MODULE *Balloting*

Specification of *SCP*'s balloting protocol following the IETF draft at:

https://datatracker.ietf.org/doc/html/draft-mazieres-dinrg-scp-05#section-3.5

This specification abstracts over some aspects of the protocol (*e.g.* increasing the ballot counter), but it does explicitely represent balloting messages. There are also some differences compared to the IETF draft, due to I suspect are omissions in the IETF draft.

Currently this specification covers only the PREPARE and COMMIT phases.

EXTENDS *DomainModel*

$$Phase \triangleq \{\text{“PREPARE”}, \text{“COMMIT”}, \text{“EXTERNALIZE”}\}$$

$$\begin{aligned}
&SCPPrepare \triangleq [\\
&\quad type : \{\text{“PREPARE”}\}\\
&, \quad ballot : Ballot\\
&, \quad prepared \ : BallotOrNull\\
&, \quad aCounter : BallotNumber\\
&, \quad hCounter : BallotNumber\\
&, \quad cCounter : BallotNumber]
\end{aligned}$$

$$\begin{aligned}
&SCPCommit \triangleq [\\
&\quad type : \{\text{“COMMIT”}\}\\
&, \quad ballot : Ballot\\
&, \quad preparedCounter : BallotNumber\\
&, \quad hCounter : BallotNumber\\
&, \quad cCounter : BallotNumber]
\end{aligned}$$

$$\begin{aligned}
&SCPExternalize \triangleq [\\
&\quad type : \{\text{“EXTERNALIZE”}\}\\
&, \quad commit : Ballot\\
&, \quad hCounter : BallotNumber]
\end{aligned}$$

$$\begin{aligned}
&Message \triangleq\\
&\quad SCPPrepare \cup SCPCommit \cup SCPExternalize
\end{aligned}$$

Some well-formedness conditions on messages:

$$\begin{aligned}
&MessageInvariant(m) \triangleq\\
&\quad \land\ m.type = \text{“PREPARE”} \Rightarrow\\
&\qquad \land\ m.ballot.counter > 0\\
&\qquad \land\ m.prepared.counter > -1 \Rightarrow\\
&\qquad\quad \land\ m.prepared \preceq m.ballot\\
&\qquad\quad \land\ m.aCounter \leq m.prepared.counter\\
&\qquad \land\ m.prepared.counter = -1 \Rightarrow m.aCounter = 0\\
&\qquad \land\ m.cCounter \leq m.hCounter\\
&\quad \land\ m.type = \text{“COMMIT”} \Rightarrow\\
&\qquad \land\ m.cCounter > 0
\end{aligned}$$

$\land \ m.cCounter \leq m.ballot.counter$
$\land \ m.cCounter \leq m.hCounter$

TODO: Page 13 mentions that we should have $m.hCounter \leq m.ballot.counter$ in a *PREPARE* message
This seems superfluous.
I guess the sender should have increased its ballot counter before sending the message, but it's not a safety problem.

Meaning of the messages in terms of logical, federated-voting messages on abort/commit statements. We will use this to show that this specification refines the *AbstractBallotingWithPrepare* specification.

$LogicalMessages(m) \triangleq$
CASE $m.type =$ "PREPARE" $\rightarrow$ [
$voteToAbort \mapsto \{b \in Ballot :$
$LessThanAndIncompatible(b, m.ballot)\},$
$acceptedAborted \mapsto \{b \in Ballot :$
$\lor LessThanAndIncompatible(b, m.prepared)$
$\lor b.counter < m.aCounter\},$
$confirmedAborted \mapsto$
IF $m.hCounter = 0$ THEN $\{\}$
ELSE $\{b \in Ballot :$
LET $h \triangleq [counter \mapsto m.hCounter, value \mapsto m.ballot.value]$
IN $LessThanAndIncompatible(b, h)\},$
$voteToCommit \mapsto$ IF $m.cCounter = 0$ THEN $\{\}$
ELSE $\{b \in Ballot :$
$\land m.cCounter \leq b.counter \land b.counter \leq m.hCounter$
$\land b.value = m.ballot.value\},$
$acceptedCommitted \mapsto \{\}$]
$\square$ $m.type =$ "COMMIT" $\rightarrow$ [
$voteToAbort \mapsto \{b \in Ballot : b.value \neq m.ballot.value\},$
$acceptedAborted \mapsto$
LET $maxPrepared \triangleq [counter \mapsto m.preparedCounter, value \mapsto m.ballot.value]$
IN $\{b \in Ballot : LessThanAndIncompatible(b, maxPrepared)\},$
$confirmedAborted \mapsto$
LET $maxPrepared \triangleq [counter \mapsto m.hCounter, value \mapsto m.ballot.value]$
IN $\{b \in Ballot : LessThanAndIncompatible(b, maxPrepared)\},$
$voteToCommit \mapsto \{b \in Ballot :$
$m.cCounter \leq b.counter \land b.value = m.ballot.value\},$
$acceptedCommitted \mapsto \{b \in Ballot :$
$\land m.cCounter \leq b.counter \land b.counter \leq m.hCounter$
$\land b.value = m.ballot.value\}$]

VARIABLES
ballot *ballot*[*n*] is the current ballot being prepared or committed by node *n*
, *phase* *phase*[*n*] is the current phase of node *n*
, *prepared* *prepared*[*n*] is the highest accepted-prepared ballot by node *n*
, *aCounter* *aCounter*[*n*] is such that all lower ballots are accepted as aborted
h and *c* track:

in the *PREPARE* phase, the highest and lowest confirmed-prepared ballot
in the *COMMIT* phase, the highest and lowest accepted committed ballot
in the *EXTERNALIZE* phase, the highest and lowest confirmed committed ballot
In phase *PREPARE*, *h.value* could be different from *ballot.value*

, h
, c
, $sent$ $sent[n]$ is the set of messages sent by node n
, byz the set of *Byzantine* nodes

$Init \triangleq$
$\land\ ballot = [n \in N \mapsto nullBallot]$
$\land\ phase = [n \in N \mapsto$ "PREPARE"$]$
$\land\ prepared\ = [n \in N \mapsto nullBallot]$
$\land\ aCounter = [n \in N \mapsto 0]$
$\land\ h = [n \in N \mapsto nullBallot]$
$\land\ c = [n \in N \mapsto nullBallot]$
$\land\ sent = [n \in N \mapsto \{\}]$
$\land\ byz \in FailProneSet$

$TypeOK \triangleq$
$\land\ ballot \in [N \to BallotOrNull]$
$\land\ phase \in [N \to Phase]$
$\land\ prepared\ \in [N \to BallotOrNull]$
$\land\ aCounter \in [N \to BallotNumber]$
$\land\ h \in [N \to BallotOrNull]$
$\land\ c \in [N \to BallotOrNull]$
$\land\ sent \in [N \to \text{SUBSET } Message]$
$\land\ byz \in \text{SUBSET } N$

faulty nodes can send any message they want
$ByzStep \triangleq \exists\, msgs \in [byz \to \text{SUBSET } Message] :$
$\land\ sent' = [n\ \in N \mapsto \text{IF } n \notin byz \text{ THEN } sent[n] \text{ ELSE } msgs[n]]$
$\land\ \text{UNCHANGED } \langle ballot, phase, prepared, aCounter, h, c, byz\rangle$

We start by specifying how a node updates its local state depending on the messages it receives.

At any point in time, we may increase the ballot counter and set the ballot value to the value of the highest confirmed prepared ballot, if any, or, if none, arbitrarily.

$IncreaseBallotCounter(n, b) \triangleq$
$\land\ b > 0$
$\land\ b > ballot[n].counter$
$\land\ \text{IF } h[n].counter > 0 \text{ THEN}$
$ballot' = [ballot \text{ EXCEPT } ![n] = [counter \mapsto b, value \mapsto h[n].value]]$
ELSE
$\exists\, v \in V : ballot' = [ballot \text{ EXCEPT } ![n] = [counter \mapsto b, value \mapsto v]]$
TODO: optimization

```
    ∧ IF b = 1
      THEN c' = [c EXCEPT ![n] = ballot'[n]]
      ELSE UNCHANGED c
    ∧ UNCHANGED ⟨phase, prepared, aCounter, h, c, sent, byz⟩

VotesToPrepare(b, m) ≜
    ∨ ∧ m.type = "PREPARE"
      ∧ ∨ ∧ b.counter ≤ m.ballot.counter
          ∧ b.value = m.ballot.value
        ∨ ∧ b.counter ≤ m.prepared.counter
          ∧ b.value = m.prepared.value
        ∨ b.counter < m.aCounter
    ∨ ∧ m.type = "COMMIT"
      ∧ b.value = m.ballot.value

AcceptsPrepared(b, m) ≜
    ∨ ∧ m.type = "PREPARE"
      ∧ ∨ ∧ b.counter ≤ m.prepared.counter
          ∧ b.value = m.prepared.value
        ∨ b.counter < m.aCounter
    ∨ ∧ m.type = "COMMIT"
      ∧ b.counter ≤ m.preparedCounter
      ∧ b.value = m.ballot.value

 whether b is aborted given aCounter and prepared:
Aborted(b, a, p) ≜
    ∨ b.counter < a
    ∨ LessThanAndIncompatible(b, p)

 update prepared and aCounter given a new accepted-prepared ballot
UpdatePrepared(n, b) ≜
    TODO: what's commented out might be needed for liveness:
    IF prepared[n] ≺ b
    THEN
      ∧ prepared' = [prepared EXCEPT ![n] = b]
      ∧ IF prepared[n].counter > −1 ∧ prepared[n].value ≠ b.value
        THEN aCounter' = [aCounter EXCEPT ![n] =
               IF prepared[n].value < b.value
               THEN prepared[n].counter
               ELSE prepared[n].counter + 1]
        ELSE UNCHANGED aCounter
    ELSE
      IF b.value ≠ prepared[n].value ∧ b.counter ≥ aCounter[n]
      THEN aCounter' = [aCounter EXCEPT ![n] =
             IF prepared[n].value < b.value
             THEN prepared[n].counter
```

```
        ELSE  prepared[n].counter + 1]
    ELSE
        ELSE  UNCHANGED aCounter

 Update what is accepted as prepared:
AcceptPrepared(n, b) ≜
    ∧  prepared[n] ≺ b
    ∧  ∨ ∃ Q ∈ Quorum : ∀ m ∈ Q : ∃ msg ∈ sent[m] : VotesToPrepare(b, msg)
       ∨ ∃ B ∈ BlockingSet : ∀ m ∈ B : ∃ msg ∈ sent[m] : AcceptsPrepared(b, msg)
    ∧  UpdatePrepared(n, b)
     Reset c to nullBallot if it has been aborted:
    ∧  IF c[n].counter > −1 ∧ Aborted(c[n], aCounter′[n], prepared′[n])
       THEN c′ = [c EXCEPT ![n] = nullBallot]
       ELSE UNCHANGED c
    ∧  UNCHANGED ⟨ballot, phase, h, sent, byz⟩

 Update what is confirmed as prepared:
ConfirmPrepared(n, b) ≜
    ∧  h[n] ≺ b
    ∧  ∃ Q ∈ Quorum : ∀ m ∈ Q : ∃ msg ∈ sent[m] : AcceptsPrepared(b, msg)
    ∧  h′ = [h EXCEPT ![n] = b]
     TODO what if we confirm prepared something that's lower and incompatible with prepared?
     Should we update aCounter? (see commented-out part of UpdatePrepared)
    ∧  IF prepared[n] ≺ b  confirmed prepared implies accepted prepared
       THEN UpdatePrepared(n, b)
       ELSE UNCHANGED ⟨prepared, aCounter⟩
     Update c (either reset to nullBallot, if it has been aborted, or set it to b):
    ∧  IF   ∧ c[n].counter > −1
            ∧ ∨ Aborted(c[n], aCounter′[n], prepared′[n])
              ∨ LessThanAndIncompatible(c[n], b)
       THEN c′ = [c EXCEPT ![n] = nullBallot]
       ELSE
          IF   ∧ c[n].counter = −1
               ∧ b = ballot[n]
               ∧ ¬Aborted(b, aCounter′[n], prepared′[n])
           THEN c′ = [c EXCEPT ![n] = b]
           ELSE UNCHANGED c
    ∧  IF b.counter > 0 ∧ ballot[n].counter < b.counter
       THEN ballot′ = [ballot EXCEPT ![n] = b]  not strictly necessary, but might help curb the statespace
       ELSE UNCHANGED ballot
    ∧  UNCHANGED ⟨phase, sent, byz⟩

 NOTE this should be consistent with LogicalMessages
VotesToCommit(b, m) ≜
    ∨  ∧  m.type = "PREPARE"
       ∧  m.cCounter > 0
```

$\land\ m.cCounter \leq b.counter$
$\land\ b.counter \leq m.hCounter$
$\land\ b.value = m.ballot.value$
$\lor\ \land\ m.type = \text{“COMMIT”}$
$\land\ m.cCounter \leq b.counter$
$\land\ b.value = m.ballot.value$

NOTE this should be consistent with *LogicalMessages*

$AcceptsCommitted(b,\ m) \triangleq$
$\land\ m.type = \text{“COMMIT”}$
$\land\ b.value = m.ballot.value$
$\land\ m.cCounter \leq b.counter$
$\land\ b.counter \leq m.hCounter$

$AcceptCommitted(n,\ b) \triangleq$
$\land\ b = ballot[n]$ *TODO* okay?
$\land$ IF $phase[n] = \text{“PREPARE”}$
THEN $phase' = [phase$ EXCEPT $![n] = \text{“COMMIT”}] \land c' = [c$ EXCEPT $![n] = b]$
ELSE UNCHANGED $\langle phase,\ c\rangle$
$\land\ phase[n] = \text{“COMMIT”} \Rightarrow h[n] \prec b$
$\land\ \lor \exists\, Q \in Quorum : \forall\, m \in Q : \exists\, msg \in sent[m] : VotesToCommit(b,\ msg)$
$\lor \exists\, B \in BlockingSet : \forall\, m \in B : \exists\, msg \in sent[m] : AcceptsCommitted(b,\ msg)$
$\land\ h' = [h$ EXCEPT $![n] = b]$
$\land$ IF $prepared[n] \prec b$ accepted committed implies accepted prepared
THEN $UpdatePrepared(n,\ b)$
ELSE UNCHANGED $\langle prepared,\ aCounter\rangle$
$\land$ UNCHANGED $\langle ballot,\ sent,\ byz\rangle$

Now we specify what messages can be sent by a node

Summarize what has been prepared, under the constraint that prepared is less than or equal to ballot:

$SummarizePrepared(n) \triangleq$
IF $prepared[n] \preceq ballot[n]$
THEN $[prepared \mapsto prepared[n],\ aCounter \mapsto aCounter[n]]$
ELSE
IF $ballot[n].value > prepared[n].value \lor aCounter[n] > ballot[n].counter$
THEN [
$prepared \mapsto [counter \mapsto ballot[n].counter,\ value \mapsto prepared[n].value]$,
$aCounter \mapsto Min(aCounter[n],\ ballot[n].counter)$]
ELSE
IF $aCounter[n] = ballot[n].counter$
TODO okay?
THEN [
$prepared \mapsto [counter \mapsto ballot[n].counter,\ value \mapsto ballot[n].value]$,
$aCounter \mapsto aCounter[n]$]
ELSE [

```
                prepared   ↦ [counter ↦ ballot[n].counter − 1, value ↦ prepared[n].value],
                aCounter ↦ Min(aCounter[n], ballot[n].counter − 1)]

SendPrepare(n) ≜
  ∧ ballot[n].counter > 0
  ∧ phase[n] = "PREPARE"
  ∧ LET msg ≜ [
          type ↦ "PREPARE"
    ,   ballot ↦ ballot[n]
    ,   prepared   ↦ SummarizePrepared(n).prepared
    ,   aCounter ↦ SummarizePrepared(n).aCounter
    ,   hCounter ↦
          IF h[n].counter > −1 ∧ h[n].value = ballot[n].value
          THEN h[n].counter
          ELSE 0
    ,   cCounter ↦ Max(c[n].counter, 0)]
    IN
      sent′ = [sent EXCEPT ![n] = sent[n] ∪ {msg}]
  ∧ UNCHANGED ⟨ballot, phase, prepared, aCounter, h, c, byz⟩

SendCommit(n) ≜
  ∧ phase[n] = "COMMIT"
  ∧ LET msg ≜ [
          type ↦ "COMMIT"
    ,   ballot ↦ ballot[n]
    ,   preparedCounter ↦ prepared[n].counter
    ,   hCounter ↦ h[n].counter
    ,   cCounter ↦ c[n].counter]
    IN
      sent′ = [sent EXCEPT ![n] = sent[n] ∪ {msg}]
  ∧ UNCHANGED ⟨ballot, phase, prepared, aCounter, h, c, byz⟩

SendExternalize(n) ≜
  ∧ phase[n] = "EXTERNALIZE"
  ∧ LET msg ≜ [
          type ↦ "EXTERNALIZE"
    ,   commit ↦ ballot[n]
    ,   hCounter ↦ h[n].counter]
    IN
      sent′ = [sent EXCEPT ![n] = sent[n] ∪ {msg}]
  ∧ UNCHANGED ⟨ballot, phase, prepared, aCounter, h, c, byz⟩
```

We can now give the full specification

```
Next ≜
  ∨ ByzStep
```

$\lor \exists\, n \in N \setminus byz :$
$\lor \exists\, cnt \in BallotNumber : IncreaseBallotCounter(n,\, cnt)$
$\lor \exists\, b \in Ballot :$
$\lor\ AcceptPrepared(n,\, b)$
$\lor\ ConfirmPrepared(n,\, b)$
$\lor\ AcceptCommitted(n,\, b)$
$\lor SendPrepare(n)$
$\lor SendCommit(n)$
$\lor SendExternalize(n)$

$vars \triangleq \langle ballot,\, phase,\, prepared,\, aCounter,\, h,\, c,\, sent,\, byz \rangle$

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

We now turn to correctness properties

$Invariant \triangleq$
$\land\ TypeOK$
$\land\ \forall\, n \in N \setminus byz :$
$\land\ \forall\, m \in sent[n] : MessageInvariant(m)$
$\land\ ballot[n].counter = -1 \lor ballot[n].counter > 0$
$\land\ prepared[n].counter > -1 \Rightarrow aCounter[n] \leq prepared[n].counter$
$\land\ prepared[n].counter = -1 \Rightarrow aCounter[n] = 0$
$\land\ h[n] \preceq prepared[n]$
$\land\ c[n].counter = -1 \lor c[n].counter > 0$
$\land\ c[n].counter \leq h[n].counter$
$\land\ c[n].counter \leq ballot[n].counter$
$\land\ c[n].counter > 0 \Rightarrow$
$\land\ c[n].value = h[n].value$
$\land\ c[n].value = prepared[n].value$
$\land\ c[n].value = ballot[n].value$

Next we instantiate the *AbstractBalloting* specification

$voteToAbort \triangleq [n \in N \mapsto \text{UNION } \{LogicalMessages(m).voteToAbort : m \in sent[n]\}]$
$acceptedAborted \triangleq [n \in N \mapsto \text{UNION } \{LogicalMessages(m).acceptedAborted : m \in sent[n]\}]$
$confirmedAborted \triangleq [n \in N \mapsto \text{UNION } \{LogicalMessages(m).confirmedAborted : m \in sent[n]\}]$
$voteToCommit \triangleq [n \in N \mapsto \text{UNION } \{LogicalMessages(m).voteToCommit : m \in sent[n]\}]$
$acceptedCommitted \triangleq [n \in N \mapsto \text{UNION } \{LogicalMessages(m).acceptedCommitted : m \in sent[n]\}]$
$externalized \triangleq [n \in N \mapsto \{\}]$

$AB \triangleq \text{INSTANCE } AbstractBalloting$

We have a correct refinement:
THEOREM $Spec \Rightarrow AB\,!\,Spec$

To check the refinement with TLC:

$$\begin{array}{l} \mathit{InitRefinement} \triangleq \\ \quad AB!\mathit{Init} \\ \mathit{NextRefinement} \triangleq \\ \quad \Box[AB!\mathit{Next}]_{vars} \end{array}$$