemma 5.9, we have that for every ρ, $0 < \rho < r$, there is a process p_j that satisfies the properties we stated above. Finally, by assumption, p_i is such that $A(i, r)[\ell] \neq\perp$ and for all $r' \in \{x \in \mathcal{R} : x \leq r - 1\}$, $A(i, r')[\ell] =\perp$.

This concludes the proof of the lemma.

□

Lemma 5.11 *Let ϕ be an execution of ROC. If $p_i, p_j \in$ Correct(ϕ), then $p_i.d = p_j.d$ in ϕ.*

Proof:

By Lemma 5.5, every correct process decides in ϕ (no correct process stops). Now let r, $0 \leq r \leq t$, be a round in which no process crashes. Such a round exists in ϕ by assumption (no more than t processes can fail in an execution, where t is $|\Pi|$ subtracted the size of the smallest survivor set).

We first show by induction on the values of ρ, $r + 1 \leq \rho \leq t + 1$, the following proposition:

$$\begin{aligned}
&\bigwedge \quad \forall p_{c_1}, p_{c_2} \in Correct(\phi) : A(c_1, \rho) = A(c_2, \rho) \\
&\bigwedge \quad \forall p_\ell \in Live(\rho), p_c \in Correct(\phi) : A(\ell, \rho) \subseteq A(c, \rho)
\end{aligned}$$

The base case is $\rho = r + 1$. According to the algorithm, every process p_i that is live at round r sends a message containing $A(i, r)$ to every other process. According to C-Liveness and the assumption that no process crashes in round r, for every process $p_c \in Correct(\phi)$, $p_c.s(r + 1) = Live(r)$. This implies that for every $p_{c_1}, p_{c_2} \in Correct(\phi)$, $M_{c_1}^{r+1} = M_{c_2}^{r+1}$. By Lemma 5.4, $M_c^{r+1} = A(c, r+1)$ for every $p_c \in Correct(\phi)$. This implies that for every $p_{c_1}, p'_{c_2} \in Correct(\phi)$, $A(c_1, r + 1) = A(c_2, r + 1)$.

It remains to show the second part of the proposition for the base case. Let p_ℓ be a process in $Live(r + 1)$ and p_c be a process in $Correct(\phi)$. By the failure assumptions, we have that $p_\ell.s(r + 1) \subseteq Live(r)$. This implies that $p_\ell.s(r + 1) \subseteq p_c.s(r + 1)$. If $p_\ell.s(r + 1) \subseteq p_c.s(r + 1)$, then $M_\ell^{r+1} \subseteq M_c^{r+1}$. By Lemma 5.4, $M_\ell^{r+1} = A(\ell, r + 1)$ and $M_c^{r+1} = A(c, r + 1)$, which implies that $A(\ell, r + 1) \subseteq A(c, r + 1)$. This concludes the proof of the base case.

Suppose that the proposition holds for every ρ. We show for $\rho + 1$. By the induction hypothesis and the algorithm, for every process $p_c \in Correct(\phi)$, $A(c, \rho) = M_c^{\rho+1}$. By Lemma 5.4, for every $p_c \in Correct(\phi)$, $A(c, \rho + 1) =$

$M_c^{\rho+1}$. We conclude that for every $p_{c_1}, p_{c_2} \in Correct(\phi)$, $A(c_1, \rho + 1) = A(c_2, \rho + 1)$.

By our failure assumptions, a faulty process may receive an arbitrary subset of the messages sent to it in a round. Let p_ℓ be a process in $Live(\rho + 1)$ and p_c be a process in $Correct(\phi)$. By the induction hypothesis and the algorithm, $M_\ell^{\rho+1} \subseteq M_c^{\rho+1}$. By Lemma 5.4, $A(\ell, \rho + 1) = M_\ell^{\rho+1}$ and $A(c, \rho + 1) = M_c^{\rho+1}$. We conclude that $A(\ell, \rho + 1) \subseteq A(c, \rho + 1)$, This concludes the proof of the induction step.

From the previous proposition, we have that $A(i, t+1) = A(j, t + 1)$. By the algorithm, p_i decides upon $A(i, t + 1)$ and p_j decides upon $A(j, t+1)$. Consequently, $p_i.d = p_j.d$. This concludes the proof of the lemma.

□

Lemma 5.12 *Let ϕ be an execution of ROC and p_i, p_j be two processes in* Live$(t + 1)$*, and p_ℓ be a process in Π. If $(A(i, t + 1)[\ell] \neq \perp)$ and for all $r \in \{z \in \mathcal{R} : z \leq t\}$, $(A(i, r)[\ell] = \perp)$, then $p_j.d[\ell] \neq \perp$.*

Proof:

By the algorithm, once p_j sets the value of $p_j.A[\ell]$ to a value different than $\perp$, p_j does not change it in subsequent rounds. Thus, we only need to show that there is some round r in which p_j sets $p_j[\ell]$ to a value different than $\perp$.

Suppose that $((A(i, t + 1)[\ell] \neq \perp) \wedge (\forall r \in \{z \in \mathcal{R} : z \leq t\} : A(i, r)[\ell] = \perp))$. Assuming that p_i and p_j can be either correct or faulty, there are four possible cases, and we analyze each case separately as follows:

- p_i and p_j are correct in ϕ. By Lemma 5.11, we have that $p_i.d = p_j.d$;

- p_i is faulty and p_j is correct in ϕ. If p_i decides in ϕ, then p_i is in $Live(t + 1)$. By Lemma 5.10, there is a chain ω_ℓ such that $|\omega_\ell| = t+2$, and $\omega_\ell[t+1] = i$. Since there are at most t failures by assumption, there is at least one correct process in ω_ℓ. Moreover, such correct process must be in a position r of the chain such that $0 \leq r \leq t$. Thus, $A(j, t + 1)[\ell]$ must be different than $\perp$, by Lemma 5.6 and the algorithm;

- p_i is correct and p_j is faulty in ϕ. If p_i is correct, then, by Lemma 5.10, there is a chain ω_ℓ such that $|\omega_\ell| = t + 2$, and $\omega_\ell[t + 1] = i$. Because we assume that p_i is correct, for every r, $0 \leq r \leq t - 1$ and $\kappa = \omega_\ell[r]$, p_κ must crash at round r of ϕ. Otherwise, $A(i, r)[\ell] \neq \perp$ for some $r < t + 1$. Because $p_j \in Live(t + 1)$ by assumption, there must be at least $t + 1$ faulty processes, violating our assumptions for survivor sets. This case is hence not possible;

- p_i and p_j are faulty in ϕ. By Lemma 5.10, there is a chain ω_ℓ such that $|\omega_\ell| = t + 2$, and $\omega_\ell[t + 1] = i$. By Lemma 5.7, there are at most two correct processes in any chain. Thus, ω_ℓ contains t faulty processes. Consequently, there must be an r, $0 \leq r \leq t$, such that $j = \omega_\ell[r]$, and $A(j, r) \neq \perp$.

From the previous analysis, we have that $A(j, t + 1) \neq \perp$. By the algorithm, we have p_j decides upon $A(j, t + 1)$. Consequenty, $p_j.d[\ell] \neq \perp$. This concludes the proof of the lemma.

□

Lemma 5.13 *Let ϕ be an execution of ROC, p_i, p_j be two processes in* Live$(t + 1)$*, and S_i, S_j be two survivor sets in S_Π such that for all $r \in \{z \in \mathcal{R} : z \leq t + 1\}$, $S_i \subseteq p_i.sr(r)$, $S_j \subseteq p_j.sr(r)$, and $S_i \cap S_j \neq \emptyset$. $p_i.d = p_j.d$ in ϕ.*

Proof:

By the algorithm, once a process p_i sets the value of $p_i.A[\ell]$ at round $r \in \{z \in \mathcal{R} : z \leq t\}$ to a value different than $\perp$, it does not change it on subsequent rounds. We then have to show that if p_i learns about the initial value of p_ℓ at round r (that is, $A(i, r)[\ell] \neq \perp$ and for all $r' \in \{z \in \mathcal{R} : z \leq r - 1\}$, $A(i, r')[\ell] \neq \perp$), then there is a round r' such that p_j learns the initial value of p_ℓ at round r' (that is, $A(j, r')[\ell] \neq \perp$ and for all $r'' \in \{z \in \mathcal{R} : z \leq r' - 1\}$, $A(j, r'')[\ell] \neq \perp$). By Lemma 5.2, if $A(i, r)[\ell] = A(j, r')[\ell] \neq \perp$, then $A(i, r)[\ell] = A(j, r')[\ell] = A(\ell, 0)[\ell]$. We now analyze each case separately.

First, suppose that $((A(i, t + 1)[\ell] \neq \perp) \wedge (\forall r \in \{z \in \mathcal{R} : z \leq t\} : A(i, r)[\ell] = \perp))$. This follows directly from Lemma 5.12.

Now, suppose that $(A(i, t)[\ell] \neq \perp) \wedge (\forall r \in \{z \in \mathcal{R} : z \leq t - 1\} : A(i, r)[\ell] = \perp)$:

- p_i and p_j are correct in ϕ. By Lemma 5.11, we have that $p_i.d = p_j.d$.

- p_i is faulty and p_j is correct in ϕ. From Lemma 5.10, there is a chain ω_ℓ such that $|\omega_\ell| = t+1$, and $\omega_\ell[t] = i$. Because there are at most t faulty processes by assumption, there must be a correct process in ω_ℓ. That is, there must be some r, $0 \leq r \leq t-1$, such that $\omega_\ell[r]$ is the identifier of a correct process in ϕ. It follows that $A(j, r+1)$ must be different than $\perp$, by Lemma 5.6.

- p_i is correct and p_j is faulty in ϕ. From Lemma 5.10, there is a chain ω_ℓ such that $|\omega_\ell| = t+1$, and $\omega_\ell[t] = i$. Because p_j is faulty, either there is some r such that $\omega_\ell[r] = j$ or there are at most $t-1$ faulty processes in ω_ℓ. If the former holds, then we are done. If the latter holds, then $\omega_\ell[t-1]$ must be the identifier of a correct process. Otherwise there is some $r \in \{z \in \mathcal{R} : z \leq t-1\}$ such that $A(i,r)[\ell] \neq\perp$ (by Lemma 5.6). In addition, because ω_ℓ contains at least $t-1$ faulty process and p_j is faulty, any $p_\chi \in (S_i \cap S_j)$ must be correct. Thus, if there are at most $t-1$ faulty processes in ω_ℓ and $\omega_\ell[t-1]$ is the identifier of a correct process, then $A(\chi, t) \neq\perp$. Since by assumption $S_j \subseteq p_j.sr(r)$ for every $r \in \{z \in \mathcal{R} : z \leq t+1\}$, we have that $A(j, t+1) \neq\perp$, by the algorithm and Lemma 5.4;

- p_i and p_j are faulty in ϕ. From Lemma 5.10, there is a chain ω_ℓ such that $|\omega_\ell| = t+1$, and $\omega_\ell[t] = i$. Because ω_ℓ contains exactly $t+1$ process identifiers, at least one must be correct, and by Lemma 5.7, at most two correct processes. We then have that either there is some r such that $\omega_\ell[r] = j$ or $\omega_\ell[r] \neq j$ for every r. In the former case, we have that $A(j,r)[\ell] \neq\perp$ for some $r < t$. In the latter, ω_ℓ must have $t-1$ faulty processes (at most two correct processes and p_j is not in ω_ℓ). We then have that $p_\chi \in (S_i \cap S_j)$ is either faulty or correct. If p_χ is faulty, then either there is $r \in \{z \in \mathcal{R} : z \leq t-1\}$ such that $\omega_\ell[r] = \chi$ or $\chi = j$. The case that $\omega_\ell[r] = \chi$ is straightforward. The second case, in which $\chi = j$, follows from our assumptions that $S_i \subseteq p_i.sr(r)$ and $S_j \subseteq p_j.sr(r)$ for every $r \in \{z \in \mathcal{R} : z \leq t+1\}$, the algorithm ($M(i,r)$ contains a message from p_j for every round $r \in \{z \in \mathcal{R} : 1 \leq z \leq t+1\}$), and by Lemma 5.4.

  Now suppose p_χ is correct. Because there is some correct process in $\omega_ell[r]$, for $r \in \{z \in \mathcal{R} : z \leq t-1\}$, by Lemma 5.6, it must be the case that $A(\chi, t-1)[\ell] \neq\perp$ and consequently $A(j,t)[\ell] \neq\perp$, by the algorithm and Lemma 5.4.

Finally, suppose that $\exists r \in \{z \in \mathcal{R} : z \leq t-1\} : (A(i,r)[\ell] \neq\perp) \wedge (\forall r' \in \{z \in \mathcal{R} : z \leq r-1\} : A(i,r')[\ell] = \perp)$. By assumption $S_i \subseteq p_i.sr(\rho)$ for all $\rho \in \{z \in \mathcal{R} : z \leq t+1\}$. This implies by the algorithm that $A(\chi, r+1) \subseteq A(i, r+2)$, $p_\chi \in S_i \cap S_j$. Because $S_j \subseteq p_j.sr(\rho)$, for all $\rho \in \{z \in \mathcal{R} : z \leq t+1\}$, and $p_\chi \in S_j$, we then have by the algorithm and Lemma 5.4 that $A(j, r+2)[\ell]$ must be different than $\perp$, and equal to $A(i,r)[\ell]$ by Lemma 5.2.

From the previous argument, we conclude that $A(i, t+1) = A(j, t+1)$. By the algorithm, we have that p_i decides upon $A(i, t+1)$ and p_j decides upon $A(j, t+1)$. Again by the algorithm, we have that $p_i.d = p_j.d$.

□

Lemma 5.14 *Let ϕ be an execution of* ROC *and p_i, p_j be two processes in* Live$(t+1)$ *such that $p_j \in p_i.sr(r)$ for every $r \in \{z \in \mathcal{R} : z \leq t+1\}$. $p_j.d \subseteq p_i.d$ in ϕ.*

Proof:

By the algorithm, once a process p_j sets the value of $p_j.A[\ell]$ to a value different than $\perp$ in a round r, for some $p_\ell \in \Pi$ and some $0 \leq r \leq t+1$, it does not change it in subsequent rounds. If $A(j, t+1)[\ell] \neq\perp$, then there is some round ρ, $0 \leq \rho \leq t+1$, such that $A(j,\rho)[\ell] \neq\perp$ and for all $\rho' \in \{z \in \mathcal{R} : z \leq \rho-1\}$, $A(j,\rho')[\ell] \neq\perp$. We then have to show that for every $p_\ell \in \Pi$ such that $A(j, t+1)[\ell] \neq\perp$, there is some ϱ such that $A(i,\varrho)[\ell] \neq\perp$, $\varrho \in \{z \in \mathcal{R} : z \leq t+1\}$.

Let p_ℓ be a process such that $A(j,\rho)[\ell] \neq\perp$ and for all $\rho' \in \{x : 0 \leq x < \rho\}$, $A(j,\rho')[\ell] \neq\perp$. Suppose that $\rho = t+1$. This case follows directly from Lemma 5.12. Now suppose that $\rho \leq t$. Because p_j sends a message to p_i in every round by assumption, $M_i^{\rho+1}[\ell]$ must be different than $\perp$, and $A(i, \rho+1)[\ell] = M_i^r[\ell]$ by Lemma 5.4. Thus, $\varrho \leq \rho+1$. We conclude that if $A(j, t+1)[\ell] \neq\perp$, for some $p_\ell \in \Pi$, then $A(i, t+1)[\ell] \neq\perp$. By Lemma 5.2, $A(i, t+1) = A(j, t+1) = p_\ell.a$. By the algorithm, p_i decides upon $A(i, t+1)$ and p_j decides upon $A(j, t+1)$.

Consequently, $p_j.d \subseteq p_i.d$.

□

Lemma 5.15 *Let ϕ be an execution of ROC. If p_i, p_j, and p_ℓ decide in ϕ, then either $p_i.d = p_j.d$, $p_i.d = p_\ell.d$, or $p_j.d = p_\ell.d$.*

Proof:
If p_i, p_j, and p_ℓ decide in ϕ, then there are survivor sets S_i, S_j, and S_ℓ such that $S_i \subseteq p_i.sr(r)$, $S_j \subseteq p_j.sr(r)$, and $S_\ell \subseteq p_\ell.sr(r)$, for all r, $0 \leq r \leq t+1$. By the (3,2)-Intersection property, either $S_i \cap S_j \neq \emptyset$, $S_i \cap S_\ell \neq \emptyset$, or $S_j \cap S_\ell \neq \emptyset$. By Lemma 5.13, we then have that either $p_i.d = p_j.d$, $p_i.d = p_\ell.d$, or $p_j.d = p_\ell.d$.

□

Lemma 5.16 *Let ϕ be an execution of ROC and p_i be a correct process in ϕ. If p_j decide in ϕ, then $p_j.d \subseteq p_i.d$.*

Proof:
By Lemma 5.5, $p_i \in Live(t+1)$ (p_i decides in ϕ). If p_j is correct, then the Lemma follows from Lemma 5.11. Now suppose that p_j commits at least one receive-omission fault in ϕ. By assumption, both p_i and p_j decide in ϕ. Because p_i is correct, we have that there is m_j from p_j in $M(i,r)$ for every r, $0 \leq r \leq t+1$. By the algorithm, $p_j \in p_i.s(r)$ for every r, $0 \leq r \leq t+1$. This means that p_i receives a message from p_j in every round of the execution. By Lemma 5.4 and the algorithm, we then have that if $A(j,r)[\ell] \neq \perp$, for some $p_\ell \in \Pi$ and $0 \leq r \leq t$, then $A(i,r+1)[\ell] = A(j,r)[\ell]$. It remains to show that if $A(j,t+1)[\ell] \neq \perp$, and $A(j,r)[\ell] = \perp$, $p_\ell \in \Pi$, for every $r \in \{z \in \mathcal{R} : z \leq t\}$, then $A(i,t+1)[\ell] = A(j,t+1)[\ell]$. By Lemma 5.12, we have that if $A(j,t+1)[\ell] \neq \perp$, then $A(i,t+1)[\ell] \neq \perp$. By Lemma 5.2, $A(i,t+1)[\ell] = A(j,t+1)[\ell] = p_\ell.a$. We conclude that $p_j.d \subseteq p_i.d$.

□

Lemma 5.17 *Let ϕ be an execution of ROC. If there are two processes p_i and p_j, $p_i, p_j \in$ Live$(t+1)$, then either $p_i.d \subseteq p_j.d$ or $p_j.d \subseteq p_i.d$.*

Proof:
If at least one of p_i and p_j is correct, then the proof follows from Lemma 5.16. Now suppose both p_i and p_j are faulty. Because both p_i and p_j decide in ϕ by assumption, there are survivor sets S_i and S_j such that $(S_i \subseteq p_i.sr(r)) \wedge (S_j \subseteq p_j.sr(r))$ for every r, $0 \leq r \leq t+1$. If $S_i \cap S_j \neq \emptyset$, then the lemma follows because by Lemma 5.13 $p_i.d = p_j.d$. Suppose now the contrary: $S_i \cap S_j = \emptyset$. By assumption, there must be a survivor set S_c containing only correct processes. By the (3,2)-Intersection property, either $S_i \cap S_c \neq \emptyset$ or $S_j \cap S_c \neq \emptyset$. Let p_c be a process in S_c. We then have by Lemma 5.13 that either p_i and p_c decide upon the same value or p_j and p_c decide upon the same value. Suppose without loss of generality that p_i and p_c decide upon the same value. We hence have from Lemma 5.16 that $p_j.d \subseteq p_i.d$. This concludes the proof of the lemma.

□

Lemma 5.18 *Let ϕ be an execution and* vals *be* $\{d : \exists p_i \in \Pi \text{ s.t. } (p_i.d = d)\} \setminus \mathcal{N}$. *For every $d_f, d_c \in$ vals, $d_f \subseteq d_c$, there are survivor sets $S_f, S_c \in S_\Pi$ such that the following properties holds:*

$$\begin{array}{llll} \bigwedge & \forall p \in S_f : & \vee & p \textit{ crashes} \\ & & \vee & p.d = d_f \\ \bigwedge & \forall p \in S_c : & \wedge & p.d = d_c \\ & & \wedge & p \textit{ is not faulty} \end{array}$$

Proof:
By Lemma 5.5, $Correct(\phi) \subseteq Live(t+1)$. By the algorithm, every non-faulty process p_i is such that $p_i.d[i] = p_i.a$. We then have that *vals* contains at least one value. By Lemma 5.15, there cannot be more than three different decision values, and if there are two values d and d', then either $d \subseteq d'$ or $d' \subseteq d$ by Lemma 5.17. We analyze these two cases separately.

First, suppose that *vals* contains a single value, say d, and $d_f = d_c = d$. By assumption, there is a survivor set S_i such that S_i contains only non-faulty processes. By Lemma 5.11, every process $p_i \in S_i$ is such that $p_i.d = d$. If we make $S_c = S_f = S_i$, then our claim holds.

Now suppose that *vals* contains two values d_f and d_c, $d_f \subseteq d_c$. Let p_i be a process such that $p_i \in Live(t+1)$ and $p_i.d = d_f$. By the algorithm, there is survivor set S_i such that $S_i \subseteq p_i.sr(r)$, for every $r \in \{z \in \mathcal{R} : z \leq$

$t+1\}$. Let p_j be a process in S_i. If $p_j \in \mathit{Live}(t+1)$, then there is a $S_j \in S_\Pi$ such that $S_j \subseteq p_j.sr(r)$, for every $r \in \{z \in \mathcal{R} : z \leq t+1\}$. Now let S'_c be a survivor set such that $S'_c \subseteq \mathit{Correct}(\phi)$. By the (3,2)-Intersection property, either $S_j \cap S'_c \neq \emptyset$ or $S_j \cap S_i \neq \emptyset$. Note that $S_i \cap S'_c$ must be empty, otherwise $p_i.d = d_c$ according to Lemma 5.13, contradicting our initial assumption.

If $S_j \cap S'_c \neq \emptyset$, then by Lemma 5.13 we have that $p_j.d = d_c$ because $p_j.d = p_c.d$ for every $p_c \in S'_c$ (Lemma 5.13) and $p_c.d$, $p_c \in S'_c$, must be equal to d_c (Lemma 5.16). By Lemma 5.14, however, we have that $p_j.d \subseteq p_i.d$. This implies that $p_i.d = d_c$, again contradicting our initial assumption. It therefore must be the case that $S_i \cap S_j$ is not empty. By Lemma 5.13, we have that $p_i.d = p_j.d = d_f$.

Now suppose that $p_j \notin \mathit{Live}(t+1)$. We then have that p_j crashes in ϕ, and $p_j.d = \mathcal{N}$. We therefore have have that $p_j \in S_i$ either decides upon d_f or crashes in ϕ.

It remains to show the second part of the properties in the statement of the lemma. By Lemma 5.16, every correct process must decide upon d_c. Thus, every process p_c in S is such that $p_c.d = d_c$ in ϕ.

To conclude, if we make $S_f = S_i$ and $S_c = S'_c$, then our claim holds, as we wanted to show.

□

Lemma 5.19 *Let ϕ be an execution of ROC such that there are survivor sets $S_f, S_c \in S_\Pi$ and values $v_f, v_c \in V$, $v_f \neq v_c$, such that the following holds:*

$$\begin{array}{l} \bigwedge \ \forall p \in S_f : p.a \in \{v_f, \perp\} \\ \bigwedge \ \forall p \in S_c : p.a = v_c \\ \bigwedge \ \forall p \in S_c : p \textit{ is not faulty} \end{array}$$

If exists $p_i, p_\ell \in \Pi$ such that $p_i.d[\ell] = v_f$, then for all $p_j \in \mathrm{Live}(t+1)$, $v_f \in p_j.d$.

Proof:

Suppose that $p_i.d[\ell] = v_f$, for some $p_i, p_\ell \in \Pi$. By the algorithm, if a process p_j does not crash in an execution of ROC, then there is some survivor set S_j such that $S_j \subseteq p_j.sr(r)$ for every $r \in \{z \in \mathcal{R} : z \leq t+1\}$. S_f and S_c must be disjoint, otherwise there is some process with two different initial values. By the (3,2)-Intersection property, either $S_j \cap S_f \neq \emptyset$ or $S_j \cap S_c \neq \emptyset$. If $S_j \cap S_f \neq \emptyset$, then: 1) $A(j,r)[x] = M^r_j[x]$ by Lemma 5.4, for some $p_x \in S_j \cap S_f$, $p_x.a \neq \perp$, and for every $r > 0$; 2) $M^r_j[x] = p_x.a$ by the algorithm and Lemma 5.3. We then have by the algorithm that $A(j, t+1)[x] = p_j.d[x] = p_x.a = v_f$.

If $S_j \cap S_c \neq \emptyset$, then $p_j.d = p_c.d$ by Lemma 5.13, for every $p_c \in S_c$. By Lemma 5.16, $p_i.d \subseteq p_c.d$ for every non-faulty p_c. That is, we have that $p_c.d[\ell] = p_i.d[\ell] = v_f$. We then have that $p_j.d[\ell] = p_c.d[\ell] = p_\ell.a = v_f$. This concludes the proof of the lemma.

□

Theorem 5.20 *Algorithm ROC satisfies Termination.*

Proof:

This is straightforward from the algorithm: every process that does not crash in an execution of ROC decides at round $t+1$.

□

Theorem 5.21 *Algorithm ROC satisfies Agreement.*

Proof:

By Lemma 5.16, if a process p_j decides in ϕ, then $p_j.d \subseteq p_i.d$ for every non-faulty p_i. This implies that for every p_ℓ such that $p_j.d[\ell] \neq \perp$, we have that $p_i.d[\ell] = p_j.d[\ell]$ for every non-faulty p_i.

□

Theorem 5.22 *Algorithm ROC satisfies RO Uniformity.*

Proof:

By Lemma 5.5, every correct process decides in ϕ. By the algorithm, for every non-faulty process p_i, $p_i.d[i] = p_i.a$. Thus, there must be at least one non-$\perp$ decision value. By Lemma 5.15, there cannot be three processes in an execution of ROC such that each process decides upon a different value. This shows the first statement of the property: $1 \leq |\mathit{vals}| \leq 2$, where $\mathit{vals} = \{p_1.d, \ldots, p_n.d\} \setminus \mathcal{N}$ in any execution of ROC. The second statement follows directly from Lemma 5.17. The third statement follows directly from Lemma 5.18.

□

Theorem 5.23 *Algorithm ROC satisfies Validity.*

Proof:
If $p_i \in Live(t+1)$ in some execution ϕ of ROC, then by Lemmas 5.5 and 5.16 $p_i.d \subseteq p_j.d$, for every $p_j \in Correct(\phi)$. By the algorithm, $p_i.d[i]$ must be equal to $p_i.a$. We consequently have that $p_j.d[i]$ must be equal to $p_i.a$. This proves the first statement in the specification of Validity.

If p_i crashes in an execution ϕ of ROC, then by Lemmas 5.2 and 5.11 either $p_j.d[i] = \perp$ or $p_j.d[i] = p_i.a$, for every $p_j \in Correct(\phi)$. This shows the second statement in the definition of validity. The third statement follows directly from Lemma 5.19.

□

With Theorems 5.20, 5.23, 5.21, and 5.22, we show that ROC implements the four RO Consensus properties, thereby showing Proposition 5.1.

6 Correctness of WLE

Algorithm WLE proceeds in iterations of an infinite repeat loop. In each iteration, processes execute two phases, and in each phase a process participates in the execution of an algorithm that implements RO Consensus. For the following description, we assume that such an algorithm is ROC. As shown in Figure 2, a process that does not crash in an execution of WLE executes infinitely many iterations of the repeat loop. According to our system model, we split an execution of an algorithm into rounds. We then number the iterations of an execution of WLE and assume that round numbers map to iteration numbers. That is, there is mapping $Iteration : \mathcal{R} \rightarrow \mathcal{I}$, where $\mathcal{R}$ is the set of round numbers as before, and $\mathcal{I} = \mathbb{Z}^*$ is the set of iteration numbers. In addition, we further assume that iteration numbers increase monotonically with round numbers, and the number of rounds executed in an iteration is fixed, being a function of the number of rounds in an execution of ROC[4]. For the purpose of the proofs that follow, we only need to assume that each phase executes at least two rounds. Note that ROC requires $t+1$ rounds, and $t+1 \geq 2$ if $t \geq 1$. We therefore have that each phase must have at least two rounds, assuming systems in which processes can fail. In fact, because processes can fail by crashing and we assume cores to model failures patterns, we can use the same argument as in [3] to show that $t+1$ is a lower bound on the number of rounds.

According to the discussion in the previous paragraph, we associate an iteration number with each iteration of the algorithm in an execution. In the following, we use iteration numbers to refer to iterations of the algorithm. In proving the correctness of WLE, we also use the following definitions:

- $vals_i$, $i \in \{1,2\}$ is the set $\{d : \exists p_i \in \Pi(p_i.d = d)\} \setminus \mathcal{N}$ after executing ROC at phase i of some iteration ζ, $\zeta \in \mathcal{I}$;
- a process p_i finishes a phase $x \in \{1,2\}$ of some iteration ζ in an execution ϕ if it neither stops nor crashes before executing the last step of that phase;
- An iteration ζ of an execution ϕ of WLE starts after a time τ if the first step s of every process that executes at least one step in ζ is such that $\mathcal{T}(s) > \tau$.

As in Section 4, we assume that WLE uses a system profile $\langle \Pi, C_\Pi, S_\Pi \rangle$ and that this profile satisfies (3,2)-Intersection.

Now we show the following proposition.

Proposition 6.1 WLE implements Weak Leader Election.

We show proposition 6.1 with the following set of theorems, each one proving a property of Weak Leader Election.

Theorem 6.2 *Algorithm WLE satisfies Safety.*

Proof:
Let $\phi = \langle F, I, S, T, \mathcal{T} \rangle$ be an execution of WLE. We have to show that $|\{p_i \in \Pi : p_i.elected\}| < 2$ for every $\tau \in T$. First, we show that in an iteration of ϕ, at most one process is elected. By the RO Uniformity property of RO Consensus, there is at least one decision value and at most two different decision values as a result of phase 1. That

[4] Because there are infinitely many executions of ROC in an execution of WLE and round numbers monotonically increase with time, the round numbers in the pseudocode for ROC are relative to the first round in which an execution of ROC starts.

is, $1 \leq |vals_1| \leq 2$. Suppose $|vals_1| = 1$. By the algorithm, every process p_i that finishes phase 2 uses a list x in its decision value $p_i.d$, where x has the minimum number of non-$\bot$ values among all lists in $p_i.d$. x must be the initial value of some processes by Validity. By assumption, there is a single initial value in phase 2, which implies that every process that finishes phase 2 uses the same list x when deciding whether to set *elected* or not.

Now suppose that $|vals_1| = 2$. From RO Uniformity, we have that there are values $d_1, d_2 \in vals_1$ and $S_1, S_2 \in S_\Pi$ such that:

$$\begin{array}{rll} \wedge \ \forall p \in S_1 : & \vee & p \text{ crashes} \\ & \vee & p.d = d_1 \\ \wedge \ \forall p \in S_2 : & \wedge & p.d = d_2 \\ & \wedge & p \text{ is not faulty} \end{array}$$

By the algorithm, a process that finishes phase 1 of an iteration executes ROC once more in phase 2 with its decision value of the previous phase as its initial value. If the above properties hold, then the only processes in S_1 that do not have d_1 as initial value are the ones that crash before phase 2 starts. Let's call this set $Crash_1$. By Validity, if some process p_i decides upon a value $p_i.d$ such that $d_1 \in p_i.d$, then every process p_j that finishes phase 2 is such that $d_1 \in p_j.d$. We then again have that there is a single process that can be elected because every process that finishes phase 2 has d_1 in its decision value and $d_1 \subseteq d_2$ by Agreement.

It remains to show that if p_i is elected in iteration ζ, and p_j is elected in iteration ζ', and $\zeta > \zeta\prime$, then there is no $\tau \in T$ such that both $p_i.elected$ and $p_j.elected$ are true at time τ. By the algorithm, every process that starts the execution of phase 1 in an iteration, first sets its flag *elected* to false. If the iteration is the first of ϕ, then p_j cannot be elected in a previous iteration, and the hypothesis is vacuosly true. Now suppose an iteration $\zeta > 0$. By assumption, every process that executes a phase of an iteration ζ executes at least two rounds. By P-Liveness, no process can start a new round $r + 1$, $r \geq 0$, without having every other live process executing at least one step of r. If a process p_i starts phase 2 of an iteration at time τ, then every process that has not crashed by τ must have executed at least one step of phase 1 of ζ. Otherwise, there is a non-crashed process p_j such that p_i executes the first step of a round $r + 1$ of ROC whereas p_j has not executed any steps of r. This implies that no process can finish phase 2 of an iteration without having all non-crashed processes setting *elected* to false.

This concludes the proof of the theorem.

□

Theorem 6.3 *Algorithm WLE satisfies LE-Liveness.*

Proof:

We have to show that for every execution $\phi = \langle F, I, S, T, \mathcal{T} \rangle$ of WLE and for every $\tau \in T$, there is some iteration after τ such that $|\{p_i \in \Pi : p_i.elected\}| > 1$.

Proof by contradiction. Suppose an execution $\phi = \langle F, I, S, T, \mathcal{T} \rangle$ of WLE and a time $\tau \in T$ such that $p_i.elected$ is false forever after τ for every p_i. By Validity and RO Uniformity, in every iteration ζ of ϕ, $\zeta \in \mathcal{I}$, there is a value $v \in vals_1$ such that v is the list with the least number of non-$\bot$ values, and for every process p_i that finishes phase 2 of iteration ζ, $v \in p_i.d$. Every process that finishes phase 2 of iteration ζ selects the same value i as the first index of v mapping to a value in v with a non-$\bot$ value. If process p_i evaluates the last "if" statement of phase 2, then it sets $p_i.elected$ to true. If p_i crashes, however, then it does not set $p_i.elected$ to true, and no process is elected at iteration ζ. By the assumption that all failures are benign, crashing (or stopping which is equivalent to crashing in our model) is the only possibility for having no process elected in an iteration ζ. By the assumption that t ($|\Pi|$ subtracted the size of the smallest survivor set) is the largest number of processes that can crash in ϕ, there can be at most t iterations after τ such that $|\{p_i \in \Pi : p_i.elected\}| = 0$.

□

Theorem 6.4 *Algorithm WLE satisfies FF-Stability.*

Proof:

Suppose ϕ is a failure-free execution and *zeta* is an iteration of ϕ. By agreement, every process decides upon the same value in both phases of iteration ζ. We then have that every process p_i uses the same value x to determine whether it sets $p_i.elected$ to true in ϕ. Moreover, we have that $x[i] = p_i$ for every i, by Validity. Assuming that $\Pi = \{p_1, p_2, \ldots, p_n\}$, we have by the algorithm that p_1 sets $p_1.\mathit{elected}$ to true at phase 2 of ζ.

□

7 Adding E-Stability

WLEallows for executions in which processes alternate forever as leaders. Such behavior, however, is not desirable. As leadership moves to one process to another, the responsibility of accomplishing the tasks of a leader also move. Recall that the original motivation is to embed such a leader election algorithm into a Primary-Backup protocol. This cause unnecessary overhead such as requests being forwarded to the correct Primary or even system instabilities if changes occur too frequently.

In fact, if such executions are allowed, then there is a simpler algorithm that trivially satisfies Safety, LE-Liveness and FF-Stability. This algorithm works by having every process broadcast the set of process it heard from in the previous round. If processes detect that some failure has occurred either by not hearing from some process or by learning that some process has not learned from another process, then processes are elected in a round-robin fashion from this round on. A process decides when to switch to round-robin mode depending on how it learns about failures. If a process learns from other process, then it starts immediately, i.e, in the current round. Otherwise, it waits until the following round. It is important to observe that failures must be detected by all non-crashed processes within two round. That is, every process that is alive at round $r + 2$ detects any failure that occurs by round r. If processes detect no failures, then some fixed process can be constantly elected.

An algorithm satisfying E-Stability guarantees that in every execution eventually there are no more leader changes. Note that with Stability only, failure-free executions are allowed to have multiple leaders elected over, and hence does not render FF-Stability unnecessary. For this discussion, we consider that the problem is still described by three properties: Safety, LE-Liveness, and E-Stability. We now show how to modify WLE (and ROC) to also satisfy this property. We call S-WLE the modified version of WLE, and S-ROC the modified version of ROC to distinguish between the original versions and the modified versions.

First, instead of initializing $p_i.s(0)$ to Π as in ROC $p_i.s(0)$ is initialize to a parameter $p_i.P$. We also roll forward the value of $p_i.s(t+1)$ in WLE instead of having $p_i.s(0)$ constant as in ROC. That is, in an iteration $\zeta > 0$, $p_i.s(0)$ in Phase 1 is $p_i.s(t+1)$ in Phase 2 of iteration $\zeta - 1$. If $\zeta = 0$, then $p_i.s(0)$ in Phase 1 is Π. For the initial value of $p_i.s(0)$ in Phase 2, we use $p_i.s(t+1)$ of Phase 1 of the same iteration. For clarity, we repeat the pseudcode for these algorithms with the respective modifications in Figures 4 and 5. Note that the main modifications in S-ROC are: 1) S-ROC has two parameters instead of one; 2) in round 1, p_i checks whether $p_i.s(1) \subseteq p_i.s(0)$. S-WLE is different from WLE by initializing $p_i.P$ to Π and by moving $p_i.s(t+1)$ forward.

It is straightforward to see that the proof of Section 5 is valid for S-ROC if the following constraint holds for every execution $\phi = \langle F, I, S, T, \mathcal{T} \rangle$ of S-ROC: if $p_c \in \mathit{Correct}(\phi)$ and $p_i \in \Pi$ is in a process in $p_c.s(1)$, then $p_j \in p_c.P$. That is, $p_c.P$ must contain all the processes that send messages to p_c at round zero if p_c is not faulty. Otherwise, p_c can falsely detect that it is faulty, and decide upon $\mathcal{N}$, violating Validity. If p_f is faulty, then there are no restrictions on the input $p_f.P$. Intuitively, a faulty process $p_f.P$ can receive any subset of processes sent to it. Consequently, it is not possible to impose a similar constraint as we did for correct processes. Differently from correct processes, if faulty process stops, it does not violate any of the RO Consensus properties.

According to the modifications described previously, $p_i.P$ is the set of processes from which p_i receives a message in the last round of the previous execution of S-ROC (Π if it is the execution of S-ROC in Phase 1 of iteration 0). By assumption and by the algorithm, once a process crashes (or stops) it never sends messages again in an execution of S-WLE. Thus, if p_c is a correct process, then $p_c.P$ must contain all the processes that p_c receives messages from in an execution of S-ROC, satisfying our constraint on $p_c.P$ for correct processes.

Because the proofs of Theorems 6.2 and 6.3 rely solely on the properties of RO Consensus, we also have that these proofs hold for S-WLE. It remains to show that S-

```
**Algorithm** S-ROC on input $p_i.a$, $p_i.P$
round 0:
    $p_i.s(0) \leftarrow p_i.P$; $p_i.sr(0) \leftarrow p_i.s(0)$
    $p_i.A\,[i] \leftarrow p_i.a$
    for all $p_k \in \Pi$, $p_k \neq p_i$ : $p_i.A\,[i] \leftarrow \bot$
    $p_i.A' \leftarrow p_i.A$
    send $p_i.A$ to all

round 1:
    $p_i.sr(1) \leftarrow p_i.s(1)$
    if $\vee p_i.s(1) \not\subseteq p_i.s(0)$
      $\vee \nexists S \in S_\Pi : S \subseteq p_i.s(1)$
    then decide $[\bot, \ldots, \bot]$
    else
        for each message $m_j \in M_i$, $p_k \in \Pi$:
            if $(p_i.A\,[k] = \bot)$ $p_i.A\,[k] \leftarrow m_j.A\,[k]$
    send $p_i.A$ to all

round $r$: $2 \leq r \leq t$:
    $p_i.sr(r) \leftarrow p_i.s(r) \setminus \{p_j : \exists m_j \in M_i :$
                                   $p_i.A' \not\subseteq m.A\}$
    if $\vee p_i.s(r) \not\subseteq p_i.s(r-1)$
      $\vee \nexists S \in S_\Pi : S \subseteq p_i.sr(r)$
    then decide $[\bot, ..., \bot]$
    else
        $p_i.A' \leftarrow p_i.A$
        for each message $m \in M_i$, $p_k \in \Pi$:
            if $(p_i.A\,[k] = \bot)$ $p_i.A\,[k] \leftarrow m.A\,[k]$
    send $p_i.A$ to all

round $t+1$:
    $p_i.sr(t+1) \leftarrow p_i.s(t+1) \setminus \{p_j : \exists m_j \in M_i :$
                                   $p_i.A \not\subseteq m.A\}$
    if $\vee$ $p_i.s(t+1) \not\subseteq p_i.s(t)$
      $\vee \nexists S \in S_\Pi : S \subseteq p_i.sr(t+1)$
    then decide $[\bot, ..., \bot]$
    else for each message $m \in M_i$, $p_k \in \Pi$:
      if $(p_i.A\,[k] = \bot)$ $p_i.A[k] \leftarrow m.A[k]$
    decide $p_i.A$
```

Figure 4: Algorithm run by process p_i.

```
**Algorithm** S-WLE
$P \leftarrow \Pi$
repeat {
$p_i$.elected $\leftarrow$ FALSE
Phase 1:
  Run ROC with
      $p_i.a \leftarrow i$; $p_i.P \leftarrow P$.
  $P \leftarrow p_i.s(t+1)$
  if $(p_i.d = [\bot, \ldots, \bot])$ then stop
Phase 2:
  Run ROC with
      $p_i.a \leftarrow p_i.d$ from Phase 1; $p_i.P \leftarrow P$.
  $P \leftarrow p_i.s(t+1)$
  if $(p_i.d = [\bot, \ldots, \bot])$ then stop
  let $x$ be a value of $p_i.d\,[1 \ldots n]$
      such that $p_i.d\,[x] \neq [\bot, \ldots, \bot]$
      and it has the least number of non-$\bot$ values
  if ($p_i$ is the first index of $x$ such that $x[i] \neq \bot$)
      then $p_i$.elected $\leftarrow$ TRUE
}
```

Figure 5: Algorithm run by process p_i.

WLE satisfies E-Stability. First, we present a few definitions to be used in the proof of E-Stability. By Theorem 6.2, in every iteration ζ of an execution ϕ of S-WLE it is the case that either one process p_i is such that $p_i.elected$ evaluates to true at the end of ζ or no process p_i is such that $p_i.elected$ evaluates to true at the end of ζ. We then use $Leader(\zeta, \phi)$ to denote the process p_i, $p_i.elected$ evaluates to true at the end of iteration ζ of ϕ, or $\bot$ if no such process exists. Finally, we need some terminology to refer to values that processes decide across iterations and in the two different phases of an iteration. We then use $\mathcal{D}^{\rho}_{\zeta}(i)$ to denote $p_i.d$ at the end of phase $\rho \in \{1, 2\}$ of iteration ζ.

Before we show our main result of this section, we state and prove a few lemmas.

Lemma 7.1 *Let ϕ be an execution of S-WLE. For every iteration ζ of S-WLE, if p_i finishes phase 1 of both ζ and $\zeta + 1$, then $\mathcal{D}^1_{\zeta+1}(i) \subseteq \mathcal{D}^1_{\zeta}(i)$.*

Proof:

Proof by contradiction. Suppose that there is an iteration ζ such that the assertion $\mathcal{D}^1_{\zeta+1}(i) \subseteq \mathcal{D}^1_{\zeta}(i)$ is false. This implies that there is some process p_ℓ, $\ell \neq i$, for which $\mathcal{D}^1_{\zeta+1}(i)[\ell] \neq\perp$ and $\mathcal{D}^1_{\zeta}(i)[\ell] =\perp$. By Lemma 5.10, in the execution of S-ROC in phase 1 of iteration $\zeta + 1$, there is a chain ω_ℓ such that $\omega_\ell[r] = i$, $r > 0$. By assumption, for every process p_j such that $\omega_\ell[\rho] = j, \rho \leq r$, $p_{j'}$ must be in $p_j.P$, where $\omega_\ell[\rho - 1] = j'$. Otherwise, by the algorithm, some process with identifier $\omega_\ell[\rho]$, $\rho \leq r$, stops before sending any messages in round ρ.

Now suppose that $\omega_\ell[1] = j$. By the algorithm, $p_j.sr(r)$ contains some survivor set S_j, for every round $r \in \{z \in \mathcal{R} : z \leq t + 1\}$ of the execution of S-ROC in iteration ζ. Equivalently, there is such a survivor set S_i for p_i. By assumption, there is some survivor set S_c such that $S_c \subseteq Correct(\phi)$. By (3,2)-Intersection, S_i either intersects S_j or intersects S_c. If S_i intersects S_j, then by Lemma 5.13, $\mathcal{D}^1_{\zeta}(i)[\ell] \neq\perp$, contradicting our initial assumption. If S_i intersects S_c, then by Lemma 5.14 $\mathcal{D}^2_{\zeta}(c) \subseteq \mathcal{D}^2_{\zeta}(i)$, for some non-faulty $p_c \in Correct(\phi)$. By Agreement, $\mathcal{D}^2_{\zeta}(c)[\ell] \neq\perp$, and consequently $\mathcal{D}^1_{\zeta}(i)[\ell] \neq\perp$, again contradicting our initial assumption.

This completes the proof of the lemma.

□

Lemma 7.2 *Let ϕ be an execution of* S-WLE *and ζ be an iteration of* S-WLE*. No process is elected in ζ unless some process crashes or stops in ζ.*

Proof:

By RO Uniformity, we have that $1 \leq vals_i \leq 2$, $i \in \{1, 2\}$. By Validity and RO Uniformity, there is some value d in $vals_1$ such that $d \in p_i.d$ for every process p_i that finishes phase 2 of iteration ζ, and d contains the least number of non-$\perp$ values. Because $d \neq \mathcal{N}$, there must be a process p_e such that e is the smallest index of d such that $d[e] \neq\perp$. We then have that p_e sets $p_e.elected$ to true unless p_e crashes. Thus, if $Leader(\zeta, \phi) =\perp$, then p_e must crash or stop in ζ.

□

Lemma 7.3 *Let ϕ be an execution of* S-WLE*, ζ and ζ' be iterations of ϕ, $\zeta + 1 < \zeta'$, such that* $\text{Leader}(\zeta, \phi) = \text{Leader}(\zeta', \phi) = p_e$*. If* $\text{Leader}(\zeta + 1, \phi) = p_{e'}$*, $e \neq e'$, then there is a process p_i that crashes or stops in some iteration ζ'', $\zeta \leq \zeta'' \leq \zeta'$.*

Proof:

If p_e is elected in iteration ζ of ϕ, then there is a value d in $vals_1$ of ζ such that e is the smallest index of d with a non-$\perp$ value, and $d \in \mathcal{D}^2_{\zeta}(e)$. Now if $p_{e'} j$ is elected in iteration $\zeta + 1$, then there is a value d' in $vals_1$ of $\zeta + 1$ such that e' is the smallest index of d' with a non-$\perp$ value, and $d' \in \mathcal{D}^2_{\zeta}(e')$. By assumption, p_e is elected again in iteration ζ'. As before, there must be a value d'' in $vals_1$ of ζ' such that e is the smallest index of d'' with a non-$\perp$ value.

There are two possibilities regarding the identifiers e and e': 1) $e' < e$; 2) $e < e'$. If $e' < e$, then there must be a second value d_c in $vals_1$ of ζ such that $d \subseteq d_c$ (by assumption, p_e has not crashed by iteration $\zeta' > \zeta + 1$; by validity, every non-faulty process p_c is such that $p_e.a \in p_c.d$). Let p_i be a process such that $\mathcal{D}^1_{\zeta}(i) = d$. Suppose by way of contradiction that p_i finishes phase 2 of $\zeta + 1$. By Lemma 7.1, $\mathcal{D}^1_{\zeta+1}(i) \subseteq \mathcal{D}^1_{\zeta}(i)$, and $\mathcal{D}^1_{\zeta+1}(i)[e'] =\perp$. By Validity, there is some $p_c \in Correct(\phi)$ such that $\mathcal{D}^1_{\zeta+1}(c)[e'] \neq\perp$. Note that $p_{e'}$ does not crash or stop in iteration $\zeta + 1$ or in a previous iteration. By RO Uniformity, $\mathcal{D}^1_{\zeta+1}(i) \subseteq \mathcal{D}^1_{\zeta+1}(c) = d'$. By RO Uniformity and Validity, $\mathcal{D}^1_{\zeta+1}(i)$ must be the value used by every process that completes phase 2 of iteration $\zeta + 1$ to determine whether it elects itself or not. Since e' is not the smallest index of $\mathcal{D}^1_{\zeta+1}(i)$ that evaluates to a non-$\perp$ value, $p_{e'}$ is not elected in $\zeta + 1$. This contradicts our initial assumption. We hence have that p_i must crash or stop by iteration $\zeta + 1$.

Now if $e < e'$, then $d\prime[e] =\perp$ by assumption (e is the smallest index in d' with a non-$\perp$ value). We use a similar argument as in the first case. Suppose by way of contradiction that $p_{e'}$ finishes phase 2 of ζ'. By Lemma 7.1, $\mathcal{D}^1_{\zeta'}(e') \subseteq \mathcal{D}^1_{\zeta+1}(e')$, and $\mathcal{D}^1_{\zeta'}(e')[e] =\perp$. By Validity, there is some $p_c \in Correct(\phi)$ such that $\mathcal{D}^1_{\zeta'}(c)[e] \neq\perp$. Note that p_e does not crash or stop in iteration ζ' or in a previous iteration. By RO Uniformity, $\mathcal{D}^1_{\zeta'}(i) \subseteq \mathcal{D}^1_{\zeta'}(c) = d''$. By RO Uniformity and Validity, $\mathcal{D}^1_{\zeta'}(i)$ must be the value used by every process that completes phase 2 of iteration ζ' to determine whether it elects itself or not. Since e is not the smallest index of $\mathcal{D}^1_{\zeta'}(i)$ that evaluates to a non-$\perp$ value, p_e is not elected in ζ'. This contradicts our initial assumption. We conclude that p_e is not elected in ζ' unless $p_{e'}$ crashes or stops by iteration ζ'.

Finally, we have that either $p_{e'}$ crashes or stops by iteration ζ' or some faulty process p_i crashes or stops by iteration $\zeta + 1$. This concludes the proof of the lemma.

□

Theorem 7.4 S-WLE *satisfies E-Stability.*

Proof:

Let ϕ be an execution of S-WLE. By LE-Liveness, infinitely often some process is elected in ϕ. By Lemma 7.2, an iteration ζ has no leader elected only if some some process crashes in ζ, and by assumption there is a finite number of processes that crash or stop. Thus, there is a bounded number of iterations that have no leader elected.

Let t be a time such that every iteration that starts after t has a leader elected. Such a t exists by the previous argument. We then have that every iteration that starts after t has a leader elected, and it remains to show that there is some $t' \geq t$ and some process p_e such that for every iteration ζ that starts after t', p_e is elected in both ζ and $\zeta + 1$. Suppose by way of contradiction that there is no such t' in ϕ. Let p_e be a process that is elected infinitely often after. Such a process must exist because the set of processes is finite. By assumption, there is an infinite sequence of non-consecutive iterations $\zeta_1 < \zeta_2 < \zeta_3 \ldots$ such that p_e is elected in ζ_i but not in $\zeta_i + 1$. By Lemma 7.3, for every $i \in \mathbb{Z}$, there is an iteration ζ, $\zeta_i \leq \zeta \leq \zeta_{i+1}$, such that some process crashes or stops in ζ. By assumption, the number of processes crashing or stopping is bounded. Consequently, there cannot be such an infinite sequence. We conclude that there must be some $t' \geq t$ and some process p_e such that for every iteration ζ that starts after t', p_e is elected in both ζ and $\zeta + 1$.

□

8 A discussion on the Primary-Backup approach

Developing a Primary-Backup protocol that uses WLE is future work. We can, however, make a few observations regarding the use of an algorithm as WLE (or S-WLE) for a Primary-Backup protocol. As mentioned previously, WLE enables faulty processes to be elected. In a Primary-Backup system, this feature impacts on liveness, although not on correctness. Often, there is a time bound in the replies to client requests, and it is impossible to meet such bounds if the primary can be faulty. A immediate consequence of electing faulty processes is that service time is not bounded during the period of time a faulty process remains as the primary. As discussed before, processes that commit failures (but do not stop or crash) are detected. In practice, we rely on an off-line mechanism to detect these anomalies and take the appropriate measures that can be for example, to remove faulty processes from the system.

It is possible, however, that faulty processes go through an iteration of WLE undetected as such, and fail to reply to client requests due to receive-omission failures . To solve this problem, we can require clients to broadcast requests to all the replicas and the primary to broadcast replies to all the backup replicas as well. Correct processes are also capable of detecting failures in such cases, although they may not be able to ”warn” the faulty primary that it is actually faulty. Recall that failure detection for omission failures requires twofold replication.

Finally, the iterations of the repeat loop of WLE are consecutive without any delay in between for expositional purposes. In practice, iterations should be delayed until failures are detected, they are manually triggered, or if none of these are desirable or possible, some time threshold is reached.

9 Conclusions

We described in this paper a weaker version of the leader election problem and an algorithm that solves this problem. This version of the problem, unlike the traditional definition of leader election, enables faulty processes to be elected. The main advantage of enabling it is requiring a lower degree of replication.

There are other interesting features of the WLE algorithm. First, it is uses cores and survivor sets instead of a threshold. This enables more flexible characterizations of systems with an heterogeneous set of processes. Second, it uses an unusual type of Intersection property, *i.e.*, (3,2)-Intersection. This property generalizes a degree of replication of the form $n > (3t/2)$, where t is the threshold on the number of failures in any execution. Finally, correct processes are able to detect faulty processes. By Lemma 5.16, non-crashed faulty processes decide upon lists with fewer values, and one can build an alarm system by collecting decision values by the end of every iteration.

Although we have not thoroughly investigated using WLE to build Primary-Backup systems, we believe our algorithm provide practical benefits compared to previous solutions.

References

[1] L. Lamport, "The Part-Time Parliament," *ACM Transactions on Computer Systems*, vol. 16, pp. 133–169, May 1998.

[2] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg, "Optimal primary-backup protocols," in *6th International Workshop on Distributed Algorithms (WDAG)*, pp. 362–378, Nov 1992.

[3] F. Junqueira and K. Marzullo, "Lower Bound on the Number of Rounds for Synchronous Consensus with Dependent Process Failures," Tech. Rep. CS2003-0734, UCSD, 2001.