MODULE *PODCommit*

This specification is the very basic version of POD (Proof of Devotion) from *Nebulas*. In this specification, we have the following assumptions to simplity the basic idea.

- No dumber node.
- No dynasty change.
- No node change or abdication.
- Assume one node only propose one value.
- Assume there is no failure node, and eventually all nodes should be consistent.
- We don't consider the liveness problem.
- We don't consider normal nodes besides validators.

CONSTANT *Validator*, The set of validators
    Majority 1+ n * 2/3 validators

VARIABLES *vrState*, *vrState*[*r*] is the state of validator
    vrPrepared, *vrPrepared*[*r*] is the set of validators from which *r* has received "Prepared" messages for *v*'s proposal
    vrCommitted, *vrCommitted*[*r*] is the set of validators from which *r* has received "vote" messages for *v*'s proposal
    vrFinal, *vrFinal*[*r*] is the final value, which the proposer.
    msgs

In the protocol, processes communicate with one another by sending messages. For simplicity, we represent message passing with the variable *msgs* whose value is the set of all messages that have been sent. A message is sent by adding it to the set *msgs*. An action that, in an implementation, would be enabled by the receipt of a certain message is here enabled by the presence of that message in *msgs*. For simplicity, messages are never removed from *msgs*. This allows a single message to be received by multiple receivers. Receipt of the same message twice is therefore allowed; but in this particular protocol, that shouldn't be a problem.

ASSUME
$\land\ Majority \subseteq \text{SUBSET}\ Validator$
$\land\ \forall\ MS1,\ MS2,\ MS3 \in Majority : MS1 \cap MS2 \cap MS3 \neq \{\}$

All we assume about the set *Majority* of majorities is that any three majorities have non-empty intersection, which makes sure *Majority* is at least 2/3 validators.

$Messages\ \triangleq$

The set of all possible messages. The *ins* field indicates the sender. For "propose" message, the "*ins*" field means she propose a block. Since we do not mind the proposed value, we do not record the proposed value here. The *acc* field indicates the sender of a message.

$[type : \{\text{"propose"}\},\ ins : Validator,\ acc : Validator]$
$\quad \cup$
$[type : \{\text{"prepare"}\},\ ins : Validator,\ acc : Validator]$
$\quad \cup$
$[type : \{\text{"vote"}\},\ ins : Validator,\ acc : Validator]$

$PODTypeOK\ \triangleq$
$\land\ vrState \in [Validator \rightarrow \{\text{"working"}, \text{"prepared"}, \text{"committed"}, \text{"finality"}\}]$
$\land\ vrPrepared \in [Validator \rightarrow \{Validator\}]$
$\land\ vrCommitted \in [Validator \rightarrow \{Validator\}]$
$\land\ vrFinal \in [Validator \rightarrow Validator \cup \{\text{"none"}\}]$

```
    ∧ msgs ⊆ Messages

PODInit ≜   The initial predicate
    ∧ vrState = [v ∈ Validator ↦ "working"]
    ∧ vrPrepared = [v ∈ Validator ↦ {}]
    ∧ vrCommitted = [v ∈ Validator ↦ {}]
    ∧ vrFinal = [v ∈ Validator ↦ "none"]
    ∧ msgs = {}
```

THE ACTIONS

```
Send(m) ≜ msgs' = msgs ∪ {m}
  An action expression that describes the sending of message m.

PreparedSet(set, r) ≜ {m ∈ set : m.acc = r}
CommittedSet(set, r) ≜ {m ∈ set : m.acc = r}
```

Validator ACTIONS

```
ValidatorPropose(r) ≜
    Validator try to propose a block
    ∧ vrState[r] = "working"
    ∧ vrState' = [vrState EXCEPT ![r] = "prepared"]
    ∧ vrPrepared' = [vrPrepared EXCEPT ![r] = {[type ↦ "prepare", ins ↦ r, acc ↦ r]}]
    ∧ Send([type ↦ "propose", ins ↦ r, acc ↦ r])
    ∧ Send([type ↦ "prepare", ins ↦ r, acc ↦ r])
    ∧ UNCHANGED ⟨vrCommitted, vrFinal⟩

ValidatorChooseToCommit ≜
    Validator try to vote a block
    ∧ LET ChooseToCommit(r, v) ≜
              ∧ LET Prepared ≜ {m.ins : m ∈ PreparedSet(vrPrepared[r], v)}
                IN  Prepared ∈ Majority
              ∧ vrState[r] = "prepared"
              ∧ vrState' = [vrState EXCEPT ![r] = "committed"]
              ∧ vrCommitted' = [vrCommitted EXCEPT ![r] = vrCommitted[r] ∪ {[type ↦ "vote", ins ↦ r, acc
              ∧ Send([type ↦ "vote", ins ↦ r, acc ↦ v])
      IN
         ∀ r ∈ Validator, v ∈ Validator : ChooseToCommit(r, v)
    ∧ UNCHANGED ⟨vrPrepared, vrFinal⟩

ValidatorChooseToFinal ≜
    Validator try to final a block.
    ∧ LET ChooseToFinal(r, v) ≜
              ∧ LET Committed ≜ {m.ins : m ∈ CommittedSet(vrCommitted[r], v)}
```

```
            IN  $Committed \in Majority$
          $\land vrState[r] = \text{“committed”}$
          $\land vrState' = [vrState \text{ EXCEPT } ![r] = \text{“finality”}]$
          $\land vrFinal' = [vrFinal \text{ EXCEPT } ![r] = v]$
      IN
         $\forall r \in Validator, v \in Validator : ChooseToFinal(r, v)$
    $\land$ UNCHANGED $\langle vrPrepared, vrCommitted \rangle$
```

```
RECV messages
$RecvPropose(r, v) \triangleq$
     The action when recv a prepare message.
    $\land vrState[r] = \text{“working”}$
    $\land \exists m \in msgs :$
        $\land m.type = \text{“propose”}$
        $\land m.ins = v$
    $\land vrState' = [vrState \text{ EXCEPT } ![r] = \text{“prepared”}]$
    $\land Send([type \mapsto \text{“prepare”}, ins \mapsto r, acc \mapsto v])$
    $\land vrPrepared' = [vrPrepared \text{ EXCEPT } ![r] = \{[type \mapsto \text{“prepare”}, ins \mapsto r, acc \mapsto v]\}]$
    $\land$ UNCHANGED $\langle vrCommitted, vrFinal \rangle$

$RecvPrepare(r, from, v) \triangleq$
    The action when recv a prepare message.
    $\land vrState[r] = \text{“prepared”}$
    $\land \exists m \in msgs :$
        $\land m.type = \text{“prepare”}$
        $\land m.acc = v$
        $\land m.ins = from$
    $\land vrPrepared' = [vrPrepared \text{ EXCEPT } ![r] = vrPrepared[r] \cup \{[type \mapsto \text{“prepare”}, ins \mapsto r, acc \mapsto v]\}]$
    $\land$ UNCHANGED $\langle vrCommitted, vrState, vrFinal \rangle$

$RecvVote(r, from, v) \triangleq$
    The action when recv a vote message.
    $\land vrState[r] = \text{“prepared”}$
    $\land \exists m \in msgs :$
        $\land m.type = \text{“vote”}$
        $\land m.acc = v$
        $\land m.ins = from$
    $\land vrCommitted' = [vrCommitted \text{ EXCEPT } ![r] = vrCommitted[r] \cup \{[type \mapsto \text{“prepare”}, ins \mapsto r, acc \mapsto v]$
    $\land$ UNCHANGED $\langle vrPrepared, vrState \rangle$
```

```
$PODNext \triangleq$
    $\lor \exists r \in Validator : ValidatorPropose(r)$
    $\lor ValidatorChooseToCommit$
    $\lor\ ValidatorChooseToFinal$
    $\lor\ \exists\, r, v \in Validator : RecvPropose(r, v)$
    $\lor\ \exists\, r, from, v \in Validator : \lor\ RecvPrepare(r, from, v)$
                                              $\lor\ RecvVote(r, from, v)$
```

$PODConsistent \triangleq$

A state predicate asserting that two *Validators* have not arrived at conflicting decisions. It is an invariant of the specification. Actually, *PoD* don't need this, so no consistency requirement.

```
  $\land\ \forall\, r1, r2 \in Validator : \neg \land\ vrState[r1] = \text{“aborted”}$
                 $\land\ vrState[r2] = \text{“finality”}$
  $\land$ LET $FinalValidators \triangleq \{r \in Validator :$
                   $vrState[r] = \text{“finality”}\}$
    IN  $\forall\, r1, r2 \in FinalValidators:$
              $vrFinal[r1] = vrFinal[r2]$
```

$$PODSpec \triangleq PODInit \land \Box[PODNext]_{\langle vrState,\, vrPrepared,\, vrCommitted,\, vrFinal \rangle}$$

THEOREM $PODSpec \Rightarrow \Box\,(PODTypeOK \land PODConsistent)$

* Modification History
* Last modified Sat *Jan* 06 20:12:31 *CST* 2018 by *xuepeng*
* Created *Wed Jan* 03 23:52:11 *CST* 2018 by *xuepeng*