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*, *Variants*

$Phase \triangleq \{\text{"PREPARE"}, \text{"COMMIT"}, \text{"EXTERNALIZE"}\}$

```
@typeAlias: message =
  PREPARE({ballot : $ ballot, prepared : $ ballot, aCounter : Int, hCounter : Int, cCounter : Int})
 | COMMIT({ballot : $ ballot, preparedCounter : Int, hCounter : Int, cCounter : Int});
```

$$
\begin{aligned}
SCPPrepare \triangleq\ & \{Variant(\text{"PREPARE"}, m) : m \in [\\
& \quad ballot : Ballot \\
& , \quad prepared : BallotOrNull \\
& , \quad aCounter : BallotNumber \\
& , \quad hCounter : BallotNumber \cup \{-1\} \\
& , \quad cCounter : BallotNumber \cup \{-1\}]\}
\end{aligned}
$$

$$
\begin{aligned}
SCPCommit \triangleq\ & \{Variant(\text{"COMMIT"}, m) : m \in [\\
& \quad ballot : Ballot \\
& , \quad preparedCounter : BallotNumber \\
& , \quad hCounter : BallotNumber \cup \{-1\} \\
& , \quad cCounter : BallotNumber \cup \{-1\}]\}
\end{aligned}
$$

$$
\begin{aligned}
& SCPExternalize \triangleq \{Variant(\text{"EXTERNALIZE"}, m) : m \in [\\
& \quad \text{commit} : Ballot \\
& , hCounter : BallotNumber]\}
\end{aligned}
$$

```
@type: Set($ message);
```

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

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

$Init \triangleq$

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

$TypeOK \triangleq$

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

faulty nodes can send any message they want

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

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:
TODO why is it usefull to have this constraint?

$SummarizePrepared(n) \triangleq$

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  ↦ [counter ↦ ballot[n].counter, value ↦ ballot[n].value],
                    aCounter ↦ 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 ≜ Variant("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 −1  TODO okay?
                ,   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 ≜ Variant("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 ≜ Variant("EXTERNALIZE", [
                    commit  ↦ ballot[n]
                , hCounter ↦ h[n].counter])
      IN
          sent′ = [sent EXCEPT ![n] = sent[n] ∪ {msg}]
    ∧ UNCHANGED ⟨ballot, phase, prepared, aCounter, h, c, byz⟩
```

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. In practice, this happens when according to a timer.

```
IncreaseBallotCounter(n, b) ≜
    ∧ b > 0
    ∧ b > ballot[n].counter
    ∧ IF h[n].counter > 0 THEN
            ballot' = [ballot EXCEPT ![n] = [counter ↦ b, value ↦ h[n].value]]
        ELSE
            ∃ v ∈ V : ballot' = [ballot EXCEPT ![n] = [counter ↦ b, value ↦ v]]
     TODO: optimization
     ∧ IF b = 1
       THEN c' = [c EXCEPT ![n] = ballot'[n]]
       ELSE UNCHANGED c
    ∧ UNCHANGED ⟨phase, prepared, aCounter, h, c, sent, byz⟩
```

We now specify how a node updates its local state depending on the messages it receives.

```
 @type: ($ballot, $message) ⇒ Bool;
VotesToPrepare(b, taggedMsg) ≜
    IF VariantTag(taggedMsg) = "PREPARE"
     THEN LET m ≜ VariantGetUnsafe("PREPARE", taggedMsg)IN
        ∧ ∨ ∧ b.counter ≤ m.ballot.counter
              ∧ b.value = m.ballot.value
          ∨ ∧ b.counter ≤ m.prepared.counter
              ∧ b.value = m.prepared.value
          ∨ b.counter < m.aCounter
     ELSE IF VariantTag(taggedMsg) = "COMMIT"
     THEN LET m ≜ VariantGetUnsafe("COMMIT", taggedMsg)IN
        ∧ b.value = m.ballot.value
     ELSE TRUE

 @type: ($ballot, $message) ⇒ Bool;
AcceptsPrepared(b, taggedMsg) ≜
    IF VariantTag(taggedMsg) = "PREPARE"
     THEN LET m ≜ VariantGetUnsafe("PREPARE", taggedMsg)IN
        ∧ ∨ ∧ b.counter ≤ m.prepared.counter
              ∧ b.value = m.prepared.value
          ∨ b.counter < m.aCounter
     ELSE IF VariantTag(taggedMsg) = "COMMIT"
     THEN LET m ≜ VariantGetUnsafe("COMMIT", taggedMsg)IN
        ∧ b.counter ≤ m.preparedCounter
        ∧ b.value = m.ballot.value
     ELSE TRUE

 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
 @type: ($ballot, $message) ⇒ Bool;
VotesToCommit(b, taggedMsg) ≜
    IF VariantTag(taggedMsg) = "PREPARE"
     THEN LET m ≜ VariantGetUnsafe("PREPARE", taggedMsg)IN
        ∧ m.cCounter > 0
        ∧ m.cCounter ≤ b.counter
        ∧ b.counter ≤ m.hCounter
        ∧ b.value = m.ballot.value
     ELSE IF VariantTag(taggedMsg) = "COMMIT"
     THEN LET m ≜ VariantGetUnsafe("COMMIT", taggedMsg)IN
        ∧ m.cCounter ≤ b.counter
        ∧ b.value = m.ballot.value
     ELSE TRUE

 NOTE this should be consistent with LogicalMessages
 @type: ($ballot, $message) ⇒ Bool;
AcceptsCommitted(b, taggedMsg) ≜
    IF VariantTag(taggedMsg) = "COMMIT"
     THEN
        LET m ≜ VariantGetUnsafe("COMMIT", taggedMsg)
        IN
            ∧ b.value = m.ballot.value
            ∧ m.cCounter ≤ b.counter
            ∧ b.counter ≤ m.hCounter
     ELSE FALSE

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⟩
```

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

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

Some well-formedness conditions on messages:
@type: $message $\Rightarrow$ *Bool*;
$MessageInvariant(taggedMsg) \triangleq$
IF $VariantTag(taggedMsg) = \text{“PREPARE”}$
THEN LET $m \triangleq VariantGetUnsafe(\text{“PREPARE”},\, taggedMsg)$ IN
$\land\ m.ballot.counter > 0$
$\land\ m.prepared.counter > -1 \Rightarrow$
$\land\ m.prepared \preceq m.ballot$
$\land\ m.aCounter \leq m.prepared.counter$
$\land\ m.prepared.counter = -1 \Rightarrow m.aCounter = 0$
$\land\ m.cCounter \leq m.hCounter$
ELSE IF $VariantTag(taggedMsg) = \text{“COMMIT”}$
THEN LET $m \triangleq VariantGetUnsafe(\text{“COMMIT”},\, taggedMsg)$ IN
$\land\ m.cCounter > 0$
$\land\ m.cCounter \leq m.ballot.counter$
$\land\ m.cCounter \leq m.hCounter$

```
    ELSE  TRUE
```

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.

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

Meaning of the messages in terms of logical, federated-voting messages.
We will use this to show that this specification refines the *AbstractBalloting* specification.
@type: $message ⇒ {*voteToAbort* : *Set*($ *ballot*), *acceptedAborted* : *Set*($ *ballot*), *confirmedAborted* : *Set*($ *ballot*), *voteToCom*

$$
\begin{array}{l}
LogicalMessages(taggedMsg) \triangleq \\
\quad \text{IF } VariantTag(taggedMsg) = \text{“PREPARE”} \\
\quad \text{THEN LET } m \triangleq VariantGetUnsafe(\text{“PREPARE”}, taggedMsg) \text{IN} \quad [\\
\qquad voteToAbort \mapsto \{b \in Ballot : \\
\qquad\quad LessThanAndIncompatible(b, m.ballot)\}, \\
\qquad acceptedAborted \mapsto \{b \in Ballot : \\
\qquad\quad \lor LessThanAndIncompatible(b, m.prepared) \\
\qquad\quad \lor b.counter < m.aCounter\}, \\
\qquad confirmedAborted \mapsto \\
\qquad\quad \text{IF } m.hCounter = 0 \text{ THEN } \{\} \\
\qquad\quad \text{ELSE LET } hbal \triangleq [counter \mapsto m.hCounter, value \mapsto m.ballot.value] \text{IN} \\
\qquad\qquad \{b \in Ballot : LessThanAndIncompatible(b, hbal)\}, \\
\qquad voteToCommit \mapsto \text{IF } m.cCounter = 0 \text{ THEN } \{\} \\
\qquad\quad \text{ELSE } \{b \in Ballot : \\
\qquad\qquad \land\, m.cCounter \leq b.counter \land b.counter \leq m.hCounter \\
\qquad\qquad \land\, b.value = m.ballot.value\}, \\
\qquad acceptedCommitted \mapsto \{\}] \\
\quad \text{ELSE IF } VariantTag(taggedMsg) = \text{“COMMIT”} \\
\quad \text{THEN LET } m \triangleq VariantGetUnsafe(\text{“COMMIT”}, taggedMsg) \text{IN} \quad [\\
\qquad voteToAbort \mapsto \{b \in Ballot : b.value \neq m.ballot.value\}, \\
\qquad acceptedAborted \mapsto
\end{array}
$$

```
        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}]
ELSE [voteToAbort ↦ {}, acceptedAborted ↦ {}, confirmedAborted ↦ {}, voteToCommit ↦ {}, accepted
```

Next we instantiate the *AbstractBalloting* specification

```
voteToAbort ≜ [n ∈ N ↦ UNION {LogicalMessages(m).voteToAbort : m ∈ sent[n]}]
acceptedAborted ≜ [n ∈ N ↦ UNION {LogicalMessages(m).acceptedAborted : m ∈ sent[n]}]
confirmedAborted ≜ [n ∈ N ↦ UNION {LogicalMessages(m).confirmedAborted : m ∈ sent[n]}]
voteToCommit ≜ [n ∈ N ↦ UNION {LogicalMessages(m).voteToCommit : m ∈ sent[n]}]
acceptedCommitted ≜ [n ∈ N ↦ UNION {LogicalMessages(m).acceptedCommitted : m ∈ sent[n]}]
@type: NODE → Set( $ ballot);
externalized ≜ [n ∈ N ↦ {}]

AB ≜ INSTANCE AbstractBalloting
```

We have a correct refinement:

```
THEOREM Spec ⇒ AB!Spec
```

To check the refinement with *TLC*:

```
InitRefinement ≜
    AB!Init
NextRefinement ≜
    □[AB!Next]_vars
```

For *Apalache*:

```
ABNext ≜ AB!Next
```