```
┌──────────────── MODULE PieceInclusionProof ────────────────┐
EXTENDS Integers, TLC, Sequences, FiniteSets

SIZE ≜ 256     Must be a power of 2. TODO: Make CONSTANT and add ASSUME .
HEIGHT ≜ 9     TODO: Calculate this from SIZE, as log2(SIZE) + 1.

  --algorithm PieceInclusionProof

variables
    HashCounter = −1;
    HashRecord = ⟨⟩;

macro RepCompress(a, b, height, var)begin
    if (⟨a, b, height⟩ ∈ DOMAIN HashRecord) then
        var := HashRecord[⟨a, b, height⟩];
     else
        HashCounter := HashCounter − 1;
        HashRecord := (⟨a, b, height⟩ :> HashCounter) @@ HashRecord;
        var := HashCounter
    end if ;
end macro ;

process test_hash = "test hash"
variables h1, h2, h3;

begin
    L1:
        RepCompress(1, 2, 0, h1);
    L2:
        RepCompress(1, 2, 0, h2);
    L3:
        RepCompress(2, 1, 0, h3);

        assert h1 = h2;
        assert h1 ≠ h3;
end process ;

fair process merkle_tree = "merkle tree"
variables h,
    input, row, rowSize, nextRow, index, proof_element, root, challenge,
    cursor_index, cursor_row, cursor_element, proof_index, challenge_path_acc, place_acc,
    rows = ⟨⟩     ;
    height = −1;

    proof_path = ⟨⟩;
    proof_elements = ⟨⟩;
begin
    BuildTree:
        $input := [i \in 1 \,..\, SIZE \mapsto i]$ ;
        $row := input$ ;
        $rows := \langle\rangle$     ;
    $RowLoop$:
        $height := height + 1$ ;
        $rows := Append(rows, row)$ ;
          It would be nice to make this assert an invariant, but how do we make an invariant
          over a process variable?
        **assert** $height > 1 \Rightarrow Len(rows[height - 1]) = 2 * Len(rows[height])$ ;

        $nextRow := \langle\rangle$ ;
        $index := 1$ ;
        $rowSize := Cardinality(\text{DOMAIN } row)$ ;
        **if** $rowSize > 1$ **then**
            $HashRow$:
                $RepCompress(row[index], row[index + 1], height, h)$ ;
                $nextRow := Append(nextRow, h)$ ;
            $Advance$: $index := index + 2$ ;
            **if** $index < Cardinality(\text{DOMAIN } row)$ **then**
                **goto** $HashRow$ ;
             **else**
                $row := nextRow$ ;
            **end if** ;
            $Repeat$:  **goto** $RowLoop$ ;
         **else**
            **assert** $Len(rows) = HEIGHT$ ;
        **end if** ;
    $Proofs$:
        $challenge := 1$ ;
    $MakeProof$:
        $cursor\_index := challenge$ ;
        $cursor\_row := 1$ ;
        $cursor\_element := rows[cursor\_row][cursor\_index]$ ;
        $proof\_path := \langle\rangle$ ;
        $proof\_elements := \langle\rangle$ ;
    $S1$:
        **if** $cursor\_index \% 2 = 1$ **then**
            $proof\_path := Append(proof\_path, \text{FALSE})$ ;
            $proof\_element := rows[cursor\_row][cursor\_index + 1]$ ;
            $RepCompress(rows[cursor\_row][cursor\_index],$
                            $proof\_element,$
                            $cursor\_row - 1,$
                            $cursor\_element)$ ;
         **else**
            $proof\_path := Append(proof\_path, \text{TRUE})$ ;
            $proof\_element := rows[cursor\_row][cursor\_index - 1]$ ;
            $RepCompress(proof\_element,$
                          $rows[cursor\_row][cursor\_index],$
                          $cursor\_row - 1,$
                          $cursor\_element)$ ;
        **end if** ;

        $proof\_elements := Append(proof\_elements, proof\_element)$ ;
    $ProofLoop$:
        $cursor\_row := cursor\_row + 1$ ;
        $cursor\_index := (cursor\_index + 1) \div 2$ ;
        **if** $cursor\_row < Len(rows)$ **then**
            **goto** $S1$ ;
        **end if** ;
    $FinishProof$:
        $root := rows[Len(rows)][1]$ ;

    $CheckProof$:
        $proof\_index := 1$ ;
        $height := 0$ ;
        $cursor\_index := challenge$ ;
        $cursor\_element := rows[height + 1][cursor\_index]$ ;
        $challenge\_path\_acc := 0$ ;
        $place\_acc := 1$ ;
    $ProofCheckLoop$:
        **if** $proof\_path[proof\_index]$ **then**
            $RepCompress(proof\_elements[proof\_index],$
                          $cursor\_element,$
                          $height,$
                          $cursor\_element)$ ;
            $challenge\_path\_acc := challenge\_path\_acc + place\_acc$ ;
        **else**
            $RepCompress(cursor\_element,$
                          $proof\_elements[proof\_index],$
                          $height,$
                          $cursor\_element)$ ;
        **end if** ;
        $place\_acc := place\_acc * 2$ ;

        $proof\_index := proof\_index + 1$ ;
        $height := height + 1$ ;
        **if** $height < Len(proof\_elements)$ **then**
            **goto** $ProofCheckLoop$ ;
        **end if** ;

    $CheckRoot$:
        assert $cursor\_element = root$ ;
        assert $challenge\_path\_acc = challenge - 1$ ;   challenges are 1-indexed because TLA+.

    $IncrementChallenge$:
        $challenge := challenge + 1$ ;
        if $challenge \leq Len(input)$ then
            goto $MakeProof$ ;
        end if ;
end process ;

end algorithm ;

 BEGIN TRANSLATION
CONSTANT $defaultInitValue$
VARIABLES $HashCounter, HashRecord, pc, h1, h2, h3, h, input, row, rowSize,$
          $nextRow, index, proof\_element, root, challenge, cursor\_index,$
          $cursor\_row, cursor\_element, proof\_index, challenge\_path\_acc,$
          $place\_acc, rows, height, proof\_path, proof\_elements$

$vars \triangleq \langle HashCounter, HashRecord, pc, h1, h2, h3, h, input, row, rowSize,$
        $nextRow, index, proof\_element, root, challenge, cursor\_index,$
        $cursor\_row, cursor\_element, proof\_index, challenge\_path\_acc,$
        $place\_acc, rows, height, proof\_path, proof\_elements \rangle$

$ProcSet \triangleq$ {"test hash"} $\cup$ {"merkle tree"}

$Init \triangleq$  Global variables
          $\land HashCounter = -1$
          $\land HashRecord = \langle\rangle$
           Process $test\_hash$
          $\land h1 = defaultInitValue$
          $\land h2 = defaultInitValue$
          $\land h3 = defaultInitValue$
           Process $merkle\_tree$
          $\land h = defaultInitValue$
          $\land input = defaultInitValue$
          $\land row = defaultInitValue$
          $\land rowSize = defaultInitValue$
          $\land nextRow = defaultInitValue$
          $\land index = defaultInitValue$
          $\land proof\_element = defaultInitValue$
          $\land root = defaultInitValue$
          $\land challenge = defaultInitValue$
          $\land cursor\_index = defaultInitValue$
          $\land cursor\_row = defaultInitValue$
          $\land cursor\_element = defaultInitValue$
          $\land proof\_index = defaultInitValue$
          $\land\ challenge\_path\_acc = defaultInitValue$
          $\land\ place\_acc = defaultInitValue$
          $\land\ rows = \langle\rangle$
          $\land\ height = -1$
          $\land\ proof\_path = \langle\rangle$
          $\land\ proof\_elements = \langle\rangle$
          $\land\ pc = [self \in ProcSet \mapsto \text{CASE } self = \text{“test hash”} \rightarrow \text{“L1”}$
                                           $\Box \quad self = \text{“merkle tree”} \rightarrow \text{“BuildTree”}]$

$L1 \triangleq$ $\land\ pc[\text{“test hash”}] = \text{“L1”}$
        $\land$ IF $(\langle 1, 2, 0\rangle \in \text{DOMAIN } HashRecord)$
              THEN $\land\ h1' = HashRecord[\langle 1, 2, 0\rangle]$
                   $\land$ UNCHANGED $\langle HashCounter, HashRecord\rangle$
              ELSE $\land\ HashCounter' = HashCounter - 1$
                   $\land\ HashRecord' = (\langle 1, 2, 0\rangle :> HashCounter')\ @@\ HashRecord$
                   $\land\ h1' = HashCounter'$
        $\land\ pc' = [pc \text{ EXCEPT } ![\text{“test hash”}] = \text{“L2”}]$
        $\land$ UNCHANGED $\langle h2, h3, h, input, row, rowSize, nextRow, index,$
                      $proof\_element, root, challenge, cursor\_index, cursor\_row,$
                      $cursor\_element, proof\_index, challenge\_path\_acc,$
                      $place\_acc, rows, height, proof\_path, proof\_elements\rangle$

$L2 \triangleq$ $\land\ pc[\text{“test hash”}] = \text{“L2”}$
        $\land$ IF $(\langle 1, 2, 0\rangle \in \text{DOMAIN } HashRecord)$
              THEN $\land\ h2' = HashRecord[\langle 1, 2, 0\rangle]$
                   $\land$ UNCHANGED $\langle HashCounter, HashRecord\rangle$
              ELSE $\land\ HashCounter' = HashCounter - 1$
                   $\land\ HashRecord' = (\langle 1, 2, 0\rangle :> HashCounter')\ @@\ HashRecord$
                   $\land\ h2' = HashCounter'$
        $\land\ pc' = [pc \text{ EXCEPT } ![\text{“test hash”}] = \text{“L3”}]$
        $\land$ UNCHANGED $\langle h1, h3, h, input, row, rowSize, nextRow, index,$
                      $proof\_element, root, challenge, cursor\_index, cursor\_row,$
                      $cursor\_element, proof\_index, challenge\_path\_acc,$
                      $place\_acc, rows, height, proof\_path, proof\_elements\rangle$

$L3 \triangleq$ $\land\ pc[\text{“test hash”}] = \text{“L3”}$
        $\land$ IF $(\langle 2, 1, 0\rangle \in \text{DOMAIN } HashRecord)$
              THEN $\land\ h3' = HashRecord[\langle 2, 1, 0\rangle]$
                   $\land$ UNCHANGED $\langle HashCounter, HashRecord\rangle$
              ELSE $\land\ HashCounter' = HashCounter - 1$
                   $\land\ HashRecord' = (\langle 2, 1, 0\rangle :> HashCounter')\ @@\ HashRecord$
                   $\land\ h3' = HashCounter'$
        $\land\ Assert(h1 = h2, \text{“Failure of assertion at line 34, column 9.”})$
        $\land\ Assert(h1 \neq h3', \text{“Failure of assertion at line 35, column 9.”})$
        $\land\ pc' = [pc \text{ EXCEPT } ![\text{“test hash”}] = \text{“Done”}]$
        $\land$ UNCHANGED $\langle h1, h2, h, input, row, rowSize, nextRow, index,$
```

```
                          proof_element, root, challenge, cursor_index, cursor_row,
                          cursor_element, proof_index, challenge_path_acc,
                          place_acc, rows, height, proof_path, proof_elements⟩

test_hash ≜ L1 ∨ L2 ∨ L3

BuildTree ≜ ∧ pc["merkle tree"] = "BuildTree"
            ∧ input′ = [i ∈ 1 . . SIZE ↦ i]
            ∧ row′ = input′
            ∧ rows′ = ⟨⟩
            ∧ pc′ = [pc EXCEPT !["merkle tree"] = "RowLoop"]
            ∧ UNCHANGED ⟨HashCounter, HashRecord, h1, h2, h3, h, rowSize,
                           nextRow, index, proof_element, root, challenge,
                           cursor_index, cursor_row, cursor_element,
                           proof_index, challenge_path_acc, place_acc,
                           height, proof_path, proof_elements⟩

RowLoop ≜ ∧ pc["merkle tree"] = "RowLoop"
          ∧ height′ = height + 1
          ∧ rows′ = Append(rows, row)
          ∧ Assert(height′ > 1 ⇒ Len(rows′[height′ − 1]) = 2 ∗ Len(rows′[height′]),
                   "Failure of assertion at line 57, column 9.")
          ∧ nextRow′ = ⟨⟩
          ∧ index′ = 1
          ∧ rowSize′ = Cardinality(DOMAIN row)
          ∧ IF rowSize′ > 1
                THEN ∧ pc′ = [pc EXCEPT !["merkle tree"] = "HashRow"]
                ELSE ∧ Assert(Len(rows′) = HEIGHT,
                              "Failure of assertion at line 74, column 13.")
                     ∧ pc′ = [pc EXCEPT !["merkle tree"] = "Proofs"]
          ∧ UNCHANGED ⟨HashCounter, HashRecord, h1, h2, h3, h, input, row,
                         proof_element, root, challenge, cursor_index,
                         cursor_row, cursor_element, proof_index,
                         challenge_path_acc, place_acc, proof_path,
                         proof_elements⟩

HashRow ≜ ∧ pc["merkle tree"] = "HashRow"
          ∧ IF (⟨(row[index]), (row[index + 1]), height⟩ ∈ DOMAIN HashRecord)
                THEN ∧ h′ = HashRecord[⟨(row[index]), (row[index + 1]), height⟩]
                     ∧ UNCHANGED ⟨HashCounter, HashRecord⟩
                ELSE ∧ HashCounter′ = HashCounter − 1
                     ∧ HashRecord′ = (⟨(row[index]), (row[index + 1]), height⟩ :> HashCounter′) @@ Ha
                     ∧ h′ = HashCounter′
          ∧ nextRow′ = Append(nextRow, h′)
          ∧ pc′ = [pc EXCEPT !["merkle tree"] = "Advance"]
          ∧ UNCHANGED ⟨h1, h2, h3, input, row, rowSize, index,
                              $proof\_element, root, challenge, cursor\_index,$
                              $cursor\_row, cursor\_element, proof\_index,$
                              $challenge\_path\_acc, place\_acc, rows, height,$
                              $proof\_path, proof\_elements\rangle$

$Advance \triangleq$ $\land\ pc[$“merkle tree”$] =$ “Advance”
              $\land\ index' = index + 2$
              $\land$ IF $index' < Cardinality($DOMAIN $row)$
                   THEN $\land\ pc' = [pc$ EXCEPT $![$“merkle tree”$] =$ “HashRow”$]$
                        $\land\ row' = row$
                   ELSE $\land\ row' = nextRow$
                        $\land\ pc' = [pc$ EXCEPT $![$“merkle tree”$] =$ “Repeat”$]$
              $\land$ UNCHANGED $\langle HashCounter, HashRecord, h1, h2, h3, h, input,$
                              $rowSize, nextRow, proof\_element, root, challenge,$
                              $cursor\_index, cursor\_row, cursor\_element,$
                              $proof\_index, challenge\_path\_acc, place\_acc, rows,$
                              $height, proof\_path, proof\_elements\rangle$

$Repeat \triangleq$ $\land\ pc[$“merkle tree”$] =$ “Repeat”
             $\land\ pc' = [pc$ EXCEPT $![$“merkle tree”$] =$ “RowLoop”$]$
             $\land$ UNCHANGED $\langle HashCounter, HashRecord, h1, h2, h3, h, input, row,$
                             $rowSize, nextRow, index, proof\_element, root,$
                             $challenge, cursor\_index, cursor\_row, cursor\_element,$
                             $proof\_index, challenge\_path\_acc, place\_acc, rows,$
                             $height, proof\_path, proof\_elements\rangle$

$Proofs \triangleq$ $\land\ pc[$“merkle tree”$] =$ “Proofs”
             $\land\ challenge' = 1$
             $\land\ pc' = [pc$ EXCEPT $![$“merkle tree”$] =$ “MakeProof”$]$
             $\land$ UNCHANGED $\langle HashCounter, HashRecord, h1, h2, h3, h, input, row,$
                             $rowSize, nextRow, index, proof\_element, root,$
                             $cursor\_index, cursor\_row, cursor\_element,$
                             $proof\_index, challenge\_path\_acc, place\_acc, rows,$
                             $height, proof\_path, proof\_elements\rangle$

$MakeProof \triangleq$ $\land\ pc[$“merkle tree”$] =$ “MakeProof”
                $\land\ cursor\_index' = challenge$
                $\land\ cursor\_row' = 1$
                $\land\ cursor\_element' = rows[cursor\_row'][cursor\_index']$
                $\land\ proof\_path' = \langle\rangle$
                $\land\ proof\_elements' = \langle\rangle$
                $\land\ pc' = [pc$ EXCEPT $![$“merkle tree”$] =$ “S1”$]$
                $\land$ UNCHANGED $\langle HashCounter, HashRecord, h1, h2, h3, h, input,$
                                $row, rowSize, nextRow, index, proof\_element, root,$
                                $challenge, proof\_index, challenge\_path\_acc,$
                                $place\_acc, rows, height\rangle$
```

```
S1 ≜  ∧ pc["merkle tree"] = "S1"
      ∧ IF cursor_index%2 = 1
           THEN ∧ proof_path′ = Append(proof_path, FALSE)
                ∧ proof_element′ = rows[cursor_row][cursor_index + 1]
                ∧ IF (⟨(rows[cursor_row][cursor_index]), proof_element′, (cursor_row − 1)⟩ ∈ DOMAIN H
                     THEN ∧ cursor_element′ = HashRecord[⟨(rows[cursor_row][cursor_index]), proof_e
                          ∧ UNCHANGED ⟨HashCounter, HashRecord⟩
                     ELSE ∧ HashCounter′ = HashCounter − 1
                          ∧ HashRecord′ = (⟨(rows[cursor_row][cursor_index]), proof_element′, (cursor
                          ∧ cursor_element′ = HashCounter′
           ELSE ∧ proof_path′ = Append(proof_path, TRUE)
                ∧ proof_element′ = rows[cursor_row][cursor_index − 1]
                ∧ IF (⟨proof_element′, (rows[cursor_row][cursor_index]), (cursor_row − 1)⟩ ∈ DOMAIN H
                     THEN ∧ cursor_element′ = HashRecord[⟨proof_element′, (rows[cursor_row][cursor_
                          ∧ UNCHANGED ⟨HashCounter, HashRecord⟩
                     ELSE ∧ HashCounter′ = HashCounter − 1
                          ∧ HashRecord′ = (⟨proof_element′, (rows[cursor_row][cursor_index]), (cursor
                          ∧ cursor_element′ = HashCounter′
      ∧ proof_elements′ = Append(proof_elements, proof_element′)
      ∧ pc′ = [pc EXCEPT !["merkle tree"] = "ProofLoop"]
      ∧ UNCHANGED ⟨h1, h2, h3, h, input, row, rowSize, nextRow, index, root,
                   challenge, cursor_index, cursor_row, proof_index,
                   challenge_path_acc, place_acc, rows, height⟩

ProofLoop ≜  ∧ pc["merkle tree"] = "ProofLoop"
             ∧ cursor_row′ = cursor_row + 1
             ∧ cursor_index′ = ((cursor_index + 1) ÷ 2)
             ∧ IF cursor_row′ < Len(rows)
                  THEN ∧ pc′ = [pc EXCEPT !["merkle tree"] = "S1"]
                  ELSE ∧ pc′ = [pc EXCEPT !["merkle tree"] = "FinishProof"]
             ∧ UNCHANGED ⟨HashCounter, HashRecord, h1, h2, h3, h, input,
                          row, rowSize, nextRow, index, proof_element, root,
                          challenge, cursor_element, proof_index,
                          challenge_path_acc, place_acc, rows, height,
                          proof_path, proof_elements⟩

FinishProof ≜  ∧ pc["merkle tree"] = "FinishProof"
               ∧ root′ = rows[Len(rows)][1]
               ∧ pc′ = [pc EXCEPT !["merkle tree"] = "CheckProof"]
               ∧ UNCHANGED ⟨HashCounter, HashRecord, h1, h2, h3, h, input,
                            row, rowSize, nextRow, index, proof_element,
                            challenge, cursor_index, cursor_row,
                            cursor_element, proof_index, challenge_path_acc,
                            place_acc, rows, height, proof_path,
                            proof_elements⟩
```

```
CheckProof ≜ ∧ pc["merkle tree"] = "CheckProof"
             ∧ proof_index′ = 1
             ∧ height′ = 0
             ∧ cursor_index′ = challenge
             ∧ cursor_element′ = rows[height′ + 1][cursor_index′]
             ∧ challenge_path_acc′ = 0
             ∧ place_acc′ = 1
             ∧ pc′ = [pc EXCEPT ![“merkle tree”] = "ProofCheckLoop"]
             ∧ UNCHANGED ⟨HashCounter, HashRecord, h1, h2, h3, h, input,
                          row, rowSize, nextRow, index, proof_element,
                          root, challenge, cursor_row, rows, proof_path,
                          proof_elements⟩

ProofCheckLoop ≜ ∧ pc["merkle tree"] = "ProofCheckLoop"
                 ∧ IF proof_path[proof_index]
                      THEN ∧ IF (⟨(proof_elements[proof_index]), cursor_element, height⟩ ∈ DOMAIN Ha
                                THEN ∧ cursor_element′ = HashRecord[⟨(proof_elements[proof_index]
                                     ∧ UNCHANGED ⟨HashCounter,
                                                  HashRecord⟩
                                ELSE ∧ HashCounter′ = HashCounter − 1
                                     ∧ HashRecord′ = (⟨(proof_elements[proof_index]), cursor_eleme
                                     ∧ cursor_element′ = HashCounter′
                           ∧ challenge_path_acc′ = challenge_path_acc + place_acc
                      ELSE ∧ IF (⟨cursor_element, (proof_elements[proof_index]), height⟩ ∈ DOMAIN Ha
                                THEN ∧ cursor_element′ = HashRecord[⟨cursor_element, (proof_eleme
                                     ∧ UNCHANGED ⟨HashCounter,
                                                  HashRecord⟩
                                ELSE ∧ HashCounter′ = HashCounter − 1
                                     ∧ HashRecord′ = (⟨cursor_element, (proof_elements[proof_inde
                                     ∧ cursor_element′ = HashCounter′
                           ∧ UNCHANGED challenge_path_acc
                 ∧ place_acc′ = place_acc ∗ 2
                 ∧ proof_index′ = proof_index + 1
                 ∧ height′ = height + 1
                 ∧ IF height′ < Len(proof_elements)
                      THEN ∧ pc′ = [pc EXCEPT !["merkle tree"] = "ProofCheckLoop"]
                      ELSE ∧ pc′ = [pc EXCEPT !["merkle tree"] = "CheckRoot"]
                 ∧ UNCHANGED ⟨h1, h2, h3, h, input, row, rowSize, nextRow,
                              index, proof_element, root, challenge,
                              cursor_index, cursor_row, rows, proof_path,
                              proof_elements⟩

CheckRoot ≜ ∧ pc["merkle tree"] = "CheckRoot"
            ∧ Assert(cursor_element = root,
                     "Failure of assertion at line 140, column 9.")
```

```
                    ∧ Assert(challenge_path_acc = challenge − 1,
                             "Failure of assertion at line 141, column 9.")
                    ∧ pc′ = [pc EXCEPT ![“merkle tree”] = “IncrementChallenge”]
                    ∧ UNCHANGED ⟨HashCounter, HashRecord, h1, h2, h3, h, input,
                                 row, rowSize, nextRow, index, proof_element, root,
                                 challenge, cursor_index, cursor_row,
                                 cursor_element, proof_index, challenge_path_acc,
                                 place_acc, rows, height, proof_path,
                                 proof_elements⟩

IncrementChallenge ≜ ∧ pc[“merkle tree”] = “IncrementChallenge”
                     ∧ challenge′ = challenge + 1
                     ∧ IF challenge′ ≤ Len(input)
                          THEN ∧ pc′ = [pc EXCEPT ![“merkle tree”] = “MakeProof”]
                          ELSE ∧ pc′ = [pc EXCEPT ![“merkle tree”] = “Done”]
                     ∧ UNCHANGED ⟨HashCounter, HashRecord, h1, h2, h3, h,
                                  input, row, rowSize, nextRow, index,
                                  proof_element, root, cursor_index,
                                  cursor_row, cursor_element, proof_index,
                                  challenge_path_acc, place_acc, rows,
                                  height, proof_path, proof_elements⟩

merkle_tree ≜ BuildTree ∨ RowLoop ∨ HashRow ∨ Advance ∨ Repeat
                 ∨ Proofs ∨ MakeProof ∨ S1    ∨ ProofLoop ∨ FinishProof
                 ∨ CheckProof ∨ ProofCheckLoop ∨ CheckRoot
                 ∨ IncrementChallenge

Next ≜ test_hash ∨ merkle_tree
          ∨  Disjunct to prevent deadlock on termination
            ((∀ self ∈ ProcSet : pc[self] = “Done”) ∧ UNCHANGED vars)

Spec ≜ ∧ Init ∧ □[Next]_vars
       ∧ WF_vars(merkle_tree)

Termination ≜ ◇(∀ self ∈ ProcSet : pc[self] = “Done”)

 END TRANSLATION
```