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 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 ← {"$v1$", "$v2$", "$v3$", "$v4$"}
FakeValidator ← {"$f1$"}
ByzQuorum ← {{"$v1$", "$v2$", "$v3$", "$f1$"}, {"$v4$", "$v2$", "$v3$", "$f1$"}, {"$v1$", "$v4$", "$v3$", "$f1$"}, {"$v1$", "$v2$", "$v4$", "$f1$"},{"$v1$", "$v2$", "$v3$", "$v4$"}}

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

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

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 ← {[*id* ↦ 1, *parent* ↦ 0], [*id* ↦ 2, *parent* ↦ 1], [*id* ↦ 3, *parent* ↦ 2]}

Here we define the following record as *Genesis* block.

```
49  Genesis ≜ [id ↦ 1, parent ↦ 0]

51  Basic assumption abouth blocks that all block id and parent id should be natural number.
52  ASSUME BA ≜ ∀ b ∈ Blocks : b ∈ Block

54 ├──────────────────────────────────────────────────────────────────────┤
55  Here we define the set Message of all possible messages.
56  fr is the finalized round, which is represented by the last finalized block. TBA when there is no finalized one

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

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

62  BMessage ≜ PathMessage ∪ PrefixMessage ∪ ....

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

69 ├──────────────────────────────────────────────────────────────────────┤
70  We now give the algorithm.
71  --algorithm PoDCon
72    variables localBlocks   = [v ∈ ByzValidator ↦ {Genesis}],          Local chain
73              beaconChain = [v ∈ ByzValidator ↦ ⟨Genesis⟩],            chain that records finalized blocks
74              votedPath = [v ∈ ByzValidator ↦ {}],                  voted path in the first round
75              prefixPaths = [v ∈ ByzValidator ↦ {}],                all posible prefix paths of a byzvalidator
76              votedPrefix = [v ∈ ByzValidator ↦ {}],                voted prefix in the second round
77              msgs = {} ;                                           all messages

79    define
80      Here we need some useful operatos, and some of them are defined in Block.tla
81          IsChain(blocks)
82          IsPath(blocks)
83          Prefix(chains)
84          GetPath(s, t, blocks)
85          LongestPath(paths)

87          Get the set of all elements in seq
88        SeqToSet(seq) ≜ {seq[i] : i ∈ 1 .. Len(seq)}

90          True for did not vote the path or any path conflicting before.
91          The first block of the path should be finalized which means shoule be in the beaconChain
92        DidNotVotePath(v, path) ≜ LET head ≜ HeadBlock(path)
93                                       finalized_blocks ≜ SeqToSet(beaconChain[v])
94                                 IN
95                                       ∧ ∀ b    ∈ path \ {head} : b ∉ finalized_blocks
96                                       ∧ head ∈ finalized_blocks
97    end define ;
```

```
99    Phase of receiving new blocks
100   macro ReceiveNewBlock()begin
101       For test here
102       localBlocks[self] := AddBlocks(Blocks, localBlocks[self]) ;
103   end macro ;

105   Phase of voting for paht
106   macro VoteForPath()begin
107       with s = beaconChain[self][Len(beaconChain[self])],    get the last block in beacon chain as the initiative blo
108            t = TailBlock(localBlocks[self]) do               get the last block in local blocks as the terminated bl
109            if IsPrev(s, t, localBlocks[self]) then           IsPrev() will return false if s = t, which means the vo
110               with path = GetPath(s, t, localBlocks[self]) do
111                  if DidNotVotePath(self, path) then
112                     votedPath[self] := path ;                  empty the set when go to final height vote pathse
113                     msgs := msgs ∪ {[type ↦ "path_vote", sender ↦ self, val ↦ path, fr ↦ s.id]} ;
114                  else
115                     skip ;
116                  end if ;
117               end with ;
118            else
119               skip ;
120            end if ;
121       end with ;
122   end macro ;

125     Phase of voting for longest common prefix, TBA
126   macro VoteForCommonPrefix()begin
127       if votedPath[self] ≠ {} then
128         wait until
129           await ∃ Q ∈ ByzQuorum : ∧∀ v ∈ (Q ∩ Validator) : votedPath[v] ≠ {}
130                                   ∧ self ∈ Q ;                  may not to have this
131           with quorum_set = {Q ∈ ByzQuorum : ∧∀ v ∈ (Q ∩ Validator) : votedPath[v] ≠ {}
132                                              ∧ self ∈ Q} do
133              with all_prefixs = {GetPrefix({votedPath[v] : v ∈ (q ∩ Validator)}) : q ∈ quorum_set} do
134                 votedPrefix[self] := LongestPath(all_prefixs) ;
135                 msgs := msgs ∪ {[type ↦ "prefix_vote", sender ↦ self, val ↦ votedPrefix[self], fr ↦ HeadB
136              end with ;
137           end with ;
138         else
139           skip ;
140         end if ;
141   end macro ;

143   macro PhaseFinalHeightVote()begin
144       if votedPath[self] ≠ {} ∧ votedPrefix[self] ≠ {} then
```

```
145             localBlocks[self] := AddBlocks(votedPrefix[self], localBlocks[self])
146          else
147             skip;
148          end if ;
149       end macro ;

151       macro FakingValidator()begin

153          skip;
154       end macro ;

156       We combine these actions into separate process decalrations for validators and fake validators
157    fair process v ∈ Validator
158     begin vote:
159       while TRUE do
160                either
161                      ReceiveNewBlock();
162                  or
163                      VoteForPath();
164                  or
165                      VoteForCommonPrefix();
166                    or
167                        PhaseFinalHeightVote();
168                  or
169                     skip;
170                end either ;
171       end while ;
172         skip;
173     end process ;

175       Fake validators
176     process fv ∈ FakeValidator
177     begin fake_vote:
178       while TRUE do
179          skip;              do nothing
180       end while ;
181     end process ;


184 end algorithm ;
185   BEGIN TRANSLATION
186 VARIABLES localBlocks, beaconChain, votedPath, prefixPaths, votedPrefix, msgs

188   define statement
189 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 ↦ votedPref
                 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 Tue Jun 18 11:53:08 CST 2019 by tangzaiyang
\* Created Wed Jun 05 14:48:17 CST 2019 by tangzaiyang
```