```
MODULE PoDCon
EXTENDS Integers, FiniteSets, Sequences

CONSTANTS Blocks,                 Given blocks
          Genesis,                Genesis block definition
          Validator,              The set of honest validators
          FakeValidator,          The set of malicious or crashed validators
          ByzQuorum               Set of n honest validators with f fake validators, where n ≥ 2f+1. Each byzan

Height ≜ Nat                      Block height

INSTANCE BlockChain WITH Nodes ← Blocks, Genesis ← Genesis

--algorithm PoDCon
  variables localBlocks   = [v ∈ Validator ↦ {}],          Local chain
            beaconChain = [v ∈ Validator ↦ ⟨Genesis⟩],     chain that records status of blocks
            votedPath = [v ∈ Validator ↦ {}],              voted path in the first round
            prefixPaths = [v ∈ Validator ↦ {}],            all posible prefix paths of a validator
            msgs = {} ;                                    all messages

  define
    Here we define some useful operatos, and some of them are defined in BlockChain.tla
      IsChain(blocks)
      IsPath(blocks)
      Prefix(chains)
      GetPath(s, t, blocks)                \*Get a path with a given source block and target block, TBA

    DidNotVotePath(v, path) ≜ TRUE         Did not vote the path or any path conflicting before.
    GetLonestPath(chains) ≜ Head(chains)   Get the lonest one of a set of given paths. TBA
  end define ;

  macro Genesis()begin
      localBlocks[self] := localBlocks[self]; \* ∪ {[id : 0, parent : "NULL"]};
    beaconChain[self] := Append(beaconChain[self], Genesis) ;
  end macro ;

  macro ReceiveNewBlock()begin
      For test here
    localBlocks[self] := Blocks ;
      PrintT(localBlocks[self]);
  end macro ;


  macro VoteForPath()begin
    with s = beaconChain[self][Len(beaconChain[self])],   get the last block in beacon chain as the initiative block
```

```
            t = Tail(localBlocks[self]) do                    get the last block in local blocks as the terminated bloc
          if IsPrev(s, t, localBlocks[self]) then
            with path = GetPath(s, t, localBlocks[self]) do
              when ∧ IsPath(path)
                   ∧ DidNotVotePath(self, path) ;
              votedPath[self] := path ;
              msgs := msgs ∪ {[type ↦ "path_vote", sender ↦ self, val ↦ path]} ;
            end with ;
          else
            skip ;
          end if ;
        end with ;
    end macro ;

    macro VoteForCommonPrefix()begin
        with Q ∈ ByzQuorum do
            with receivedPaths = {votedPath[v] : v ∈ Q} do
                if IsChain(votedPath[q])??
              prefixPaths[self] := prefixPaths[self] ∪ Prefix(receivedPaths) ;
          end with ;
        end with ;
        msgs := msgs ∪ {[type ↦ "prefix_vote", sender ↦ self, val ↦ GetLonestPath(prefixPaths[self])]} ;
    end macro ;

    macro PhaseFinalHeightVote()begin

    end macro ;

    macro FakingValidator()begin

        skip ;
    end macro ;

  We combine these actions into separate process decalrations for validators and fake validators
process v ∈ Validator
begin vote:
    Genesis() ;
    while TRUE do
        either
            VoteForPath();
          or
            VoteForCommonPrefix();
          or
            PhaseFinalHeightVote();
          or
            skip;
```

```
96                  end either;
97          end while; \
98          For test here

100       ReceiveNewBlock() ;
101         VoteForPath();
102         skip;
103     end process ;

105     Fake validators
106     process fv ∈ FakeValidator
107     begin fake_vote:
108       skip ;
109         ReceiveNewBlock();
110         while TRUE do
111               FakingValidator();
112         end while;
113     end process ;


116 end algorithm ;
117   BEGIN TRANSLATION
118 VARIABLES localBlocks, beaconChain, votedPath, prefixPaths, msgs, pc

120   define statement
121 DidNotVotePath(v, path) ≜ TRUE
122 GetLonestPath(chains) ≜ Head(chains)


125 vars ≜ ⟨localBlocks, beaconChain, votedPath, prefixPaths, msgs, pc⟩

127 ProcSet ≜ (Validator) ∪ (FakeValidator)

129 Init ≜   Global variables
130           ∧ localBlocks   = [v ∈ Validator ↦ {}]
131           ∧ beaconChain = [v ∈ Validator ↦ ⟨Genesis⟩]
132           ∧ votedPath = [v ∈ Validator ↦ {}]
133           ∧ prefixPaths = [v ∈ Validator ↦ {}]
134           ∧ msgs = {}
135           ∧ pc = [self ∈ ProcSet ↦ CASE self ∈ Validator → "vote"
136                                      □   self ∈ FakeValidator → "fake_vote"]

138 vote(self) ≜  ∧ pc[self] = "vote"
139               ∧ beaconChain' = [beaconChain EXCEPT ![self] = Append(beaconChain[self], Genesis)]
140               ∧ localBlocks'   = [localBlocks EXCEPT ![self]   = Blocks]
141               ∧ pc' = [pc EXCEPT ![self] = "Done"]
142               ∧ UNCHANGED ⟨votedPath, prefixPaths, msgs⟩

144 v(self) ≜ vote(self)
```

```
146 fake_vote(self) ≜  ∧ pc[self] = "fake_vote"
147                    ∧ TRUE
148                    ∧ pc' = [pc EXCEPT ![self] = "Done"]
149                    ∧ UNCHANGED ⟨localBlocks, beaconChain, votedPath,
150                                 prefixPaths, msgs⟩

152 fv(self) ≜ fake_vote(self)

154 Next ≜ (∃ self ∈ Validator : v(self))
155         ∨ (∃ self ∈ FakeValidator : fv(self))
156         ∨  Disjunct to prevent deadlock on termination
157           ((∀ self ∈ ProcSet : pc[self] = "Done") ∧ UNCHANGED vars)

159 Spec ≜ Init ∧ □[Next]_vars

161 Termination ≜ ◇(∀ self ∈ ProcSet : pc[self] = "Done")

163  END TRANSLATION

165 ├──────────────────────────────────────────────────────────────────────┤

167 TypeOK ≜  ∧ ∀ i ∈ Validator :  ∧ localBlocks[i] ⊆ Blocks
168                               ∧ votedPath[i] ⊆ Blocks
169                               ∧ prefixPaths[i] ⊆ Blocks
170                                ∧ ∃ b ∈ beaconChain[i] : b = Genesis
171             ∧ ∀ j ∈ Validator : votedPath[j] = {"1"}

173 └──────────────────────────────────────────────────────────────────────┘
```

* Modification History
* Last modified *Wed Jun* 05 20:25:09 *CST* 2019 by *tangzaiyang*
* Created *Wed Jun* 05 14:48:17 *CST* 2019 by *tangzaiyang*