```
──────────────────────────── MODULE curp ────────────────────────────
EXTENDS FiniteSets, Naturals, Sequences

Constants:
   commands : set of records like[key ↦ "key", value ↦ "value"].
   replicas : set of replicas.

CONSTANTS commands, replicas

ASSUME IsFiniteSet(replicas)

Variables :
   leader : records the leader of each epoch.
   epoch : current epoch(the number of leader changes).
   proposedCmds : the set of proposed commands.
   proposeRequests : the set of propose could be received by each replica.
   proposeResponses : the set of responses in each epoch to each proposed
                  command.
   specPools : the speculative pool of each replica.
   uncommittedCmds : the sequence of back − end protocol uncommitted commands.
   committedCmds : the sequence of back − end protocol committed commands.
   commitMsgs : the set of commit messages could be received by each replica.
   specExecPrevCmd : the index of the last same − key command in the committed
                 sequence at the time the leader responds to the proposal.

VARIABLES leader, epoch, proposedCmds, proposeRequests,
          proposeResponses, specPools, uncommittedCmds,
          committedCmds, commitMsgs, specExecPrevCmd

The epoch space.
epoches ≜ Nat

Special noLeader value, for future epoches.
noLeader ≜ CHOOSE r : r ∉ replicas

In N = 2 ∗ f + 1 replicas :
   quorum : a set of replicas that contains at least f + 1 replicas.
   superQuorum : a set of replicas that contains at least f + (f + 1)/2 + 1
             replicas.
   recoverQuorum : a set of replicas that contains at least(f + 1)/2 + 1
              replicas.
quorums ≜
    LET f ≜ Cardinality(replicas) ÷ 2
        size ≜ f + 1
    IN  {q ∈ SUBSET replicas : Cardinality(q) ≥ size}

superQuorums ≜
    LET f ≜ Cardinality(replicas) ÷ 2
        size ≜ f + (f + 1) ÷ 2 + 1
    IN  $\{q \in \textsc{subset}\ replicas : Cardinality(q) \geq size\}$

$recoverQuorums \triangleq$
    LET $f \triangleq Cardinality(replicas) \div 2$
        $size \triangleq (f + 1) \div 2 + 1$
    IN  $\{q \in \textsc{subset}\ replicas : Cardinality(q) \geq size\}$
```

Helper function for converting a set to a set containing all sequences containing the elements of the set exactly once and no other elements.

```
$SetToSeqs(set) \triangleq$
    LET $len \triangleq 1 \,..\, Cardinality(set)$ IN
        $\{f \in [len \rightarrow set] : \forall\, i, j \in len : i \neq j \Rightarrow f[i] \neq f[j]\}$
```

Helper function for getting the index of the last element in a sequence satisfying the predicate.

```
$GetIdxInSeq(seq, Pred(\_)) \triangleq$
    LET $I \triangleq \{i \in 1 \,..\, Len(seq) : Pred(seq[i])\}$ IN
        IF $I \neq \{\}$ THEN CHOOSE $i \in I : \forall\, j \in I : j \leq i$ ELSE $0$
```

Propose a command.
This is done by the client sending a proposeRequest to all replicas.

```
$Propose(cmd) \triangleq$
    $\land$ $proposedCmds' = proposedCmds \cup \{cmd\}$
    $\land$ $proposeRequests' =$
          $[r \in replicas \mapsto proposeRequests[r] \cup \{cmd\}]$
    $\land$ UNCHANGED $\langle leader, epoch, specPools, proposeResponses,$
                        $uncommittedCmds, committedCmds, commitMsgs,$
                        $specExecPrevCmd\rangle$
```

How the leader process a proposeRequest.

```
$ProcessProposeLeader(r, cmd) \triangleq$
    LET $specPoolHasConflict \triangleq$
              $\exists\, specCmd \in specPools[r] : specCmd.key = cmd.key$
        $uncommittedCmdsHasConflict \triangleq$
            $GetIdxInSeq(uncommittedCmds,$ LAMBDA $e : e.key = cmd.key) \neq 0$
    IN
        $\land$ $proposeRequests' =$
            $[proposeRequests$ EXCEPT $![r] = @ \setminus \{cmd\}]$
        $\land$ $specPools' =$
            $[specPools$ EXCEPT $![r] =$
                IF $\neg specPoolHasConflict$ THEN $@ \cup \{cmd\}$ ELSE $@]$
        $\land$ $uncommittedCmds' = Append(uncommittedCmds, cmd)$
        $\land$ $proposeResponses' =$
            $[proposeResponses$ EXCEPT $![cmd][epoch] =$
                IF $\neg specPoolHasConflict \land \neg uncommittedCmdsHasConflict$
                THEN @ ∪ {r}
                ELSE  @]
        ∧ specExecPrevCmd′ =
            [specExecPrevCmd EXCEPT ![cmd] =
                IF ¬specPoolHasConflict ∧ ¬uncommittedCmdsHasConflict
                 THEN GetIdxInSeq(committedCmds, LAMBDA e : e.key = cmd.key)
                 ELSE  @]
        ∧ UNCHANGED ⟨leader, epoch, proposedCmds, committedCmds,
                        commitMsgs⟩
```

How a non − leader replica process a proposeRequest.

```
ProcessProposeNonLeader(r, cmd) ≜
    LET specPoolHasConflict ≜
            ∃ specCmd ∈ specPools[r] : specCmd.key = cmd.key
    IN
        ∧ proposeRequests′ =
            [proposeRequests EXCEPT ![r] = @ \ {cmd}]
        ∧ specPools′ =
            [specPools EXCEPT ![r] =
                IF ¬specPoolHasConflict THEN @ ∪ {cmd} ELSE  @]
        ∧ proposeResponses′ =
            [proposeResponses EXCEPT ![cmd][epoch] =
                IF ¬specPoolHasConflict THEN @ ∪ {r} ELSE  @]
        ∧ UNCHANGED ⟨leader, epoch, proposedCmds, uncommittedCmds,
                        committedCmds, commitMsgs, specExecPrevCmd⟩
```

Syncing a command using the back − end protocol(Raft). The implementation details are omitted.

A replica may not be able to receive the commit message at the exact time the the leader sends it.

```
Commit ≜
    ∧   committedCmds′ = Append(committedCmds, Head(uncommittedCmds))
    ∧   commitMsgs′ =
          [r ∈ replicas ↦ commitMsgs[r] ∪ {Head(uncommittedCmds)}]
    ∧   uncommittedCmds′ = Tail(uncommittedCmds)
    ∧   UNCHANGED ⟨leader, epoch, specPools, proposedCmds, proposeRequests,
                      proposeResponses, specExecPrevCmd⟩
```

How a replica process a commit message.

```
ProcessCommitMsg(r, cmd) ≜
    ∧ commitMsgs′ =
        [commitMsgs EXCEPT ![r] = @ \ {cmd}]
    ∧ specPools′ = [specPools EXCEPT ![r] = @ \ {cmd}]
    ∧ UNCHANGED ⟨leader, epoch, proposedCmds, proposeRequests,
                    proposeResponses, uncommittedCmds, committedCmds,
                  $specExecPrevCmd \rangle$
```

Leader Change Action

The new leader should gather at least a quorum of replicas specPool to recover the commands.

Commands existed in the *specPool* of a *RecoverQuorum* of replicas need to be recovered.

```
$LeaderChange(l) \triangleq$
    $\land\ leader' = [e \in epoches \mapsto \text{IF } e = epoch + 1 \text{ THEN } l \text{ ELSE } leader[e]]$
    $\land\ epoch' = epoch + 1$
    $\land\ \exists\, q \in quorums :$
        $\text{LET } specCmds \triangleq \text{UNION } \{specPools[r] : r \in q\}$
                $newSpecPool \triangleq$
                    $\{cmd \in specCmds :$
                        $\{r \in q : cmd \in specPools[r]\} \in recoverQuorums\}$
        $\text{IN}$
                $\land\ specPools' = [specPools \text{ EXCEPT } ![l] = newSpecPool]$
                $\land\ uncommittedCmds' \in SetToSeqs(newSpecPool)$
    $\land\ \text{UNCHANGED } \langle proposedCmds, proposeRequests, proposeResponses,$
                      $committedCmds, commitMsgs, specExecPrevCmd \rangle$
```

The initial state of the system.

```
$Init \triangleq$
    $\exists\, r \in replicas :$
        $\text{LET } initEpoch \triangleq 1 initLeader \triangleq r\text{IN}$
            $\land\ leader = [e \in epoches \mapsto$
                $\text{IF } e = initEpoch \text{ THEN } initLeader \text{ ELSE } noLeader]$
            $\land\ epoch = initEpoch$
            $\land\ proposedCmds = \{\}$
            $\land\ proposeRequests = [replica \in replicas \mapsto \{\}]$
            $\land\ proposeResponses =$
                $[cmd \in commands \mapsto [e \in epoches \mapsto \{\}]]$
            $\land\ specPools = [replica \in replicas \mapsto \{\}]$
            $\land\ uncommittedCmds = \langle\rangle$
            $\land\ committedCmds = \langle\rangle$
            $\land\ commitMsgs = [replica \in replicas \mapsto \{\}]$
            $\land\ specExecPrevCmd = [cmd \in commands \mapsto 0]$

$Next \triangleq$
    $\lor\ \exists\, cmd \in (commands \setminus proposedCmds) : Propose(cmd)$
    $\lor\ \exists\, r \in replicas : \exists\, cmd \in proposeRequests[r] :$
        $\text{IF } leader[epoch] = r$
         $\text{THEN } ProcessProposeLeader(r, cmd)$
         $\text{ELSE } ProcessProposeNonLeader(r, cmd)$
    $\lor\ uncommittedCmds \neq \langle\rangle \land Commit$
    $\lor\ \exists\, r \in replicas : \exists\, cmd \in commitMsgs[r] : ProcessCommitMsg(r, cmd)$
    $\lor\ \exists\, l \in replicas : LeaderChange(l)$
```

$$
\begin{array}{l}
\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\langle \mathit{leader},\ \mathit{epoch},\ \mathit{specPools},\ \mathit{proposedCmds},} \\
\qquad\qquad\qquad\qquad\qquad \mathit{proposeRequests},\ \mathit{proposeResponses}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{uncommittedCmds},\ \mathit{committedCmds},\ \mathit{commitMsgs}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{specExecPrevCmd}\rangle
\end{array}
$$

Type Invariants

$$
\begin{array}{l}
\mathit{TypeOK} \triangleq \\
\quad \land\ \mathit{leader} \in [\mathit{epoches} \to (\mathit{replicas} \cup \{\mathit{noLeader}\})] \\
\quad \land\ \mathit{epoch} \in \mathit{epoches} \\
\quad \land\ \mathit{proposedCmds} \subseteq \mathit{commands} \\
\quad \land\ \mathit{proposeRequests} \in [\mathit{replicas} \to \textsc{subset}\ \mathit{commands}] \\
\quad \land\ \mathit{proposeResponses} \in [\mathit{commands} \to [\mathit{epoches} \to \textsc{subset}\ \mathit{replicas}]] \\
\quad \land\ \mathit{specPools} \in [\mathit{replicas} \to \textsc{subset}\ \mathit{commands}] \\
\quad \land\ \mathit{uncommittedCmds} \in \textsc{union}\ \{\mathit{SetToSeqs}(s) : s \in \textsc{subset}\ \mathit{commands}\} \\
\quad \land\ \mathit{committedCmds} \in \textsc{union}\ \{\mathit{SetToSeqs}(s) : s \in \textsc{subset}\ \mathit{commands}\} \\
\quad \land\ \mathit{commitMsgs} \in [\mathit{replicas} \to \textsc{subset}\ \mathit{commands}] \\
\quad \land\ \mathit{specExecPrevCmd} \in [\mathit{commands} \to 0\,..\,\mathit{Cardinality}(\mathit{commands})]
\end{array}
$$

Stability Property

This is the key property of *CURP*:

1. If a command is committed by *CURP*, command will eventually be synced by the back-end protocol.

2. If a command is committed by *CURP*, when the command is synced be the back-end protocol, there will never be a command with the same key between the command and the recorded previous same-key command in the synced sequence.

$$
\begin{array}{l}
\mathit{Stability} \triangleq \\
\quad \forall\, \mathit{cmd} \in \mathit{commands} : \forall\, e \in \mathit{epoches} : \\
\qquad (\land\ \mathit{leader}[e] \in \mathit{proposeResponses}[\mathit{cmd}][e] \\
\qquad\ \ \land\ \mathit{proposeResponses}[\mathit{cmd}][e] \in \mathit{superQuorums}) \Rightarrow \\
\qquad\quad \textsc{let}\ \mathit{idx} \triangleq \mathit{GetIdxInSeq}(\mathit{committedCmds},\ \textsc{lambda}\ t : t = \mathit{cmd}) \\
\qquad\qquad\quad \mathit{prevExecCmds} \triangleq \mathit{SubSeq}(\mathit{committedCmds},\ 1,\ \mathit{idx}) \\
\qquad\quad \textsc{in} \\
\qquad\qquad\quad \land\ \mathit{idx} \neq 0 \\
\qquad\qquad\quad \land\ \mathit{GetIdxInSeq}(\mathit{prevExecCmds},\ \textsc{lambda}\ t : t.\mathit{key} = \mathit{cmd}.\mathit{key}) = \\
\qquad\qquad\qquad \mathit{specExecPrevCmd}[\mathit{cmd}]
\end{array}
$$

$$
\textsc{theorem}\ \mathit{Spec} \Rightarrow \Box\, \mathit{TypeOK} \land \Diamond\, \mathit{Stability}
$$