```
┌──────────────────────────── MODULE Block ────────────────────────────┐

LOCAL INSTANCE TLC              For Assert()
LOCAL INSTANCE FiniteSets       For Cardinality()
LOCAL INSTANCE Sequences        For Len()
LOCAL INSTANCE Integers         For 1 . . n

Block ≜ [id : Nat, parent : Nat]          Genesis block: [id:1, parent:0]

 For test
 {[id ↦ 1, parent ↦ 0], [id ↦ 2, parent ↦ 1], [id ↦ 3, parent ↦ 2], [id ↦ 4, parent ↦ 3]}
 {[id ↦ 1, parent ↦ 0], [id ↦ 2, parent ↦ 1], [id ↦ 3, parent ↦ 2], [id ↦ 5, parent ↦ 3], [id ↦ 6, parent ↦ 5]}

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

LegalBlock(b) ≜ ∧ b ∈ Block
                ∧ b.id ≠ b.parent

Equal(b1, b2) ≜ ∧ LegalBlock(b1)
                ∧ LegalBlock(b2)
                ∧ b1.id = b2.id
                ∧ b1.parent = b2.parent

 Add new block to local blocks. Do nothing if there is a same block or conflicting block
AddBlock(b, blocks) ≜ IF ¬LegalBlock(b) THEN Assert(FALSE, “Illegal block!”)
                       Do nothing, if the given set has same block.
                      ELSE IF ∃ tmpBlock ∈ blocks : tmpBlock.id = b.id THEN Print(“Conflicting block!”, b
                           ELSE blocks ∪ {b}

 Add new blocks to local blocks
AddBlocks(bs, blocks) ≜ IF ∃ b ∈ bs : ¬LegalBlock(b) THEN Assert(FALSE, “Illegal block!”)
                        ELSE LET repeated_set ≜ {b ∈ bs : ∃ tmpBlock ∈ blocks : tmpBlock.id = b.id}IN
                             blocks ∪ (bs \ (repeated_set))


 Determine whether the given blocks is a path
IsPath(blocks) ≜ LET path ≜ {} ∪ blocksIN
                   IF path = {} ∨ ∃ fb ∈ path : ¬LegalBlock(fb) THEN FALSE
                    ELSE IF Cardinality(path) = 1 THEN TRUE
                         ELSE IF ∃ b1 ∈ path : ∃ b2 ∈ path \ {b1} : ∨ b1.parent = b2.parent
                                                                   ∨ b1.id = b2.id
                                                                     THEN FALSE
                              ELSE IF ∃ head ∈ path : ∧ ∀ other ∈ path \ {head} : ∧ other.id ≠ head.pa
                                                                                 ∧ other.parent ∈ {b.i
                                                      ∧ head.parent ∉ {b.id    : b ∈ path}
                                                                           THEN TRUE
```

```
                                    ELSE  FALSE


IF  ∃ b ∈ blocks :  ∨ ¬LegalBlock(b)
                    ∨ ∃ b1 ∈ blocks \ {b} :  ∨ b.id = b1.id
                                             ∨ b.parent = b1.parent
                        THEN FALSE
     ELSE  IF  Cardinality(blocks) = 1  THEN TRUE        \* single block is a path
       ELSE  -IF
         ELSE  LET id_set ≜ {b2.id : b2 ∈ blocks}IN

IF  ∧ Cardinality(blocks) = 1
    ∧ ∀ b ∈ blocks : LegalBlock(b)
        THEN TRUE
  ELSE  ∧ ∀ b1 ∈ blocks:  ∧ LegalBlock(b1)

                          ∧ ∀ b2 ∈  (blocks \ {b1}) :  ∧ b2.parent ≠ b1.parent
                                                       ∧ b2.id ≠ b1.id

        ∧ LET id_set ≜ {ab.id : ab ∈ blocks}
          IN  LET head ≜ CHOOSE h ∈ blocks : h.parent ∉ id_set
              IN  ∀ nb ∈ blocks \ {head} : nb.parent ∈ id_set


Determine whether there is path which starts from s to t
HasPath(s, t, blocks) ≜ ∃ path ∈ SUBSET blocks :  ∧ s ∈ path
                                                  ∧ t ∈ path
                                                  ∧ IsPath(path)

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

Return the tail of a given path or chain
TailBlock(blocks) ≜ IF blocks = {} THEN Assert(FALSE, "Set is null")
                    ELSE  IF Cardinality(blocks) = 1 THEN CHOOSE b ∈ blocks : TRUE
                          ELSE  CHOOSE b ∈ blocks :  ∧ IsPath(blocks \ {b})
                                                     ∧ IsPath(blocks)
                                                     ∧ ∃ bp ∈ blocks : b.parent = bp.id

IsPrefix(path1, path2) ≜  ∧ IsPath(path1)
                          ∧ IsPath(path2)
                          ∧ path1 ⊆ path2
                          ∧ HeadBlock(path1) = HeadBlock(path2)
```

```
IsPrefixForAll(path, paths) ≜ ∀ tmpPath ∈ paths : IsPrefix(path, tmpPath)


LongestPath(paths) ≜ CHOOSE longest ∈ paths : ∀ tmpPath ∈ paths : ∧ Cardinality(longest) ≥ Cardinality(tr
                                                                  ∧ IsPath(tmpPath)


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


GetPath(s, t, blocks) ≜ IF ¬HasPath(s, t, blocks) THEN Print("No path!", ⟨s, t, blocks⟩)
                        ELSE LET all ≜ SUBSET blocksIN
                          CHOOSE path ∈ all : ∧ IsPath(path)
                                              ∧ s ∈ path
                                              ∧ t ∈ path
                                              ∧ HeadBlock(path) = s
                                              ∧ TailBlock(path)  = t


 Determine whether the given block s is ancestor of t
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



IsChain(blocks) ≜ ∀ b ∈ blocks : ∧ LegalBlock(b)
                                   Each block in chain must have a path to the head block
                                 ∧ LET head ≜ HeadBlock(blocks)IN   HasPath(head, b, blocks)
```

* Modification History
* Last modified *Mon Jun* 17 13:48:21 *CST* 2019 by *tangzaiyang*
* Created *Thu Jun* 06 11:21:13 *CST* 2019 by *tangzaiyang*