## MODULE *PoDCon*

This module specifies the *PoD* consensus algorithm. It is an abstraction and generalization of the *PoD* algorithm described in https://*github.com/freeof*123/*blue_paper*/blob/master/en/*main.pdf*

EXTENDS *Integers*, *FiniteSets*, *Sequences*

Here we import a module which defines the structure of block and chain.

INSTANCE *Block*

---

Validators are the nodes that verify the finality of blocks. We pretend that which validators are honest and which are malicious is specified in advance.

The basic idea is that the honest validators have to execute the *PoD* algorithm, while the malicious ones may try to prevent them with unpredictable actions.

*Validator* is the set of honest validators and *FakeValidator* is the set of malicious or crashed validators.
*ByzQuorum* is the set of $n$ honest validators with at most f fake validators, where $n \geq 2f+1$.
Each byzantine quorum has $3f+1$ validators.

CONSTANTS *Validator*,
*FakeValidator*,
*ByzQuorum*

We define *ByzValidator* to be the set of all real or fake validators.

$$ByzValidator \triangleq Validator \cup FakeValidator$$

Constants input for *TLC* Model:
$Validator \leftarrow \{\text{“}v1\text{”}, \text{“}v2\text{”}, \text{“}v3\text{”}, \text{“}v4\text{”}\}$
$FakeValidator \leftarrow \{\text{“}f1\text{”}\}$
$ByzQuorum \leftarrow \{\{\text{“}v1\text{”}, \text{“}v2\text{”}, \text{“}v3\text{”}, \text{“}f1\text{”}\}, \{\text{“}v4\text{”}, \text{“}v2\text{”}, \text{“}v3\text{”}, \text{“}f1\text{”}\}, \{\text{“}v1\text{”}, \text{“}v4\text{”}, \text{“}v3\text{”}, \text{“}f1\text{”}\},$
$\{\text{“}v1\text{”}, \text{“}v2\text{”}, \text{“}v4\text{”}, \text{“}f1\text{”}\},\{\text{“}v1\text{”}, \text{“}v2\text{”}, \text{“}v3\text{”}, \text{“}v4\text{”}\}\}$

The following are the assumptions about validators and quorums that are needed to ensure satety of the algorithm.

ASSUME $BQA \triangleq$
$\quad \land\ Validator \cap FakeValidator = \{\}$
$\quad \land\ \forall\, Q \in ByzQuorum : Q \subseteq ByzValidator$
$\quad \land\ \forall\, Q1,\ Q2 \in ByzQuorum : Q1 \cap Q2 \cap Validator \neq \{\}$

---

Blocks are the set of blocks. Each block is represented as a record which contains the block *id* (hash) and a pointer to the parent *id* (hash). For brevity, we omit the payload of block.

CONSTANTS *Blocks*

Constants input for *TLC* Model:
$Blocks \leftarrow \{[id \mapsto 1,\ parent \mapsto 0,\ type \mapsto \text{“}normal\text{”}],\ [id \mapsto 2,\ parent \mapsto 1,\ type \mapsto \text{“}normal\text{”}],\ [id \mapsto 3,\ parent \mapsto 2,\ type \mapsto \text{“}norm$

Basic assumption abouth blocks that all block *id* and parent *id* should be natural number.

```
ASSUME ∀ b ∈ Blocks : b ∈ NormalBlock
├──────────────────────────────────────────────────────────────┤
 The length of each epoch
CONSTANT EpochLength

ASSUME EpochLength ∈ Nat
 Constants input for TLC Model:
 EpochLength ← 3
├──────────────────────────────────────────────────────────────┤
 Here we define the set Message of all possible messages.
 round is the finalized round, which is represented by the last finalized block. TBA when there is no finalized one

PathMessage ≜ [type : {"path_vote"}, sender : ByzQuorum, val : Blocks, round : Nat]

PrefixMessage ≜ [type : {"prefix_vote"}, sender : ByzQuorum, val : Blocks, round : Nat]

BlockMessage ≜ [type : {"block_vote"}, sender : ByzQuorum, val : Blocks, round : Nat]

BMessage ≜ PathMessage ∪ PrefixMessage ∪ BlockMessage

 The following lemma is the simple fact about these set of messages.
LEMMA BMessageLemma ≜ ∀ m ∈ BMessage : ∧ (m ∈ PathMessage) ≡ (m.type = "path_vote")
                                       ∧ (m ∈ PrefixMessage) ≡ (m.type = "prefix_vote")

├──────────────────────────────────────────────────────────────┤
 We now give the algorithm.
 --algorithm PoDCon

   variables localBlocks = [v ∈ ByzValidator ↦ {Genesis}],           Local blocks
             finalizedChain  = [v ∈ ByzValidator ↦ ⟨Genesis⟩],        chain that records finalized blocks
             votedPath = [v ∈ ByzValidator ↦ {}],                     voted path in the first round
               prefixPaths =  [v ∈ ByzValidator ↦ {}],         \*all posible prefix paths of a byzvalidator
             votedPrefix = [v ∈ ByzValidator ↦ {}],                   voted prefix in the second round
             votedBlock = [v ∈ ByzValidator ↦ Empty],                 voted block in the final round
             msgs = {} ;                                              all messages

   define
     Here we need some useful operators, and some of them are defined in Block.tla
         Get the set of all elements in seq
       SeqToSet(seq) ≜ {seq[i] : i ∈ 1 .. Len(seq)}

         True for did not vote the path or any path conflicting before. TBA
       DidNotVotePath(v, path) ≜ TRUE LET finalized_blocks ≜ SeqToSet(finalizedChain[v])
                                       IN   ∀ b ∈ path : b ∉ finalized_blocks

   end define ;

     Phase of receiving new blocks
   macro ReceiveNewBlock()begin
```

```
      For test here
    $localBlocks[self] := AddBlocks(Blocks, localBlocks[self])$ ;
end macro ;

  Phase of voting for paht
macro $VoteForPath()$ begin
    with $s = finalizedChain[self][Len(finalizedChain[self])]$,      get the last block in beacon chain as the initiativ
         $t = EndBlock(localBlocks[self])$ do                        get the last block in local blocks as the terminated bloc
         if $IsPrev(s, t, localBlocks[self])$ then                   $IsPrev()$ will return false if $s = t$, which means the vot
            with $path = GetPath(s, t, localBlocks[self])$ do
                if $DidNotVotePath(self, path)$ then
                    $votedPath[self] := path$ ;                      empty the set when go to final height vote pathse
                    $msgs := msgs \cup \{[type \mapsto$ "path_vote", $sender \mapsto self, val \mapsto path, round \mapsto s.id]\}$ ;
                 else
                    skip ;
                end if ;
            end with ;
          else
            skip ;
         end if ;
    end with ;
end macro ;

macro $VoteForPath1()$ begin
    with $endBlock = EndBlock(localBlocks[self])$ do
        if $GetHeight(endBlock, localBlocks[self]) = EpochLength$ then          $TBA$ here
            with $path = GetBackTrace(endBlock, EpochLength, localBlocks[self])$ do
                if $DidNotVotePath(self, path)$ then
                    $votedPath[self] := path$ ;
                    $msgs := msgs \cup \{[type \mapsto$ "path_vote", $sender \mapsto self, val \mapsto path, round \mapsto HeadBlock(p$
                 else
                    skip ;
                end if ;
            end with ;
        else
            skip ;
        end if ;
    end with ;
end macro ;

  Phase of voting for longest common prefix, $TBA$
macro $VoteForCommonPrefix()$ begin
    if $votedPath[self] \neq \{\}$ then
      wait until received paths from at least one byz quorum
        await $\exists\, Q \in ByzQuorum : \land \forall\, v \in (Q \cap Validator) : votedPath[v] \neq \{\}$
                                         $\land\, self \in Q$ ;                     no need here maybe
            with $quorum\_set = \{Q \in ByzQuorum : \land \forall v \in (Q \cap Validator) : votedPath[v] \neq \{\}$
                                                  $\land\ self \in Q\}$ do
                with $all\_prefixs = \{GetPrefix(\{votedPath[v] : v \in (q \cap Validator)\}) : q \in quorum\_set\}$ do
                    choose the longest prefix for honest validators
                    $votedPrefix[self] := LongestPath(all\_prefixs)$ ;
                    $msgs := msgs \cup \{[type \mapsto$ "prefix_vote", $sender \mapsto self, val \mapsto votedPrefix[self], round \mapsto He$
                end with ;
            end with ;
          else
            skip ;
          end if ;
    end macro ;

    macro $PhaseFinalHeightVote()$ begin
        if $votedPath[self] \neq \{\} \land votedPrefix[self] \neq \{\}$ then
          wait until received prefixs from at least one byz quorum
            await $\exists Q \in ByzQuorum : \land \forall v \in (Q \cap Validator) : votedPrefix[v] \neq \{\}$
                                          $\land\ self \in Q$ ;
            with $prefix\_set = \{votedPrefix[v] : v \in ByzValidator\}$ do
                  $votedBlock[self] := TailBlock(LongestPath(prefix\_set))$ ;
                  $msgs := msgs \cup \{[type \mapsto$ "block_vote", $sender \mapsto self, val \mapsto votedBlock[self], round \mapsto Head$
            end with
         else
            skip ;
        end if ;
    end macro ;

    macro $FakingValidator()$ begin

        skip ;
    end macro ;

   We combine these actions into separate process decalrations for validators and fake validators
fair process $v \in Validator$
 begin $vote$:
    while TRUE do
            either
                  $ReceiveNewBlock()$ ;
              or
                  $VoteForPath()$ ;
                    $VoteForPath1()$;
              or
                  $VoteForCommonPrefix()$ ;
              or
                  $PhaseFinalHeightVote()$ ;
            end either ;
    end while ;
      skip;
  end process ;

    Fake validators
  process fv ∈ FakeValidator
  begin fake_vote:
    while TRUE do
        skip ;                   do nothing
    end while ;
  end process ;


end algorithm ;
 BEGIN TRANSLATION
VARIABLES localBlocks, finalizedChain, votedPath, votedPrefix, votedBlock,
          msgs

 define statement
SeqToSet(seq) ≜ {seq[i] : i ∈ 1 . . Len(seq)}


DidNotVotePath(v, path) ≜ TRUE


vars ≜ ⟨localBlocks, finalizedChain, votedPath, votedPrefix, votedBlock,
         msgs⟩

ProcSet ≜ (Validator) ∪ (FakeValidator)

Init ≜  Global variables
         ∧ localBlocks = [v ∈ ByzValidator ↦ {Genesis}]
         ∧ finalizedChain = [v ∈ ByzValidator ↦ ⟨Genesis⟩]
         ∧ votedPath = [v ∈ ByzValidator ↦ {}]
         ∧ votedPrefix = [v ∈ ByzValidator ↦ {}]
         ∧ votedBlock = [v ∈ ByzValidator ↦ Empty]
         ∧ msgs = {}

v(self) ≜ ∧ ∨ ∧ localBlocks′ = [localBlocks EXCEPT ![self] = AddBlocks(Blocks, localBlocks[self])]
              ∧ UNCHANGED ⟨votedPath, votedPrefix, votedBlock, msgs⟩
            ∨ ∧ LET s ≜ finalizedChain[self][Len(finalizedChain[self])]IN
                  LET t ≜ EndBlock(localBlocks[self])IN
                    IF IsPrev(s, t, localBlocks[self])
                       THEN ∧ LET path ≜ GetPath(s, t, localBlocks[self])IN
                                IF DidNotVotePath(self, path)
                                   THEN ∧ votedPath′ = [votedPath EXCEPT ![self] = path]
                                        ∧ msgs′ = (msgs ∪ {[type ↦ "path_vote", sender ↦ self, val ↦
                                   ELSE ∧ TRUE
```

```
                                      $\land$ UNCHANGED $\langle votedPath, msgs \rangle$
                          ELSE  $\land$ TRUE
                                $\land$ UNCHANGED $\langle votedPath, msgs \rangle$
                  $\land$ UNCHANGED $\langle localBlocks, votedPrefix, votedBlock \rangle$
                $\lor$ $\land$ IF $votedPath[self] \neq \{\}$
                       THEN $\land \exists Q \in ByzQuorum : \land \forall v \in (Q \cap Validator) : votedPath[v] \neq \{\}$
                                                         $\land self \in Q$
                            $\land$ LET $quorum\_set \triangleq \{Q \in ByzQuorum : \land \forall v \in (Q \cap Validator) : votedPath[$
                                                                          $\land self \in Q\}$IN
                              LET $all\_prefixs \triangleq \{GetPrefix(\{votedPath[v] : v \in (q \cap Validator)\}) : q \in qu$
                                $\land votedPrefix' = [votedPrefix$ EXCEPT $![self] = LongestPath(all\_prefixs)]$
                                $\land msgs' = (msgs \cup \{[type \mapsto$ "prefix_vote", $sender \mapsto self, val \mapsto votedPref$
                       ELSE $\land$ TRUE
                            $\land$ UNCHANGED $\langle votedPrefix, msgs \rangle$
                  $\land$ UNCHANGED $\langle localBlocks, votedPath, votedBlock \rangle$
                $\lor$ $\land$ IF $votedPath[self] \neq \{\} \land votedPrefix[self] \neq \{\}$
                       THEN $\land \exists Q \in ByzQuorum : \land \forall v \in (Q \cap Validator) : votedPrefix[v] \neq \{\}$
                                                         $\land self \in Q$
                            $\land$ LET $prefix\_set \triangleq \{votedPrefix[v] : v \in ByzValidator\}$IN
                               $\land votedBlock' = [votedBlock$ EXCEPT $![self] = TailBlock(LongestPath(prefix\_$
                               $\land msgs' = (msgs \cup \{[type \mapsto$ "block_vote", $sender \mapsto self, val \mapsto votedBlock'[$
                       ELSE $\land$ TRUE
                            $\land$ UNCHANGED $\langle votedBlock, msgs \rangle$
                  $\land$ UNCHANGED $\langle localBlocks, votedPath, votedPrefix \rangle$
              $\land$ UNCHANGED $finalizedChain$

$fv(self) \triangleq$ $\land$ TRUE
              $\land$ UNCHANGED $\langle localBlocks, finalizedChain, votedPath,$
                                 $votedPrefix, votedBlock, msgs \rangle$

$Next \triangleq (\exists self \in Validator : v(self))$
           $\lor (\exists self \in FakeValidator : fv(self))$

$Spec \triangleq \land Init \land \Box[Next]_{vars}$
         $\land \forall self \in Validator : \mathrm{WF}_{vars}(v(self))$

 END TRANSLATION

|----------------------------------------------------------------------------|
 *********************** Invariants                ***************************
$ChainCorrectness \triangleq \forall i \in Validator : \land localBlocks[i] \subseteq Block$
                                          $\land votedPath[i] \subseteq Blocks$
                                            $\land prefixPaths[i] \subseteq Blocks$

$GenesisInvariants \triangleq \forall i \in ByzValidator : \land Genesis \in localBlocks[i]$
                                               $\land Genesis = finalizedChain[i][1]$
```

```
************************ Properties                          ***************************
Liveness ≜ ∀ i ∈ Validator : ∧ ◇(Blocks = localBlocks[i])
                             ∧ ◇(Blocks = votedPath[i])     for test
                             ∧ ◇(Blocks = votedPrefix[i])   for test
```

\* Modification History
\* Last modified *Wed Jul* 03 11:58:09 *CST* 2019 by *tangzaiyang*
\* Created *Wed Jun* 05 14:48:17 *CST* 2019 by *tangzaiyang*