Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

MAKER

Veridise Inc.

February 4, 2023

► **Prepared For:**

Maciek Kamiński | Maker Foundation
makerdao.com

► **Prepared By:**

Jacob Van Geffen
Shankara Pailoor
Jon Stephens

► **Contact Us:** contact@veridise.com

► **Version History:**

January 18, 2023 Draft

Contents

1 Executive Summary

From July 25th 2022 to February 1st 2023, MakerDAO engaged Veridise to review the security of their Multi Collateral Dai Protocol for StarkNet. The review focused on verifying the functional correctness of various operations within the Cairo version of the VAT contract of the DAI Stablecoin, and included commits starting from commit 3a6bf6c and ending with commit 9914ac5. Veridise conducted this assessment over 14 person-months, with two senior research scientists and one research engineer. The auditing strategy involved tool-assisted analysis of the source code performed by Veridise engineers. Specifically, Medjai was used to formally verify the implementation of the protocol based on functional specifications. Some enhancements to Medjai were developed specifically to enable this verification.

Summary of issues detected. Through the process of verifying the VAT contract, Veridise engineers found a bug that affected functions using the `safe_math` library, such as the `fold` function. The bug was caused by a mismatch of assumptions made by the `safe_math` libraries and assumptions presumed by the caller of such functions. The bug was fixed in commit 8713f85.

Code assessment. The code provided by MakerDAO for the VAT contract defines an essential part of MakerDAO's DAI stablecoin. It includes operations that alter user balances, set permissions, and update metadata used by other computations. An important distinction between the Cairo VAT contract and the original Solidity version is that Cairo `uint256` values have a wider range of possible errors. Specifically, since Cairo represents values as field elements with a prime less than 2^{256}, `uint256` values are represented by two such field elements. However, the prime is also much larger than 2^{128}, meaning that there are a large range of field element pairs that correspond to invalid representations of a `uint256`. Additionally, there is no automatic protection against these invalid representations. For this reason, part of proving the correctness of the protocol includes proving that all uses of a `uint256` are safe. Many of the properties in chapter 4 reference the validity of `uint256` values for this reason.

Disclaimer. We hope that this report is informative 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 system is secure in all dimensions. In no event shall Veridise 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.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
DAI Stablecoin VAT	3a6bf6c - 9914ac5	Cairo	StarkNet

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
May 3 - June 3, 2022	Medjai	2	14 person-months

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	1	0
High-Severity Issues	0	0
Moderate-Severity Issues	0	0
Low-Severity Issues	0	0
Informational-Severity Issues	0	0
Undetermined-Severity Issues	0	0
TOTAL	1	1

Table 2.4: Category Breakdown.

Name	Number
Logic Error	1

Table 2.5: Verification Summary.

Type	Number
Behavior Validation	18
Revert Necessary and Sufficient Conditions	17

3 Detailed Description of Bugs

During the course of our audit, Medjai identified a bug while verifying V-MCD-PROP-034. The bug was caused by a mismatch of assumptions between the caller and callee of arithmetic operations in `safe_math.cairo`, and affected operations that used safe math arithmetic such as `fold`. This section describes the bug in detail, describes the suggested (and implemented) fix. It also outlines Veridise's recommendations for continued verification of the protocol as a whole.

Table 3.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-VAT-VUL-001	Incorrect uint256 math	Critical	Fixed

3.0.1 V-VAT-VUL-001: Incorrect uint256 math

Severity	Critical	Commit	c751ae5
Type	Logic Error	Status	Fixed
Files			vat.cairo
Functions			fold

The MakerDAO safe math library contains a variety of arithmetic operations over `uint256` values, all with varying assumptions on those parameters. For example, two of the arithmetic operations `add` and `_add` are shown here:

```
// unsigned wad + unsigned wad -> unsigned wad
func add{range_check_ptr, bitwise_ptr: BitwiseBuiltin*}(lhs: Uint256, rhs: Uint256)
    -> (
    res: Uint256
) {
    ...
}

// unsigned wad + signed wad -> unsigned wad
// function _add(uint256 x, int256 y) internal pure returns (uint256 z) {
//     z = y >= 0 ? x + uint256(y) : x - uint256(-y);
// }
func _add{range_check_ptr, bitwise_ptr: BitwiseBuiltin*}(x: Uint256, y: Int256) -> (
    res: Uint256) {
    ...
}
```

As the comments illustrate, `add` operates over unsigned `uint256` values while `_add` takes a signed value as its second argument. Since `uint256` values are also used to represent signed integers, its easy for developers to mismatch caller and callee assumptions. Such an error was found in the `fold` function shown here:

```
// // --- Rates ---
// function fold(bytes32 i, address u, int256 rate_) external auth {
@external
func fold{
    syscall_ptr: felt*, pedersen_ptr: HashBuiltin*, range_check_ptr, bitwise_ptr:
    BitwiseBuiltin*
}(i: felt, u: felt, rate: Int256) {
    ...

    // int256 rad  = _int256(ilk.Art) * rate_;
    let (rad) = _mul(ilk.Art, rate);

    let (dai) = _dai.read(u);
    let (dai) = add(dai, rad);
    _dai.write(u, dai);

    let (debt) = _debt.read();
    let (debt) = add(debt, rad);
    _debt.write(debt);

    ...
}
```

Importantly, `rad` is assumed to be an *signed* value, since `rate` may be negative. However, the `add` operation used assumes that the second operation (`rad` in these cases) is an *unsigned* value. As a result, when `rad` is negative, `fold` may actually add a large amount to `dai(u)` and `debt` instead of subtracting.

Bug fix The fix to this bug came in two parts. First, the `fold` function was updated to use the correct addition operation (i.e. changing `add` to `_add`). Second, the safe math library was updated with the auxiliary type `Int256` to make more clear the callee assumptions on arithmetic operations.

Recommendations As the MakerDAO code base evolves and changes over time, we recommend rerunning this automatic verification process in order to ensure any new changes do not affect the correctness of the protocol. Since the properties specify functional correctness of protocol operations, we do not expect any necessary changes to the specifications when verifying changes to the protocol. Should new operations be added, we recommend specifying the correctness of those operations using the [V] language in a similar way to the specified properties in chapter 4. Doing so will allow Medjai to verify the correctness of these new operations as well.

4 Verified Properties

In this section, we describe the properties verified by our tools. For each property, we log the relevant functions and the type of property. Table 4.1 summarizes the verified properties:

Table 4.1: Summary of Verified Properties.

ID	Description	Status
V-MCD-PROP-001	Constructor correctly initializes state	Verified
V-MCD-PROP-002	`rely` correctly sets `ward`	Verified
V-MCD-PROP-003	`rely` reverts *iff* conditions are met	Verified
V-MCD-PROP-004	`deny` correctly sets `ward`	Verified
V-MCD-PROP-005	`deny` reverts *iff* conditions are met	Verified
V-MCD-PROP-006	`init` correctly sets `rate` value of `ilks`	Verified
V-MCD-PROP-007	`init` reverts *iff* conditions are met	Verified
V-MCD-PROP-008	`file` correctly sets `Line`	Verified
V-MCD-PROP-009	`file` reverts *iff* conditions are met	Verified
V-MCD-PROP-010	`file_ilk` correctly updates state	Verified
V-MCD-PROP-011	`file_ilk` reverts *iff* conditions are met	Verified
V-MCD-PROP-012	`cage` correctly sets `live`	Verified
V-MCD-PROP-013	`cage` reverts *iff* conditions are met	Verified
V-MCD-PROP-014	`hope` correctly sets `can`	Verified
V-MCD-PROP-015	`hope` reverts *iff* conditions are met	Verified
V-MCD-PROP-016	`nope` correctly sets `can`	Verified
V-MCD-PROP-017	`nope` reverts *iff* conditions are met	Verified
V-MCD-PROP-018	`slip` correctly updates `gem`	Verified
V-MCD-PROP-019	`slip` reverts *iff* conditions are met	Verified
V-MCD-PROP-020	`flux` correctly updates `gem` for `src` and `dst`	Verified
V-MCD-PROP-021	`flux` reverts *iff* conditions are met	Verified
V-MCD-PROP-022	`move` correctly updates `dai` for `src` and `dst`	Verified
V-MCD-PROP-023	`move` reverts *iff* conditions are met	Verified
V-MCD-PROP-024	`frob` correctly updates various parts of state	Verified
V-MCD-PROP-025	`frob` reverts *iff* conditions are met	In Progress
V-MCD-PROP-026	`fork` correctly updates `urns`	Verified
V-MCD-PROP-027	`fork` reverts *iff* conditions are met	Verified
V-MCD-PROP-028	`grab` correctly updates various parts of state	Verified
V-MCD-PROP-029	`grab` reverts *iff* conditions are met	Verified
V-MCD-PROP-030	`heal` correctly updates various parts of state	Verified
V-MCD-PROP-031	`heal` reverts *iff* conditions are met	Verified
V-MCD-PROP-032	`suck` correctly updates various parts of state	Verified
V-MCD-PROP-033	`suck` reverts *iff* conditions are met	Verified
V-MCD-PROP-034	`fold` correctly updates various parts of state	Verified
V-MCD-PROP-035	`fold` reverts *iff* conditions are met	Verified

4.1 Detailed Description of Formal Verification Results

In this section, we describe how each property was verified, including both an English description of the property as well as the formal property verified. The specifications for these properties are mostly based on the specifications for the Solidity versions of the functions, with some alterations. For example, additional checks on the validity of `uint256` values are required due to the structural differences between `uint256` in Cairo versus Solidity. While most properties were able to be verified by Medjai in a fully automatic way, some more complex properties required additional manual effort. These efforts are described for each property below.

4.1.1 V-MCD-PROP-001: Constructor correctly initializes state

Commit	9914ac5	**Status**	Verified
Files		vat.cairo	
Functions		constructor	

Description This is a correctness property for the constructor of the `vat` contract. Specifically, the property specifies that the `live` value is set to 1, `ward` value for the parameterized address is set to 1, and the `ward` value for any other addresses does not change.

Formal Specification The following shows the formal specification for the V-MCD-PROP-001 property:

```
vars: contract c, address otherUsr
spec: finished(c.constructor(ward),
               otherUsr != ward
               |=>
               wards(ward) = 1 &&
               wards(otherUsr) = old(wards(otherUsr)) &&
               live() = 1)
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.2 V-MCD-PROP-002: `rely` correctly sets `ward`

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			rely

Description This is a correctness property for the function `rely` in the case that the function finishes (i.e. does not revert). Specifically, the property specifies that the `ward` value for the parameterized address is set to 1, while the `ward` value for any other addresses does not change.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, address otherUsr
spec: finished(c.rely(usr), otherUsr != usr |=>
                            wards(usr) = 1 && wards(otherUsr) = old(wards(otherUsr)))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.3 V-MCD-PROP-003: `rely` reverts *iff* conditions are met

Commit	9914ac5	**Status**	Verified
Files		vat.cairo	
Functions		rely	

Description There are two conditions under which `rely` reverts:

1. The `ward` value of the message sender is not 1
2. The `live` value is not 1

This property specifies that the `rely` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: reverted(c.rely(usr), ward(get_caller_address()) != 1 || live() != 1)
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.4 V-MCD-PROP-004: `deny` correctly sets `ward`

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			deny

Description This is a correctness property for the function `deny` in the case that the function finishes (i.e. does not revert). Specifically, the property specifies that the `ward` value for the parameterized address is set to 0, while the `ward` value for any other addresses does not change.

Formal Specification The following shows the formal specification for the property:

```
1 vars: contract c, address otherUsr
2 spec: finished(c.deny(usr), otherUsr != usr |=>
3                             wards(usr) = 0 && wards(otherUsr) = old(wards(otherUsr)))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.5 V-MCD-PROP-005: `deny` reverts *iff* conditions are met

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			deny

Description There are two conditions under which `deny` reverts:

1. The `ward` value of the message sender is not 1
2. The `live` value is not 1

This property specifies that the `deny` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: reverted(c.deny(usr), ward(get_caller_address()) != 1 || live() != 1)
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.6 V-MCD-PROP-006: `init` correctly sets `rate` value of `ilks`

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			init

Description This is a correctness property for the function `init` in the case that the function finishes (i.e. does not revert). Specifically, the property specifies that the value of `ilks` at index `ilk` is updated such that the rate is set to 10^{27}, and all other values are unchanged.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: let ray() := Uint256(low=10 ** 27, high=0);
      finished(c.init(ilk),
                  ilks(ilk).rate = ray()
               && ilks(ilk).Art = old(ilks(ilk).Art)
               && ilks(ilk).spot = old(ilks(ilk).spot)
               && ilks(ilk).line = old(ilks(ilk).line)
               && ilks(ilk).dust = old(ilks(ilk).dust))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.7 V-MCD-PROP-007: `init` reverts *iff* conditions are met

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			init

Description There are two conditions under which `init` reverts:

1. The `ward` value of the message sender is not 1
2. The initial value of `rate` for the ilk parameter is not 0

This property specifies that the `init` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: reverted(c.init(ilk), ward(get_caller_address()) != 1 || ilks(ilk).rate != 0)
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.8 V-MCD-PROP-008: `file` correctly sets `Line`

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			file

Description This is a correctness property for the function `file` in the case that the function finishes (i.e. does not revert). Specifically, the property specifies that `file` correctly sets the value of `Line` to the parameter `data` of the function.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: finished(c.file(what, data), c.Line() = data)
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.9 V-MCD-PROP-009: `file` reverts *iff* conditions are met

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			file

Description There are three conditions under which `file` reverts:

1. The `ward` value of the message sender is not 1
2. The `live` value is not 1
3. The `what` value passed to the function is not `"Line"`

This property specifies that the `file` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: reverted(c.file(what, data),
               ward(get_caller_address()) != 1 ||
               live() != 1 ||
               what != 0
   x4c696e650000000000000000000000000000000000000000000000000000000000000000)
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.10 V-MCD-PROP-010: `file_ilk` correctly updates state

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			file_ilk

Description This is a correctness property for the function `file_ilk` in the case that the function finishes (i.e. does not revert). Specifically, the property specifies that `file_ilk` updates the `spot`, `line`, and `dust` values of `ilks(ilk)` correctly based on the parameterized `what` value.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: let w1() := 0x73706f7400000000000000000000000000000000000000000000000000000000;
      let w2() := 0x6c696e6500000000000000000000000000000000000000000000000000000000;
      let w3() := 0x6475737400000000000000000000000000000000000000000000000000000000;
      finished(c.file_ilk(ilk, what, data),
               what  = w1() -> ilks(ilk).spot = data
            && what != w1() -> ilks(ilk).spot = old(ilks(ilk).spot)
            && what  = w2() -> ilks(ilk).line = data
            && what != w2() -> ilks(ilk).line = old(ilks(ilk).line)
            && what  = w3() -> ilks(ilk).dust = data
            && what != w3() -> ilks(ilk).dust = old(ilks(ilk).dust)
            && ilks(ilk).Art = old(ilks(ilk).Art)
            && ilks(ilk).rate = old(ilks(ilk).rate))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.11 V-MCD-PROP-011: `file_ilk` reverts *iff* conditions are met

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			file_ilk

Description There are three conditions under which `file_ilk` reverts:

1. The `ward` value of the message sender is not 1
2. The `live` value is not 1
3. The `what` value passed to the function is not `"Line"`

This property specifies that the `file_ilk` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
1 vars: contract c
2 spec: reverted(c.file_ilk(ilk, what, data),
3                ward(get_caller_address()) != 1 ||
4                live() != 1 ||
5                (what != 0
     x73706f7400000000000000000000000000000000000000000000000000000000 &&
6                 what != 0
     x6c696e6500000000000000000000000000000000000000000000000000000000 &&
7                 what != 0
     x6475737400000000000000000000000000000000000000000000000000000000))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.12 V-MCD-PROP-012: `cage` correctly sets `live`

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			cage

Description This is a correctness property for the function `cage` in the case that the function finishes. Specifically, the property specifies that the value of `live` is set to 0 after `cage` is called.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: finished(c.cage(), c.live() = 0)
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.13 V-MCD-PROP-013: `cage` reverts *iff* conditions are met

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			cage

Description This property specifies that the `cage` function should revert *if and only if* the `ward` value of the message sender is not 1.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: reverted(c.cage(), ward(get_caller_address()) != 1)
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.14 V-MCD-PROP-014: `hope` correctly sets `can`

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			hope

Description This is a correctness property for the function `hope` in the case that the function finishes. Specifically, the property specifies that the value of `can` corresponding to both the caller address and `usr` parameter address is set to 1. All other values of `can` should remain unchanged.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, address otherFrom, address otherTo
spec: finished(c.hope(usr),
                   otherFrom != get_caller_address() || otherTo != usr
               |=> can(get_caller_address(), usr) = 1
                   && can(otherFrom, otherTo) = old(can(otherFrom, otherTo)))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.15 V-MCD-PROP-015: `hope` reverts *iff* conditions are met

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			hope

Description This property specifies that the `hope` function should not revert.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: reverted(c.hope(usr), false)
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.16 V-MCD-PROP-016: `nope` correctly sets `can`

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			nope

Description This is a correctness property for the function `nope` in the case that the function finishes. Specifically, the property specifies that the value of `can` corresponding to both the caller address and `usr` parameter address is set to 0. All other values of `can` should remain unchanged.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, address otherFrom, address otherTo
spec: finished(c.nope(usr),
                   otherFrom != get_caller_address() || otherTo != usr
               |=> can(get_caller_address(), usr) = 0
                   && can(otherFrom, otherTo) = old(can(otherFrom, otherTo)))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.17 V-MCD-PROP-017: `nope` reverts *iff* conditions are met

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			nope

Description This property specifies that the `nope` function should not revert.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: reverted(c.nope(usr), false)
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.18 V-MCD-PROP-018: `slip` correctly updates `gem`

Commit	9914ac5	**Status**	Verified
Files		vat.cairo	
Functions		slip	

Description This is a correctness property for the function `slip` in the case that the function finishes. Specifically, the property specifies that the value of `can` corresponding to both the caller address and `usr` parameter address is set to 0. All other values of `can` should remain unchanged.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, felt otherIlk, address otherUsr
spec: finished(c.slip(ilk, usr, wad),
                   otherIlk != ilk || otherUsr != usr
               |=> mathint(gem(ilk, usr)) = old(mathint(gem(ilk, usr))) + mathint(wad
    )
                   gem(otherIlk, otherUsr) = old(gem(otherIlk, otherUsr)))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.19 V-MCD-PROP-019: `slip` reverts *iff* conditions are met

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			slip

Description There are three conditions under which `slip` reverts:

1. The `ward` value of the message sender is not 1
2. The new `gem` value is outside of the range of valid `uint256` values
3. The `wad` parameter represents an invalid `uint256`

This property specifies that the `slip` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: let max_uint256() :=
    115792089237316195423570985008687907853269984665640564039457584007913129639935;
      let sum(l, r) := mathint(uint256(l)) + mathint(int256(r));
      reverted(c.slip(ilk, usr, wad),
               !valid_uint256(wad) ||
               ward(get_caller_address()) != 1 ||
               sum(gem(ilk, usr), wad) < 0 ||
               sum(gem(ilk, usr), wad) > max_uint256())
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.20 V-MCD-PROP-020: `flux` correctly updates `gem` for `src` and `dst`

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			flux

Description This is a correctness property is broken up into two specifications based on the starting state.

1. If `src` and `dst` are equivalent, then the value of `gem` for the address remains the same.
2. If `src` and `dst` are different, then `gem(ilk, src)` decreases and `gem(ilk, dst)` increases by the correct amount. Specifically, the computed `uint256` value represents the correct mathematical integer.

In both cases, the `gem` value for other addresses should not change.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, felt otherIlk, address otherUsr
spec: finished(c.flux(ilk, src, dst, wad),
    src != dst && (otherIlk != ilk || (otherUsr != src && otherUsr != dst))
    |=>
    mathint(gem(ilk, src)) = old(mathint(gem(ilk, src))) - mathint(wad) &&
    mathint(gem(ilk, dst)) = old(mathint(gem(ilk, dst))) + mathint(wad) &&
    gem(otherIlk, otherUsr) = old(gem(otherIlk, otherUsr)))
```

```
vars: contract c, felt anyIlk, address anyUsr
spec: finished(c.flux(ilk, src, dst, wad),
                  src = dst |=> gem(anyIlk, anyUsr) = old(gem(anyIlk, anyUsr)))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.21 V-MCD-PROP-021: `flux` reverts *iff* conditions are met

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			flux

Description There are four conditions under which `flux` reverts:

1. The `wish` value for `src` is false (i.e. `src` is not the message sender and the value of `can` for `src` and the message sender is not 1)
2. The original value for `gem` at `src` is less than the `wad` parameter
3. The resulting `gem` value for `dst` would overflow
4. The `wad` parameter represents an invalid `uint256`

This property specifies that the `flux` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: let max_uint256() :=
    115792089237316195423570985008687907853269984665640564039457584007913129639935;
      let sum(l, r) := mathint(uint256(l)) + mathint(int256(r));
      reverted(c.flux(ilk, src, dst, wad),
               !valid_uint256(wad) ||
               wish(src, get_caller_address()) != 1 ||
               gem(ilk, src) < wad ||
               (src != dst && sum(gem(ilk, dst), wad) > max_uint256()))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.22 V-MCD-PROP-022: `move` correctly updates `dai` for `src` and `dst`

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			move

Description This is a correctness property is broken up into two specifications based on the starting state.

1. If `src` and `dst` are equivalent, then the value of `dai` for the address remains the same.
2. If `src` and `dst` are different, then `dai(src)` decreases and `gem(dst)` increases by the correct amount. Specifically, the computed `uint256` value represents the correct mathematical integer.

In both cases, the `dai` value for other addresses should not change.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, address otherUsr
spec: finished(c.move(src, dst, rad),
               src != dst && otherUsr != src && otherUsr != dst
               |=>
               mathint(dai(src)) = old(mathint(dai(src))) - mathint(rad) &&
               mathint(dai(dst)) = old(mathint(dai(dst))) + mathint(rad) &&
               dai(otherUsr) = old(dai(otherUsr)))
```

```
vars: contract c, address anyUsr
spec: finished(c.move(src, dst, rad),
                   src = dst |=> dai(anyUsr) = old(dai(anyUsr)))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.23 V-MCD-PROP-023: `move` reverts *iff* conditions are met

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			move

Description There are four conditions under which `move` reverts:

1. The `wish` value for `src` is false (i.e. `src` is not the message sender and the value of `can` for `src` and the message sender is not 1)
2. The original value for `dai` at `src` is less than the `rad` parameter
3. The resulting `dai` value for `dst` would overflow
4. The `rad` parameter represents an invalid `uint256`

This property specifies that the `move` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: let max_uint256() :=
    115792089237316195423570985008687907853269984665640564039457584007913129639935;
      let sum(l, r) := mathint(uint256(l)) + mathint(int256(r));
      reverted(c.move(src, dst, rad),
               !valid_uint256(rad) ||
               wish(src, get_caller_address()) != 1 ||
               dai(src) < rad ||
               (src != dst && sum(dai(dst), rad) > max_uint256()))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.24 V-MCD-PROP-024: `frob` correctly updates various parts of state

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			frob

Description This is a correctness property specifies that the `frob` function modifies various parts of the contract state, while keeping others constant:

1. The `ink` and `art` values of urns(i, u) increase, while `urns` for unspecified addresses remains constant.
2. The `Art` value of `ilks(i)` increases, while all other values of `ilks(i)` remain constant.
3. The `debt` value increases by the correct amount.
4. `gem(i, v)` increases, but remains constant for other addresses.
5. `dai(w)` increases, but remains constant for other addresses.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, felt otherIlk, address otherU, address otherV, address otherW
spec: let diff(l, r) := mathint(uint256(l)) - mathint(int256(r));
      finished(c.frob(i, u, v, w, dink, dart)
      (otherIlk != ilk || otherU != u) &&
      (otherIlk != ilk || otherV != v) &&
      otherW != W
      |=>
      mathint(urns(i, u).ink) = old(mathint(urns(i, u).ink)) + mathint(dink) &&
      mathint(urns(i, u).art) = old(mathint(urns(i, u).art)) + mathint(dart) &&
      mathint(ilks(i).Art) = old(mathint(ilks(i).Art)) + mathint(dart)  &&
      mathint(debt()) = old(mathint(debt())) + old(mathint(ilks(i).rate)) * mathint(
    dart)  &&
      mathint(gem(i, v)) = diff(old(gem(i, v)), dink) &&
      mathint(dai(w)) = old(mathint(dai(w))) + old(mathint(ilks(i).rate)) * mathint(
    dart) &&
      urns(otherIlk, otherU) = old(urns(otherIlk, otherU)) &&
      ilks(i).rate = old(ilks(i).rate) &&
      ilks(i).spot = old(ilks(i).spot) &&
      ilks(i).line = old(ilks(i).line) &&
      ilks(i).dust = old(ilks(i).dust) &&
      gem(otherIlk, otherV) = old(gem(otherIlk, otherV)) &&
      dai(otherW) = old(dai(otherW)))
```

Verification Methodology This query required some manual effort from Veridise engineers to fully verify. The main reason for this is the arithmetic used in `frob`: multiplication for `uint256` implemented with finite field arithmetic proves difficult to reason about quickly. To enable verification of this property with Medjai, engineers at Veridise manually decomposed the verification task into multiple independent pieces.

The first way that Veridise engineers performed this decomposition was modeling the multiplication functions within `safe_math.cairo`. In doing so, Medjai was able to use this simpler model to prove the property above, then prove that the model correctly simulates the actual implementation of multiplication.

The second step in decomposing the problem was to verify the property separately for different assumptions made on the inputs to `frob`. Medjai was able to prove the property for `frob` for each input case separately, and then together prove that those cases covered all possible inputs. As a result, Medjai was able to provide a piece-wise proof that the correctness property for `frob` holds.

4.1.25 V-MCD-PROP-025: `frob` reverts *iff* conditions are met

Commit	9914ac5	**Status**	Verified
Files		vat.cairo	
Functions		frob	

Description There are twenty conditions under which `frob` reverts:

1. `live` is not 1
2. The `rate` value of `ilks(i)` is 0
3. The resulting `ink` value of `urns(i, u)` would overflow
4. The resulting `art` value of `urns(i, u)` would overflow
5. The resulting `Art` of `ilks(i)` value would overflow
6. The `rate` value of `ilks(i)` is above the max `uint256` value
7. `rate * dart` over- or underflows
8. `rate` times the new `urns(i, u).art` value overflows
9. `debt + rate * dart` would overflow
10. `rate` times the new `ilks(i).Art` value overflows
11. The parameter `dart > 0`, and `rate * ilks(i).Art` is greater than `ilks(i).line` or the new debt value is greater than `Line`
12. The new value for `ink * spot` overflows
13. Both `rate` times the new `art` is greater than `spot` times the new `ink` and either `dart > 0` or `dink < 0`
14. Both `wish(u)` is false and either `dart > 0` or `dink < 0`
15. Both `wish(v)` is false and `dink > 0`
16. Both `wish(w)` is false and `dart < 0`
17. The new `art` value is greater than 0, and `rate` times the new `art` value is less than `dust`
18. The updated `gem` value overflows
19. The updated `dai` value overflows
20. The `dink` or `dart` parameter represents an invalid `uint256`

This property specifies that the `frob` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: let diff(l, r) := mathint(uint256(l)) - mathint(int256(r));
      let sum(l, r) := mathint(uint256(l)) + mathint(int256(r));
      let mul(l, r) := mathint(uint256(l)) * mathint(int256(r));
      let max_Uint256() :=
    115792089237316195423570985008687907853269984665640564039457584007913129639935;
      let max_Int256() :=
    57896044618658097711785492504343953926634992332820282019728792003956564819967;
      let min_Int256() :=
    -57896044618658097711785492504343953926634992332820282019728792003956564819968;
      let ink(usr) := urns(i, usr).ink;
      let art(usr) := urns(i, usr).art;
      let Art() := ilks(i).Art;
      let rate() := ilks(i).rate;
      let spot() := ilks(i).spot;
      let dust() := ilks(i).dust;
      let line() := ilks(i).line;
      let caller() := c.get_caller_address();
      reverted(c.frob(i, u, v, w, dink, dart),
               !valid_uint256(dink) ||
               !valid_uint256(dart) ||
               live() != 1 ||
               rate() = 0 ||
               sum(ink(i), dink) < 0 ||
               sum(ink(i), dink) > max_Uint256() ||
               sum(art(i), dart) < 0 ||
               sum(art(i), dart) > max_Uint256() ||
               sum(Art(), dart) < 0 ||
               sum(Art(), dart) > max_Uint256() ||
               mathint(rate()) > max_Int256() ||
               mul(rate(), dart) > max_Int256() ||
               mul(rate(), dart) < min_Int256() ||
               mul(rate(), sum(art(i), dart)) > max_Uint256() ||
               sum(debt(), mul(rate(), dart)) < 0 ||
               sum(debt(), mul(rate(), dart)) > max_Uint256() ||
               mul(rate(), sum(Art(), dart)) > max_Uint256() ||
               (dart > 0 && (mul(rate(), sum(Art(), dart)) > line() ||
                             sum(debt(), mul(rate(), dart)) > Line())) ||
               mul(sum(ink(i), ink), spot()) > max_Uint256() ||
               ((dart > 0 || dink < 0) && (mul(rate(), sum(Art(), dart)) > mul(sum(
    ink(i), dink), spot()))) ||
               ((dart > 0 || dink < 0) && wish(u, caller()) != 1) ||
               (dink < 0 && wish(v, caller()) != 1) ||
               (dart > 0 && wish(w, caller()) != 1) ||
               (sum(Art(), dart) > 0 && mul(rate(), sum(Art(), dart) < dust())) ||
               diff(gem(i, v), dink) < 0 ||
               diff(gem(i, v), dink) > max_Uint256() ||
               sum(dai(w), mul(rate(), dart)) < 0 ||
               sum(dai(w), mul(rate(), dart)) > max_Uint256())
```

Verification Methodology Veridise engineers were able to verify this property of `frob` using similar techniques as those described for V-MCD-PROP-024. Specifically, we model arithmetic operations as described previously, and only include necessary assumptions. In general, Medjai proves `if and only if` properties in two cases: showing that `frob` reverting implies that one condition is true, and separately that the disjunction of the conditions implies that `frob` reverts. For this property in particular, Medjai proved that each revert condition listed was sufficient independent of others. This allowed Medjai to perform smaller (and easier) verification tasks. By doing so, Medjai was able to verify that this property holds.

4.1.26 V-MCD-PROP-026: `fork` correctly updates `urns`

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			fork

Description This is a correctness property is broken up into two specifications based on the starting state.

1. If `src` and `dst` are equivalent, then the value of `urns` for the address remains the same.
2. If `src` and `dst` are different, then both the `ink` and `art` values of `urns(ilk, src)` decrease and values of `urns(ilk, dst)` increase by the correct amount. Specifically, all computed `uint256` values represent the correct mathematical integer.

In both cases, the `urns` value for other addresses should not change.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, felt otherIlk, address otherUsr
spec: finished(c.fork(ilk, src, dst, dink, dart),
      src != dst && (otherIlk != ilk || (otherUsr != src && otherUsr != dst))
      |=>
      mathint(urns(ilk, src).ink) = old(mathint(urns(ilk, src).ink)) - mathint(dink)
      && mathint(urns(ilk, src).art)
         = old(mathint(urns(ilk, src).art)) - mathint(dart)
      && mathint(urns(ilk, dst).ink)
         = old(mathint(urns(ilk, dst).ink)) + mathint(dink)
      && mathint(urns(ilk, dst).art)
         = old(mathint(urns(ilk, dst).art)) + mathint(dart)
      && mathint(gem(ilk, dst)) = old(mathint(gem(ilk, dst))) + mathint(wad)
      && urns(otherIlk, otherUsr).ink = old(urns(otherIlk, otherUsr).ink)
      && urns(otherIlk, otherUsr).art = old(urns(otherIlk, otherUsr).art))
```

```
vars: contract c, felt anyIlk, address anyUsr
spec: finished(c.fork(ilk, src, dst, dink, dart),
                  src = dst
               |=> urns(anyIlk, anyUsr).ink = old(urns(anyIlk, anyUsr).ink)
                  && urns(anyIlk, anyUsr).art = old(urns(anyIlk, anyUsr).art))
```

Verification Methodology Like `frob`, `fork` contains arithmetic involving multiplication in a finite field. In order to enable verification through Medjai, Veridise engineers again used a model for `uint256` multiplication and separately verified the correctness of this model. By doing so, Medjai was able to fully verify this property.

4.1.27 V-MCD-PROP-027: `fork` reverts *iff* conditions are met

Commit	9914ac5	**Status**	Verified
Files		vat.cairo	
Functions		fork	

Description There are fourteen conditions under which `fork` reverts:

1. The resulting `ink` value of `urns(ilk, src)` would overflow
2. The resulting `art` value of `urns(ilk, src)` would overflow
3. The resulting `ink` value of `urns(ilk, dst)` would overflow
4. The resulting `art` value of `urns(ilk, dst)` would overflow
5. `rate` times the new `urns(ilk, src).art` value overflows
6. `rate` times the new `urns(ilk, dst).art` value overflows
7. Either `wish(src)` or `wish(dst)` is false
8. `spot` times the new `urns(ilk, src).ink` value overflows
9. `rate` times the new `urns(ilk, src).art` is too large
10. `spot` times the new `urns(ilk, dst).ink` value overflows
11. `rate` times the new `urns(ilk, dst).art` is too large
12. Both the updated `urns(ilk, src).art` is non-zero, and `urns(ilk, src).art * rate < dust`
13. Both the updated `urns(ilk, dst).art` is non-zero, and `urns(ilk, dst).art * rate < dust`
14. The `dink` or `dart` parameter represents an invalid `uint256`

This property specifies that the `fork` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: let diff(l, r) := mathint(uint256(l)) - mathint(int256(r));
      let sum(l, r) := mathint(uint256(l)) + mathint(int256(r));
      let mul(l, r) := mathint(uint256(l)) * mathint(int256(r));
      let max_uint256() :=
    115792089237316195423570985008687907853269984665640564039457584007913129639935;
      let ink(u) := urns(ilk, u).ink;
      let art(u) := urns(ilk, u).art;
      let rate() := ilks(ilk).rate;
      let spot() := ilks(ilk).spot;
      let dust() := ilks(ilk).dust;
      let caller() := c.get_caller_address();
      reverted(c.fork(ilk, src, dst, dink, dart),
               !valid_uint256(dink) ||
               !valid_uint256(dart) ||
               diff(ink(src), dink) < 0 ||
               diff(ink(src), dink) > max_uint256() ||
               diff(art(src), dart) < 0 ||
               diff(art(src), dart) > max_uint256() ||
               (src != dst && sum(ink(dst), dink) < 0) ||
               (src != dst && sum(ink(dst), dink) > max_uint256()) ||
               (src != dst && sum(art(dst), dart) < 0) ||
               (src != dst && sum(art(dst), dart) > max_uint256()) ||
               mul(art(src), rate()) > max_uint256() ||
               mul(art(dst), rate()) > max_uint256() ||
               c.wish(src, caller()) != 1 ||
               c.wish(dst, caller()) != 1 ||
               mul(ink(src), spot()) > max_uint256() ||
               mul(art(src), rate()) > mul(ink(src), spot()) ||
               mul(ink(dst), spot()) > max_uint256() ||
               mul(art(src), rate()) > dust() && art(src) != 0 ||
               mul(art(dst), rate()) > dust() && art(dst) != 0)
```

Verification Methodology Veridise engineers verified this property using methodology similar to that described for property V-MCD-PROP-025. The two techniques used here specifically include modeling arithmetic and individually proving revert conditions are sufficient to force `fork` to revert.

4.1.28 V-MCD-PROP-028: `grab` correctly updates various parts of state

Commit	9914ac5	**Status**	Verified
Files		vat.cairo	
Functions		grab	

Description This is a correctness property specifies that the `grab` function modifies various parts of the contract state, while keeping others constant:

1. The `ink` and `art` values of urns(i, u) increase, while `urns` for unspecified addresses remains constant.
2. The `Art` value of `ilks(i)` increases, while all other values of `ilks(i)` remain constant.
3. `gem(i, v)` decreases, but remains constant for other addresses.
4. `sin(w)` decreases, but remains constant for other addresses.
5. `vice` decreases

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, felt otherIlk, address otherU, address otherV, address otherW
spec: finished(c.grab(i, u, v, w, dink, dart)
      (otherIlk != ilk || otherU != u) &&
      (otherIlk != ilk || otherV != v) &&
      otherW != W
      |=>
      mathint(urns(i, u).ink) = old(mathint(urns(i, u).ink)) + mathint(dink)
      && mathint(urns(i, u).art) = old(mathint(urns(i, u).art)) + mathint(dart)
      && mathint(ilks(i).Art) = old(mathint(ilks(i).Art)) + mathint(dart)
      && mathint(gem(i, v)) = old(mathint(gem(i, v))) - mathint(dink)
      && mathint(sin(w))
         = old(mathint(sin(w))) - old(mathint(ilks(i).rate)) * mathint(dart)
      && mathint(vice())
         = old(mathint(vice())) - old(mathint(ilks(i).rate)) * mathint(dart)
      && urns(otherIlk, otherU) = old(urns(otherIlk, otherU))
      && ilks(i).rate = old(ilks(i).rate)
      && ilks(i).spot = old(ilks(i).spot)
      && ilks(i).line = old(ilks(i).line)
      && ilks(i).dust = old(ilks(i).dust)
      && gem(otherIlk, otherV) = old(gem(otherIlk, otherV))
      && sin(otherW) = old(sin(otherW)))
```

Verification Methodology Like `frob` and `fork`, `grab` contains arithmetic involving multiplication in a finite field. In order to enable verification through Medjai, Veridise engineers again used a model for `uint256` multiplication and separately verified the correctness of this model. By doing so, Medjai was able to fully verify this property.

4.1.29 V-MCD-PROP-029: `grab` reverts *iff* conditions are met

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			grab

Description There are ten conditions under which `grab` reverts:

1. `ward` is not 1
2. The resulting `ink` value of `urns(i, u)` would overflow
3. The resulting `art` value of `urns(i, u)` would overflow
4. The resulting `Art` of `ilks(i)` value would overflow
5. The `rate` value of `ilks(i)` is above the max `uint256` value
6. `rate * dart` over- or underflows
7. The updated `gem` value overflows
8. The updated `sin` value overflows
9. The updated `vice` value overflows
10. The `dink` or `dart` parameter represents an invalid `uint256`

This property specifies that the `grab` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: let diff(l, r) := mathint(uint256(l)) - mathint(int256(r));
      let sum(l, r) := mathint(uint256(l)) + mathint(int256(r));
      let mul(l, r) := mathint(uint256(l)) * mathint(int256(r));
      let max_Uint256() :=
    115792089237316195423570985008687907853269984665640564039457584007913129639935;
      let max_Int256() :=
    57896044618658097711785492504343953926634992332820282019728792003956564819967;
      let min_Int256() :=
    -57896044618658097711785492504343953926634992332820282019728792003956564819968;
      let ink(usr) := urns(i, usr).ink;
      let art(usr) := urns(i, usr).art;
      let Art() := ilks(i).Art;
      let rate() := ilks(i).rate;
      let spot() := ilks(i).spot;
      let dust() := ilks(i).dust;
      let caller() := c.get_caller_address();
      reverted(c.grab(i, u, v, w, dink, dart),
               !valid_uint256(dink) ||
               !valid_uint256(dart) ||
               wards(caller()) != 1 ||
               sum(ink(i), dink) < 0 ||
               sum(ink(i), dink) > max_Uint256() ||
               sum(art(i), dart) < 0 ||
               sum(art(i), dart) > max_Uint256() ||
               sum(Art(), dart) < 0 ||
               sum(Art(), dart) > max_Uint256() ||
               mathint(rate()) > max_Int256() ||
               mul(rate(), dart) > max_Int256() ||
               mul(rate(), dart) < min_Int256() ||
               diff(gem(i, v), dink) < 0 ||
               diff(gem(i, v), dink) > max_Uint256 ||
               diff(sin(w), mul(rate(), dart)) > max_Uint256() ||
               diff(sin(w), mul(rate(), dart)) < 0 ||
               diff(vice(), mul(rate(), dart)) > max_Uint256() ||
               diff(vice(), mul(rate(), dart)) < 0)
```

Verification Methodology Veridise engineers verified this property using methodology similar to that described for property V-MCD-PROP-025. The two techniques used here specifically include modeling arithmetic and individually proving revert conditions are sufficient to force `grab` to revert.

4.1.30 V-MCD-PROP-030: `heal` correctly updates various parts of state

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			heal

Description This is a correctness property specifies that the `heal` function modifies various parts of the contract state, while keeping others constant:

1. The `dai` of the message sender decreases
2. The `sin` of the message sender decreases
3. `vice` decreases
4. `debt` decreases
5. `dai` for other addresses remains constant
6. `sin` for other addresses remains constant

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, address otherUsr
spec: let addr() = get_caller_address();
      finished(c.heal(rad),
                  otherUsr != addr()
               |=> mathint(dai(addr())) = old(mathint(dai(addr()))) - mathint(rad)
                  && mathint(sin(addr())) = old(mathint(sin(addr()))) - mathint(rad)
                  && mathint(vice()) = old(mathint(vice())) - mathint(rad)
                  && mathint(debt()) = old(mathint(debt())) - mathint(rad)
                  && dai(otherUsr) = old(dai(otherUsr))
                  && sin(otherUsr) = old(sin(otherUsr)))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.31 V-MCD-PROP-031: `heal` reverts *iff* conditions are met

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			heal

Description There are five conditions under which `heal` reverts:

1. The original value of `dai` for the message sender is less than the input `rad`
2. The original value of `sin` for the message sender is less than the input `rad`
3. The original value of `vice` is less than the input `rad`
4. The original value of `debt` is less than the input `rad`
5. The `rad` parameter represents an invalid `uint256`

This property specifies that the `heal` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: let caller() := c.get_caller_address();
      reverted(c.heal(rad),
               !valid_uint256(rad) ||
               dai(caller()) < rad ||
               sin(caller()) < rad ||
               vice() < rad ||
               debt() < rad)
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.32 V-MCD-PROP-032: `suck` correctly updates various parts of state

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			suck

Description This is a correctness property specifies that the `suck` function modifies various parts of the contract state, while keeping others constant:

1. The `dai(v)` increases
2. The `sin(u)` increases
3. `vice` increases
4. `debt` increases
5. `dai` for other addresses remains constant
6. `sin` for other addresses remains constant

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, address otherUsr
spec: let addr() = get_caller_address();
      finished(c.suck(rad),
                  otherUsr != addr()
              |=> mathint(dai(addr())) = old(mathint(dai(addr()))) + mathint(rad)
                  && mathint(sin(addr())) = old(mathint(sin(addr()))) + mathint(rad)
                  && mathint(vice()) = old(mathint(vice())) + mathint(rad)
                  && mathint(debt()) = old(mathint(debt())) + mathint(rad)
                  && dai(otherUsr) = old(dai(otherUsr))
                  && sin(otherUsr) = old(sin(otherUsr)))
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.33 V-MCD-PROP-033: `suck` reverts *iff* conditions are met

Commit	9914ac5	**Status**	Verified
Files			vat.cairo
Functions			suck

Description There are six conditions under which `suck` reverts:

1. `ward` is not 1
2. The new value of `sin(u)` overflows
3. The new value of `dai(v)` overflows
4. The new value of `vice` overflows
5. The new value of `debt` overflows
6. The `rad` parameter represents an invalid `uint256`

This property specifies that the `suck` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: let max_uint256() :=
    115792089237316195423570985008687907853269984665640564039457584007913129639935;
      let sum(l, r) := mathint(uint256(l)) + mathint(int256(r));
      let caller() := c.get_caller_address();
      reverted(c.suck(rad),
               !valid_uint256(rad) ||
               wards(caller()) != 1 ||
               sum(dai(caller()), rad) > max_uint256() ||
               sum(sin(caller()), rad) > max_uint256() ||
               sum(vice(), rad) > max_uint256() ||
               sum(debt(), rad) > max_uint256())
```

Verification Methodology This property was verified by Medjai fully automatically.

4.1.34 V-MCD-PROP-034: `fold` correctly updates various parts of state

Commit	9914ac5	**Status**	Verified
Files		vat.cairo	
Functions		fold	

Description This is a correctness property specifies that the `fold` function modifies various parts of the contract state, while keeping others constant:

1. `ilks(i).rate` increases
2. `dai(u)` increases
3. `debt` increases
4. Other parameters of `ilks(i)` remain constant

Formal Specification The following shows the formal specification for the property:

```
vars: contract c, address otherUsr
spec: finished(c.fold(i, u, rate),
      otherUsr != u
      |=>
      mathint(ilks(i).rate) = old(mathint(ilks(i).rate)) + mathint(rate)
      && mathint(dai(u))
         = old(mathint(dai(u))) + old(mathint(ilks(i).Art)) * mathint(rate)
      && mathint(debt())
         = old(mathint(debt())) + old(mathint(ilks(i).Art)) * mathint(rate)
      && ilks(i).Art = old(ilks(i).Art)
      && ilks(i).spot = old(ilks(i).spot)
      && ilks(i).line = old(ilks(i).line)
      && ilks(i).dust = old(ilks(i).dust)
      && dai(otherUsr) = old(dai(otherUsr))
```

Verification Methodology This property was verified by Medjai fully automatically.

Uncovered vulnerability During the process of verifying this property, Medjai originally uncovered a bug whose root cause was due to a mismatch of assumptions when using the `safe_math` library functions. After fixing the bug, the property was verified by Medjai.

4.1.35 V-MCD-PROP-035: `fold` reverts *iff* conditions are met

Commit	9914ac5	Status	Verified
Files			vat.cairo
Functions			fold

Description There are eight conditions under which `fold` reverts:

1. `ward` is not 1
2. `live` is not 1
3. The new value of `rate` overflows
4. The original value of `ilks(i).Art` overflows
5. The new value of `ilks(i).Art` overflows
6. The original value of `dai(u)` overflows
7. The original value of `debt` overflows
8. The `rate` parameter represents an invalid `uint256`

This property specifies that the `fold` function should revert *if and only if* at least one of these conditions is met.

Formal Specification The following shows the formal specification for the property:

```
vars: contract c
spec: let max_Uint256() :=
    115792089237316195423570985008687907853269984665640564039457584007913129639935;
      let max_Int256() :=
    57896044618658097711785492504343953926634992332820282019728792003956564819967;
      let min_Int256() :=
    -57896044618658097711785492504343953926634992332820282019728792003956564819968;
      let sum(l, r) := mathint(uint256(l)) + mathint(int256(r));
      let mul(l, r) := mathint(uint256(l)) * mathint(int256(r));
      let caller() := c.get_caller_address();
      reverted(c.fold(i, u, rate),
               !valid_uint256(rate) ||
               wards(caller()) != 1 ||
               live() != 1 ||
               sum(ilks(i).rate, rate) > max_Uint256() ||
               sum(ilks(i).rate, rate) < 0 ||
               ilks(i).Art > max_Int256() ||
               mul(ilks(i).Art, rate) > max_Int256() ||
               mul(ilks(i).Art, rate) < min_Int256() ||
               sum(dai(u), mul(ilks(i).Art, rate)) > max_Uint256() ||
               sum(dai(u), mul(ilks(i).Art, rate)) < 0 ||
               sum(debt(), mul(ilks(i).Art, rate)) > max_Uint256() ||
               sum(debt(), mul(ilks(i).Art, rate)) < 0)
```

Verification Methodology This property was verified by Medjai fully automatically.