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.

EXTENDS *Naturals*, *TLC*

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 "*val*" field means she propose a block. Since we do not mind the proposed value, we do not record the proposed value here. The "sender" field indicates the sender of a message.

$[type : \{\text{"propose"}\},\ val : Validator,\ sender : Validator]$
$\cup$
$[type : \{\text{"prepare"}\},\ val : Validator,\ sender : Validator]$
$\cup$
$[type : \{\text{"vote"}\},\ val : Validator,\ sender : Validator]$

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

$\land\ msgs \subseteq Messages$

$PODInit \triangleq$ The initial predicate

$\land\ vrState = [v \in Validator \mapsto \text{“working”}]$
$\land\ vrPrepared = [v \in Validator \mapsto \{\}]$
$\land\ vrCommitted = [v \in Validator \mapsto \{\}]$
$\land\ vrFinal = [v \in Validator \mapsto \text{“none”}]$
$\land\ msgs = \{\}$
$\land\ Print(\text{“init”}, \text{TRUE})$

THE ACTIONS

$Send(m) \triangleq msgs' = msgs \cup \{m\}$

An action expression that describes the sending of message m.

$PreparedSet(set, r) \triangleq \{m \in set : m.val = r\}$
$CommittedSet(set, r) \triangleq \{m \in set : m.val = r\}$

Validator ACTIONS

$ValidatorPropose(r) \triangleq$

Validator try to propose a block

$\land\ vrState[r] = \text{“working”}$
$\land\ vrState' = [vrState \text{ EXCEPT } ![r] = \text{“prepared”}]$
$\land\ vrPrepared' = [vrPrepared \text{ EXCEPT } ![r] = \{[type \mapsto \text{“prepare”}, val \mapsto r, sender \mapsto r]\}]$
$\land\ msgs' = msgs \cup \{[type \mapsto \text{“propose”}, val \mapsto r, sender \mapsto r],$
$\qquad\qquad [type \mapsto \text{“prepare”}, val \mapsto r, sender \mapsto r]\}$
$\land\ \text{UNCHANGED } \langle vrCommitted, vrFinal\rangle$

$ChooseToCommit(r, v) \triangleq$
$\land\ \text{LET } Prepared \triangleq \{m.sender : m \in PreparedSet(vrPrepared[r], v)\}$
$\quad \text{IN} \quad \land\ Prepared \in Majority$
$\land\ vrState[r] = \text{“prepared”}$
$\land\ vrState' = [vrState \text{ EXCEPT } ![r] = \text{“committed”}]$
$\land\ vrCommitted' = [vrCommitted \text{ EXCEPT } ![r] = vrCommitted[r] \cup$
$\qquad \{[type \mapsto \text{“vote”}, val \mapsto v, sender \mapsto r]\}]$
$\land\ Send([type \mapsto \text{“vote”}, val \mapsto v, sender \mapsto r])$

$ValidatorChooseToCommit \triangleq$

Validator try to vote a block

$\land\ \exists\, r, v \in Validator : ChooseToCommit(r, v)$
$\land\ \text{UNCHANGED } \langle vrPrepared, vrFinal\rangle$

$ValidatorChooseToFinal \triangleq$

Validator try to final a block.

$\land$ LET $ChooseToFinal(r,\ v) \triangleq$
  $\land$ LET $Committed \triangleq \{m.sender : m \in CommittedSet(vrCommitted[r],\ v)\}$
    IN $\land\ Committed \in Majority$
  $\land\ vrState[r] =$ "committed"
  $\land\ vrState' = [vrState$ EXCEPT $![r] =$ "finality"$]$
  $\land\ vrFinal' = [vrFinal$ EXCEPT $![r] = v]$
 IN
  $\exists\, r \in Validator,\ v \in Validator : ChooseToFinal(r,\ v)$
$\land$ UNCHANGED $\langle vrPrepared,\ vrCommitted,\ msgs\rangle$

RECV messages

$RecvPropose(r,\ v) \triangleq$

The action when recv a prepare message.

$\land\ vrState[r] =$ "working"
$\land \exists\, m \in msgs :$
  $\land\ m.type =$ "propose"
  $\land\ m.val = v$
$\land\ vrState' = [vrState$ EXCEPT $![r] =$ "prepared"$]$
$\land\ Send([type \mapsto$ "prepare", $val \mapsto v,\ sender \mapsto r])$
$\land\ vrPrepared' = [vrPrepared$ EXCEPT $![v] = \{[type \mapsto$ "prepare", $val \mapsto v,\ sender \mapsto r]\}]$
$\land$ UNCHANGED $\langle vrCommitted,\ vrFinal\rangle$

$RecvPrepare(r,\ from,\ v) \triangleq$

The action when recv a prepare message.

$\land\ vrState[r] =$ "prepared"
$\land \exists\, m \in msgs :$
  $\land\ m.type =$ "prepare"
  $\land\ m.val = v$
  $\land\ m.sender = from$
$\land\ vrPrepared' = [vrPrepared$ EXCEPT $![r] = vrPrepared[r] \cup$
                $\{[type \mapsto$ "prepare", $val \mapsto v,\ sender \mapsto from]\}]$
$\land$ UNCHANGED $\langle vrCommitted,\ vrState,\ vrFinal,\ msgs\rangle$

$RecvVote(r,\ from,\ v) \triangleq$

The action when recv a vote message.

$(\ \land\ vrState[r] =$ "prepared"
$\land \exists\, m \in msgs :$
  $\land\ m.type =$ "vote"
  $\land\ m.val = v$
  $\land\ m.sender\ = from$
$\land\ vrCommitted' = [vrCommitted$ EXCEPT $![r] = vrCommitted[r] \cup$
                $\{[type \mapsto$ "vote", $val \mapsto v,\ sender \mapsto from]$,
                 $[type \mapsto$ "vote", $val \mapsto v,\ sender \mapsto r]\}]$
$\land\ vrState' = [vrState$ EXCEPT $![r] =$ "committed"$]$

$\land\ Send([type \mapsto \text{“vote”},\ val \mapsto v,\ sender \mapsto r])$
$\land$ UNCHANGED $\langle vrPrepared,\ vrFinal\rangle)$
$\lor\ (\ \land\ vrState[r] = \text{“committed”}$
$\land\ \exists\, m \in msgs :$
$\land\ m.type = \text{“vote”}$
$\land\ m.val = v$
$\land\ m.sender\ = from$
$\land\ vrCommitted' = [vrCommitted$ EXCEPT $![r] = vrCommitted[r] \cup$
$\{[type \mapsto \text{“vote”},\ val \mapsto v,\ sender \mapsto from]\}]$
$\land$ UNCHANGED $\langle vrPrepared,\ vrFinal,\ vrState,\ msgs\rangle)$

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

$PODConsistent\ \triangleq$

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.

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

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

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

---

* Modification History
* Last modified Sat *Jan* 13 19:20:32 *CST* 2018 by *xuepeng*
* Created *Wed Jan* 03 23:52:11 *CST* 2018 by *xuepeng*