MODULE *BitcoinChain*

EXTENDS *Naturals*, *FiniteSets*, *Sequences*

CONSTANTS

Nodes,	Set of nodes in the network
MaxBlocks,	Maximum number of blocks that can be created. We need this to cap the model run.
GenesisHash	Hash of the genesis block

VARIABLES

blocksByNode,	*blocksByNode*[*n*] is the set of blocks known by node *n*
chainsByNode,	*chainsByNode*[*n*] is the *DAG* of blocks for node *n*
tipsByNode,	*tipsByNode*[*n*] is the current tip of the chain for node *n*
workByNode,	*workByNode*[*n*] is a function mapping each block to its cumulative work
confirmedByNode,	*confirmedByNode*[*n*] is the set of confirmed blocks for node *n*
network,	*network*[*n*] is the set of blocks in transit to node *n*
nextBlockId	Counter used to generate unique block *IDs*

$vars \triangleq \langle blocksByNode,\ chainsByNode,\ tipsByNode,\ workByNode,\ confirmedByNode,\ network,\ nextBlockId \rangle$

Type definitions

$Block \triangleq [id : Nat,\ prevHash : Nat,\ nonce : Nat,\ timestamp : Nat]$

$Chain \triangleq [blocks : \text{SUBSET } Nat,\ edges : \text{SUBSET } (Nat \times Nat)]$

Helper functions

$WorkFor(b) \triangleq 1$ Simplified work calculation, each block contributes 1 unit of work

Initial state

$$
\begin{aligned}
Init \triangleq\ & \land\ blocksByNode = [n \in Nodes \mapsto \{[id \mapsto 0,\ prevHash \mapsto GenesisHash,\ nonce \mapsto 0,\ timestamp \mapsto 0]\}] \\
& \land\ chainsByNode = [n \in Nodes \mapsto [blocks \mapsto \{0\},\ edges \mapsto \{\}]] \\
& \land\ tipsByNode = [n \in Nodes \mapsto 0] \\
& \land\ workByNode = [n \in Nodes \mapsto [i \in \{0\} \mapsto 1]] \quad \text{Initialize work for genesis block to 1} \\
& \land\ confirmedByNode = [n \in Nodes \mapsto \{\}] \\
& \land\ network = [n \in Nodes \mapsto \{\}] \\
& \land\ nextBlockId = 1
\end{aligned}
$$

Action: Create a new block

$$
\begin{aligned}
CreateBlock(n) \triangleq\ & \land\ nextBlockId < MaxBlocks \\
& \land\ \text{LET} \\
& \qquad newBlock \triangleq [id \mapsto nextBlockId, \\
& \qquad\qquad prevHash \mapsto tipsByNode[n], \\
& \qquad\qquad nonce \mapsto nextBlockId, \quad \text{Simplified nonce} \\
& \qquad\qquad timestamp \mapsto nextBlockId] \quad \text{Simplified timestamp} \\
& \qquad updatedChain \triangleq [\\
& \qquad\quad blocks \mapsto chainsByNode[n].blocks \cup \{nextBlockId\}, \\
& \qquad\quad edges \mapsto chainsByNode[n].edges \cup \{\langle tipsByNode[n],\ nextBlockId \rangle\}
\end{aligned}
$$

```
        ]
        newWork ≜ workByNode[n][tipsByNode[n]] + WorkFor(newBlock)
        updatedWork ≜ [workByNode[n] EXCEPT ![nextBlockId] = newWork]
        prevTip ≜ tipsByNode[n] Previous tip before creating the new block
    IN
     Update the blocks known by the local node
    ∧ blocksByNode' = [blocksByNode EXCEPT ![n] = @ ∪ {newBlock}]
     Update the chain for the local node
    ∧ chainsByNode' = [chainsByNode EXCEPT ![n] = updatedChain]
     Update the tip for the local node
    ∧ tipsByNode' = [tipsByNode EXCEPT ![n] = nextBlockId]
     Track total work for the new block at this node, even if it is a constant for now
    ∧ LET
        newWorkMap ≜ [b ∈ DOMAIN workByNode[n] ∪ {nextBlockId} ↦
            IF b = nextBlockId THEN newWork ELSE workByNode[n][b]]
        IN
            workByNode' = [workByNode EXCEPT ![n] = newWorkMap]
     The new block will be received by all other nodes in the network
    ∧ network' = [m ∈ Nodes ↦ IF m ≠ n THEN network[m] ∪ {newBlock} ELSE network[m]]
     Update the global nextBlockId
    ∧ nextBlockId' = nextBlockId + 1
     Local node immediately confirms the previous tip
    ∧ confirmedByNode' = [confirmedByNode EXCEPT ![n] = confirmedByNode[n] ∪ {prevTip}]

 Action: Receive a block from the network
ReceiveBlockWithPreviousKnown(n, block) ≜
    ∧ block.prevHash ∈ chainsByNode[n].blocks
    ∧
        LET
            prevBlock ≜ block.prevHash
            updatedChain ≜ [
                                blocks ↦ chainsByNode[n].blocks ∪ {block.id},
                                edges ↦ chainsByNode[n].edges ∪ {⟨prevBlock, block.id⟩}
                            ]
            newWork ≜ workByNode[n][tipsByNode[n]] + WorkFor(block)
            newTip ≜ IF newWork > workByNode[n][tipsByNode[n]]
                     THEN block.id
                     ELSE tipsByNode[n]
        IN
         Add the block to the local node's known blocks
        ∧ blocksByNode' = [blocksByNode EXCEPT ![n] = @ ∪ {block}]
         Take the next block in the receieve queue
        ∧ network' = [network EXCEPT ![n] = @ \ {block}]
         Update the chain for the local node by adding a back ref to the previous block
        ∧ chainsByNode' = [chainsByNode EXCEPT ![n] = updatedChain]
```

```
        Update the work seen by the node up to the block
      ∧ LET
          newWorkMap ≜ [b ∈ DOMAIN workByNode[n] ∪ {block.id} ↦
              IF b = block.id THEN newWork ELSE workByNode[n][b]]
        IN
            workByNode' = [workByNode EXCEPT ![n] = newWorkMap]
        Confirm the previous tip if we have a new tip
      ∧ IF newWork > workByNode[n][tipsByNode[n]] THEN
            ∧ confirmedByNode' = [confirmedByNode EXCEPT ![n] = confirmedByNode[n] ∪ {tipsByNode[n
        ELSE
            ∧ confirmedByNode' = confirmedByNode
        Update the tip for the local node
      ∧ tipsByNode' = [tipsByNode EXCEPT ![n] = block.id]
      ∧ UNCHANGED ⟨nextBlockId⟩

 Next state relation
Next ≜
    ∨ ∃ n ∈ Nodes : CreateBlock(n)
    ∨ ∃ n ∈ Nodes : ∃ block ∈ network[n] : ReceiveBlockWithPreviousKnown(n, block)

 Invariants
TypeInvariant ≜
    ∧ ∀ n ∈ Nodes :
        ∧ nextBlockId ∈ Nat
        ∧ tipsByNode[n] ∈ Nat
        ∧ chainsByNode[n].blocks ⊆ Nat
        ∧ chainsByNode[n].edges ⊆ (Nat × Nat)
        ∧ ∀ blockId ∈ DOMAIN workByNode[n] :
            ∧ blockId ∈ Nat
            ∧ workByNode[n][blockId] ∈ Nat
        ∧ confirmedByNode[n] ⊆ Nat
        ∧ network[n] ⊆ Block
        ∧ ∀ b ∈ blocksByNode[n] :
            ∧ b.id ∈ Nat
            ∧ b.prevHash ∈ Nat
            ∧ b.nonce ∈ Nat
            ∧ b.timestamp ∈ Nat

 Properties
TipHasMostWork ≜
    ∀ n ∈ Nodes :
      ∀ b ∈ chainsByNode[n].blocks :
        workByNode[n][tipsByNode[n]] ≥ workByNode[n][b]

 Complete specification
Spec ≜ Init ∧ □[Next]_vars
```