Formal Verification Report of AAVE Token V3

Summary

This document describes the specification and verification of AAVE Token V3 using the Certora Prover. The work was undertaken from July 15th to August 10th, 2022. The latest commit reviewed and run through the Certora Prover was e869de2 on September 27th 2023.

The scope of this verification is AAVE token V3 code which includes the following contracts:

- `AaveV3Token.sol`

And its parent contracts:

- `BaseAaveToken.sol`
- `BaseAaveTokenV2.sol`

This project has been a part of a joint Certora and Aave community program. Contributors from the community have conducted independent formal verification of the code, where Certora has provided an initial setup for writing a specification.

19 out of the 25 community participants submitted spec files containing formal specifications resulting in 275 properties in total. Out of the 275 correctness rules, 240 quality rules passed our professional review and credited their authors with grants.

Selected rules written by the community are included in this report in the *Community* section.

Certora also performed a manual audit of these contracts.

During this verification process, the Certora Prover discovered issues in the code which are listed in the tables below.

All the rules and specification files are publicly available and can be found in AAVE Token V3 repository.

List of Main Issues Discovered

Severity: Low

Discovered independently by the following contributors: https://github.com/Elpacos https://github.com/himanshu-Bhatt https://github.com/jessicapointing and Certora team.

Discover

Issue:	Precision loss during voting power transfer
Description:	When calculating delegated balance on token transfer, the new delegated balance of a delegate was calculated with a small precision loss that violated the property $delegatee1Power_{t1} = delegatee1Power_{t0} - z/10^{10} * 10^{10}$ after a delegator to delegatee1 transfers z amount of tokens.
Property Violated:	vpTransferWhenOnlyOneIsDelegating (Property #6) and others
AAVE Response:	The issue was fixed in commit a287d134 and the relevant property was modified to be $delegatee1Power_{t1} = delegatee1Power_{t0} - account1Balance_{t0}/10^{10} * 10^{10} + account1Balance_{t1}/10^{10} * 10^{10}$

List of Issues Discovered Independently By The Community

Severity: High

Found by the following contributors: https://github.com/Elpacos

Issue:	Wrong parameters order in a `_transferWithDelegation` call
Description:	This issue was present in an intermediary version of the code given to the community to verify, but not in the finalized version that Certora has verified. It was introduced for a short period of time during development, and immediately fixed by the AAVE team.
Property Violated:	multiple properties
AAVE Response:	The issue was fixed in commit 190c03f4

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 the Certora Prover does not check any cases not covered by the specification.

We hope that this information is useful, but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract 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.

Summary of Formal Verification

Overview of the AAVE Token V3

`AaveV3Token.sol` is the main contract. It inherits from `BaseAaveToken.sol` and `BaseAaveTokenV2.sol`.

The following description is taken from the token repository:

AAVE is an ERC20 token deployed on Ethereum, which main utility is participating in the Aave governance system via voting on proposals or creating them.

AAVE is a transparent proxy contract, and its current implementation is version 2.

Together with all the standard ERC20 functionalities, the current implementation includes extra logic mainly for the management and accounting of voting and proposition power. Due to the design/architecture of the Aave governance v2 system, of which AAVE is the main voting asset, the current AAVE implementation makes the token transfers quite gas-consuming, as multiple snapshots of data (voting and proposition power) need to be stored all the time.

With a new iteration of the Aave governance in the Aave/BGD roadmap down the line, snapshots on the token will not be required anymore for its integration with the governance system. So this new version 3 of AAVE consists mainly of removing the snapshotting, together with adding extra minor meta-transactions capabilities.

Assumptions and Simplifications Made During Verification

The invariants in `general.spec` were proven on a slightly modified version of the token code. To bypass a current limitation of the Certora prover, we've refactored the `delegationState` field of the `_balances` struct to be a `uint8` instead of a `DelegationState` enum type. The `AaveV3Token.sol` code was modified to accomodate this change.

These changes can be seen in the patch file in the Certora branch of the token repository.

To create this harness, we run `make munged` command from the `certora` directory (on the `certora` branch).

- We unroll loops. Violations that require a loop to execute more than twice will not be detected.

Notations

✓ indicates the rule is formally verified on the latest reviewed commit. We write ✓* when the rule was verified on the simplified assumptions described above in "Assumptions and Simplifications Made During Verification".

❌ indicates the rule was violated under one of the tested versions of the code.

🔁 indicates the rule is timing out.

Our tool uses Hoare triples of the form {p} C {q}, which means that if the execution of program C starts in any state satisfying p, it will end in a state satisfying q. This logical structure is reflected in the included formulae for many of the properties below. Note that p and q here can be thought of as analogous to `require` and `assert` in Solidity.

The syntax {p} (C1 ~ C2) {q} is a generalization of Hoare rules, called relational properties. {p} is a requirement on the states before C1 and C2, and {q} describes the states after their executions. Notice that C1 and C2 result in different states. As a special case, C1 ~op C2, where op is a getter, indicating that C1 and C2 result in states with the same value for op.

Our tool consists of a special struct type variable called environment, usually denoted by e. This complex type includes the various block data context accessible by solidity (e.g. block.timestamp, msg.sender, msg.value etc.) These fields are accessible via the environment variable.

Community

The following properties were written and verified by contributors from the Aave community

1. **permitIntegrity** Integrity of permit function - successful permit function increases the nonce of owner by 1 and also changes the allowance of owner to spender. Contributed by https://github.com/parth-15

```
    {
        nonceBefore = getNonce(owner)
    }
    <
        permit(owner, spender, value, deadline, v, r, s)
    >
    {
        allowance(owner, spender) == value && getNonce(owner) == nonceBefore +
    }
```

2. **addressZeroNoPower** Address 0 has no voting or proposition power. Contributed by https://github.com/JayP11

```
{
        getPowerCurrent(0, VOTING_POWER) == 0 && getPowerCurrent(0, PROPOSITIO
}
```

3. **metaDelegateByTypeOnlyCallableWithProperlySignedArguments** Verify that `metaDelegateByType` can only be called with a signed request. Contributed by https://github.com/kustosz

```
    {
        ecrecover(v,r,s) != delegator
    }
    <
        metaDelegateByType@withrevert(delegator, delegatee, delegationType, de
    >
    {
        lastReverted == true
    }
```

4. **metaDelegateNonRepeatable** Verify that it's impossible to use the same arguments to call `metaDalegate` twice. Contributed by https://github.com/kustosz

```
    {
        hash1 = computeMetaDelegateHash(delegator, delegatee, deadline, nonce)
```

```
        hash2 = computeMetaDelegateHash(delegator, delegatee, deadline, nonce
        ecrecover(hash1, v, r, s) == delegator
    }
    <
        metaDelegate(e1, delegator, delegatee, v, r, s)
        metaDelegate@withrevert(e2, delegator, delegatee, delegationType, dead
    >
    {
        lastReverted == true
    }
```

5. **delegatingToAnotherUserRemovesPowerFromOldDelegatee** Power of the previous delegate is removed when the delegatee delegates to another delegate. Contributed by https://github.com/priyankabhanderi

```
    {
        _votingBalance = getDelegatedVotingBalance(alice)
    }
    <
        delegateByType(alice, VOTING_POWER)
        delegateByType(bob, VOTING_POWER)
    >
    {
        alice != bob => getDelegatedVotingBalance(alice) == _votingBalance
    }
```

6. **powerChanges** Voting and proposition power change only as a result of specific functions. Contributed by https://github.com/top-sekret

```
    {
        powerBefore = getPowerCurrent(alice, type)
    }
    <
        f(e, args)
    >
    {
       powerAfter = getPowerCurrent(alice, type)
       powerAfter != powerBefore =>
        f.selector == delegate(address).selector ||
        f.selector == delegateByType(address, uint8).selector ||
        f.selector == metaDelegate(address, address, uint256, uint8, bytes32,
        f.selector == metaDelegateByType(address, address, uint8, uint256, uin
        f.selector == transfer(address, uint256).selector ||
        f.selector == transferFrom(address, address, uint256).selector
    }
```

7. **delegateIndependence** Changing a delegate of one type doesn't influence the delegate of the other type. Written by https://github.com/top-sekret

```
{
    delegateBefore = type == 1 ? getPropositionDelegate(e.msg.sender) : ge
}
<
    delegateByType(e, delegatee, 1 - type)
>
{
   delegateBefore = type == 1 ? getPropositionDelegate(e.msg.sender) : get
   delegateBefore == delegateAfter
}
```

8. **votingPowerChangesWhileNotBeingADelegatee** Verify that voting power increases/decreases while not being a voting delegatee yourself. Contributed by https://github.com/Zarfsec

```
{
    votingPowerBefore = getPowerCurrent(a, VOTING_POWER)
    balanceBefore = balanceOf(a)
    isVotingDelegatorBefore = getDelegatingVoting(a)
    isVotingDelegateeBefore = getDelegatedVotingBalance(a) != 0
}
<
    f(e, args)
>
{
    votingPowerAfter = getPowerCurrent(a, VOTING_POWER()
    balanceAfter = getBalance(a)
    isVotingDelegatorAfter = getDelegatingVoting(a);
    isVotingDelegateeAfter = getDelegatedVotingBalance(a) != 0

    votingPowerBefore < votingPowerAfter <=>
    (!isVotingDelegatorBefore && !isVotingDelegatorAfter && (balanceBefore
    (isVotingDelegatorBefore && !isVotingDelegatorAfter && (balanceBefore
    &&
    votingPowerBefore > votingPowerAfter <=>
    (!isVotingDelegatorBefore && !isVotingDelegatorAfter && (balanceBefore
    (!isVotingDelegatorBefore && isVotingDelegatorAfter && (balanceBefore
}
```

9. **propositionPowerChangesWhileNotBeingADelegatee** Verify that proposition power increases/decreases while not being a voting delegatee yourself. Contributed by https://github.com/Zarfsec

```
{
    propositionPowerBefore = getPowerCurrent(a, PROPOSITION_POWER)
    balanceBefore = balanceOf(a)
    isPropositionDelegatorBefore = getDelegatingProposition(a)
    isPropositionDelegateeBefore = getDelegatedPropositionBalance(a) != 0
```

```
}
<
    f(e, args)
>
{
    propositionPowerAfter = getPowerCurrent(a, PROPOSITION_POWER()
    balanceAfter = getBalance(a)
    isPropositionDelegatorAfter = getDelegatingProposition(a);
    isPropositionDelegateeAfter = getDelegatedPropositionBalance(a) != 0

    propositionPowerBefore < propositionPowerAfter <=>
    (!isPropositionDelegatorBefore && !isPropositionDelegatorAfter && (bal
    (isPropositionDelegatorBefore && !isPropositionDelegatorAfter && (bala
    &&
    propositionPowerBefore > propositionPowerAfter <=>
    (!isPropositionDelegatorBefore && !isPropositionDelegatorAfter && (bal
    (!isPropositionDelegatorBefore && isPropositionDelegatorAfter && (bala
}
```

10. **allowanceStateChange** Allowance only changes as a result of specific subset of functions. Contributed by https://github.com/oracleorb

```
{
    allowanceBefore = allowance(owner, spender)
}
<
    f(e, args)
>
{
   allowance(owner, spender) != allowanceBefore =>f.selector==approve(addr
        || f.selector==increaseAllowance(address,uint256).selector
        || f.selector==decreaseAllowance(address,uint256).selector
        || f.selector==transferFrom(address,address,uint256).selector
        || f.selector==permit(address,address,uint256,uint256,uint8,bytes3

}
```

Formal Properties for AaveTokenV3

The following properties were written and verified by Certora

Delegation Invariants

1. **delegateCorrectness** ✓ User's delegation flag is switched on iff user is delegating to an address other than his own own or 0

```
{
    (getVotingDelegate(account) == account || getVotingDelegate(account) =
```

```
        &&
        (getPropositionDelegate(account) == account || getPropositionDelegate(
    }
```

2. **sumOfVBalancesCorrectness ✓** Sum of delegated voting balances and undelegated voting balances is equal to total supply

$$\sum balances[u'].delegatedVotingBalance * 10^{10} + \sum balanceOf(u) = totalSupply()$$

where getVotingDelegate(u) == 0

```
    {
        sumDelegatedVotingBalances + sumUndelegatedVotingBalances == totalSupp
    }
```

3. **sumOfPBalancesCorrectness ✓** Sum of delegated proposition balances and undelegated proposition balances is equal to total supply.

$$\sum balances[u'].delegatedPropositionBalance * 10^{10} + \sum balanceOf(u) = totalSupply()$$

where getPropositionDelegate(u) == 0

```
    {
        sumDelegatedPropositionBalances + sumUndelegatedPropositionBalances ==
    }
```

Delegation Properties

4. **powerWhenNotDelegating ✓** If an account is not receiving delegation of power (one type) from anybody, and that account is not delegating that power to anybody, the power of that account must be equal to its token balance.

```
    {
        dvb = _balances[account].delegatedVotingBalance
        votingPower = getPowerCurrent(account, VOTING_POWER)
        (dvb == 0 && !isDelegatingVoting(account)) => votingPower == balanceOf
    }
```

5. **vpTransferWhenBothNotDelegating ✓** When both accounts are not delegating: On transfer of z amount of tokens from account1 to account2, voting power holds the

$$account1Power_{t1} = account1Power_{t0} - z$$

$$account2Power_{t1} = account2Power_{t0} + z$$

following properties:

```
{
    !isDelegatingVoting(account1) && !isDelegatingVoting(account2)
    account1PowerBefore = getPowerCurrent(account1, VOTING_POWER)
    account2PowerBefore = getPowerCurrent(account2, VOTING_POWER)
    account3PowerBefore = getPowerCurrent(account3, VOTING_POWER)
}
<
    transferFrom(account1, account2, z)
>
{
    getPowerCurrent(account1, VOTING_POWER) == account1PowerBefore - z
    getPowerCurrent(account2, VOTING_POWER) == account2PowerBefore + z
    getPowerCurrent(account3, VOTING_POWER) == account3PowerBefore
}
```

6. **ppTransferWhenBothNotDelegating ✓** When both account1 and account2 are not delegating: On transfer of z amount of tokens from account1 to account2, proposition power holds the following properties:

$$account1Power_{t1} = account1Power_{t0} - z$$

$$account2Power_{t1} = account2Power_{t0} + z$$

```
{
    !isDelegatingProposition(account1) && !isDelegatingProposition(account2
    account1PowerBefore = getPowerCurrent(account1, PROPOSITION_POWER)
    account2PowerBefore = getPowerCurrent(account2, PROPOSITION_POWER)
    account3PowerBefore = getPowerCurrent(account3, PROPOSITION_POWER)
}
<
    transferFrom(account1, account2, z)
>
{
    getPowerCurrent(account1, PROPOSITION_POWER) == account1PowerBefore - z
    getPowerCurrent(account2, PROPOSITION_POWER) == account2PowerBefore + z
    getPowerCurrent(account3, PROPOSITION_POWER) == account3PowerBefore
}
```

7. **vpDelegateWhenBothNotDelegating ✓** When both account1 and account2 are not delegating: After account1 will delegate his voting power to account2

$$account1Power_{t1} = account1Power_{t0} - account1Balance$$

$$account2Power_{t1} = account2Power_{t0} + account1Balance/10^{10} * 10^{10}$$

$$account1PowerDelegatee_{t1} = account2$$

```
 {
     account1 = e.msg.sender
     !isDelegatingVoting(account1) && !isDelegatingVoting(account2)
```

```
        account1PowerBefore = getPowerCurrent(account1, VOTING_POWER)
        account2PowerBefore = getPowerCurrent(account2, VOTING_POWER)
        account3PowerBefore = getPowerCurrent(account3, VOTING_POWER)
    }
    <
        delegate(account2)
    >
    {
        getPowerCurrent(account1, VOTING_POWER) == account1PowerBefore - balan
        getPowerCurrent(account2, VOTING_POWER) == account2PowerBefore + balan
        getPowerCurrent(account3, VOTING_POWER) == account3PowerBefore
    }
```

8. **ppDelegateWhenBothNotDelegating✓**

When both account1 and account2 are not delegating: After account1 will delegate his proposition power to account2

$$account1Power_{t1} = account1Power_{t0} - account1Balance$$

$$account2Power_{t1} = account2Power_{t0} + account1Balance/10^{10} * 10^{10}$$

$$account1PowerDelegatee_{t1} = account2$$

```
    {
        account1 = e.msg.sender
        !isDelegatingProposition(account1) && !isDelegatingProposition(account
        account1PowerBefore = getPowerCurrent(account1, PROPOSITION_POWER)
        account2PowerBefore = getPowerCurrent(account2, PROPOSITION_POWER)
        account3PowerBefore = getPowerCurrent(account3, PROPOSITION_POWER)
    }
    <
        delegate(account2)
    >
    {
        getPowerCurrent(account1, PROPOSITION_POWER) == account1PowerBefore -
        getPowerCurrent(account2, PROPOSITION_POWER) == account2PowerBefore +
        getPowerCurrent(account3, PROPOSITION_POWER) == account3PowerBefore
    }
```

9. **vpTransferWhenOnlyOneIsDelegating ✓** When account1 is delegating voting power to delegatee1 and account2 is not delegating voting power: On transfer of z amount of tokens from account1 to account2

$$account1Power_{t1} = account1Power_{t0} = 0$$

$$delegatee1Power_{t1} = delegatee1Power_{t0} - account1Balance_{t0}/10^{10} * 10^{10} + account1Balance_{t1}/10^{10} * 10^{10}$$

$$account2Power_{t1} = account2Power_{t0} + z$$

```
{
    isDelegatingVoting(account1) && !isDelegatingVoting(account2)
    account1PowerBefore = getPowerCurrent(account1, VOTING_POWER)
    account2PowerBefore = getPowerCurrent(account2, VOTING_POWER)
    account3PowerBefore = getPowerCurrent(account3, VOTING_POWER)
    delegatee1PowerBefore = getPowerCurrent(delegatee1, VOTING_POWER)
    balanceAccount1Before = balanceOf(account1)
}
<
    transferFrom(account1, account2, z)
>
{
    getPowerCurrent(account1, VOTING_POWER) == account1PowerBefore == 0
    getPowerCurrent(delegatee1, VOTING_POWER) == delegatee1PowerBefore - b
    getPowerCurrent(account2, VOTING_POWER) == account2PowerBefore + z
    getPowerCurrent(account3, VOTING_POWER) == account3PowerBefore
}
```

10. **ppTransferWhenOnlyOneIsDelegating** ✓ When account1 is delegating proposition power to delegatee1 and account2 is not delegating proposition power: On transfer of z amount of tokens from account1 to account2

$$account1Power_{t1} = account1Power_{t0} = 0$$

$$delegatee1Power_{t1} = delegatee1Power_{t0} - account1Balance_{t0}/10^{10} * 10^{10} + account1Balance_{t1}/10^{10} * 10^{10}$$

$$account2Power_{t1} = account2Power_{t0} + z$$

```
{
    isDelegatingProposition(account1) && !isDelegatingProposition(account2
    account1PowerBefore = getPowerCurrent(account1, PROPOSITION_POWER)
    account2PowerBefore = getPowerCurrent(account2, PROPOSITION_POWER)
    account3PowerBefore = getPowerCurrent(account3, PROPOSITION_POWER)
    delegatee1PowerBefore = getPowerCurrent(delegatee1, PROPOSITION_POWER)
    balanceAccount1Before = balanceOf(account1)
}
<
    transferFrom(account1, account2, z)
>
{
    getPowerCurrent(account1, PROPOSITION_POWER) == account1PowerBefore ==
    getPowerCurrent(delegatee1, PROPOSITION_POWER) == delegatee1PowerBefor
    getPowerCurrent(account2, PROPOSITION_POWER) == account2PowerBefore +
    getPowerCurrent(account3, PROPOSITION_POWER) == account3PowerBefore
}
```

11. **vpStopDelegatingWhenOnlyOneIsDelegating** ✓ When account1 is delegating voting power to delegatee1 and account2 is not delegating voting power: After

account will stop delegating voting power to delegatee1

$$account1Power_{t1} = account1Power_{t0} + account1Balance$$

$$delegatee1Power_{t1} = delegatee1Power_{t0} - account1Balance/10^{10} * 10^{10}$$

```
{
    account1 == msg.sender && isDelegatingVoting(account1)
    account1PowerBefore = getPowerCurrent(account1, VOTING_POWER)
    account2PowerBefore = getPowerCurrent(account2, VOTING_POWER)
    account3PowerBefore = getPowerCurrent(account3, VOTING_POWER)
    delegatee1PowerBefore = getPowerCurrent(delegatee1, VOTING_POWER)
    balanceAccount1Before = balanceOf(account1)
}
<
    delegate(0)
>
{
    getPowerCurrent(account1, VOTING_POWER) == account1PowerBefore + balan
    getPowerCurrent(delegatee1, VOTING_POWER) == delegatee1PowerBefore - b
    getPowerCurrent(account3, VOTING_POWER) == account3PowerBefore
}
```

12. **ppStopDelegatingWhenOnlyOneIsDelegating** ✓ When account1 is delegating proposition power to delegatee1 and account2 is not delegating proposition power: After account will stop delegating proposition power to delegatee1

$$account1Power_{t1} = account1Power_{t0} + account1Balance$$

$$delegatee1Power_{t1} = delegatee1Power_{t0} - account1Balance/10^{10} * 10^{10}$$

```
{
    account1 == msg.sender && isDelegatingProposition(account1)
    account1PowerBefore = getPowerCurrent(account1, PROPOSITION_POWER)
    account2PowerBefore = getPowerCurrent(account2, PROPOSITION_POWER)
    account3PowerBefore = getPowerCurrent(account3, PROPOSITION_POWER)
    delegatee1PowerBefore = getPowerCurrent(delegatee1, PROPOSITION_POWER)
    balanceAccount1Before = balanceOf(account1)
}
<
    delegate(0)
>
{
    getPowerCurrent(account1, PROPOSITION_POWER) == account1PowerBefore +
    getPowerCurrent(delegatee1, PROPOSITION_POWER) == delegatee1PowerBefor
    getPowerCurrent(account3, PROPOSITION_POWER) == account3PowerBefore
}
```

13. **vpChangeDelegateWhenOnlyOneIsDelegating ✓** When account1 is delegating voting power to delegatee1 and account2 is not delegating voting power: After account1 will delegate power to delegatee2

$$account1Power_{t1} = account1Power_{t0} = 0$$
$$delegatee1Power_{t1} = delegatee1Power_{t0} - account1Balance/10^{10} * 10^{10}$$
$$delegatee2Power_{t1} = delegatee2Power_{t0} + account1Balance/10^{10} * 10^{10}$$
$$account1PowerDelegatee_{t1} = delegatee2$$

```
{
    account1 == msg.sender && isDelegatingVoting(account1)
    account1PowerBefore = getPowerCurrent(account1, VOTING_POWER)
    account3PowerBefore = getPowerCurrent(account3, VOTING_POWER)
    delegatee1PowerBefore = getPowerCurrent(delegatee1, VOTING_POWER)
    delegatee2PowerBefore = getPowerCurrent(delegatee1, VOTING_POWER)

}
<
    delegate(delegatee2)
>
{
    getPowerCurrent(account1, VOTING_POWER) == account1PowerBefore == 0
    getPowerCurrent(delegatee1, VOTING_POWER) == delegatee1PowerBefore - b
    getPowerCurrent(delegatee2, VOTING_POWER) == delegatee2PowerBefore + b
    getPowerCurrent(account3, VOTING_POWER) == account3PowerBefore
}
```

14. **ppChangeDelegateWhenOnlyOneIsDelegating ✓** When account1 is delegating voting power to delegatee1 and account2 is not delegating voting power: After account1 will delegate power to delegatee2

$$account1Power_{t1} = account1Power_{t0} = 0$$
$$delegatee1Power_{t1} = delegatee1Power_{t0} - account1Balance/10^{10} * 10^{10}$$
$$delegatee2Power_{t1} = delegatee2Power_{t0} + account1Balance/10^{10} * 10^{10}$$
$$account1PowerDelegatee_{t1} = delegatee2$$

```
{
    account1 == msg.sender && isDelegatingProposition(account1)
    account1PowerBefore = getPowerCurrent(account1, PROPOSITION_POWER)
    account3PowerBefore = getPowerCurrent(account3, PROPOSITION_POWER)
    delegatee1PowerBefore = getPowerCurrent(delegatee1, PROPOSITION_POWER)
    delegatee2PowerBefore = getPowerCurrent(delegatee1, PROPOSITION_POWER)

}
```

```
<
    delegate(delegatee2)
>
{
    getPowerCurrent(account1, PROPOSITION_POWER) == account1PowerBefore ==
    getPowerCurrent(delegatee1, PROPOSITION_POWER) == delegatee1PowerBefor
    getPowerCurrent(delegatee2, PROPOSITION_POWER) == delegatee2PowerBefor
    getPowerCurrent(account3, PROPOSITION_POWER) == account3PowerBefore
}
```

15. **vpOnlyAccount2IsDelegating ✓** Account1 not delegating voting power to anybody, account2 is delegating voting power to delegatee2: On transfer of z tokens from account1 to account 2

$$account1Power_{t1} = account1Power_{t0} - z$$

$$account2Power_{t1} = account2Power_{t0} = 0$$

$$delegatee2Power_{t1} = delegatee2Power_{t0} - account2Balance_{t0}/10^{10} * 10^{10} + account2Balance_{t1}/10^{10} * 10^{10}$$

```
{
    isDelegatingVoting(account1) && isDelegatingVoting(account2)
    delegatee2 == getVotingDelegate(account2)
    account1PowerBefore = getPowerCurrent(account1, VOTING_POWER)
    account3PowerBefore = getPowerCurrent(account3, VOTING_POWER)
    delegatee2PowerBefore = getPowerCurrent(delegatee2, VOTING_POWER)
    account2BalanceBefore == balanceOf(account2)
}
<
    transferFrom(account1, account2, z)
>
{
    getPowerCurrent(account1, VOTING_POWER) == account1PowerBefore - z
    getPowerCurrent(account2, VOTING_POWER) == 0
    getPowerCurrent(delegatee2, VOTING_POWER) == delegatee2PowerBefore - a
    getPowerCurrent(account3, VOTING_POWER) == account3PowerBefore
}
```

16. **ppOnlyAccount2IsDelegating ✓** Account1 not delegating proposition power to anybody, account2 is delegating proposition power to delegatee2: On transfer of z tokens from account1 to account 2

$$account1Power_{t1} = account1Power_{t0} - z$$

$$account2Power_{t1} = account2Power_{t0} = 0$$

$$delegatee2Power_{t1} = delegatee2Power_{t0} - account2Balance_{t0}/10^{10} * 10^{10} + account2Balance_{t1}/10^{10} * 10^{10}$$

```
{
    isDelegatingProposition(account1) && isDelegatingProposition(account2)
    account1PowerBefore = getPowerCurrent(account1, PROPOSITION_POWER)
    account3PowerBefore = getPowerCurrent(account3, PROPOSITION_POWER)
```

```
        delegatee2PowerBefore = getPowerCurrent(delegatee2, PROPOSITION_POWER)
        account2BalanceBefore == balanceOf(account2)
    }
    <
        transferFrom(account1, account2, z)
    >
    {
        getPowerCurrent(account1, PROPOSITION_POWER) == account1PowerBefore -
        getPowerCurrent(account2, PROPOSITION_POWER) == 0
        getPowerCurrent(delegatee2, PROPOSITION_POWER) == delegatee2PowerBefor
        getPowerCurrent(account3, PROPOSITION_POWER) == account3PowerBefore
    }
```

17. **vpTransferWhenBothAreDelegating** ✓ Account1 is delegating voting power to delegatee1, account2 is delegating voting power to delegatee2: On transfer of z tokens from account1 to account2

$$account1Power_{t1} = account1Power_{t0} = 0$$

$$delegatee1Power_{t1} = delegatee1Power_{t0} - account1Balance_{t0}/10^{10} * 10^{10} + account1Balance_{t1}/10^{10} * 10^{10}$$

$$account2Power_{t1} = account2Power_{t0} = 0$$

$$delegatee2Power_{t1} = delegatee2Power_{t0} - account2Balance_{t0}/10^{10} * 10^{10} + account2Balance_{t1}/10^{10} * 10^{10}$$

```
    {
        isDelegatingVoting(account1) && isDelegatingVoting(account2)
        account1PowerBefore = getPowerCurrent(account1, VOTING_POWER)
        account2PowerBefore = getPowerCurrent(account2, VOTING_POWER)
        account3PowerBefore = getPowerCurrent(account3, VOTING_POWER)
        delegatee1PowerBefore = getPowerCurrent(delegatee1, VOTING_POWER)
        delegatee2PowerBefore = getPowerCurrent(delegatee2, VOTING_POWER)
        account1BalanceBefore = balanceOf(account1)
        account2BalanceBefore = balanceOf(account2)
    }
    <
        transferFrom(account1, account2, z)
    >
    {
        getPowerCurrent(account1, VOTING_POWER) == account1PowerBefore == 0
        getPowerCurrent(account2, VOTING_POWER) == account2PowerBefore == 0
        getPowerCurrent(delegatee1, VOTING_POWER) == delegatee1PowerBefore - a
        getPowerCurrent(delegatee2, VOTING_POWER) == delegatee2PowerBefore - a
    }
```

18. **ppTransferWhenBothAreDelegating** ✓ Account1 is delegating proposition power to delegatee1, account2 is delegating proposition power to delegatee2: On transfer of z

tokens from account1 to account2

$$account1Power_{t1} = account1Power_{t0} = 0$$

$$delegatee1Power_{t1} = delegatee1Power_{t0} - account1Balance_{t0}/10^{10} * 10^{10} + account1Balance_{t1}/10^{10} * 10^{10}$$

$$account2Power_{t1} = account2Power_{t0} = 0$$

$$delegatee2Power_{t1} = delegatee2Power_{t0} - account2Balance_{t0}/10^{10} * 10^{10} + account2Balance_{t1}/10^{10} * 10^{10}$$

```
{
    isDelegatingProposition(account1) && isDelegatingProposition(account2)
    account1PowerBefore = getPowerCurrent(account1, PROPOSITION_POWER)
    account2PowerBefore = getPowerCurrent(account2, PROPOSITION_POWER)
    account3PowerBefore = getPowerCurrent(account3, PROPOSITION_POWER)
    delegatee1PowerBefore = getPowerCurrent(delegatee1, PROPOSITION_POWER)
    delegatee2PowerBefore = getPowerCurrent(delegatee2, PROPOSITION_POWER)
    account1BalanceBefore = balanceOf(account1)
    account2BalanceBefore = balanceOf(account2)
}
<
    transferFrom(account1, account2, z)
>
{
    getPowerCurrent(account1, PROPOSITION_POWER) == account1PowerBefore ==
    getPowerCurrent(account2, PROPOSITION_POWER) == account2PowerBefore ==
    getPowerCurrent(delegatee1, PROPOSITION_POWER) == delegatee1PowerBefor
    getPowerCurrent(delegatee2, PROPOSITION_POWER) == delegatee2PowerBefor
}
```

19. **delegationTypeIndependence** ✓ Only delegate() and metaDelegate() may change both voting and proposition delegates of an account at once.

```
{
    delegateVBefore = getVotingDelegate(account)
    delegatePBefore = getPropositionDelegate(account)
}
<
    f(e, args)
>
{
    delegateVAfter = getVotingDelegate(account)
    delegatePAfter = getPropositionDelegate(account)
    (delegateVBefore == delegateVAfter || delegatePBefore == delegatePAfte
}
```

20. **cantDelegateTwice** ✓ Delegating twice to the same delegate _delegate changes the delegate's voting power only once.

```
{
    votingPowerBefore = getPowerCurrent(_delegate, VOTING_POWER)
```

```
        propositionPowerBefore = getPowerCurrent(_delegate, PROPOSITION_POWER)
    }
    <
        delegate(_delegate)
        votingPowerAfter = getPowerCurrent(_delegate, VOTING_POWER)
        propositionPowerAfter = getPowerCurrent(_delegate, PROPOSITION_POWER)
        delegate(_delegate)
    >
    {
        getPowerCurrent(_delegate, VOTING_POWER) == votingPowerAfter
        getPowerCurrent(_delegate, PROPOSITION_POWER) == propositionPowerAfter
    }
```

ERC20 Properties

21. **transferCorrect** ✓ Token transfer works correctly. Balances are updated if not reverted. If reverted then the transfer amount was too high, or the recipient is 0.

```
    {
        balanceFromBefore = balanceOf(msg.sender)
        balanceToBefore = balanceOf(to)
    }
    <
        transfer(to, amount)
    >
    {
        lastReverted => to = 0 || amount > balanceOf(msg.sender)
        !lastReverted => balanceOf(to) = balanceToBefore + amount &&
                         balanceOf(msg.sender) = balanceFromBefore - amount
    }
```

22. **transferFromCorrect** ✓ Token transferFrom function works correctly. Balances are updated if not reverted. If reverted then the transfer amount was too high, or the recipient is 0, or the allowance was not sufficient

```
    {
        balanceFromBefore = balanceOf(from)
        balanceToBefore = balanceOf(to)
    }
    <
        transferFrom(from, to, amount)
    >
    {
        lastreverted => to = 0 || amount > balanceOf(from)
        !lastreverted => balanceOf(to) = balanceToBefore + amount &&
                         balanceOf(from) = balanceFromBefore - amount
    }
```

23. **zeroAddressNoBalance** ✓ Balance of address 0 is always 0

```
{ balanceOf(0) = 0 }
```

24. **NoChangeTotalSupply** ✓ Contract calls don't change token total supply.

```
    {
        supplyBefore = totalSupply()
    }
    < f(e, args)>
    {
        supplyAfter = totalSupply()
        supplyBefore == supplyAfter
    }
```

25. **ChangingAllowance** ✓ Allowance changes correctly as a result of calls to approve, transfer, increaseAllowance, decreaseAllowance

```
    {
        allowanceBefore = allowance(from, spender)
    }
    <
        f(e, args)
    >
    {
        f.selector = approve(spender, amount) => allowance(from, spender) = am
        f.selector = transferFrom(from, spender, amount) => allowance(from, sp
        f.selector = decreaseAllowance(spender, delta) => allowance(from, spen
        f.selector = increaseAllowance(spender, delta) => allowance(from, spen
        generic f.selector => allowance(from, spender) == allowanceBefore
    }
```

26. **TransferSumOfFromAndToBalancesStaySame** ✓ Transfer from msg.sender to b doesn't change the sum of their balances

```
    {
        balancesBefore = balanceOf(msg.sender) + balanceOf(b)
    }
    <
        transfer(b, amount)
    >
    {
        balancesBefore == balanceOf(msg.sender) + balanceOf(b)
    }
```

27. **TransferFromSumOfFromAndToBalancesStaySame** ✓ transferFrom from a to b doesn't change the sum of their balances

```
{
    balancesBefore = balanceOf(a) + balanceOf(b)
}
<
    transferFrom(a, b)
>
{
    balancesBefore == balanceOf(a) + balanceOf(b)
}
```

28. **TransferDoesntChangeOtherBalance** ✓ Transfer from msg.sender to alice doesn't change the balance of other addresses

```
{
    balanceBefore = balanceOf(charlie)
}
<
    transfer(alice, amount)
>
{
    balanceOf(charlie) == balanceBefore
}
```

29. **TransferFromDoesntChangeOtherBalance** ✓ transferFrom of tokens from alice to bob doesn't change the balance of other addresses

```
{
    balanceBefore = balanceOf(charlie)
}
<
    transferFrom(alice, bob, amount)
>
{
    balanceOf(charlie) = balanceBefore
}
```

30. **OtherBalanceOnlyGoesUp** ✓ Balance of an address, who is not a sender or a recipient in transfer functions, doesn't decrease as a result of contract calls

```
{
    balanceBefore = balanceOf(charlie)
}
<
    f(e, args)
```

```
    >
    {
        f.selector != transfer && f.selector != transferFrom => balanceOf(char
    }
```