$$\lor \exists\, i, j \in Server : RequestVote(i, j)$$
$$\lor \exists\, i \in Server : BecomeLeader(i)$$
$$\lor \exists\, i \in Server, v \in Value : ClientRequest(i, v)$$
$$\lor \exists\, i \in Server : AdvanceCommitIndex(i)$$
$$\lor \exists\, i, j \in Server : AppendEntries(i, j)$$
$$\lor \exists\, m \in \text{DOMAIN } messages : Receive(m)$$
$$\lor \exists\, m \in \text{DOMAIN } messages : DuplicateMessage(m)$$
$$\lor \exists\, m \in \text{DOMAIN } messages : DropMessage(m)$$

History variable that tracks every *log* ever:

$$\land\, allLogs' = allLogs \cup \{log[i] : i \in Server\}$$

The specification must start with the initial state and transition according to *Next*.

$$Spec \triangleq Init \land \Box[Next]_{vars}$$

B.3 Proof

Lemma 1. Each server's $currentTerm$ monotonically increases:

$$\forall\, i \in Server :$$
$$currentTerm[i] \leq currentTerm'[i]$$

Proof. This follows immediately from the specification. □

Lemma 2. There is at most one leader per term:

$$\forall\, e, f \in elections :$$
$$e.eterm = f.eterm \Rightarrow e.eleader = f.eleader$$

This is the Election Safety property of Figure 3.2.

Sketch. It takes votes from a quorum to become leader, voters may only vote once per term, and any two quorums overlap.

Proof.

1. Consider two elections, e and f, both members of $elections$, where $e.eterm = f.eterm$.

2. $e.evotes \in Quroum$ and $f.evotes \in Quorum$, since this is a necessary condition for members of $elections$.

3. Let $voter$ be an arbitrary member of $e.evotes \cap f.evotes$. Such a member must exist since any two quorums overlap.

4. Once $voter$ casts a vote for $e.eleader$ in $e.eterm$, it can not cast a vote for a different server in $e.eterm$ (the specification ensures this: once it increments its $currentTerm$, it can never vote again for the same server (Lemma 1); and until then, it safely retains its vote information).

5. $e.eleader = f.eleader$, since $voter$ voted for $e.eleader$ and $voter$ voted for $f.eleader$ in $e.eterm = f.eterm$.

□

Lemma 3. A leader's log monotonically grows during its term:

$$\begin{aligned}
&\forall\ e \in elections : \\
&\quad currentTerm[e.leader] = e.term \Rightarrow \\
&\quad\quad \forall\ index \in 1..Len(log[e.leader]) : \\
&\quad\quad\quad log'[e.leader][index] = log[e.leader][index]
\end{aligned}$$

This is the Leader Append-Only property of Figure 3.2.

Sketch. As a leader, server i only appends to its log; i won't ever get an AppendEntries request from some other server for the same term, since there is at most one leader per term; and i rejects AppendEntries requests for other terms until increasing its own term.

Proof.

1. Three variables are involved in the goal: $elections$, $currentTerm$, and log. We consider the transitions that change each of these variables in turn; otherwise, the invariant trivially holds by the inductive hypothesis.

2. When a new election is added to $elections$ (a history variable which maintains information about all successful elections), the log of the leader is not changed in the same step ($log'[e.leader] = log[e.leader]$), so the invariant is maintained.

3. $currentTerm[e.leader]$ monotonically increases by Lemma 1, so once $e.leader$ moves to a new term, it will trivially satisfy the invariant forever after.

4. *log* changes either from client requests or AppendEntries requests:

   (a) Case: client request:

      i. By the specification, the leader only appends an entry to its log, which maintains the invariant.

   (b) Case: AppendEntries request:

      i. Only servers with $state[i] = Leader$ can send AppendEntries requests for their $currentTerm$.

      ii. By Lemma 2, $e.leader$ is the only server which can ever be leader for $e.term$.

      iii. Servers don't send themselves AppendEntries requests (see specification).

      iv. $e.leader$ will process no AppendEntries requests while its term is $e.term$.

□

Lemma 4. An $\langle index, term \rangle$ pair identifies a log prefix:

$$
\begin{array}{l}
\forall\, l, m \in allLogs : \\
\quad \forall\, \langle index, term \rangle \in l : \\
\qquad \langle index, term \rangle \in m \Rightarrow \\
\qquad\quad \forall\, pindex \in 1..index : \\
\qquad\qquad l[pindex] = m[pindex]
\end{array}
$$

This is the Log Matching property of Figure 3.2.

Sketch. Only leaders create entries, and they assign the new entries term numbers that will never be assigned again by other leaders (there's at most one leader per term). Moreover, the consistency check in AppendEntries guarantees that when followers accept new entries, they do so in a way that's consistent with the leader's log at the time it sent the entries.

Assertion. If p is a prefix of some log $l \in allLogs$, then $allLogs' = allLogs \cup \{p\}$ maintains the invariant (the statement in the lemma).

1. This follows immediately from the invariant, since p's entries match l's entries, and p contributes no additional entries.

Proof by induction on an execution.

1. Initial state: all of the servers' logs are empty, so $allLogs = \langle\rangle$, and the invariant trivially holds.

2. Inductive step: logs change in one of the following ways:

   (a) Case: a leader adds one entry (client request)

      i. By the inductive hypothesis, $log[leader] \in allLogs$.

      ii. The $\langle index, term\rangle$ of the new entry cannot exist in any other entry in any log in $allLogs$, since there's only one leader per term (Lemma 2) and leaders only append to their logs (Lemma 3).

      iii. Then $allLogs' = allLogs \cup \{log[leader] \parallel \langle index, term\rangle\}$ maintains the invariant.

   (b) Case: a follower removes one entry (AppendEntries request m)

      i. The invariant still holds, since $log'[follower]$ is a prefix of $log[follower]$ (by the Assertion above).

   (c) Case: a follower adds one entry (AppendEntries request m)

      i. $m.mlog$ is a copy of the leader's log at the time the leader created the AppendEntries request.

      ii. $m.mlog \in allLogs$ by definition of $allLogs$.

      iii. In the two cases below, we show that $log'[follower]$ is a prefix of $m.mlog$.

      iv. Case: $m.mprevLogIndex = 0$

         A. $m.mentries$ is a prefix of $m.mlog$.

         B. $log[follower]$ is empty, as a necessary condition for accepting the request (the specification separates transitions for removing a conflicting entry, replying when there is no longer any change to make, and appending an entry).

         C. $log'[follower] = m.mentries$ upon accepting the request, which is a prefix of $m.mlog$.

      v. Case: $m.mprevLogIndex > 0$

         A. $start \parallel \langle m.mprevLogIndex, m.mprevLogTerm\rangle \parallel m.mentries$ is a prefix of $m.mlog$, where $start$ is some (possibly empty) log prefix.

         B. The follower accepts the request by assumption, so its log contains the entry $\langle m.mprevLogIndex, m.mprevLogTerm\rangle$.

C. By the inductive hypothesis, $log[follower]$ contains the prefix $start \parallel \langle m.mprevLogIndex, m.mprevLogTerm \rangle$.

D. $log'[follower] = start \parallel \langle m.mprevLogIndex, m.mprevLogTerm \rangle \parallel m.mentries$ upon accepting the request, which is a prefix of $m.mlog$.

vi. Because $log'[follower]$ is a prefix of $m.mlog$, the invariant is maintained (by the Assertion above).

□

Lemma 5. When a follower appends an entry to its log, its log after the append is a prefix of the leader's log at the time the leader sent the AppendEntries request:

$$
\begin{array}{l}
\forall\, i \in Server : \\
\quad state[i] \neq Leader \wedge Len(log'[i]) > Len(log[i]) \Rightarrow \\
\quad\quad \exists\, m \in \text{DOMAIN } messages : \\
\quad\quad\quad \wedge\, m.mtype = AppendEntriesRequest \\
\quad\quad\quad \wedge\, \forall\, index \in 1..Len(log'[i]) : \\
\quad\quad\quad\quad log'[i][index] = m.mlog[index]
\end{array}
$$

This restates an argument from the proof of Lemma 4 that is useful in the proofs of other lemmas. (The argument is difficult to make before Lemma 4, since that lemma's inductive hypothesis is key; however, the proof for this lemma follows easily from Lemma 4.)

Sketch. The new entry that the follower appends to its log was also present in the leader's log. Thus, by Lemma 4, the follower's new log is a prefix of what was the leader's log.

Proof. Logs change in one of the following ways:

1. Case: a leader adds one entry (client request). This invariant only applies to non-leaders.
2. Case: a follower removes one entry (AppendEntries request). This invariant only affects logs that grow in length.
3. Case: a follower adds one entry (AppendEntries request m):
   (a) $m.mlog$ is a copy of the leader's log at the time the leader created the AppendEntries request.
   (b) Thus, $m.mlog \in allLogs$.

(c) $log'[i] \in allLogs$ by definition of $allLogs$.

(d) $m.mentries$, the entry being added, is the last entry in $log'[i]$. (This extends to multiple entries for implementations that batch entries together.)

(e) $m.mentries \in m.mlog$

(f) By Lemma 4, the index and term of $m.mentries$ uniquely identifies a prefix of $m.mlog$ equal to $log'[i]$.

□

Lemma 6. A server's current term is always at least as large as the terms in its log:

$$\forall\, i \in Server :$$
$$\quad \forall\, \langle index, term \rangle \in log[i] :$$
$$\quad\quad term \leq currentTerm[i]$$

Sketch. Servers' current terms monotonically increase. When leaders create new entries, they assign them their current term. And when followers accept new entries from a leader, their current term agrees with the leader's term at the time it sent the entries.

Proof by induction on an execution.

1. Initial state: all logs are empty, so the invariant trivially holds.
2. Inductive step: $currentTerm[i]$ changes:
   (a) By Lemma 1, $currentTerm'[i] \geq currentTerm[i]$, so the invariant is maintained.
3. Inductive step: logs change in one of the following ways:
   (a) Case: a leader adds one entry (client request):
      i. By the inductive hypothesis, all entries in $log[i]$ have $term \leq currentTerm[i]$.
      ii. The new entry's term is $currentTerm[i]$.
      iii. Thus, all entries in $log'[i]$ satisfy the invariant.
   (b) Case: a follower removes one entry (AppendEntries request)
      i. The invariant still holds, since only the length of the log decreased.
   (c) Case: a follower adds one entry (AppendEntries request m):

i. By the inductive hypothesis, when the leader created the request, its current term was at least as large as the term of every entry in its log:
$\forall \langle index, term \rangle \in m.mlog : term \leq m.mterm$

ii. $log'[i]$ is a prefix of $m.mlog$ by Lemma 5.

iii. As a necessary condition for accepting the request, $currentTerm[i] = m.mterm$.

iv. Then $currentTerm[i]$ is at least as large as the term in every entry in $log'[i]$, and the invariant is maintained.

□

Lemma 7. The terms of entries grow monotonically in each log:

$$\forall\, l \in allLogs :$$
$$\quad \forall\, index \in 1..(Len(l) - 1) :$$
$$\quad\quad l[index].term \leq l[index + 1].term$$

Sketch. A leader maintains this by assigning new entries its current term, which is always at least as large as the terms in its log. When followers accept new entries, they are consistent with the leader's log at the time it sent the entries.

Proof by induction on an execution.

1. Initial state: all logs are empty, so the invariant holds.

2. Inductive step: logs change in one of the following ways:

   (a) Case: a leader adds one entry (client request)

      i. The new entry's term is $currentTerm[leader]$

      ii. $currentTerm[leader]$ is at least as large as the term of any entry in $log[leader]$, by Lemma 6.

   (b) Case: a follower removes one entry (AppendEntries request)

      i. The invariant still holds, since only the length of the log decreased.

   (c) Case: a follower adds one entry (AppendEntries request m)

      i. $log'[follower]$ is a prefix of $m.mlog$ (by Lemma 5).

      ii. $m.mlog \in allLogs$

iii. By the inductive hypothesis, the terms in $m.mlog$ monotonically grow, so the terms in $log'[follower]$ monotonically grow.

□

Definition 1. An entry $\langle index, term \rangle$ is **committed at term** t if it is present in every leader's log following t:

$$
\begin{aligned}
&committed(t) \triangleq \{\langle index, term \rangle : \\
&\qquad \forall\, election \in elections : \\
&\qquad\quad election.eterm > t \Rightarrow \\
&\qquad\qquad \langle index, term \rangle \in election.elog\}
\end{aligned}
$$

Definition 2. An entry $\langle index, term \rangle$ is **immediately committed** if it is acknowledged by a quorum (including the leader) during $term$. Lemma 8 shows that these entries are committed at $term$.

$$
\begin{aligned}
&immediatelyCommitted \triangleq \{\langle index, term \rangle \in anyLog : \\
&\qquad \land\ anyLog \in allLogs \\
&\qquad \land\ \exists\, leader \in Server,\ subquorum \in \text{SUBSET } Server : \\
&\qquad\quad \land\ subquorum \cup \{leader\} \in Quorum \\
&\qquad\quad \land\ \forall\, i \in subquorum : \\
&\qquad\qquad \exists\, m \in messages : \\
&\qquad\qquad\quad \land\ m.mtype = AppendEntriesResponse \\
&\qquad\qquad\quad \land\ m.msource = i \\
&\qquad\qquad\quad \land\ m.mdest = leader \\
&\qquad\qquad\quad \land\ m.mterm = term \\
&\qquad\qquad\quad \land\ m.mmatchIndex \geq index\}
\end{aligned}
$$

Lemma 8. *Immediately committed* entries are *committed*:

$$
\begin{aligned}
&\forall\, \langle index, term \rangle \in immediatelyCommitted : \\
&\qquad \langle index, term \rangle \in committed(term)
\end{aligned}
$$

Along with Lemma 9, this is the Leader Completeness property of Figure 3.2.

Sketch. See Section 3.6.3.

Proof.

1. Consider an entry $\langle index, term \rangle$ that is *immediately committed*.

2. Define

$$\begin{aligned} Contradicting \triangleq \{ & election \in elections : \\ & \wedge\ election.eterm > term \\ & \wedge\ \langle index, term \rangle \notin election.elog \} \end{aligned}$$

3. Let $election$ be an element in $Contradicting$ with a minimal $term$ field. That is, $\forall\ e \in Contradicting : election.eterm \leq e.eterm$.
   If more than one election has the same term, choose the earliest one. (The specification does not allow this to happen, but it is safe for a leader to step down and become leader again in the same term.)

4. It suffices to show a contradiction, which implies $Contradicting = \phi$.

5. Let $voter$ be any server that both votes in $election$ and contains $\langle index, term \rangle$ in its log during $term$ (either it acknowledges the entry as a follower or it was leader). Such a server must exist since:

   (a) A quorum of servers voted in $election$ for it to succeed.

   (b) A quorum contains $\langle index, term \rangle$ in its log during $term$, since $\langle index, term \rangle$ is immediately committed.

   (c) Any two quorums overlap.

6. Let $voterLog \triangleq election.evoterLog[voter]$, the voter's log at the time it cast its vote.

7. The voter contains the entry when it cast its vote during $election.eterm$. That is, $\langle index, term \rangle \in voterLog$:

   (a) $\langle index, term \rangle$ was in the voter's log during $term$.

   (b) The voter must have stored the entry in $term$ before voting in $election.eterm$, since:

      i. $election.eterm > term$.

      ii. The voter rejects requests with terms smaller than its current term, and its current term monotonically increases (Lemma 1).

   (c) The voter couldn't have removed the entry before casting its vote:

i. Case: No $AppendEntriesRequest$ with $mterm < term$ removes the entry from the voter's log, since $currentTerm[voter] \geq term$ upon storing the entry (by Lemma 6), and the voter rejects requests with terms smaller than $currentTerm[voter]$.

ii. Case: No $AppendEntriesRequest$ with $mterm = term$ removes the entry from the voter's log, since:

A. There is only one leader of $term$.

B. The leader of $term$ created and therefore contains the entry (Lemma 3).

C. The leader would not send any conflicting requests to $voter$ during $term$.

iii. Case: No $AppendEntriesRequest$ with $mterm > term$ removes the entry from the voter's log, since:

A. Case: $mterm > election.eterm$:

This can't happen, since $currentTerm[voter] > election.eterm$ would have prevented the voter from voting in $term$.

B. Case: $mterm = election.eterm$:

Since there is at most one leader per term (Lemma 2), this request would have to come from $election.eleader$ as a result of an earlier election in the same term ($election.eterm$).

Because a leader's log grows monotonically during its term (by Lemma 3), the leader could not have had $\langle index, term \rangle$ in its log at the start of its term.

Then there exists an earlier election with the same term in $Contradicting$; this is a contradiction.

C. Case $mterm < election.eterm$:

The leader of $mterm$ must have contained the entry (otherwise its election would also be $Contradicting$ but have a smaller term than $election$, which is a contradiction). Thus, the leader of $mterm$ could not send any conflicting entries to the voter for this index, nor could it send any conflicting entries for prior indexes: that it has this entry implies that it has the entire prefix before it (Lemma 4).

8. The log comparison during elections states the following, since $voter$ granted its vote during $election$:

$$\vee\ LastTerm(election.elog) > LastTerm(voterLog)$$

$$\lor \ \land \mathit{LastTerm}(\mathit{election}.\mathit{elog}) = \mathit{LastTerm}(\mathit{voterLog})$$
$$\land \mathit{Len}(\mathit{election}.\mathit{elog}) \geq \mathit{Len}(\mathit{voterLog})$$

In the following two steps, we take each of these cases in turn and show a contradiction.

9. Case: $\mathit{LastTerm}(\mathit{election}.\mathit{elog}) = \mathit{LastTerm}(\mathit{voterLog})$ and $\mathit{Len}(\mathit{election}.\mathit{elog}) \geq \mathit{Len}(\mathit{voterLog})$

   (a) The leader of $\mathit{LastTerm}(\mathit{voterLog})$ monotonically grew its log during its term (by Lemma 3).

   (b) The same leader must have had $\mathit{election}.\mathit{elog}$ as its log at some point, since it created the last entry.

   (c) Thus, $\mathit{voterLog}$ is a prefix of $\mathit{election}.\mathit{elog}$.

   (d) Then $\langle \mathit{index}, \mathit{term} \rangle \in \mathit{election}.\mathit{elog}$, since $\langle \mathit{index}, \mathit{term} \rangle \in \mathit{voterLog}$.

   (e) But $\mathit{election} \in \mathit{Contradicting}$ implies that $\langle \mathit{index}, \mathit{term} \rangle \notin \mathit{election}.\mathit{elog}$.

10. Case: $\mathit{LastTerm}(\mathit{election}.\mathit{elog}) > \mathit{LastTerm}(\mathit{voterLog})$

    (a) $\mathit{LastTerm}(\mathit{voterLog}) \geq \mathit{term}$, since $\langle \mathit{index}, \mathit{term} \rangle \in \mathit{voterLog}$ and terms in logs grow monotonically (Lemma 7).

    (b) $\mathit{election}.\mathit{eterm} > \mathit{LastTerm}(\mathit{election}.\mathit{elog})$ since servers increment their $\mathit{currentTerm}$ when starting an election, and Lemma 6 states that a server's $\mathit{currentTerm}$ is at least as large as the terms in its log.

    (c) Let prior be the election in $\mathit{elections}$ with $\mathit{prior}.\mathit{eterm} = \mathit{LastTerm}(\mathit{election}.\mathit{elog})$. Such an election must exist since $\mathit{LastTerm}(\mathit{election}.\mathit{elog}) > 0$ and a server must win an election before creating an entry.

    (d) By transitivity, we now have the following inequalities:

$$\begin{aligned} &\mathit{term} \leq \\ &\quad \mathit{LastTerm}(\mathit{voterLog}) < \\ &\quad\quad \mathit{LastTerm}(\mathit{election}.\mathit{elog}) = \mathit{prior}.\mathit{eterm} < \\ &\quad\quad\quad \mathit{election}.\mathit{eterm} \end{aligned}$$

    (e) $\langle \mathit{index}, \mathit{term} \rangle \in \mathit{prior}.\mathit{elog}$, since $\mathit{prior} \notin \mathit{Contradicting}$ ($\mathit{election}$ was assumed to have the lowest term of any election in $\mathit{Contradicting}$, and $\mathit{prior}.\mathit{eterm} < \mathit{election}.\mathit{eterm}$).

(f) $prior.elog$ is a prefix of $election.elog$ since:

i. $prior.eleader$ creates entries with $prior.eterm$ by appending them to its log, which monotonically grows during $prior.eterm$ from $prior.elog$.

ii. Thus, any entry with term $prior.eterm$ must follow $prior.elog$ in all logs (by Lemma 4).

iii. $LastTerm(election.elog) = prior.eterm$

(g) $\langle index, term \rangle \in election.elog$

(h) This is a contradiction, since $election.elog$ was assumed to not contain the committed entry ($election \in Contradicting$).

□

Definition 3. An entry $\langle index, term \rangle$ is **prefix committed at term** t if there is another entry that is *committed at term* t following it in some log. Lemma 9 shows that these entries are *committed at term* t.

$$\begin{aligned} prefixCommitted(t) \triangleq \{ & \langle index, term \rangle \in anyLog : \\ & \wedge\ anyLog \in allLogs \\ & \wedge\ \exists\ \langle rindex, rterm \rangle \in anyLog : \\ & \quad \wedge\ index < rindex \\ & \quad \wedge\ \langle rindex, rterm \rangle \in committed(t) \} \end{aligned}$$

Lemma 9. *Prefix committed* entries are *committed* in the same term:

$$\forall\ t : prefixCommitted(t) \subseteq committed(t)$$

Along with Lemma 8, this is the Leader Completeness property of Figure 3.2.

Sketch. If an entry is committed, it identifies a prefix of a log in which every entry is committed, since those entries will also be present in every future leader's log.

Proof.

1. Consider an arbitrary entry $\langle index, term \rangle \in prefixCommitted(t)$.

2. There exists an entry $\langle rindex, rterm \rangle \in committed(t)$ following $\langle index, term \rangle$ in some log, by definition of $prefixCommitted(t)$.

3. $\langle rindex, rterm \rangle$ uniquely identifies the log prefix containing $\langle index, term \rangle$ (Lemma 4).

4. Every leader following t contains $\langle index, term \rangle$, since every leader following t contains $\langle rindex, rterm \rangle$.

5. $\langle index, term \rangle \in committed(t)$ by definition of $committed(t)$.

□

Theorem 1. Servers only apply entries that are *committed* in their current term:

$$
\begin{aligned}
&\forall\, i \in Server : \\
&\quad \land\ commitIndex[i] \leq Len(log[i]) \\
&\quad \land\ \forall\, \langle index, term \rangle \in log[i] : \\
&\qquad\qquad index \leq commitIndex[i] \Rightarrow \\
&\qquad\qquad\quad \langle index, term \rangle \in committed(currentTerm[i])
\end{aligned}
$$

This is equivalent to the State Machine Safety property of Figure 3.2. (The bound on the commit index is needed to strengthen the inductive hypothesis.)

Sketch. A leader only advances its $commitIndex$ to cover entries that are immediately committed or prefix committed. Followers update their $commitIndex$ from the leader's only when they have a prefix of the leader's log.

Proof by induction on an execution.

1. Initial state: trivially holds for empty logs (and $commitIndex[i]$ is initialized to 0).

2. Inductive step: the set of entries committed at $currentTerm[i]$ changes:

   (a) Once an entry is committed at $currentTerm[i]$, all leaders of subsequent terms will have the entry (by the definition of $committed$).

   (b) Thus, the set of committed entries at $currentTerm[i]$ monotonically grows.

3. Inductive step: $commitIndex[i]$ changes:

   (a) When $commitIndex[i]$ decreases (if implementations allow this to happen), the inductive hypothesis suffices to show the invariant holds.

   (b) When $commitIndex[i]$ increases, it covers entries present in i's log that are committed:

i. Case: follower completes accepting AppendEntries request m:

A. Upon processing the request, the follower's log is a prefix of a prior version of the leader's log, $m.mlog$ (by Lemma 5).

B. Every entry up through $commitIndex'[i]$ in $m.mlog$ is committed by the inductive hypothesis (they were marked committed in the leader's log when it sent the request).

ii. Case: leader i processes AppendEntries response:

A. If the leader sets a new $commitIndex$, the conditions in the specification ensure that $commitIndex'[i] \in immediatelyCommitted$.

B. Every entry in the leader's log with index up to $commitIndex'[i]$ is prefix committed at $currentTerm[i]$.

4. Inductive step: $currentTerm[i]$ changes:

(a) By Lemma 1, $currentTerm'[i] > currentTerm[i]$.

(b) $committed(currentTerm[i]) \subseteq committed(currentTerm'[i])$ by the definition of $committed$.

(c) Thus, the inductive hypothesis suffices to show the invariant holds.

5. Inductive step: logs change in one of the following ways:

(a) Case: a leader adds one entry (client request):

i. Newly created entries are not marked committed, so the invariant holds.

(b) Case: a follower removes one entry (AppendEntries request m):

i. Case: the removed entry was not marked committed on the follower:
The inductive hypothesis suffices to show the invariant holds.

ii. Case: the removed entry was marked committed on the follower:

A. $m.mterm = currentTerm[i]$, since the follower accepted the request.

B. The removed entry is not in $m.mlog$, since it conflicts with the request.

C. The removed entry is not present in $m.msource$'s log at the start of its term (by Lemma 3).

D. The election for $m.mterm$ did not contain the removed entry.

E. The removed entry is not committed at $currentTerm[i]$.

F. This contradicts the inductive hypothesis; this case cannot occur.

(c) Case: a follower adds one entry (AppendEntries request m):

i. Case: the new entry is not marked committed on the follower:
The inductive hypothesis suffices to show the invariant holds.

ii. Case: the new entry is marked committed on the follower:
$commitIndex[i]$ must increase (which was already handled above).

□