```
──────────────── MODULE BitSnark ────────────────
This module specifies the transaction flow in the BitSNARK protocol.

EXTENDS Naturals, FiniteSets

CONSTANTS
    PROGRAM_SIZE,      The size of the verification program.
    PROVER_STAKE,      Size of the prover stake.
    VERIFIER_PAYMENT   Size of the verifier payment.

VARIABLES
    outputs,       The set of all currently spendable outputs.
    balances,      Balances of the participants and contracts.
    contentioned   The number of contentioned instructions.

Transactions ≜     The set of protocol transactions.
    [Proof ↦ [
        inputs ↦ {"Stakable Funds"},
        outputs ↦ {"Proof Value", "Proof Signal"}],
    ProofUncontested ↦ [
        inputs ↦ {"Proof Value", "Proof Signal", "Locked Funds"},
        outputs ↦ {"Proof Uncontested"}],
    Challenge ↦ [
        inputs ↦ {"Payable Funds", "Proof Signal"},
        outputs ↦ {"Challenge"}],
    ChallengeUncontested ↦ [
        inputs ↦ {"Proof Value"},
        outputs ↦ {"Challenge Uncontested"}],
    FirstState ↦ [
        inputs ↦ {"Proof Value"},
        outputs ↦ {"State"}],
    SubsequentState ↦ [
        inputs ↦ {"Select"},
        outputs ↦ {"State"}],
    StateUncontested ↦ [
        inputs ↦ {"State", "Locked Funds"},
        outputs ↦ {"State Uncontested"}],
    Select ↦ [
        inputs ↦ {"State"},
        outputs ↦ {"Select"}],
    SelectUncontested ↦ [
        inputs ↦ {"Select"},
        outputs ↦ {"Select Uncontested"}],
    Argument ↦ [
        inputs ↦ {"Select"},
        outputs ↦ {"Argument"}],
    $ArgumentUncontested \mapsto [$
        $inputs \mapsto \{$"Argument", "Locked Funds"$\},$
        $outputs \mapsto \{$"Argument Uncontested"$\}],$
    $ProofRefuted \mapsto [$
        $inputs \mapsto \{$"Argument"$\},$
        $outputs \mapsto \{$"Proof Refuted"$\}]]$
```

$StartingBalances \triangleq$

Nothing is staked.
Prover has funds to stake and unlock funds.
Verifier has funds to pay for a challenge and win the stake.

```
    $[staked \mapsto 0,$
     $prover \mapsto PROVER\_STAKE,$
     $verifier \mapsto VERIFIER\_PAYMENT]$
```

$IsProofValid \triangleq \text{CHOOSE } v \in \{\text{TRUE}, \text{FALSE}\} : \text{TRUE}$

```
$Init \triangleq$
    $\land\ outputs = \{$"Stakable Funds", "Payable Funds", "Locked Funds"$\}$
    $\land\ balances = StartingBalances$
    $\land\ contentioned = PROGRAM\_SIZE$
```

State Changers.

```
$Publish(transaction) \triangleq$
    $\land\ transaction.inputs \subseteq outputs$
    $\land\ outputs' = (outputs \setminus transaction.inputs) \cup transaction.outputs$
```

```
$Transfer(from, to, amount) \triangleq$
    $balances' = [$
        $balances \text{ EXCEPT}$
        $![from] = @ - amount,$
        $![to] = @ + amount]$
```

$ContentionDissection \triangleq$

Divide the *contentioned* segment into ten subsegments, dividing the remainder between as many segments as necessary. Then set the size of the new *contentioned* segment to the size of first subsegment, which will always be at least as large as any of the other subsegments.
And yes, this is practically the same as:
$contentioned' = (contentioned + 9) \div 10$

```
    $contentioned' = Cardinality(\{i \in 1 \,..\, contentioned : i \% 10 = 1\})$
```

Transaction Functions - the steps taken from one state to the next.

$Proof \triangleq$

```
    ∧ Publish(Transactions["Proof"])
    ∧ Transfer("prover", "staked", PROVER_STAKE)
    ∧ UNCHANGED contentioned

ProofUncontested ≜
    ∧ Publish(Transactions["ProofUncontested"])
    ∧ Transfer("staked", "prover", PROVER_STAKE)
    ∧ UNCHANGED contentioned

Challenge ≜
     A smart verifier will test for an existing state transaction,
     but smart verifiers aren't a part of the spec.
    ∧ Publish(Transactions["Challenge"])
    ∧ Transfer("verifier", "prover", VERIFIER_PAYMENT)
    ∧ UNCHANGED contentioned

ChallengeUncontested ≜
    ∧ Publish(Transactions["ChallengeUncontested"])
    ∧ Transfer("staked", "verifier", PROVER_STAKE)
    ∧ UNCHANGED contentioned

State ≜
    ∧ contentioned > 1
    ∧ (IF contentioned = PROGRAM_SIZE THEN
            A smart prover will test for an existing challenge,
            but smart provers aren't a part of the spec either.
          Publish(Transactions["FirstState"])
       ELSE
          Publish(Transactions["SubsequentState"]))
    ∧ UNCHANGED balances
    ∧ UNCHANGED contentioned

StateUncontested ≜
    ∧ Publish(Transactions["StateUncontested"])
    ∧ Transfer("staked", "prover", PROVER_STAKE)
    ∧ UNCHANGED contentioned

Select ≜
    ∧ contentioned > 1
    ∧ Publish(Transactions["Select"])
    ∧ ContentionDissection
    ∧ UNCHANGED balances

SelectUncontested ≜
    ∧ Publish(Transactions["SelectUncontested"])
    ∧ Transfer("staked", "verifier", PROVER_STAKE)
    ∧ UNCHANGED contentioned
```

```
Argument ≜
    ∧ contentioned = 1
    ∧ Publish(Transactions["Argument"])
    ∧ UNCHANGED contentioned
    ∧ UNCHANGED balances

ArgumentUncontested ≜
    ∧ Publish(Transactions["ArgumentUncontested"])
    ∧ Transfer("staked", "prover", PROVER_STAKE)
    ∧ UNCHANGED contentioned

ProofRefuted ≜
    ∧ IsProofValid = FALSE
    ∧ Publish(Transactions["ProofRefuted"])
    ∧ Transfer("staked", "verifier", PROVER_STAKE)
    ∧ UNCHANGED contentioned

Next ≜
    ∨ Proof
    ∨ ProofUncontested
    ∨ Challenge
    ∨ ChallengeUncontested
    ∨ State
    ∨ StateUncontested
    ∨ Select
    ∨ SelectUncontested
    ∨ Argument
    ∨ ArgumentUncontested
    ∨ ProofRefuted

vars ≜ ⟨outputs, balances, contentioned⟩

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


 Data Extraction.

AllowedOutputs ≜
    UNION {
        Transactions[name].inputs
            ∪
        Transactions[name].outputs :
            name ∈ DOMAIN Transactions}

Sum(balancesRecord) ≜
```

$balancesRecord[\text{“staked”}] +$
$balancesRecord[\text{“prover”}] +$
$balancesRecord[\text{“verifier”}]$

Safety Properties.

$OutputsTypeOK \triangleq$
$\quad outputs \subseteq AllowedOutputs$

$BalancesTypeOK \triangleq$
$\quad \land \text{DOMAIN } balances = \text{DOMAIN } StartingBalances$
$\quad \land \forall key \in \text{DOMAIN } balances : balances[key] \in 0 \,..\, Sum(StartingBalances)$

$ContentionedTypeOK \triangleq$
$\quad contentioned \in 1 \,..\, PROGRAM\_SIZE$

$TypesOK \triangleq$
$\quad \land OutputsTypeOK$
$\quad \land BalancesTypeOK$
$\quad \land ContentionedTypeOK$

$BalancesValueOK \triangleq$
$\quad Sum(balances) = Sum(StartingBalances)$

$IncentiveOK \triangleq$
$\quad \land \text{“Proof Refuted”} \in outputs \Rightarrow$
$\qquad balances[\text{“verifier”}] \geq StartingBalances[\text{“verifier”}]$
$\quad \land \text{“Argument Uncontested”} \in outputs \Rightarrow$
$\qquad balances[\text{“prover”}] \geq StartingBalances[\text{“prover”}]$

$Safe \triangleq$
$\quad \land TypesOK$
$\quad \land BalancesValueOK$
$\quad \land IncentiveOK$

$\text{THEOREM } Spec \Rightarrow \Box Safe$

Liveness Helpers.

$Final \triangleq$
$\quad \neg \text{ENABLED } Next$

$ProverWins \triangleq$
$\quad \text{“Locked Funds”} \notin outputs$

$VerifierWins \triangleq$
$\quad \text{“Locked Funds”} \in outputs$

Liveness Properties.

$Terminates \triangleq$
$\quad \Box\Diamond Final$

$StakeIsFreed \triangleq$
$\quad \Diamond(balances[\text{“staked”}] = 0)$

$HonestVerification \triangleq$
$\quad \Diamond\textsc{if}\ IsProofValid\ \textsc{then}\ ProverWins\ \textsc{else}\ VerifierWins$

$Live \triangleq$
$\quad \land Terminates$
$\quad \land StakeIsFreed$
$\quad \land HonestVerification$

$\textsc{theorem}\ Spec \Rightarrow \Box Live$