Formal Verification of Aave’s protocol-V2

Summary

This document describes the specification and verification using Certora Prover of Aave’s protocol-V2. The work was undertaken from August 2nd through October 29th, 2020. The latest commit that was reviewed and run through the Certora Prover was 750920303e33b66bc29862ea3b85206dda9ce786.

The scope was the Aave Protocol, a decentralized, open-source, and non-custodial money market protocol. Depositors can deposit assets into the protocol and earn interest on their deposits without relying on middlemen. The main contracts considered were the LendingPool, AToken, StableDebtToken, and VariableDebtToken. In addition, the UserConfiguration library was verified independently of the calling contracts.

The Certora Prover proved the implementation of the Aave protocol correct with respect to the formal rules written by the Aave and the Certora teams. During the verification process, the Certora Prover and the team’s manual review discovered a number of bugs in the code listed in the table below in Section 1.1. All the high-severity issues were promptly corrected, and the fixes were verified to satisfy the specifications up to limitations of the Certora Prover. These limitations are currently being handled by the Certora development team. Section 2 formally defines high level specifications of the protocol. All the rules are publically available in a public github: https://github.com/aave/protocol-v2/tree/master/specs

List of Main Issues Discovered

<table>
<thead>
<tr>
<th>Issue</th>
<th>Rule broken</th>
<th>Description</th>
<th>Severity</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss of user’s assets</td>
<td>Total assets of user</td>
<td>The system does not return excess payment in repay and deposit transactions</td>
<td>High</td>
<td>The ETH specific code has been removed in favor of a pull payment strategy using ERC20</td>
</tr>
<tr>
<td>Issue</td>
<td>Description</td>
<td>Severity</td>
<td>Recommendation</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>----------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Lock of assets</td>
<td>Deposit is always possible. Due to chains of links in redirection of interest, user's transaction reverts without an option to remove the chain and their assets are locked.</td>
<td>High</td>
<td>Postpone redirection feature to a new version.</td>
<td></td>
</tr>
<tr>
<td>System loss of assets</td>
<td>Additive burn. No gain due to redirection. Due to a missing update to a user's time-tracking property, a user burns his entire balance, but is left with some balance.</td>
<td>High</td>
<td>Postpone redirection feature to a new version.</td>
<td></td>
</tr>
<tr>
<td>Increase in user debt</td>
<td>Additive repay. Due to a missing update to a user's time-tracking property, a user pays his debt, but in fact his debt grows.</td>
<td>High</td>
<td>Update user's property.</td>
<td></td>
</tr>
<tr>
<td>Lock of assets</td>
<td>balanceOf should never revert. Due to division by zero, balanceOf reverts.</td>
<td>High</td>
<td>Different calculation of accrued interest.</td>
<td></td>
</tr>
<tr>
<td>Overestimating system's assets</td>
<td>When repaying, the system neglects the amount that is going to be reimbursed when computing the new interest rate.</td>
<td>High</td>
<td>Fixed computation of interest rate.</td>
<td></td>
</tr>
<tr>
<td>System loss of assets, user gain of assets</td>
<td>Additive burn (AToken). Due to rounding in conversions to Atoken, if the conversion rate is high enough, one can withdraw a small amount that will result in the system transferring underlying tokens but burning zero ATokens of the user's account.</td>
<td>Medium</td>
<td>Fixed to avoid transfer on burn of zero ATokens.</td>
<td></td>
</tr>
<tr>
<td>System loss of assets, user gain of assets, user loss of assets</td>
<td>Additive mint (Stable debt token). Due to rounding in conversions to stable debt token, if the conversion rate is high enough, one can deposit a small amount that will result in the system transferring underlying tokens but minting debt tokens to the user's account.</td>
<td>Medium</td>
<td>Fixed to avoid transfer on mint/burn of zero stable debt tokens.</td>
<td></td>
</tr>
<tr>
<td>Reduce health factor,</td>
<td>Valid operation. In case of a very low health factor, due to having the bonus as a fixed.</td>
<td>Medium</td>
<td>Known issue from V1.</td>
<td></td>
</tr>
</tbody>
</table>
Notations

1. ✔ indicates the rule is formally verified on the latest commit. We write ✔* when the rule was verified on a simplified version of the code that ignores rounding errors.
2. ☞ indicates the rule is not yet formally specified.
3. 🔁 indicates the rule is only applicable to a future version of the code with redirections.
4. We define ε to be the allowed rounding error and enforce a global invariant that $\epsilon \leq \frac{\text{normalizedIncome}}{\text{ray}}$.
5. We use $|.|$ to denote the absolute value of a numeric value.
6. We use Hoare triples of the form $\{p\} C \{q\}$, which means that if program C executes starting in any state satisfying $p$, then it will end in a state satisfying $q$. In Solidity, $p$ is similar to require and $q$ is similar to assert. The syntax $\{\text{Pre}\} P_1 \sim P_2 \{\text{Post}\}$ is a generalization of Hoare rules, called relational properties. Pre is a requirement on the states before $P_1$ and $P_2$ and $\{\text{Post}\}$ describes the states after their executions. Notice that $P_1$ and $P_2$ operate in different states. As a special case, $P_1 \sim g P_2$, where $g$ is a getter function (e.g., getBalance()), indicating that $P_1$ and $P_2$ result in states in which the value of $g()$ is the same.

In a hoare triple $\{p\} P_1 \sim P_2 \{q\}$, we may refer to the value of a getter $g()$ after $P_1$ or $P_2$ by the notation of $g_1()$ or $g_2()$, respectively.
Verification of AToken

Functions:

**totalSupply() : uint**
Returns the total supply.

**balanceOf(address u) : uint**
Returns the balance of address \( u \) with interest and redirected interest.

**normalizedIncome() : uint**
Returns the reserve’s current normalized income index in high precision format (ray).

**burn(address u, address t, uint x) : bool**
Returns true if successfully burns \( x \) ATokens from the account of \( u \) and sends \( x \) underlying tokens to \( t \).

**transfer(address u, address t, uint x)**
Transfers \( x \) ATokens from the account of \( u \) to \( t \).

Properties:

1. **Integrity of totalSupply ✔️**
   TotalSupply is the sum of all users' balances.
   
   \[ \text{totalSupply()} = \sum_{\text{address } u} \text{balanceOf}(u) \]

2. **Integrity of mint ✔️**
   Minting an amount of \( x \) tokens for user \( u \) increases their balance by \( x \), up to rounding errors (\( \varepsilon \)).
   
   \[ \{ b = \text{balanceOf}(u) \} \text{mint}(u,x) \{ b + x - \varepsilon \leq \text{balanceOf}(u) \leq b + x + \varepsilon \} \]

3. **Additivity of mint ✔️**
   Minting is additive, i.e., it can be performed either all at once or in steps.
   
   \[ \text{mint}(u,x); \text{mint}(u,y) - \text{mint}(u,x+y) \{ | \text{balanceOf}(u) - \text{balanceOf}_{2}(u) | \leq 3\varepsilon \} \]
4. **Integrity of transfer ✔**

Transferring $x$ tokens from user $u$ to user $u'$ changes the balance of $u$ and $u'$ as expected and does not change the underlying asset balance of any user or of the system.

I. \{ \begin{align*} & u \neq u' \land \text{balanceOf}(u) = \text{balanceOf}(u') \} \\
& \text{transfer}(u, u', x); \\
& \{ \begin{align*} & | \text{balanceOf}(u) - (\text{balanceOf}(u') - x) | \leq \epsilon \\
& | \text{balanceOf}(u') - (\text{balanceOf}(u) - x) | \leq \epsilon \end{align*} \}
\}

II. \{ \begin{align*} & \text{underlyingAssetBalanceOf}(u'') \} \\
& \text{transfer}(u, u', x); \\
& \{ b = \text{underlyingAssetBalanceOf}(u'') \}
\}

5. **Additivity of transfer ✔**

Transfer is additive, i.e., it can be performed either all at once or in steps.

\text{transfer}(u, u', x); \text{transfer}(u, u', y) \sim \text{transfer}(u, u', x+y)

6. **Integrity of burn ✔**

Transfer of $x$ amount of tokens from a user $u$ to a receiver user $u'$

\{ \begin{align*} & \text{balanceOf}(u) \land \text{underlyingAssetBalanceOf}(u') \} \\
& \text{burn}(u, u', x); \\
& \{ | \text{balanceOf}(u) - (\text{balanceOf}(u') - x) | \leq \epsilon \land \\
& u' \neq \text{AToken} \Rightarrow | \text{underlyingAssetBalanceOf}(u') - (\text{balanceOf}(u') - x) | \leq \epsilon \}
\}

7. **Additivity of burn ✔**

Burning is additive, i.e., it can be performed either all at once or in steps.

\text{burn}(u, u', x); \text{burn}(u, u', y) \sim \text{burn}(u, u', x+y) \text{ at the same timestamp}
8. Revert characteristic of burn ✓*

Burning of \(x\) token of user \(u\) when transferring underlying token to receiver \(t\) succeeds when the following holds:

I. \(msg\).\(sender\) is the LendingPool contract and no value sent,

II. \(totalSupply\) in AToken is more than \(amount\) to burn,

III. The system has enough underlying tokens to transfer

IV. user \(u\) which burns his asset is valid and has enough balance

V. Receiver \(t\) is valid and transferring him underlying token will not overflow

\[
\{ 
\text{msg.value} = 0 \land \text{msg.sender} = \text{LendingPool} \land \\
\text{totalSupply()} \geq x \land \\
\text{underlyingAssetBalanceOf(AToken)} > x \land \\
u \neq 0 \land \text{balanceOf(u)} < x \\
t \neq 0 \land \text{underlyingAssetBalanceOf(t)} + x < \text{MAXINT} \}
\]

\(r = \text{burn}(u, to, x)\)

\[\{r\}\]
Verification of StableDebtToken

9. Integrity of getUserLastUpdated ✓*
   The time of the last update to a user cannot lie in the future.

   \[
   \text{getUserLastUpdated}(u) \leq \text{block.timestamp};
   \]

10. Integrity of totalSupply ✓*
    TotalSupply is the sum of all users’ balances.

    \[
    \text{totalSupply}(t) = \sum \text{address } u \text{ balanceOf}(u,t)
    \]
    
    This is proven by showing properties 11 and 12 below:

11. No two balances updated ✓*
    Each possible operation changes the balance of at most one user.

12. Integrity balance and total supply ✓*
    Check that the changes to total supply are coherent with the changes to balance for each operation.

13. Integrity of mint ✓*
    Minting an amount of \( x \) tokens for user \( u \) increases their balance by \( x \), up to rounding errors.

    \[
    \{ \text{b = balanceOf}(u, t) \} \\
    \text{mint}(u, x, \text{index}) \\
    \{ \text{balanceOf}(u, t) = b + x \}
    \]

14. Additivity of mint ✓*
    Minting is additive, i.e., it can be performed either all at once or in steps.

    \[
    \text{mint}(u,x); \text{mint}(u,y) \sim \text{mint}(u,x+y)
    \]

15. Integrity of burn ✓*
    Transfer of \( x \) amount of tokens from user \( u \) where receiver is user \( u' \)
    \[
    \{ \text{bu = balanceOf}(u) \} \\
    \text{burn}(u, u', x) \\
    \{ \text{balanceOf}(u) = bu - x \}
    \]
16. Additivity of burn ✔*

Burning is additive, i.e., it can be performed either all at once or in steps.

\[ \text{burn}(u, u', x); \text{burn}(u, u', y) \sim \text{burn}(u, u', x+y) \]

17. Invertibility of mint and burn ✔*

Minting and burning are inverse operations.

\[ \{ \text{bu} = \text{balanceOf}(u) \} \]
\[ \text{mint}(u,x); \text{burn}(u, u, x) \]
\[ \{ \text{balanceOf}(u) = \text{bu} \} \]
Verification of VariableDebtToken

18. **Integrity of totalSupply ✔**
   TotalSupply is the sum of all users' balances.

   \[
   \text{totalSupply}(t) = \sum \text{balanceOf}(u, t)
   \]

   This is proven by showing properties 19 and 20 below:

19. **No two balances updated ✔**
   Each possible operation changes the balance of at most one user.

20. **Integrity balance and total supply ✔**
   The changes to total supply are coherent with the changes to balance for each operation.

21. **Integrity of mint ✔**
   Minting an amount of \( x \) tokens for user \( u \) increases their balance by \( x \), up to rounding errors.

   \[
   \begin{align*}
   &\{ \text{b} = \text{balanceOf}(u, t) \} \\
   &\text{mint}(u, x, \text{index}) \\
   &\{ \text{balanceOf}(u, t) = \text{b} + x \}
   \end{align*}
   \]

22. **Additivity of mint ✔**
   Minting is additive, i.e., it can be performed either all at once or in steps.

   \[
   \text{mint}(u, x); \text{mint}(u, y) \rightarrow \text{mint}(u, x+y)
   \]

23. **Integrity of burn ✔**
   Transfer of \( x \) amount of tokens from user \( u \) where receiver is user \( u' \)

   \[
   \begin{align*}
   &\{ \text{bu} = \text{balanceOf}(u) \} \\
   &\text{burn}(u, u', x) \\
   &\{ \text{balanceOf}(u) = \text{bu} - x \}
   \end{align*}
   \]

24. **Additivity of burn ✔**
   Minting is additive, i.e., it can be performed either all at once or in steps.

   \[
   \text{burn}(u, u', x); \text{burn}(u, u', y) \rightarrow \text{burn}(u, u', x+y)
   \]
25. Inverse of mint and burn ✓*

Minting and burning are inverse operations.

\begin{verbatim}
{ bu = balanceOf(u) }
    mint(u,x); burn(u, u, x)
{ balanceOf(u) = bu }
\end{verbatim}
Verification of UserConfiguration Library

This library implements a bitmap logic to handle the user configuration, it stores whether a user has borrowed or and deposited for collateral for each possible asset.

Functions:

setBorrowing(address u, uint256 reserveId, bool val)
Sets that user \( u \) borrowing flag for \( \text{reserveId} \) to \( \text{val} \)

setUsingAsCollateral(address u, uint256 reserveId, bool val)
Sets that user \( u \) is using \( \text{reserveId} \) as collateral to \( \text{val} \)

isUsingAsCollateralOrBorrowing(address u, uint256 reserveId) : bool
Returns true if the user \( u \) has been using a reserve \( \text{reserveId} \) for borrowing or as collateral

isBorrowing(address u, uint256 reserveId) : bool
Returns true if user \( u \) has been using a \( \text{reserveId} \) for borrowing

isUsingAsCollateral(address u, uint256 reserveId) : bool
Returns true if user \( u \) has been using a \( \text{reserveId} \) as collateral

isBorrowingAny(address u) : bool
Returns true if user \( u \) has been borrowing any reserve

isEmpty(address u) : bool
Returns true if user \( u \) has been borrowing or using as collateral any reserve
Properties:

26. Integrity of getters ✔
For each user $u$ and reserve id $r$:
I. $\text{isEmpty}(u) \Rightarrow
   (-\text{isBorrowingAny}(u) \land -\text{isUsingAsCollateralOrBorrowing}(u,r))$
II. $(\text{isBorrowingAny}(u) \lor \text{isUsingAsCollateral}(u,r)) \Rightarrow -\text{isEmpty}(user)$
III. $\text{isBorrowing}(u,r) \Rightarrow \text{isBorrowingAny}(u)$
IV. $(\text{isUsingAsCollateral}(u,r) \lor \text{isBorrowing}(u,r)) \Leftrightarrow \text{isUsingAsCollateralOrBorrowing}(u,r)$

27. Integrity of setBorrowing ✔
When setting the flag of user $u$ borrowing from reserve $r$, the value is updated as expected.
\{ $r < 128$ \}
\{ $\text{setBorrowing}(u, r, b)$ \}
\{ $\text{isBorrowing}(u,r) = b$ \}

28. No effect on other on setBorrowing ✔
When setting the flag of user $u$ borrowing from reserve $r$, the flags concerning reserve $r'$ are not changed.
\{ $r' \neq r \land r' < 128 \land r < 128 \land \text{borrow} = \text{isBorrowing}(u, r') \land \text{collateral} = \text{isUsingAsCollateral}(u,r') \}$
\{ $\text{setBorrowing}(u, r, b)$ \}
\{ $\text{borrow} = \text{isBorrowing}(u, r') \land \text{collateral} = \text{isUsingAsCollateral}(u,r')$ \}

29. Integrity of setUsingAsCollateral ✔
When setting the flag of user $u$ using as collateral reserve $r$, the value is updated as expected.
\{ $r < 128$ \}
\{ $\text{setUsingAsCollateral}(u,r,b)$ \}
\{ $\text{isBorrowing}(u,r) = b$ \}
30. No effect on other on setUsingAsCollateral ✓

When setting the flag of user $u$ using as collateral reserve $r$, the flags concerning reserve $r'$ are not changed.

$$\{ r' \neq r \land r' < 128 \land r < 128 \land \text{borrow} = \text{isBorrowing}(u, r') \land$$

$$\text{collateral} = \text{isUsingAsCollateral}(u, r') \}$$

setUsingAsCollateral($u$, $r$, $b$)

$$\{ \text{borrow} = \text{isBorrowing}(u, r') \land$$

$$\text{collateral} = \text{isUsingAsCollateral}(u, r') \}$$
Verification of LendingPool

Functions:
Properties of the system that have no side-effect (view functions)

\[
\text{getAToken}(\text{Token } t) : \text{AToken}
\]
Returns the AToken associated with a reserve of token \( t \).
i.e., reserves[\( t \)].aTokenAddress

\[
\text{getStableDebtToken}(\text{Token } t) : \text{DebtToken}
\]
i.e., reserves[\( t \)].stableDebtTokenAddress

\[
\text{getVariableDebtToken}(\text{Token } t) : \text{DebtToken}
\]
i.e., reserves[\( t \)].variableDebtTokenAddress

\[
\text{healthFactor}(\text{address } a) : \text{uint}
\]
Returns the health factor of user \( a \)

\[
\text{totalDebt}(\text{address } a) : \text{uint}
\text{totalCollateralETH}(\text{address } a) : \text{uint}
\]

\[
\text{token.balanceOf}(\text{address } a) : \text{uint}
\]
Returns the balanceOf address \( a \) in \( \text{token} \)

\[
\text{token.totalSupply}() : \text{uint}
\]
Returns the total supply of \( \text{token} \).

Operations:
All operations return true on successful execution of the operation (i.e., did not revert).
These operations have side effects on the state of the system.

\[
\text{deposit}(\text{Actor } a, \text{Token } t, \text{uint } x, \text{Actor } b) : \text{bool}
\]
Actor \( a \) deposits on behalf of actor \( b \), in the reserve of \( x \) amount of token \( t \).
Returns true on successful deposit (i.e., did not revert)

\[
\text{withdraw}(\text{Actor } a, \text{Token } t, \text{uint } x) : \text{bool}
\]
Actor \( a \) withdraws \( x \) amount of token \( t \)
borrow(Actor a, Token t, uint x, Actor b) : bool
Actor a borrows on behalf of actor b amount of x tokens t

repay(Actor a, Token t, uint x, Actor b) : bool
Actor a repays on behalf of actor b amount of x tokens t

swap(Actor a, Token t) : bool
Swap the rate of debt of Actor a for token t

rebalanceStableBorrowRate(Token t, Actor a)
FlashLoan
LiquidationCall

Properties:

31. Integrity of a reserve ✔*

The balance of a reserve for asset t is at least the sum of all deposits minus the borrowed.

I.  t.balanceOf(getAToken(t)) ≥ getAToken(t).totalSupply() -
    ( getStableDebtToken(t).totalSupply() +
      getVariableDebtToken(t).totalSupply() )

II. reserve.liquidityIndex ≤ reserve.getNormalizedIncome()

32. Total assets of user is preserved ✔*

The total assets with regard to some (external) token t is preserved. Need to take into account that operation can change the balance of two actors

We define total assets of user u with respect to token t

\[ \text{totalAssets}(t, u) \equiv t.\text{balanceOf}(u) + \text{getAToken}(t).\text{balanceOf}(u) - \]
\[ (\text{getStableDebtToken}(t).\text{balanceOf}(u) + \text{getVariableDebtToken}(t).\text{balanceOf}(u)) \]

\{ x = \text{totalAssets}(t, a) \}
    op;
\{ \text{totalAssets}(t, a) = x\}

Where op is an operation that affects the asset of at most one user
For case where \( op \) can change the balance of actor \( a \) performing on operation behalf of actor \( b \) the property is:

\[
\{ x = \text{totalAssets}(t, a) + \text{totalAssets}(t, b) \} \\
\text{borrow}(a, t, x, b) \land \text{deposit}(a, t, x, b); \\
\{ \text{assets}(t, a) + \text{totalAssets}(t, b) = x \} 
\]

33. **One can always deposit**

assuming the system is in active state, one can always deposit

\[
\{ \text{isActive}(t) \land \neg \text{isFreeze}(t) \land x \neq 0 \} \text{ ret } = \text{deposit}(u, t, x) \{ \text{ ret } \} 
\]

34. **Revert characteristic of deposit**

Deposit fails if the reserve is in inactive or freeze state or the user attempts to deposits 0 tokens

\[
\{ \neg \text{isActive}(t) \lor \text{isFreeze}(t) \lor x = 0 \} \text{ ret } = \text{deposit}(u, t, x) \{ \neg \text{ ret } \} 
\]

35. **Integrity of deposit ✔️**

When actor \( u \) deposits \( x \) tokens of asset \( t \) on behalf of actor \( b \) (can be \( a \))

The asset balance of \( u \) is decreased and the aToken of \( b \) is increased.

\[
\{ t_ = t.\text{balanceOf}(u) \land a_ = \text{getAToken}(t).\text{balanceOf}(b) \} \\
\text{deposit}(u, t, x, b); \\
\{ t.\text{balanceOf}(u) = t_ - x \land \text{getAToken}(t).\text{balanceOf}(b) = a_ + x \} 
\]

36. **Integrity of borrow ☞**

When actor \( u \) deposits \( x \) tokens of asset \( t \) on behalf of actor \( b \) (can be \( a \))

The asset balance of \( u \) is increased and the debt of \( b \) is increased.

\[
\{ t_ = t.\text{balanceOf}(u) \land a_ = \text{totalDebt}(b) \} \\
\text{borrow}(u, t, x, b); \\
\{ t.\text{balanceOf}(u) = t_ + x \land \text{totalDebt}(b) = a_ + x \} 
\]

37. **Revert characteristic of borrow**

38. **Total assets and debts of user is not influenced by others ☞**

39. **Operations at one interest rate does not change the other interest rate type debt ☞**

40. **Variable debt token is not influenced by stable rate operations ☞**
41. Each operation affects at most one reserve

42. Valid Operations with respect to health factor
\[
\{ \text{healthFactor}(a) > 1 \} \text{ Op } a \{ \text{healthFactor}(a) \geq 1 \}
\]

\[
\{ h = \text{healthFactor}(a) < 1 \} \text{ Op } \{ \text{healthFactor}(a) < h \}
\]

43. Inverse operations
Deposit; Op; Redeem \sim skip
Borrow; Op; Repay \sim skip
Where op is any operation by other users or specific operation by the user.
Here we mean that these operations return the same values that are important to us.

44. Additivity properties at the same timestamp
Deposit(x); Deposit(y) \sim Deposit(x+y)
Redeem(x); Redeem(y) \sim Redeem(x+y)
Borrow(x); Borrow(y) \sim Borrow(x+y)
Repay(x); Repay(y) \sim Repay(x+y)
Properties regarding redirection of interest

Properties that have been excluded for the current release of the AToken contract.

Functions:

getInterestRedirectionAddress(address u) : address
Returns the address to which address \( u \) redirects the interest to.

getRedirectedBalance(address u) : uint
Returns the total amount of balance redirected to address \( u \) from all redirecting users.

Properties:

45. Safety of redirection
   One can not redirect interest to himself.
   \[ \text{getInterestRedirectionAddress}(u) \neq u \]

46. Integrity when no redirection
   If account \( u \) does not redirect and has no redirection than the balance of equals the
   principle balance.
   \[ ( \text{getInterestRedirectionAddress}(u) = 0 \land \text{getRedirectedBalance}(u) = 0 ) \Rightarrow \]
   \[ \text{balanceOf}(u, t) = \text{calculateCumulatedBalance}(u, \text{principalBalanceOf}(u), t) \]

47. Integrity of interest redirection
   The balance of a user \( v \) that has redirection from user \( u \) but no principalBalance is the
   interest of user \( u \)
   \[ ( \text{amount} = \text{principalBalanceOf}(u) = \text{getRedirectedBalance}(v) \land \]
   \[ \text{principalBalanceOf}(v) = 0 ) \Rightarrow \]
   \[ \text{calculateCumulatedBalance}(u, \text{amount}, t) - \text{amount} = \text{balanceOf}(v, t) \]

48. Integrity of interest redirection on transactions
   The balance of a user \( v \) that has redirection from user \( u \) but no principalBalance is the
   interest of user \( u \).
   \[ \{ \text{amount} = \text{principalBalanceOf}(u) = \text{getRedirectedBalance}(v) \land \]
   \[ \text{principalBalanceOf}(v) = 0 \land t' \geq t \} \text{ Op_t } \{ \]
   \[ \text{calculateCumulatedBalance}(u, \text{amount}, t') - \text{principalBalanceOf}(u) = \]
   \[ \text{balanceOf}(v, t') \}
   Where \( \text{Op_t} \) is an operation performed at time \( t \) beside: redirectInterestStream,...
49. No gain due to redirection of interest

The total balance of two users, say \( u_1 \) and \( u_2 \), is not affected if one redirects the interest to the other.

\[
\text{Op } x = \text{balanceOf}(u_1, s) + \text{balanceOf}(u_2, s) /*scenario 1*/
\]

\[
\text{redirectInterestStream}(u_1, u_2); \text{Op } ;
\]

\[
y = \text{balanceOf}(u_1, s) + \text{balanceOf}(u_2, s) /*scenario 2*/
\]

\[
\{ x == y \}
\]
Properties suggested by Aave

Global invariants

- **P1**: Whenever a user transfers aTokens from A -> B, the balanceOf(aToken) of the underlying asset transferred does not change ([Integrity of the system](#)).
- **P2**: Given an asset A with $B_{sa} = \text{total borrowed at stable of } A$, any action of borrowing/repaying/liquidating at variable rate does not change $B_{sa}$.
- **P3**: Given an asset A with $B_{va} = \text{total borrowed at variable of } A$, any action of borrowing/repaying/liquidating at stable rate does not change $B_{va}$.
- **P4**: Given a user U and a set of borrowed assets A, with $B_n = \{B_{a1}, B_{a2}, ..., B_{an}\}$ the set of borrow balances for each asset $a \in A$, if U borrows another asset $a_{n+1}$, $B_{an+1}$ is equal to the amount of the asset $a_{n+1}$ being borrowed and all the balances $B_1...B_n$ are unchanged (explanation: borrowing a new assets doesn’t affect other users positions).
- **P5**: Given a user U and a set of borrowed assets A, with $B_n = \{B_{a1}, B_{a2}, ..., B_{an}\}$ the set of borrow balances for each asset $a \in A$, if U repays an asset repays an amount $M$ for any asset $a_x \in A$, then $B_{ax} = B_{ax} - M$ and all the other balances are unaffected (explanation: repaying a loan does not affect other loans).
- **P6**: On any action on a specific asset:
  - If the asset is not being borrowed at a variable rate, the variable borrow index doesn’t change.
  - If the asset is being borrowed at a variable rate, the variable borrow index increases, unless the following happens:
    - The index has been updated by another action in the same block.
    - The interest accrued is so small that is below $10^{-27}$.

  In these conditions, the variable borrow index remains the same.

- **P7**: On any action on a specific asset:
  - If the asset is not being borrowed, the liquidity index doesn’t change.
  - If the asset is being borrowed, the liquidity index increases, unless the following happens:
    - The index has been updated by another action in the same block.
    - The interest accrued is so small that is below $10^{-27}$.
In these conditions, the variable borrow index remains the same

State transitions

Deposit()

- P6: It's not possible to deposit in an inactive reserve.
- P7: It's not possible to deposit in a frozen reserve.
- P8: It's not possible to deposit 0 amount.
- P9: When a user deposits X amount of an asset, his collateral in the system is increased by X. Integrity of deposit
- P10: When a user deposits X amount of an asset, he receives exactly X amount of the corresponding aToken. Integrity of deposit
- P11: When a user deposits X amount of an asset, the protocol balance on that asset is increased by X. Integrity of the system + Integrity of deposit
- P12: On any deposit, if the asset is being borrowed at variable(totalSupply(variableDebtToken) > 0 for the particular asset), the variable borrow rate goes down as long as the deposited amount isn’t so small that the impact on the variable borrow rate is below the margin of error (10^-27)
- P13: On any deposit, if the asset is being borrowed at stable (totalSupply(stableDebtToken) > 0 for the particular asset), the stable borrow rate goes down as long as the deposited amount isn’t so small that the impact on the variable borrow rate is below the margin of error (10^-27)
- P14: On any deposit, if the asset is being borrowed (totalSupply(stableDebtToken) + totalSupply(variableDebtToken) > 0 for the particular asset), the liquidity rate goes down as long as the deposited amount isn’t so small that the impact on the liquidity rate is below the margin of error 10^-27.

Borrow()

- P15: It's not possible to borrow in an inactive reserve.
- P16: It's not possible to borrow in a frozen reserve.
- P17: It's not possible to borrow on a reserve disabled for borrowing.
- P18: It's not possible to borrow 0 amount.
- P19: It's not possible to borrow with an interest rate mode that is different than 1 (stable) or 2 (variable)
- P20: It's not possible to borrow more than the available liquidity in the reserve.
- P21: It's not possible to borrow at a stable rate in a reserve where the stable rate is not enabled.
- P22: It's not possible to borrow at a stable rate more than the percentage over the available liquidity defined by the parameters returned by `getMaxStableRateBorrowSizePercent()`.
- P23: A borrower whose health factor (HF) is below 1 can't borrow. [Valid Operation with respect to health factor]
- P24: A borrower can only borrow up to the amount which would set the HF to 1 (alternatively written, after a borrow action the HF of the borrower is always > 1) [Valid Operation with respect to health factor]
- P25: Whenever a user borrows an asset at a variable rate, the `balanceOf(user)` on the `VariableDebtToken` increases by the amount borrowed.
- P26: Whenever a user borrows an asset at a stable rate, the `balanceOf(user)` on the `StableDebtToken` increases by the amount borrowed.
- P27: When a user borrows X amount of an asset they will receive exactly X amount of that asset. [Integrity of borrow]
- P28: When a user borrows X amount of an asset, the protocol balance of that asset is decreased by X. [Integrity of borrow]
- P29: On any borrow, the variable rate goes up as long as the borrowed amount isn't so small that the impact on the variable borrow rate is below the margin of error (10^-27).
- P30: On any borrow, the stable rate goes up as long as the borrowed amount isn't so small that the impact on the variable borrow rate is below the margin of error (10^-27).

**withdraw()**

- P31: It's not possible to withdraw from an inactive reserve.
- P32: It's not possible to withdraw a 0 amount.
- **P33**: A user can't withdraw more than his current aToken balance. [Total assets of user is preserved]
- **P34**: A user can't withdraw more than the liquidity available in the reserve (the total deposited - the borrowed).
- **P35**: A user can't withdraw an amount that would set his Health Factor below 1e18 (alternatively written, after a `redeem()` the HF of the user can't be < 1e18).
- P36: After withdraw, the withdrawn amount is subtracted from the user aToken balance.
● P37: After withdraw, the user receives the exact withdrawn amount in the underlying token.
● P38: Whenever a user withdraws, the balanceOf(aToken) for the underlying asset decreases of the amount withdrawn
● P39: On any withdraw, if the asset is being borrowed at variable(totalSupply(variableDebtToken) > 0 for the particular asset), the variable borrow rate goes up as long as the withdrawn amount isn’t so small that the impact on the variable borrow rate is below the margin of error (10^-27)
● P40: On any withdraw, if the asset is being borrowed at stable (totalSupply(stableDebtToken) > 0 for the particular asset), the stable borrow rate goes up as long as the withdrawn amount isn’t so small that the impact on the variable borrow rate is below the margin of error (10^-27)
● P41: On any deposit, if the asset is being borrowed (totalSupply(stableDebtToken) + totalSupply(variableDebtToken) > 0 for the particular asset), the liquidity rate goes up as long as the withdrawn amount isn’t so small that the impact on the liquidity rate is below the margin of error 10^-27.

Critical properties to ensure no funds can be trivially stolen

- **P9:** When a user deposits X amount of an asset, his collateral in the system is increased by X. **Integrity of deposit**
- **P10:** When a user deposits X amount of an asset, he receives exactly X amount of the corresponding aToken. **Integrity of deposit**
- **P11:** When a user deposits X amount of an asset, the protocol balance on that asset is increased by X. **Integrity of the system + Integrity of deposit**
- **P23:** A borrower whose health factor is below 1 can’t borrow. **Valid Operation with respect to health factor**
- **P24:** A borrower can only borrow up to the amount which would set the HF to 1 (alternatively written, after a borrow action the HF of the borrower is always > 1) **Valid Operation with respect to health factor**
- **P25:** Whenever a user borrows an asset at a variable rate, the balanceOf(user) on the VariableDebtToken increases by the amount borrowed
- **P26:** Whenever a user borrows an asset at a stable rate, the balanceOf(user) on the StableDebtToken increases by the amount borrowed
- **P27:** When a user borrows X amount of an asset they will receive exactly X amount of that asset. **Integrity of borrow**
- P28: When a user borrows X amount of an asset, the protocol balance of that asset is decreased by X. **Integrity of borrow**
Disclaimer

The Certora Prover takes as input a contract and a specification and formally proves that the contract satisfies the specification in all scenarios. Importantly, the guarantees of the Certora Prover are scoped to the provided specification, and any cases not covered by the specification are not checked by the Certora Prover.

We hope that this information is useful, but provide no warranty of any kind, express or implied. The contents of this report should not be construed as a complete guarantee that the Aave protocol is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.