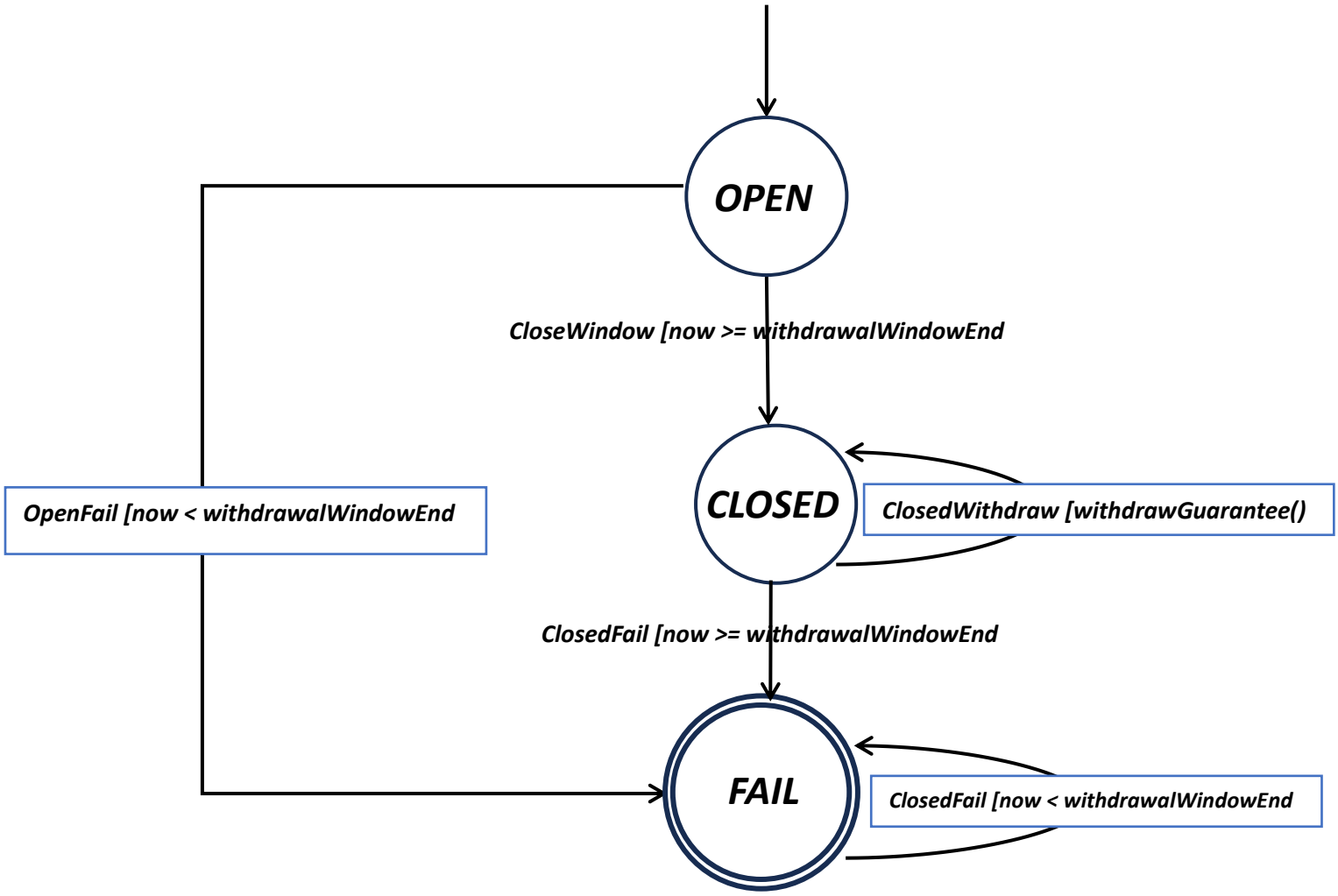
Question 2: Runtime Verification

Piggy Bank Smart Contract:

Property 1: Limited Withdrawal Time

Specification: Users can only withdraw funds from the PiggyBank contract after a specified withdrawal window has passed since the first deposit.



This DEA specifies the withdrawal window behaviour. It starts in the 'Open' state and transitions to the 'Closed' state when the withdrawal window has ended. Trying to withdraw outside the window leads to a 'Fail' state. Successful withdrawals within the window keep the contract in the 'Closed' state.

LARVA Specification:

```
monitor PiggyBankWithdrawal {
    declarations {
        uint public withdrawalWindowEnd;
    }
    initialisation {
        LARVA_DisableContract();
        withdrawalWindowEnd = 0;
    }
    reparation {
        LARVA_DisableContract();
    }
    DEA WithdrawalWindow {
        states {
            Open: initial;
            Closed;
            Fail: bad;
        }
        transitions {
            Open -[timeWindow@(now >= withdrawalWindowEnd)]-> Closed;
            Open -[timeWindow@(now < withdrawalWindowEnd)]-> Fail;
            Closed -[after(withdrawGuarantee())]-> Closed;
            Closed -[timeWindow@(now >= withdrawalWindowEnd)]-> Fail;
            Closed -[timeWindow@(now < withdrawalWindowEnd)]-> Fail;
        }
    }
}
```

Monitor Purpose: This monitor enforces the property that users can only withdraw funds from the PiggyBank contract after a specified withdrawal window has passed since the first deposit.

Declarations:

- withdrawalWindowEnd: A public variable indicating the end time of the withdrawal window.

Initialisation:

- LARVA_DisableContract(): Disables the monitoring of the contract.
- withdrawalWindowEnd = 0;: Initializes the withdrawal window end time to 0.

Reparation:

- LARVA_DisableContract(): Disables the contract in case of a violation.

DEA WithdrawalWindow:

This DEA monitors the withdrawal window behavior.

States: Open (initial state), Closed, Fail (bad state).

Transitions:

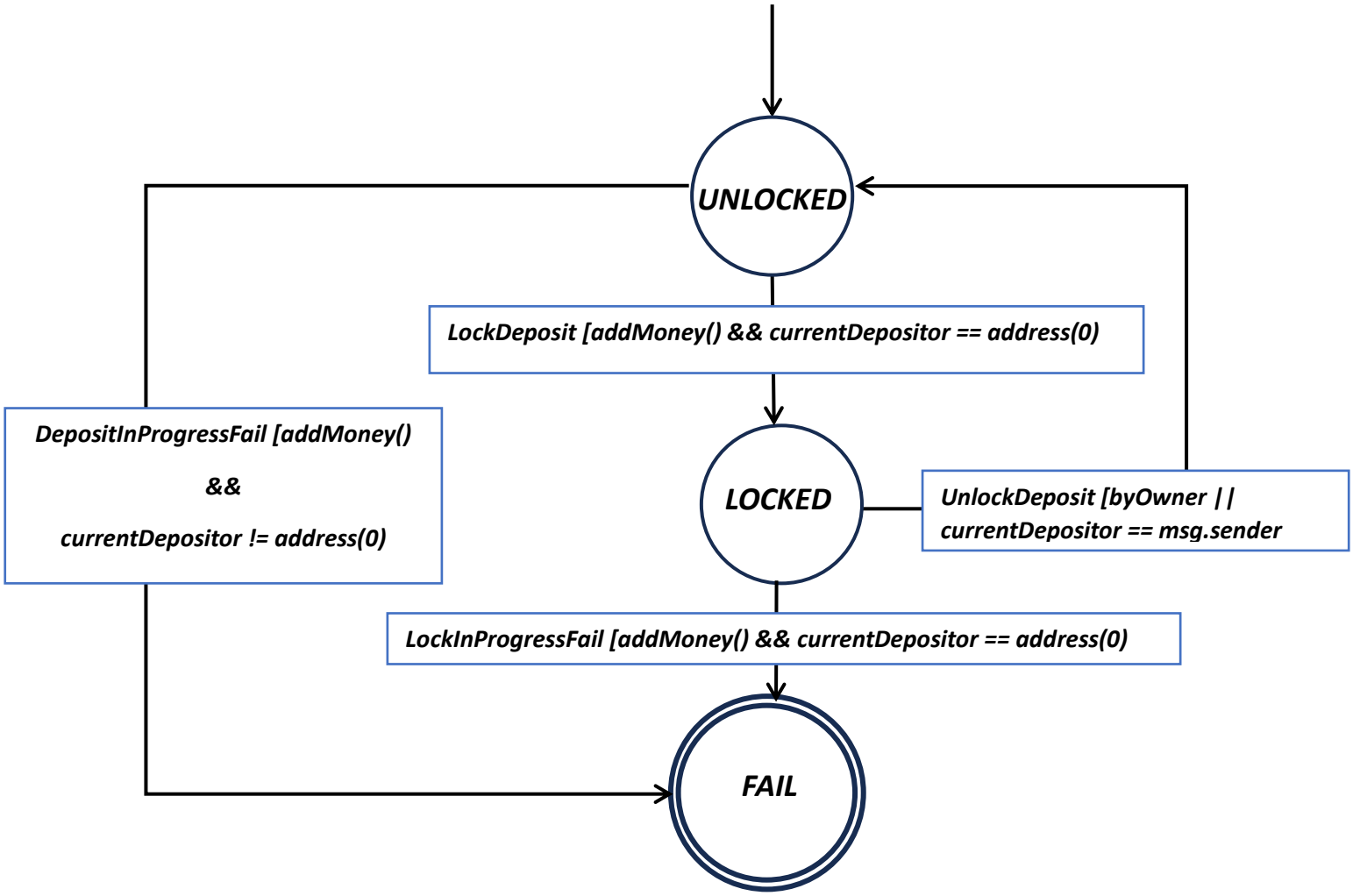
1. From Open to Closed when the withdrawal window has ended (now >= withdrawalWindowEnd).
2. From Open to Fail when attempting to withdraw before the withdrawal window (now < withdrawalWindowEnd).
3. From Closed to Closed after a successful withdrawal (withdrawGuarantee()).
4. From Closed to Fail if attempting to withdraw after the withdrawal window has ended (now >= withdrawalWindowEnd).
5. From Closed to Fail if attempting to withdraw before the withdrawal window (now < withdrawalWindowEnd).

This specification ensures that users can only withdraw funds after a specific withdrawal window has passed since the first deposit. The PiggyBankWithdrawal monitor includes the withdrawalWindowEnd variable that holds the end time of the withdrawal window. Initially, the monitoring is disabled, and the withdrawal window end time is set to 0.

The WithdrawalWindow DEA monitors the withdrawal window. The contract starts in the 'Open' state, and a transition to the 'Closed' state occurs when the withdrawal window has ended. Attempting a withdrawal outside the window leads to a 'Fail' state. After the window is closed, users can withdraw funds using the withdrawGuarantee function, but any interaction with the withdrawal window outside the specified time results in a 'Fail' state.

Property 2: Prevent Simultaneous Deposits

Specification: Only one user at a time can deposit funds into the PiggyBank contract.



This DEA enforces the property of preventing simultaneous deposits. It begins in the 'Unlocked' state. A transition to the 'Locked' state occurs when a deposit is initiated and no other deposit is in progress. If a deposit is attempted while another is ongoing, the contract enters a 'Fail' state. The contract returns to the 'Unlocked' state when the owner cancels the deposit or when the ongoing depositor confirms their transaction.

LARVA Specification:

```
monitor PiggyBankDeposit {
    declarations {
        address public currentDepositor;
    }
    initialisation {
        LARVA_DisableContract();
        currentDepositor = address(0);
    }
    reparation {
        LARVA_DisableContract();
    }
    DEA DepositLock {
        states {
            Unlocked: initial;
            Locked;
            Fail: bad;
        }
        transitions {
            Unlocked -[after(addMoney()) | currentDepositor == address(0)]->
Locked;
            Unlocked -[after(addMoney()) | currentDepositor != address(0)]->
Fail;
            Locked -[after(addMoney()) | currentDepositor == address(0)]->
Fail;
            Locked -[byOwner | currentDepositor == msg.sender]-> Unlocked;
        }
    }
}
```

Monitor Purpose: This monitor enforces the property that only one user at a time can deposit funds into the PiggyBank contract.

Declarations:

- currentDepositor: A public variable indicating the address of the current depositor.

Initialisation:

- LARVA_DisableContract(): Disables the monitoring of the contract.
- currentDepositor = address(0);: Initializes the current depositor's address to the zero address.

Reparation:

- LARVA_DisableContract(): Disables the contract in case of a violation.

This DEA monitors the deposit behavior.

States: Unlocked (initial state), Locked, Fail (bad state).

Transitions:

1. From Unlocked to Locked when a deposit is initiated and no other deposit is in progress (after(addMoney()) && currentDepositor == address(0)).
2. From Unlocked to Fail when attempting to initiate a deposit while another is ongoing (after(addMoney()) && currentDepositor != address(0)).
3. From Locked to Fail when attempting to initiate another deposit while one is already ongoing (after(addMoney()) && currentDepositor == address(0)).
4. From Locked to Unlocked when the owner cancels the deposit or the ongoing depositor confirms the transaction (byOwner || currentDepositor == msg.sender).

This specification prevents multiple users from depositing funds simultaneously into the PiggyBank contract. The PiggyBankDeposit monitor includes the currentDepositor variable that keeps track of the current depositor's address. Initially, monitoring is disabled, and currentDepositor is set to the zero address.

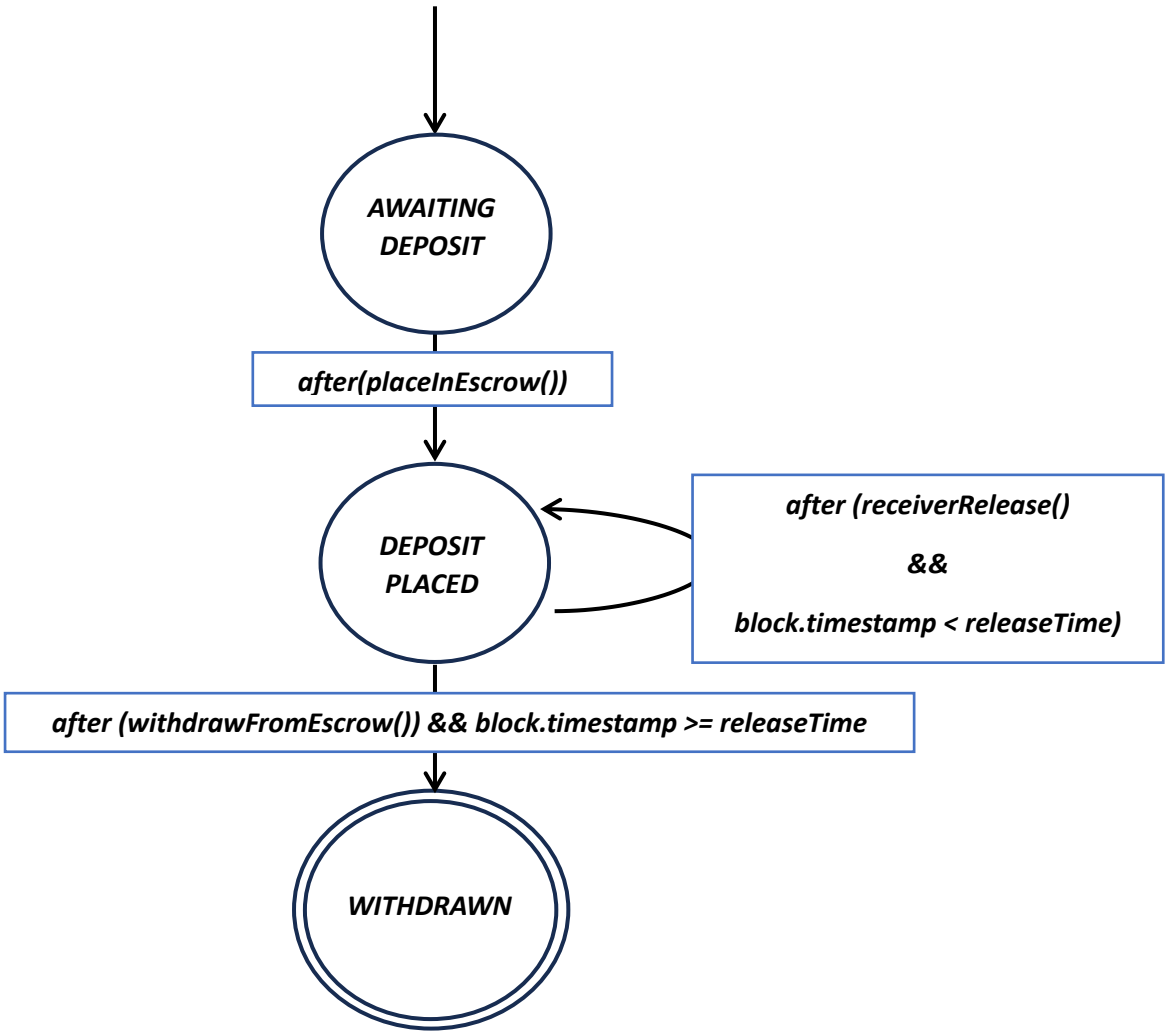
The DepositLock DEA monitors the deposit process. The contract starts in the 'Unlocked' state. A transition to the 'Locked' state occurs when a user initiates a deposit and no other deposit is in progress. If a deposit is attempted while another is ongoing, the contract enters a 'Fail' state. The contract returns to the 'Unlocked' state when the owner cancels the deposit or when the ongoing depositor confirms their transaction.

Escrow Smart Contract:

2 specification properties for the Escrow Smart Contract, along with their associated Dynamic Event Automata (DEA) and LARVA code. I'll also explain the LARVA code for each property in detail.

Specification 1: Early Release Prevention

Property: The smart contract should prevent the receiver from withdrawing funds before the release time, even if the sender releases the escrow.



In this property, we want to ensure that the receiver cannot withdraw funds before the release time, even if the sender releases the escrow. To achieve this, we introduce a new `DEA PreventEarlyRelease`.

- ***States:*** The DEA has three states: `AwaitingDeposit`, `DepositPlaced`, and `Withdrawn`.
- ***Transitions:***
    i. From `AwaitingDeposit` to `DepositPlaced` after the sender places a deposit.
    ii. From `DepositPlaced` to itself when the receiver tries to release the escrow and the release time hasn't been reached yet.
    iii. From `DepositPlaced` to `Withdrawn` after the receiver successfully withdraws funds and the release time has passed.

```
DEA PreventEarlyRelease {
    states {
        AwaitingDeposit: initial;
        DepositPlaced;
        Withdrawn;
    }

    transitions {
        AwaitingDeposit -[after(placeInEscrow())]-> DepositPlaced;
        DepositPlaced -[after(receiverRelease() && block.timestamp < re-
leaseTime)]-> DepositPlaced;
        DepositPlaced -[after(withdrawFromEscrow()) && block.timestamp >= re-
leaseTime]-> Withdrawn;
    }
}
```

LARVA Code Explanation:

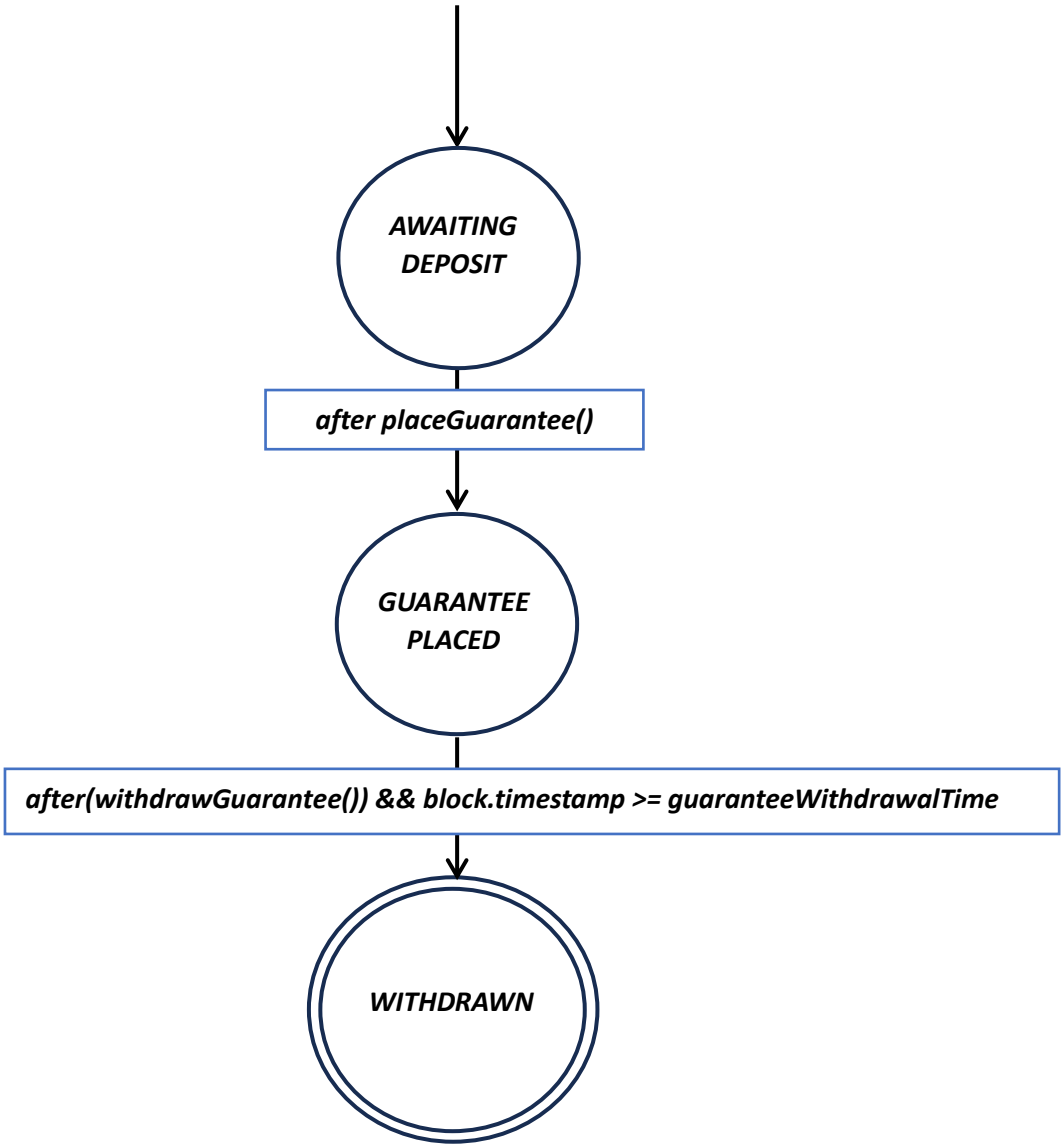
The automaton starts in the AwaitingDeposit state.

- When the placeInEscrow() function is called, it transitions to the DepositPlaced state.
- If the receiverRelease() function is called and the current time is before the release time, it remains in the DepositPlaced state.
- If the withdrawFromEscrow() function is called and the current time is greater than or equal to the release time, it transitions to the Withdrawn state.

Specification Property 2: Delayed Guarantee Withdrawal

Property: The smart contract should enforce a delay between guarantee placement and withdrawal.

Dynamic Event Automata (DEA):




```
DEA DelayedGuaranteeWithdrawal {
    states {
        AwaitingDeposit: initial;
        GuaranteePlaced;
        Withdrawn;
    }
    transitions {
        AwaitingDeposit -[after(placeGuarantee())]-> GuaranteePlaced;
        GuaranteePlaced -[after(withdrawGuarantee()) && block.timestamp >=
guaranteeWithdrawalTime]-> Withdrawn;
    }
}
```

LARVA Code Explanation:

For this property, we want to introduce a delay between placing a guarantee and being able to withdraw it. We create a new `DEA DelayedGuaranteeWithdrawal`:

- ***States:*** The DEA has three states: `AwaitingDeposit`, `GuaranteePlaced`, and `Withdrawn`.
- ***Transitions:***
    i. From `AwaitingDeposit` to `GuaranteePlaced` after the sender places a guarantee.
    ii. From `GuaranteePlaced` to `Withdrawn` after the sender successfully withdraws the guarantee and the specified withdrawal time has passed.