```
┌──────────────────────── MODULE Block ────────────────────────┐
LOCAL INSTANCE TLC            For Assert()
LOCAL INSTANCE FiniteSets     For Cardinality()
LOCAL INSTANCE Sequences      For Len()
LOCAL INSTANCE Integers       For 1 .. n

  In this module we define the structure of blocks, then we give some useful operators.

Block ≜ [id : Nat, parent : Nat, type : {"normal", "finality"}]

NormalBlock ≜ [id : Nat, parent : Nat, type : {"normal"}]

FinalityBlock ≜ [id : Nat, parent : Nat, type : {"finality"}]

  Genesis block
Genesis ≜ [id ↦ 1, parent ↦ 0, type ↦ "normal"]

  Finalized block without any height finality, which may be caused by time out.
Empty ≜ [id : {0}, parent : Nat, type : {"finality"}]

  Basic axiom for block
AXIOM BA ≜   ∧ NormalBlock ⊆ Block
             ∧ FinalityBlock ⊆ Block
             ∧ Genesis ∈ NormalBlock
             ∧ Empty ∈ FinalityBlock
  For test
  {[id ↦ 1, parent ↦ 0, type ↦ "normal"], [id ↦ 2, parent ↦ 1, type ↦ "normal"], [id ↦ 3, parent ↦ 2, type ↦ "normal"], [id ↦ 4
  {[id ↦ 1, parent ↦ 0, type ↦ "normal"], [id ↦ 2, parent ↦ 1, type ↦ "normal"], [id ↦ 3, parent ↦ 2, type ↦ "normal"], [id ↦ 5

├──────────────────────────────────────────────────────────────┤
  Useful operators

  True for genesis block
IsGenesis(b) ≜ b = Genesis

  True for empty finality block
IsEmpty(b) ≜ b ∈ Empty

  Determine whether the given block is legal.
LegalBlock(b) ≜ ∧ b.id ≠ 0
                ∧ ∨ ∧ b ∈ NormalBlock
                    ∧ b.id ≠ b.parent
                  ∨ ∧ b ∈ FinalityBlock
                    ∧ TRUE      maybe here need some requirements

  Determine wheter b1 and b2 are equivalent.
Equal(b1, b2) ≜ ∨ ∧ b1 ∈ NormalBlock
                  ∧ b2 ∈ NormalBlock
```

```
                          ∧ b1.id = b2.id
                         for finality blocks, the id can be same
                       ∨ ∧ b1 ∈ FinalityBlock
                         ∧ b2 ∈ FinalityBlock
                         ∧ b1.id = b2.id
                         ∧ b1.parent = b2.parent

 Note that Equal(b1, b2) = TRUE is not equivalent to b1 = b2, and we give the following trivial axioms
AXIOM NormalBlockEquivalency ≜ ∀ b1, b2 ∈ Block : b1 = b2 ⇒ Equal(b1, b2)

AXIOM FinalityBlockEquivalency ≜ ∀ b1, b2 ∈ Block : b1 = b2 ≡ Equal(b1, b2)


 Add new block to local blocks. Do nothing if there are same blocks or conflicting blocks
AddBlock(b, blocks) ≜ IF ¬LegalBlock(b) THEN Assert(FALSE, "Illegal block!")
                        Do nothing, if the given set has same block.
                       ELSE IF ∃ tb ∈ blocks : Equal(b, tb) THEN Print("Conflicting block!", blocks)
                            ELSE blocks ∪ {b}

 Add a set of blocks to local blocks
AddBlocks(bs, blocks) ≜ IF ∃ b ∈ bs : ¬LegalBlock(b) THEN Assert(FALSE, "Illegal block!")
                         ELSE LET repeated_set ≜ {b ∈ bs : ∃ tb ∈ blocks : Equal(b, tb)}IN
                              blocks ∪ (bs \ (repeated_set))


 True for the blocks have at least one fork.
HasFork(blocks) ≜ ∃ b1 ∈ blocks : ∃ b2 ∈ blocks \ {b1} : b1.parent = b2.parent

 Determine whether the given blocks is a tree, which has a root block
IsTree(blocks) ≜ LET tree ≜ {} ∪ blocksIN
                  IF tree = {} ∨ ∃ fb ∈ tree : ¬LegalBlock(fb) THEN FALSE
                   ELSE IF Cardinality(tree) = 1 THEN TRUE
                        ELSE IF ∃ b1 ∈ tree : ∃ b2 ∈ tree \ {b1} : Equal(b1, b2) THEN Assert(FALSE, "Equ
                              Each block in the tree should have a parent in the tree except the root block
                             ELSE IF ∃ root ∈ tree : ∧ ∀ other ∈ tree \ {root} : ∧ other.id ≠ root.parent
                                                                                 ∧ other.parent ∈ {b.id :
                                                     ∧ root.parent ∉ {b.id    : b ∈ tree}
                                                                    THEN TRUE
                                   ELSE FALSE

 Determine whether the given blocks is a path, which has no fork
IsPath(blocks) ≜ ∧ IsTree(blocks)
                 ∧ ¬HasFork(blocks)


 Simple axioms of path
AXIOM PathProperty1 ≜ ∀ blocks : ∧ IsFiniteSet(blocks)
                                 ∧ IsPath(blocks)
```

```
                                   ⇒ IsTree(blocks)

 Determine whether there is path which starts from s to t
HasPath(s, t, blocks) ≜
    LET F[m ∈ blocks] ≜  True if m is a child of s
        IF m = s THEN TRUE
         ELSE IF ∀ b ∈ blocks : b.id ≠ m.parent THEN FALSE
         ELSE LET pm ≜ CHOOSE b ∈ blocks : b.id = m.parent
              IN  F[pm]
    IN  F[t]

 Here we give another no-recursive version.
 HasPath(s, t, blocks) ≜ ∃ path ∈ SUBSET blocks : ∧ s ∈ path
                                                  ∧ t ∈ path
                                                  ∧ IsPath(path)


 Return the head of a given path
HeadBlock(blocks) ≜ IF Cardinality(blocks) = 1 THEN CHOOSE b ∈ blocks : LegalBlock(b)
                     ELSE IF IsPath(blocks) THEN CHOOSE head ∈ blocks : ∧ IsPath(blocks \ {head})
                                                                        ∧ ∀ b ∈ blocks : head.parent ≠
                          ELSE  Assert(FALSE, "Set is not a path")

 Return the tail of a given path
TailBlock(blocks) ≜ IF Cardinality(blocks) = 1 THEN CHOOSE b ∈ blocks : LegalBlock(b)
                     ELSE IF IsPath(blocks) THEN CHOOSE t ∈ blocks : ∀ b ∈ blocks : b.parent ≠ t.id
                          ELSE  Assert(FALSE, "Set is not a path")

 Return a path of given source and terminated blocks
GetPath(s, t, blocks) ≜ IF ¬HasPath(s, t, blocks) THEN Assert(FALSE, "No path")
                         ELSE LET F[m ∈ blocks] ≜
                                  IF m = s THEN {s}
                                   ELSE LET pm ≜ CHOOSE b ∈ blocks : b.id = m.parent
                                            IN  F[pm] ∪ {m}
                              IN  F[t]

 Here we give another no-recursive version.
 GetPath(s, t, blocks) ≜ IF ¬HasPath(s, t, blocks) THEN Assert(FALSE, "No path")
                    ELSE LET all ≜ SUBSET blocksIN
                     CHOOSE path ∈ all : ∧ IsPath(path)
                                         ∧ s ∈ path
                                         ∧ t ∈ path
                                         ∧ HeadBlock(path) = s
                                         ∧ TailBlock(path) = t
```

```
Return the root of a given tree
RootBlock(blocks) ≜ IF Cardinality(blocks) = 1 THEN CHOOSE b ∈ blocks : LegalBlock(b)
                      ELSE IF IsTree(blocks) THEN CHOOSE root ∈ blocks :  ∧ ¬IsTree(blocks \ {root})
                                                                          ∧ ∀ b ∈ blocks : root.parent ≠ b.i
                            ELSE  Assert(FALSE, "Set is not a tree")

Return the height of a given block
GetHeight(b, blocks) ≜ IF b ∉ blocks ∨ ¬LegalBlock(b) THEN Assert(FALSE, "Illegal block")
                        ELSE LET path ≜ GetPath(RootBlock(blocks), b, blocks)IN
                             Cardinality(path)

Return the end of a given tree
EndBlock(blocks) ≜ IF Cardinality(blocks) = 1 THEN CHOOSE b ∈ blocks : LegalBlock(b)
                     ELSE CHOOSE t ∈ blocks : ∧ IsTree(blocks \ {t})
                                              ∧ ∀ t2 ∈ blocks : ∨ ¬IsTree(blocks \ {t2})
                                                                ∨ ∧ IsTree(blocks \ {t2})
                                                                  ∧ ∨ GetHeight(t, blocks) > GetHeight(t
                                                                    ∨ ∧ GetHeight(t, blocks) = GetHeigh
                                                                      ∧ t.id ≤ t2.id
                                            ∧ TRUE \ *choose the end block with longest path
                                            from root
                                          ∧ TRUE \ *choose the end block with lowest id

Simple axioms of path
AXIOM PathProperty2 ≜ ∀ blocks : ∧ IsFiniteSet(blocks)
                                 ∧ IsPath(blocks)
                                    ⇒ ∧ HeadBlock(blocks) = RootBlock(blocks)
                                      ∧ TailBlock(blocks)  = EndBlock(blocks)


Return the parent block of a given block
GetParent(b, blocks) ≜ IF ∧ b.parent ∈ {tmp.id : tmp ∈ blocks}
                          ∧ b ∈ blocks
                        THEN
                           CHOOSE pb ∈ blocks : pb.id = b.parent
                        ELSE  Assert(FALSE, "No parent")

Get the back trace from a block b with n length
GetBackTrace(b, n, blocks) ≜
    LET F[m ∈ 0 .. n] ≜
        IF m = 1 THEN {b}
         ELSE LET secondblock ≜ HeadBlock(F[m − 1])
              IN
                IF ∀ block ∈ blocks : block.id ≠ secondblock.parent THEN Assert(FALSE, "No trace")
```

```
                    ELSE
                      LET firstblock ≜ CHOOSE block ∈ blocks : block.id = secondblock.parent
                      IN  {firstblock} ∪ F[m − 1]
   IN   F[n]


 Return the longest path
LongestPath(paths) ≜ CHOOSE longest ∈ paths : ∀ path ∈ paths : ∧ Cardinality(longest) ≥ Cardinality(path)
                                                                ∧ IsPath(tmpPath)

 True for path1 is the prefix of path2
IsPrefix(path1, path2) ≜ ∧ IsPath(path1)
                         ∧ IsPath(path2)
                         ∧ path1 ⊆ path2
                         ∧ HeadBlock(path1) = HeadBlock(path2)            may not need this

 Return the longest common prefix of given paths
GetPrefix(paths) ≜ IF ∃ p1, p2 ∈ paths : ∧ Cardinality(p1 ∩ p2) = 0
                                        ∧ HeadBlock(p1) ≠ HeadBlock(p2)
                                   THEN Print("No intersection", {})
                   ELSE  LET prefix ≜ {intersection ∈ (UNION paths) :  ∀ path ∈ paths : intersection ∈ pat
                                IN
                         IF IsPath(prefix) THEN prefix
                          ELSE  Print("No prefix", {})


 Determine whether the given block s is ancestor of t
IsPrev(s, t, blocks) ≜
   LET F[m ∈ blocks] ≜
        IF m = s THEN TRUE
         ELSE  IF ∀ b ∈ blocks : b.id ≠ m.parent THEN FALSE
         ELSE  LET pm ≜ CHOOSE b ∈ blocks : b.id = m.parent
                IN   F[pm]
   IN   F[t]
 Here we give another no-recursive version.
 IsPrev(s, t, blocks) ≜ LET path_set ≜ {sub_blocks_set ∈ (SUBSET blocks) \ {{}} : IsPath(sub_blocks_set)}IN
                        ∃ path ∈ path_set :  ∧ HeadBlock(path) = s
                                             ∧ TailBlock(path) = t
                                             ∧ s ≠ t


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

* Modification History
* Last modified *Wed Jul* 03 11:27:54 *CST* 2019 by *tangzaiyang*
* Created *Thu Jun* 06 11:21:13 *CST* 2019 by *tangzaiyang*