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 ≜ {"PREPARE", "COMMIT", "EXTERNALIZE"}

SCPPrepare ≜ [
    type : {"PREPARE"}
,   ballot : Ballot
,   prepared  : BallotOrNull
,   aCounter : BallotNumber
,   hCounter : BallotNumber
,   cCounter : BallotNumber]

SCPCommit ≜ [
    type : {"COMMIT"}
,   ballot : Ballot
,   preparedCounter : BallotNumber
,   hCounter : BallotNumber
,   cCounter : BallotNumber]

SCPExternalize ≜ [
    type : {"EXTERNALIZE"}
,   commit : Ballot
,   hCounter : BallotNumber]

Message ≜
    SCPPrepare ∪ SCPCommit ∪ SCPExternalize

  Some well-formedness conditions on messages:
MessageInvariant(m) ≜
    ∧ m.type = "PREPARE" ⇒
        ∧ m.ballot.counter > 0
        ∧ m.prepared.counter > −1 ⇒
            ∧ m.prepared  ⪯ m.ballot
            ∧ m.aCounter ≤ m.prepared.counter
        ∧ m.prepared.counter = −1 ⇒ m.aCounter = 0
        ∧ m.cCounter ≤ m.hCounter
    ∧ m.type = "COMMIT" ⇒
        ∧ m.cCounter > 0
        ∧  m.cCounter ≤ m.ballot.counter
        ∧  m.cCounter ≤ 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.
We will use this to show that this specification refines the *AbstractBalloting* specification.

```
LogicalMessages(m) ≜
   CASE m.type = "PREPARE" → [
                voteToAbort ↦ {b ∈ Ballot :
                    LessThanAndIncompatible(b, m.ballot)},
                acceptedAborted ↦ {b ∈ Ballot :
                    ∨ LessThanAndIncompatible(b, m.prepared)
                    ∨ b.counter < m.aCounter},
                confirmedAborted ↦
                    IF m.hCounter = 0 THEN {}
                     ELSE {b ∈ Ballot :
                        LET h ≜ [counter ↦ m.hCounter, value ↦ m.ballot.value]
                        IN  LessThanAndIncompatible(b, h)},
                voteToCommit ↦ IF m.cCounter = 0 THEN {}
                     ELSE {b ∈ Ballot :
                        ∧ m.cCounter ≤ b.counter ∧ b.counter ≤ m.hCounter
                        ∧ b.value = m.ballot.value},
                acceptedCommitted ↦ {}]
   □ m.type = "COMMIT" → [
             voteToAbort ↦ {b ∈ Ballot : b.value ≠ m.ballot.value},
             acceptedAborted ↦
                 LET maxPrepared ≜ [counter ↦ m.preparedCounter, value ↦ m.ballot.value]
                 IN  {b ∈ Ballot : LessThanAndIncompatible(b, maxPrepared)},
             confirmedAborted ↦
                 LET maxPrepared ≜ [counter ↦ m.hCounter, value ↦ m.ballot.value]
                 IN  {b ∈ Ballot : LessThanAndIncompatible(b, maxPrepared)},
             voteToCommit ↦ {b ∈ Ballot :
                 m.cCounter ≤ b.counter ∧ b.value = m.ballot.value},
             acceptedCommitted ↦ {b ∈ Ballot :
                 ∧ m.cCounter ≤ b.counter ∧ b.counter ≤ m.hCounter
                 ∧ 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
```

```
$Init \triangleq$
    $\land\ ballot = [n \in N \mapsto NullBallot]$
    $\land\ phase = [n \in N \mapsto \text{“PREPARE”}]$
    $\land\ prepared\ \ = [n \in N \mapsto NullBallot]$
    $\land\ aCounter = [n \in N \mapsto 0]$
    $\land\ h = [n \in N \mapsto NullBallot]$
    $\land\ c = [n \in N \mapsto NullBallot]$
    $\land\ sent = [n \in N \mapsto \{\}]$
    $\land\ byz \in FailProneSet$
```

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

faulty nodes can send any message they want

```
$ByzStep \triangleq \exists\, msgs \in [byz \rightarrow \text{SUBSET}\ Message] :$
    $\land$   $sent' = [n\ \ \in N \mapsto \text{IF}\ n \notin byz\ \text{THEN}\ sent[n]\ \text{ELSE}\ \ msgs[n]]$
    $\land$   $\text{UNCHANGED}\ \langle ballot, phase, prepared, aCounter, h, c, byz \rangle$
```

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.

```
$IncreaseBallotCounter(n, b) \triangleq$
    $\land$  $b > 0$
    $\land$  $b > ballot[n].counter$
    $\land$  $\text{IF}\ h[n].counter > 0\ \text{THEN}$
              $ballot' = [ballot\ \text{EXCEPT}\ ![n] = [counter \mapsto b, value \mapsto h[n].value]]$
         $\text{ELSE}$
              $\exists\, v \in V : ballot' = [ballot\ \text{EXCEPT}\ ![n] = [counter \mapsto b, value \mapsto v]]$
    TODO: optimization
    $\land$ IF $b = 1$
        THEN $c' = [c$ EXCEPT $![n] = ballot'[n]]$
        ELSE UNCHANGED $c$
    $\land$ UNCHANGED $\langle phase, prepared, aCounter, h, c, sent, byz \rangle$

$VotesToPrepare(b, m) \triangleq$
    $\lor$ $\land$ $m.type =$ "PREPARE"
       $\land$ $\lor$ $\land$ $b.counter \leq m.ballot.counter$
               $\land$ $b.value = m.ballot.value$
          $\lor$ $\land$ $b.counter \leq m.prepared.counter$
               $\land$ $b.value = m.prepared.value$
          $\lor$ $b.counter < m.aCounter$
    $\lor$ $\land$ $m.type =$ "COMMIT"
       $\land$ $b.value = m.ballot.value$

$AcceptsPrepared(b, m) \triangleq$
    $\lor$ $\land$ $m.type =$ "PREPARE"
       $\land$ $\lor$ $\land$ $b.counter \leq m.prepared.counter$
               $\land$ $b.value = m.prepared.value$
          $\lor$ $b.counter < m.aCounter$
    $\lor$ $\land$ $m.type =$ "COMMIT"
       $\land$ $b.counter \leq m.preparedCounter$
       $\land$ $b.value = m.ballot.value$

whether $b$ is aborted given $aCounter$ and prepared:
$Aborted(b, a, p) \triangleq$
    $\lor$ $b.counter < a$
    $\lor$ $LessThanAndIncompatible(b, p)$

update prepared and $aCounter$ given a new accepted-prepared ballot
$UpdatePrepared(n, b) \triangleq$
    $TODO$: what's commented out might be needed for liveness:
    IF $prepared[n] \prec b$
     THEN
       $\land$ $prepared' = [prepared$ EXCEPT $![n] = b]$
       $\land$ IF $prepared[n].counter > -1 \land prepared[n].value \neq b.value$
           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 $b.value \neq prepared[n].value \land b.counter \geq aCounter[n]$
        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] ≺ b
        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
```

```
         ∧ m.cCounter ≤ b.counter
         ∧ b.counter ≤ m.hCounter
         ∧ b.value = m.ballot.value
      ∨  ∧ m.type = "COMMIT"
         ∧ m.cCounter ≤ b.counter
         ∧ b.value = m.ballot.value
```

NOTE this should be consistent with *LogicalMessages*

```
AcceptsCommitted(b, m) ≜
    ∧ m.type = "COMMIT"
    ∧ b.value = m.ballot.value
    ∧ m.cCounter ≤ b.counter
    ∧ b.counter ≤ m.hCounter

AcceptCommitted(n, b) ≜
    ∧ b = ballot[n]   TODO okay?
    ∧ IF phase[n]   = "PREPARE"
        THEN phase′ = [phase EXCEPT ![n] = "COMMIT"] ∧ c′ = [c EXCEPT ![n] = b]
        ELSE UNCHANGED ⟨phase, c⟩
    ∧ phase[n] = "COMMIT" ⇒ h[n] ≺ b
    ∧  ∨ ∃ Q ∈ Quorum : ∀ m ∈ Q : ∃ msg ∈ sent[m] : VotesToCommit(b, msg)
       ∨ ∃ B ∈ BlockingSet : ∀ m ∈ B : ∃ msg ∈ sent[m] : AcceptsCommitted(b, msg)
    ∧ h′ = [h EXCEPT ![n] = b]
    ∧ IF prepared[n] ≺ b   accepted committed implies accepted prepared
        THEN UpdatePrepared(n, b)
        ELSE UNCHANGED ⟨prepared, aCounter⟩
    ∧ UNCHANGED ⟨ballot, sent, byz⟩
```

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) ≜
   IF prepared[n] ⪯ ballot[n]
    THEN [prepared ↦ prepared[n], aCounter ↦ aCounter[n]]
    ELSE
       IF ballot[n].value > prepared[n].value ∨ aCounter[n] > ballot[n].counter
        THEN [
           prepared  ↦ [counter ↦ ballot[n].counter, value ↦ prepared[n].value],
           aCounter ↦ Min(aCounter[n], ballot[n].counter)]
        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"
    $\land$ LET $msg \triangleq [$
            $type \mapsto$ "PREPARE"
        $,\ ballot \mapsto ballot[n]$
        $,\ prepared \mapsto SummarizePrepared(n).prepared$
        $,\ aCounter \mapsto SummarizePrepared(n).aCounter$
        $,\ hCounter \mapsto$
                IF $h[n].counter > -1 \land h[n].value = ballot[n].value$
                 THEN $h[n].counter$
                 ELSE $0$
        $,\ cCounter \mapsto Max(c[n].counter,\ 0)]$
        IN
            $sent' = [sent$ EXCEPT $![n] = sent[n] \cup \{msg\}]$
    $\land$ UNCHANGED $\langle ballot,\ phase,\ prepared,\ aCounter,\ h,\ c,\ byz \rangle$

$SendCommit(n) \triangleq$
    $\land\ phase[n] =$ "COMMIT"
    $\land$ LET $msg \triangleq [$
            $type \mapsto$ "COMMIT"
        $,\ ballot \mapsto ballot[n]$
        $,\ preparedCounter \mapsto prepared[n].counter$
        $,\ hCounter \mapsto h[n].counter$
        $,\ cCounter \mapsto c[n].counter]$
        IN
            $sent' = [sent$ EXCEPT $![n] = sent[n] \cup \{msg\}]$
    $\land$ UNCHANGED $\langle ballot,\ phase,\ prepared,\ aCounter,\ h,\ c,\ byz \rangle$

$SendExternalize(n) \triangleq$
    $\land\ phase[n] =$ "EXTERNALIZE"
    $\land$ LET $msg \triangleq [$
            $type \mapsto$ "EXTERNALIZE"
        $,\ commit \mapsto ballot[n]$
        $,\ hCounter \mapsto h[n].counter]$
        IN
            $sent' = [sent$ EXCEPT $![n] = sent[n] \cup \{msg\}]$
    $\land$ UNCHANGED $\langle ballot,\ phase,\ prepared,\ aCounter,\ h,\ c,\ byz \rangle$
```

We can now give the full specification

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

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

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

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*:

```
$InitRefinement \triangleq$
    $AB!Init$
$NextRefinement \triangleq$
    $\Box[AB!Next]_{vars}$
```

Debugging canaries:

```
$Canary1 \triangleq \neg($
    $\exists\, n \in N \setminus byz : phase[n] = \text{“COMMIT”}$
$)$
$Canary2 \triangleq \neg($
    $\exists\, n \in N \setminus byz : \exists\, msg \in sent[n] :$
        $\land\ msg.type = \text{“PREPARE”}$
        $\land\ msg.cCounter = 1$
$)$
$Canary3 \triangleq \neg($
    $\exists\, Q \in Quorum : \forall\, n \in Q \setminus byz : c[n].counter = 1$
$)$
```