```
1  ---------- MODULE PoDCon ----------
2  EXTENDS Integers, FiniteSets, Sequences

4  INSTANCE Block
5  ----------
6  CONSTANTS Validator,          The set of honest validators
7            FakeValidator,     The set of malicious or crashed validators
8            ByzQuorum          Set of n honest validators with f fake validators, where n ≥ 2f+1. Each byzant

10 ByzValidator ≜ Validator ∪ FakeValidator

12 ****************************************** Constants                    for TLC Model:**********************
13 Validator ← {"v1", "v2", "v3", "v4"}
14 FakeValidator ← {"f1"}
15 ByzQuorum ← {{"v1", "v2", "v3", "f1"}, {"v4", "v2", "v3", "f1"}, {"v1", "v4", "v3", "f1"}, {"v1", "v2", "v4", "f1"}, {"v1
17 ASSUME BQA ≜ ∧ Validator ∩ FakeValidator = {}
18             ∧ ∀ Q ∈ ByzQuorum : Q ⊆ ByzValidator
19             ∧ ∀ Q1, Q2 ∈ ByzQuorum : Q1 ∩ Q2 ∩ Validator ≠ {}
20 ----------
21 CONSTANTS Blocks

23 Genesis ≜ [id ↦ 1, parent ↦ 0]          Geneis block

25 ASSUME BA ≜ ∀ b ∈ Blocks : b ∈ Block

27 ****************************************** Constants                    for TLC Model:**********************
28 Blocks ← {[id ↦ 1, parent ↦ 0], [id ↦ 2, parent ↦ 1], [id ↦ 3, parent ↦ 2]}

31 ----------
32 Here we define the set Message of all possible messages.

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

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

38 BMessage ≜ PathMessage ∪ PrefixMessage ∪ ....

40 LEMMA BMessageLemma ≜ ∀ m ∈ BMessage : ∧ (m ∈ PathMessage) ≡ (m.type = "path_vote")
41                                        ∧ (m ∈ PrefixMessage) ≡ (m.type = "prefix_vote")

44 ----------
45 --algorithm PoDCon
46   variables localBlocks   = [v ∈ ByzValidator ↦ {Genesis}],          Local chain
47             beaconChain = [v ∈ ByzValidator ↦ ⟨Genesis⟩],           chain that records finalized blocks
48             votedPath = [v ∈ ByzValidator ↦ {}],                    voted path in the first round
49             prefixPaths = [v ∈ ByzValidator ↦ {}],                  all posible prefix paths of a byzvalidator
50             votedPrefix = [v ∈ ByzValidator ↦ {}],                  voted prefix in the second round
51             msgs = {} ;                                             all messages
```

```
define
  Here we need some useful operatos, and some of them are defined in Block.tla
      IsChain(blocks)
      IsPath(blocks)
      Prefix(chains)
      GetPath(s, t, blocks)
      LongestPath(paths)

      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.
      The first block of the path should be finalized which means shoule be in the beaconChain
    DidNotVotePath(v, path) ≜ LET head ≜ HeadBlock(path)
                                  finalized_blocks ≜ SeqToSet(beaconChain[v])
                              IN
                                  ∧ ∀ b ∈ path \ {head} : b ∉ finalized_blocks
                                  ∧ head ∈ 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 = beaconChain[self][Len(beaconChain[self])],    get the last block in beacon chain as the initiative blo
         t = TailBlock(localBlocks[self]) do               get the last block in local blocks as the terminated bl
         if IsPrev(s, t, localBlocks[self]) then           IsPrev() will return false if s = t, which means the vo
           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 ∪ {[type ↦ "path_vote", sender ↦ self, val ↦ path]} ;
                else
                   skip ;
               end if ;
           end with ;
         else
           skip ;
         end if ;
    end with ;
end macro ;
```

```
99      Phase of voting for longest common prefix, TBA
100   macro VoteForCommonPrefix()begin
101     if votedPath[self] ≠ {} then
102       wait until
103         await ∃ Q ∈ ByzQuorum : ∧∀ v ∈ (Q ∩ Validator) : votedPath[v] ≠ {}
104                                 ∧ self ∈ Q ;                    may not to have this
105         with quorum_set = {Q ∈ ByzQuorum : ∧∀ v ∈ (Q ∩ Validator) : votedPath[v] ≠ {}
106                                            ∧ self ∈ Q} do
107           with all_prefixs = {GetPrefix({votedPath[v] : v ∈ (q ∩ Validator)}) : q ∈ quorum_set} do
108             votedPrefix[self] := LongestPath(all_prefixs) ;
109             msgs := msgs ∪ {[type ↦ "prefix_vote", sender ↦ self, val ↦ votedPrefix[self]]} ;
110           end with ;
111         end with ;
112       else
113         skip ;
114       end if ;
115   end macro ;

118   macro PhaseFinalHeightVote()begin
119
120   end macro ;

122   macro FakingValidator()begin
123
124     skip ;
125   end macro ;

127   We combine these actions into separate process decalrations for validators and fake validators
128 fair process v ∈ Validator
129  begin vote:
130   while TRUE do
131           either
132                 ReceiveNewBlock() ;
133            or
134                 VoteForPath() ;
135            or
136                 VoteForCommonPrefix() ;
137             or
138                PhaseFinalHeightVote();
139             or
140                skip;
141           end either ;
142   end while ;
143    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, beaconChain, votedPath, prefixPaths, votedPrefix, msgs

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


DidNotVotePath(v, path) ≜ LET head ≜ HeadBlock(path)
                              finalized_blocks ≜ SeqToSet(beaconChain[v])
                          IN
                              ∧ ∀ b ∈ path \ {head} : b ∉ finalized_blocks
                              ∧ head ∈ finalized_blocks


vars ≜ ⟨localBlocks, beaconChain, votedPath, prefixPaths, votedPrefix,
        msgs⟩

ProcSet ≜ (Validator) ∪ (FakeValidator)

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

v(self) ≜  ∧ ∨ ∧ localBlocks' = [localBlocks EXCEPT ![self] = AddBlocks(Blocks, localBlocks[self])]
              ∧ UNCHANGED ⟨votedPath, votedPrefix, msgs⟩
           ∨ ∧ LET s ≜ beaconChain[self][Len(beaconChain[self])]IN
                 LET t ≜ TailBlock(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
                                              ∧ UNCHANGED ⟨votedPath, msgs⟩
                                ELSE  ∧ TRUE
                                      ∧ UNCHANGED ⟨votedPath, msgs⟩
                          ∧ UNCHANGED ⟨localBlocks, votedPrefix⟩
                        ∨ ∧ IF votedPath[self] ≠ {}
                                THEN  ∧ ∃ Q ∈ ByzQuorum : ∧ ∀ v  ∈ (Q ∩ Validator) : votedPath[v] ≠ {}
                                                          ∧ self ∈ Q
                                      ∧ LET quorum_set ≜ {Q ∈ ByzQuorum : ∧ ∀ v  ∈ (Q ∩ Validator) : votedPath[
                                                                          ∧ self ∈ Q}IN
                                          LET all_prefixs ≜ {GetPrefix({votedPath[v] : v ∈ (q ∩ Validator)}) : q ∈ qu
                                            ∧ votedPrefix′ = [votedPrefix EXCEPT ![self]  = LongestPath(all_prefixs)]
                                            ∧ msgs′ = (msgs ∪ {[type ↦ "prefix_vote", sender ↦ self, val ↦ votedPrefi
                                ELSE  ∧ TRUE
                                      ∧ UNCHANGED ⟨votedPrefix, msgs⟩
                          ∧ UNCHANGED ⟨localBlocks, votedPath⟩
                    ∧ UNCHANGED ⟨beaconChain, prefixPaths⟩

fv(self) ≜  ∧ TRUE
            ∧ UNCHANGED ⟨localBlocks, beaconChain, votedPath, prefixPaths,
                         votedPrefix, msgs⟩

Next ≜ (∃ self ∈ Validator : v(self))
       ∨ (∃ self ∈ FakeValidator : fv(self))

Spec ≜  ∧ Init ∧ □[Next]_vars
        ∧ ∀ self ∈ Validator : WF_vars(v(self))

END TRANSLATION

********** Invariants **********
ChainCorrectness ≜  ∀ i ∈ Validator :  ∧ localBlocks[i] ⊆ Blocks
                                       ∧ votedPath[i] ⊆ Blocks
                                         ∧ prefixPaths[i] ⊆ Blocks

GenesisInvariants ≜ ∀ i ∈ ByzValidator :  ∧ Genesis ∈ localBlocks[i]
                                          ∧ Genesis = beaconChain[i][1]


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

* Modification History
* Last modified *Mon Jun* 17 14:35:07 *CST* 2019 by *tangzaiyang*
* Created *Wed Jun* 05 14:48:17 *CST* 2019 by *tangzaiyang*