Audit Report

Produced by CertiK

for

23rd Dec, 2019

CertiK Audit Report For SkyPeople

Request Date: 2019-10-30
Revision Date: 2019-12-23
Platform Name: Ethereum

Contents

Disclaimer

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets.

For more information: https://certik.org/

Executive Summary

This report has been prepared for SkyPeople to discover issues and vulnerabilities in the source code of their MineralNFTMarket, MineralNFT, Mineral, Counters, Context, SafeMath, ERC20, ERC20Burnable, ERC165, ERC721, ERC721Enumerable, ERC721Metadata, ERC1132, IERC20, IERC20Receiver, IERC165, IERC721, IERC721Enumberable, IERC721Metadata, IERC721Receiver, ERC721Full, IERC721Full, SafeERC20, Address, BytesLib and Ownable smart contracts. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Vulnerability Classification

CertiK categorizes issues into three buckets based on overall risk levels:

Critical

Code implementation does not match specification, which could result in the loss of funds for contract owner or users.

Medium

Code implementation does not match the specification under certain conditions, which could affect the security standard by loss of access control.

Low

Code implementation does not follow best practices, or uses suboptimal design patterns, which could lead to security vulnerabilities further down the line.

Testing Summary

PASS

CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*

Dec 23, 2019

Type of Issues

CertiK's smart label engine applied 100% formal verification coverage on the source code. Our team of engineers has scanned the source code using proprietary static analysis tools and code-review methodologies. The following technical issues were found:

Title	Description	Issues	SWC ID
Integer Overflow/ Underflow	An overflow/underflow occurs when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function Incorrectness	Function implementation does not meet specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker can write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by miners to some degree.	1	SWC-116
Insecure Compiler Version	Using a fixed outdated compiler version or floating pragma can be problematic if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103
Insecure Randomness	Using block attributes to generate random numbers is unreliable, as they can be influenced by miners to some degree.	0	SWC-120
"tx.origin" for Authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115

Title	Description	Issues	SWC ID
Delegatecall to Untrusted Callee	Calling untrusted contracts is very dangerous, so the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default, meaning a malicious user can make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized Variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used.	0	SWC-111
Unused Variables	Unused variables reduce code quality	0	SWC-131

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Manual Review Notes

Review Details

Summary

CertiK was chosen by SkyPeople to audit the design and implementation of its soon to be released smart contract. To ensure comprehensive protection, the source code has been analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space.
Refer to White Paper:

> MNR is the primary currency of the Mineral Hub ecosystem. It will act as a primary currency across multiple games, allowing players to buy and sell in-game items as non-fungible tokens.

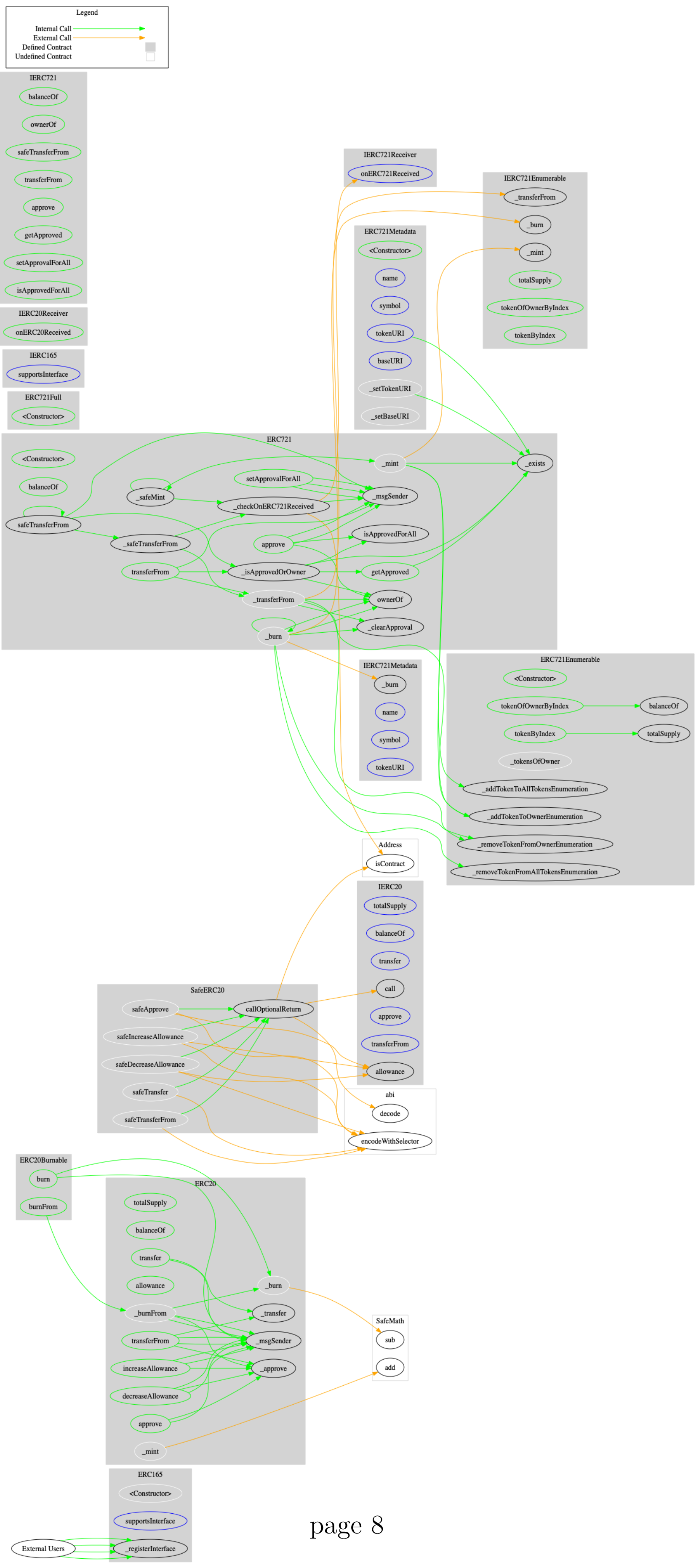
Source Code SHA-256 Checksum

- **MineralNFTMarket.sol**
  `81deb468a0c5d72679190c0737735f78d0254662c9bb58a0106cd31a4b73ce12`
- **Counters.sol**
  `9bf7f72914f5d87d3f10e51b077ba4ef591330be2ce2f20f5eebea629c2c22ba`
- **Context.sol**
  `48e51cda94a082cd59a2cb1309af279ba74fa061532ed4e2c191a25b68ad671b`
- **SafeMath.sol**
  `ed701033e29cd4d639b0d8ab13c2e1554f6b4817e692b62c3dc269470a508d94`
- **Mineral.sol**
  `b0d2c367c9f47b3906090cd5fba6d2406994194b5b39c9f7a60f7704433349ba`
- **MineralNFT.sol**
  `1e56a06bab0dadac96f928dd99c086605b7de8db18ecc10ddc9de4c229afb9e1`
- **ERC1132.sol**
  `e1b9bef8bd03022d3239bff75df220fe9a5cd156e5922024659167428a2d07ee`
- **ERC165.sol**
  `9428389a4270beedb8a550aab3117a5e0ab5d6810ded1f37349dcaea9ce20848`
- **ERC20.sol**
  `7af01925c2ef84f807d4a5c7d9bcb61a7f1650f22ff4dc70d57936ccbe6e0246`
- **ERC20Burnable.sol**
  `47e1c6135c7215dfd9903f9145ce6ba263e0ebadd2ba84b3eaf4515eec5ff6ca`
- **ERC721.sol**
  `ac454c8b1d9de703621af0cdbfaec6bc7f94b48ee74aab1aa18ac05908c8d3b0`

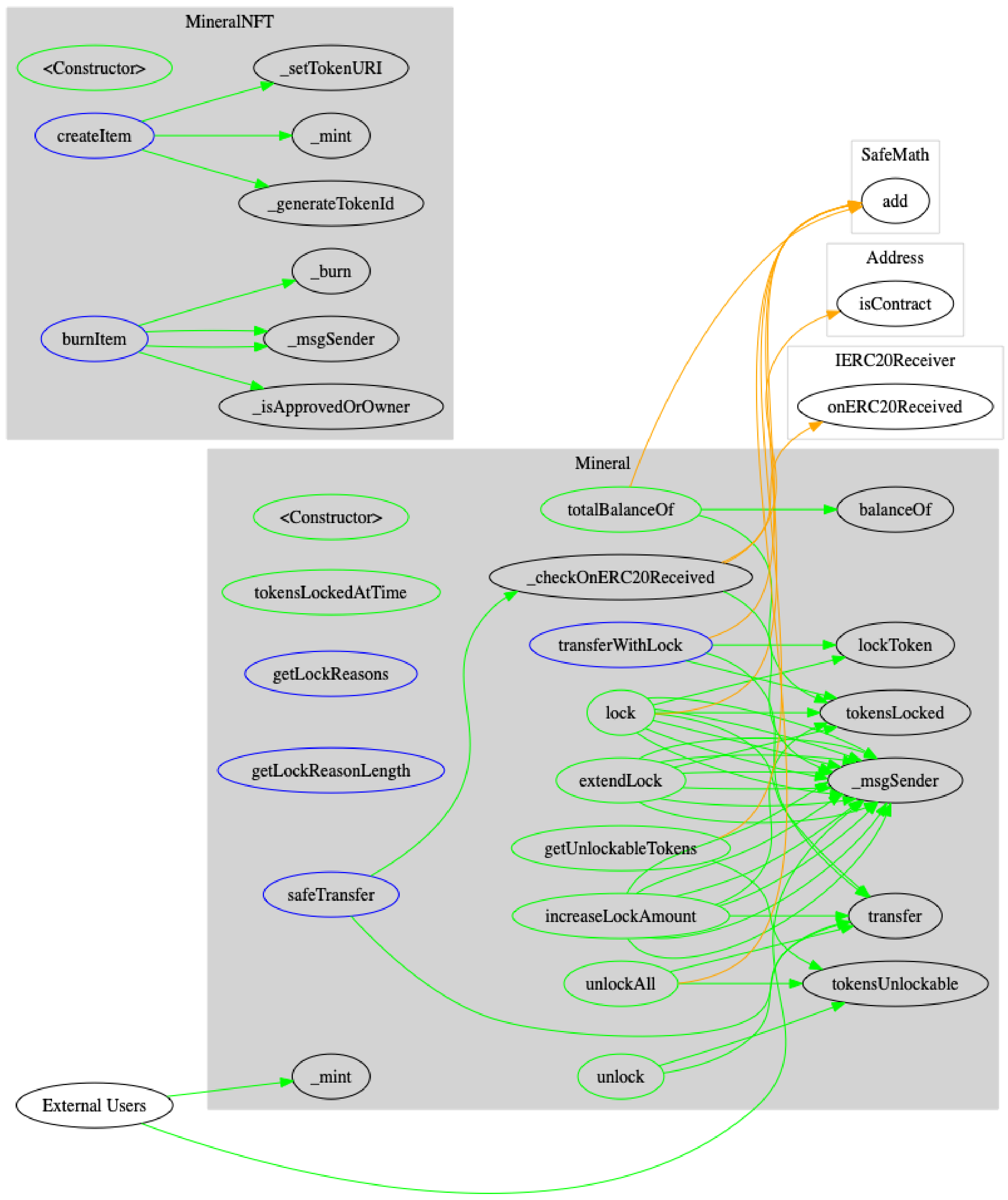
- **ERC721Enumerable.sol**
  `1084a0795b83499cffaa7d9549af34f5afd16ab3c504fdb50a8ad6b406ec86f1`
- **ERC721Full.sol**
  `f4cc48196ca3c577a8d8f63475f909023277336fd49b11b056f632711c4100b8`
- **ERC721Metadata.sol**
  `da24231ef6d1a912715caf4715a48d12deffede5d92c1d549261456a0a07b78f`
- **IERC165.sol**
  `2c51255f992f9cb9af6f4a49258e1ea95461745f2793ad7bfa866fb3c32ef961`
- **IERC20.sol**
  `a9125f9fed6949dfe63f0a681b611ba519042c54da222c00e97072895d690df2`
- **IERC20Receiver.sol**
  `e9908f36e4aef61089720c9764a06e856bf988f853ded5727d30f30c18fc5f0c`
- **IERC721.sol**
  `d24478871e68fe8017105c4bf07032033992220bfb8bfe667280b89a52f89a5f`
- **IERC721Enumerable.sol**
  `b0b6915aaf8ee7f58bc04f68e4edb91ac3a2602fa9636a572319e478c977debe`
- **IERC721Full.sol**
  `886bb7a12e412fde52cfe89672d71f14aa99910ac9f0765093f941d59f0608ac`
- **IERC721Metadata.sol**
  `8971be3b30c2b5ee7473c6ff91531b3e42c74807f745947637fccc9d499f99b2`
- **IERC721Receiver.sol**
  `9d682e2276bee3b73c4bce0547d3568890be97eea8084e5c6c67b8adce102a74`
- **SafeERC20.sol**
  `22c89d5b7342d1f24d652f9462a8aed6c592d23f37db9c2f903e27f71de7e2e0`
- **Address.sol**
  `47f17517d45f594e0c2aea25a7a7699fb11c9c146aeb276aaa03e4fb20ce068e`
- **BytesLib.sol**
  `95da1be4fdf1a6eb1d35564f141a4b3073f312b580c374ed78ff56b181d103f4`
- **Ownable.sol**
  `dda28aaf81b9acfec7d04ea23ff8dc04adbcee311a7406165766af727f691987`

Design Architect

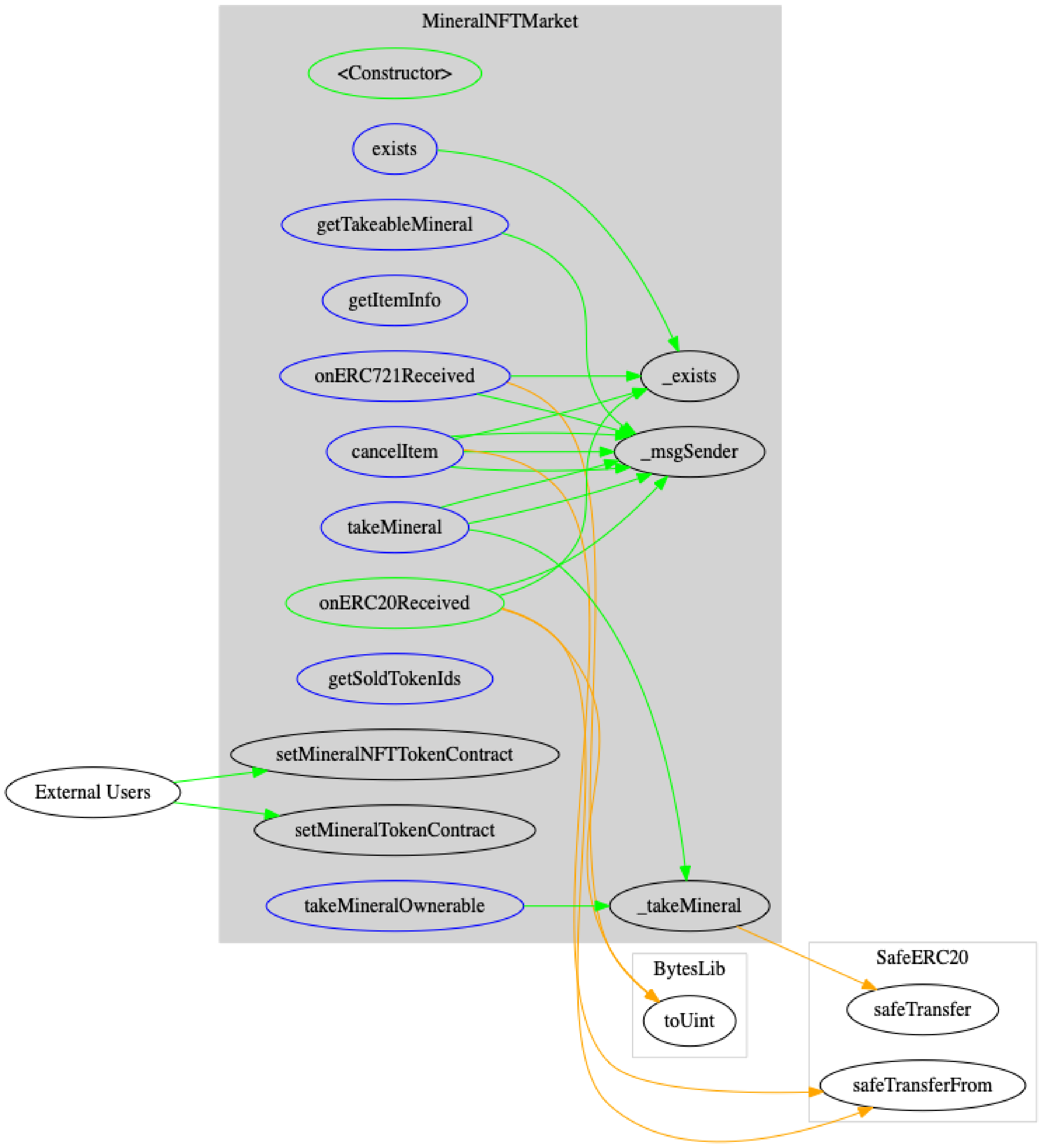
ERC Token Dependencies



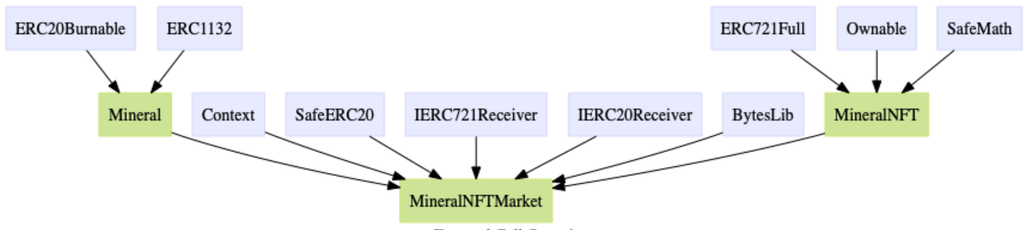
Mineral Dependencies



MineralNFTMarket Dependencies

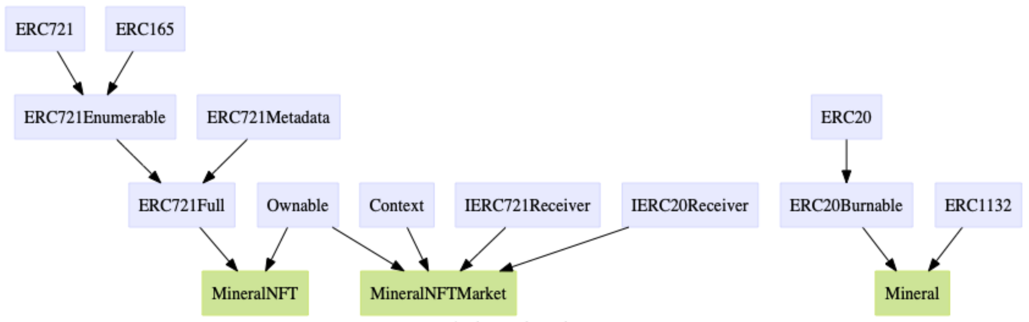


Call Overview



External Call Overview

Inheritance Overview



Inheritance Overview

Recommendations

Changelog

- ✓ $\boxed{\text{SkyPeople}}$ Upgraded to latest version of Openzeppelin contracts with usage of `safeTransfer()` and `safeTransferFrom()`.
- ✓ $\boxed{\text{SkyPeople}}$ Added functions `getLockReasons()` and `getLockReasonLength()` as two getter functions, no security issues found.

Overview

- $\boxed{\text{MINOR}}$ Recommend to use `safeTransfer()` and `safeTransferform()` when need to trigger transfer related logic.
  - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
- $\boxed{\text{MINOR}}$ Recommend to update to the latest and stable compiler version, or at least ensure the compiler versions of files in the project are the same.
  - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
- $\boxed{\text{MINOR}}$ Recommend to update to the latest version of ERC20 and ERC721 from Openzeppelin. Here is the diff comparing to Openzeppelin$_{commit\ 33047ffddcc81ba7b0349431c5065c448603d098}$ and EIP-1132 Proposer's$_{commit\ 82c68d3bd2d16c5ecda04d67b435ddaa69e5e7d4}$:
  - ERC1132.sol
    * Derived from EIP-1132 Proposer
    * SkyPeople: + `unlock()`
    * ✓ $\boxed{\text{SkyPeople}}$ This item is confirmed.
  - ERC20.sol
    * Variables `_balances`, `_allowances` and `_totalSupply` are `public` in SkyPeople but `private` in openzeppelin.
    * SkyPeople: + `safeTransfer()`
    * ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
  - ERC721.sol
    * SkyPeople: `- Counters ->` syntax diff for `increment()` or `decrement()`
    * SkyPeople: `- _safeMint()`
    * SkyPeople: `- _safeTransferFrom()`
    * ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
  - ERC721Enumerable.sol
    * SkyPeople: + `totalSupply()`
    * ✓ $\boxed{\text{SkyPeople}}$ This item is confirmed.

`MineralNFTMarket.sol`

- $\boxed{\text{MAJOR}}$ Reentrancy in function `_takeMineral()`, `cancelItem()` and `onERC20Received()`. Recommend to use the Check Effect Interaction Pattern to protect the contract from being reentered.
  - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
- $\boxed{\text{INFO}}$ Variable `from` is unused in function `onERC721Received()`
  - ✓ $\boxed{\text{SkyPeople}}$ The `from` parameter is derived from `IERC721Receiver` and the function is overriding `onERC721Received()`.
- $\boxed{\text{INFO}}$ Local variable named `owner` shadows `Ownable.owner`.
- $\boxed{\text{INFO}}$ Functions `onERC721Received()` and `onERC20Received()`: Recommend changing `public` to `external` to save gas.
- $\boxed{\text{INFO}}$ Function `takeMineralOnwerable()`: Recommend renaming to `takeMineralOwnerable()`.
  - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
- $\boxed{\text{INFO}}$ Variable `selledTokenIds`: Recommend renaming to `soldTokenIds`.
  - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
- $\boxed{\text{INFO}}$ Struct `Item`: The various statuses should be enumerated for readability and used wherever you set statuses:

```
    enum Status{enabled, sold, cancelled}
```

  - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
- $\boxed{\text{INFO}}$ Recommend changing wordings of error messages in `require`:
  - `require same token address` can be changed to `msg.sender is not nft token address`.
    - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
  - `exists item` can be changed to `item with input tokenId is existing`.
    - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
  - `require 0 < price` can be changed to `input price is not valid`.
    - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.

`/token/Mineral.sol`

- $\boxed{\text{MINOR}}$ Token name is set to be `"Mienral"`.
  - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.

- $\boxed{\text{INFO}}$ `INITIAL_SUPPLY` initialization has many digits. Consider using scientific notation.
  - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
- $\boxed{\text{INFO}}$ Functions `lock()`, `tokensLockedAtTime()`, `totalBalanceOf()`, `extendLock()`, `increaseLockAmount()`, `unlockAll()`, `unlock()`, `getUnlockableTokens()` and `transferWithLock ()`: Recommend changing `public` to `external` to save gas.
  - ✓ $\boxed{\text{SkyPeople}}$ `transferWithLock()` is updated.
- $\boxed{\text{INFO}}$ Recommend to initialize `locked` and `lockedReason` are mapped in `ERC1132.sol` for clarity.
  - ✓ $\boxed{\text{SkyPeople}}$ This item is confirmed with SkyPeople that they aware the risk of using external library, although the risk is low on the this incident.
- $\boxed{\text{DISCUSSION}}$ `Mineral` does not inherit from `Ownable`, so users couldn't directly give each other MNR in game. Is this intended functionality?
  - ✓ $\boxed{\text{SkyPeople}}$ This item is confirmed.

`/token/MineralNFT.sol`

- $\boxed{\text{INFO}}$ Function `createItem()`: Recommend changing `public` to `external` to save gas.
  - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
- $\boxed{\text{MAJOR}}$ When `id` is set to be the value of total supply, there is a possibility that the `createItem()` would never succeed, which might be considered as a Denial of Service.
  - ✓ $\boxed{\text{SkyPeople}}$ The code is updated and reflected in the latest commit.
  - Approach:
    - Let's say 10 NFTs exist, so they'll have tokenIDs 0-9.
    - Now let's burn tokenIDs 3-5 so we're left with 7 total NFTs.
    - When `createItem()` is called, the tokenID for this new NFT will be 7.
    - Since an NFT with tokenID 7 already exists, minting will fail.
    - Currently the issue is safe since all of the `_burn()` functions are internal and never being called. Recommend to add a similar logic like what openzeppelin does. Move the `id` of the latest valid element from the largest `id` value to the burned `id` value.

Static Analysis Results

TIMESTAMP_DEPENDENCY

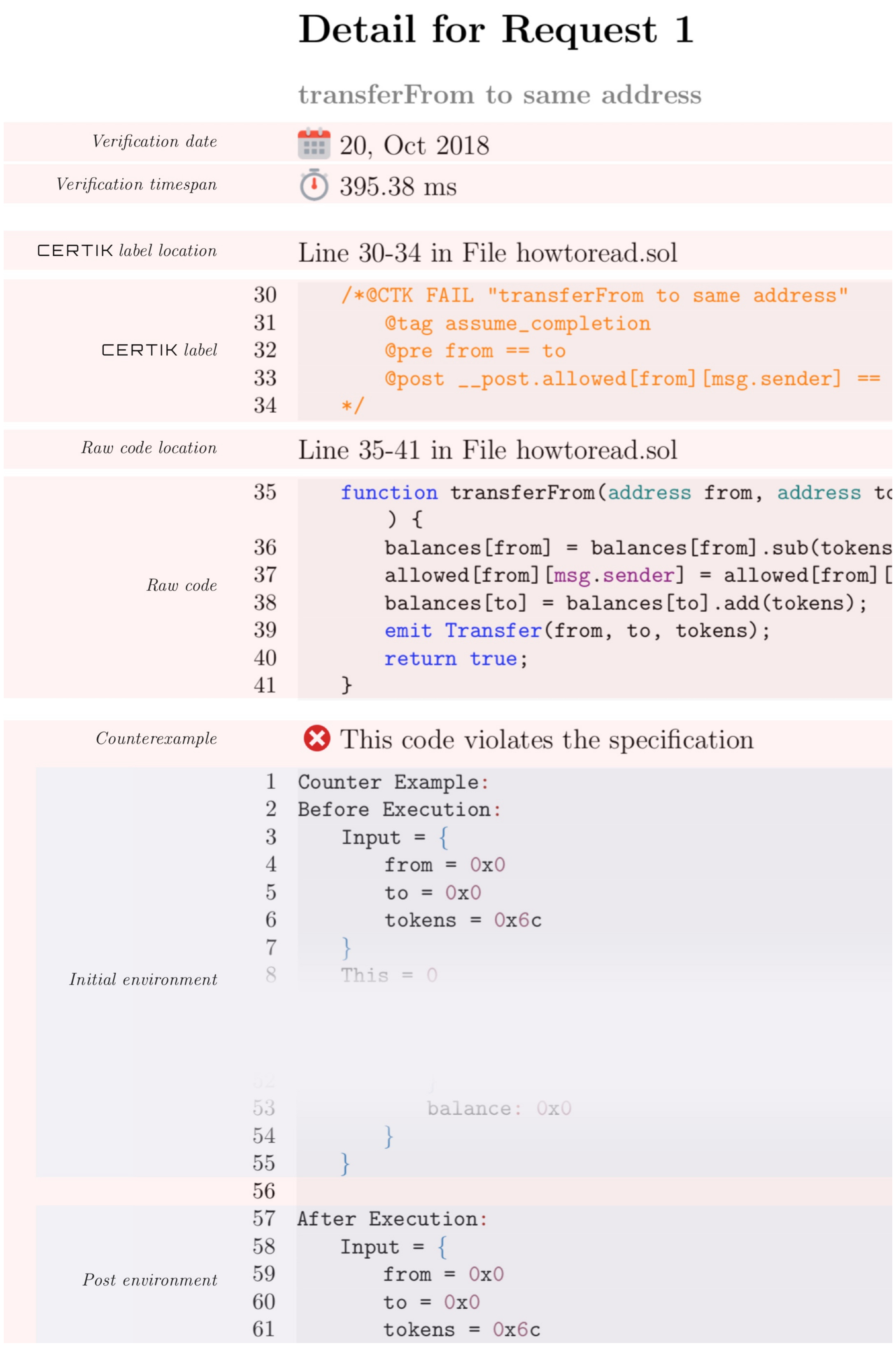
Line 236 in File Mineral.sol

```
236         if (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) //solhint-disable-line
```

❗ ”now” can be influenced by miners to some degree

Formal Verification Results

How to read



Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms

CERTIK *label location*: Line 30-34 in File howtoread.sol

CERTIK *label*

```
30  /*@CTK FAIL "transferFrom to same address"
31      @tag assume_completion
32      @pre from == to
33      @post __post.allowed[from][msg.sender] ==
34  */
```

Raw code location: Line 35-41 in File howtoread.sol

Raw code

```
35  function transferFrom(address from, address tc
        ) {
36      balances[from] = balances[from].sub(tokens
37      allowed[from][msg.sender] = allowed[from][
38      balances[to] = balances[to].add(tokens);
39      emit Transfer(from, to, tokens);
40      return true;
41  }
```

Counterexample: This code violates the specification

Initial environment

```
Counter Example:
Before Execution:
    Input = {
        from = 0x0
        to = 0x0
        tokens = 0x6c
    }
    This = 0

            balance: 0x0
        }
    }

```

Post environment

```
After Execution:
    Input = {
        from = 0x0
        to = 0x0
        tokens = 0x6c
```

Formal Verification Request 1

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
239.03 ms

Line 41 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 44-47 in File MineralNFTMarket.sol

```
    constructor(address nft, address mineral) public {
        setMineralNFTTokenContract(nft);
        setMineralTokenContract(mineral);
    }
```

✓ The code meets the specification.

Formal Verification Request 2

If method completes, integer overflow would not happen.

23, Dec 2019
1.46 ms

Line 42 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 44-47 in File MineralNFTMarket.sol

```
    constructor(address nft, address mineral) public {
        setMineralNFTTokenContract(nft);
        setMineralTokenContract(mineral);
    }
```

✓ The code meets the specification.

Formal Verification Request 3

Method will not encounter an assertion failure.

23, Dec 2019
1.37 ms

Line 43 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 44-47 in File MineralNFTMarket.sol

```
    constructor(address nft, address mineral) public {
        setMineralNFTTokenContract(nft);
        setMineralTokenContract(mineral);
    }
```

✓ The code meets the specification.

Formal Verification Request 4

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 40.97 ms

Line 48 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 55-57 in File MineralNFTMarket.sol

```
    function exists(uint id) external view returns (bool) {
        return _exists(id);
    }
```

✓ The code meets the specification.

Formal Verification Request 5

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.54 ms

Line 49 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 55-57 in File MineralNFTMarket.sol

```
    function exists(uint id) external view returns (bool) {
        return _exists(id);
    }
```

✓ The code meets the specification.

Formal Verification Request 6

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.59 ms

Line 50 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 55-57 in File MineralNFTMarket.sol

```
    function exists(uint id) external view returns (bool) {
        return _exists(id);
    }
```

✓ The code meets the specification.

Formal Verification Request 7

exists

📅 23, Dec 2019
⏱ 2.81 ms

Line 51-54 in File MineralNFTMarket.sol

```
    /*@CTK exists
      @post _items[id].price == 0 -> __return == false
      @post _items[id].price != 0 -> __return == (_items[id].status == 0)
    */
```

Line 55-57 in File MineralNFTMarket.sol

```
    function exists(uint id) external view returns (bool) {
        return _exists(id);
    }
```

✅ The code meets the specification.

Formal Verification Request 8

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.42 ms

Line 58 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 65-70 in File MineralNFTMarket.sol

```
    function _exists(uint id) internal view returns (bool) {
        if (_items[id].price == 0)
            return false;

        return _items[id].status == uint8(ItemStatus.enable);
    }
```

✅ The code meets the specification.

Formal Verification Request 9

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.4 ms

Line 59 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 65-70 in File MineralNFTMarket.sol

```
    function _exists(uint id) internal view returns (bool) {
        if (_items[id].price == 0)
            return false;

        return _items[id].status == uint8(ItemStatus.enable);
    }
```

✓ The code meets the specification.

Formal Verification Request 10

Method will not encounter an assertion failure.

23, Dec 2019
0.39 ms

Line 60 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 65-70 in File MineralNFTMarket.sol

```
    function _exists(uint id) internal view returns (bool) {
        if (_items[id].price == 0)
            return false;

        return _items[id].status == uint8(ItemStatus.enable);
    }
```

✓ The code meets the specification.

Formal Verification Request 11

__exists

23, Dec 2019
1.7 ms

Line 61-64 in File MineralNFTMarket.sol

```
    /*@CTK _exists
      @post _items[id].price == 0 -> __return == false
      @post _items[id].price != 0 -> __return == (_items[id].status == 0)
    */
```

Line 65-70 in File MineralNFTMarket.sol

```
    function _exists(uint id) internal view returns (bool) {
        if (_items[id].price == 0)
            return false;

        return _items[id].status == uint8(ItemStatus.enable);
    }
```

✓ The code meets the specification.

Formal Verification Request 12

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
22.68 ms

Line 71 in File MineralNFTMarket.sol

```
71    //@CTK NO_BUF_OVERFLOW
```

Line 77-79 in File MineralNFTMarket.sol

```
77    function getTakeableMineral() external view returns (uint256) {
78        return _takeableMineral[_msgSender()];
79    }
```

✓ The code meets the specification.

Formal Verification Request 13

If method completes, integer overflow would not happen.

23, Dec 2019
0.48 ms

Line 72 in File MineralNFTMarket.sol

```
72    //@CTK NO_OVERFLOW
```

Line 77-79 in File MineralNFTMarket.sol

```
77    function getTakeableMineral() external view returns (uint256) {
78        return _takeableMineral[_msgSender()];
79    }
```

✓ The code meets the specification.

Formal Verification Request 14

Method will not encounter an assertion failure.

23, Dec 2019
0.47 ms

Line 73 in File MineralNFTMarket.sol

```
73    //@CTK NO_ASF
```

Line 77-79 in File MineralNFTMarket.sol

```
77    function getTakeableMineral() external view returns (uint256) {
78        return _takeableMineral[_msgSender()];
79    }
```

✓ The code meets the specification.

Formal Verification Request 15

getTakeableMineral

23, Dec 2019
0.66 ms

Line 74-76 in File MineralNFTMarket.sol

```
    /*@CTK getTakeableMineral
      @post !__reverted -> __return == _takeableMineral[msg.sender]
    */
```

Line 77-79 in File MineralNFTMarket.sol

```
    function getTakeableMineral() external view returns (uint256) {
        return _takeableMineral[_msgSender()];
    }
```

✓ The code meets the specification.

Formal Verification Request 16

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
8.52 ms

Line 80 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 86-88 in File MineralNFTMarket.sol

```
    function getItemInfo(uint256 tokenId) external view returns (uint256 price, address
         owner, uint8 status) {
        return (_items[tokenId].price, _items[tokenId].owner, _items[tokenId].status);
    }
```

✓ The code meets the specification.

Formal Verification Request 17

If method completes, integer overflow would not happen.

23, Dec 2019
0.4 ms

Line 81 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 86-88 in File MineralNFTMarket.sol

```
    function getItemInfo(uint256 tokenId) external view returns (uint256 price, address
         owner, uint8 status) {
        return (_items[tokenId].price, _items[tokenId].owner, _items[tokenId].status);
    }
```

✓ The code meets the specification.

Formal Verification Request 18

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.39 ms

Line 82 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 86-88 in File MineralNFTMarket.sol

```
    function getItemInfo(uint256 tokenId) external view returns (uint256 price, address
        owner, uint8 status) {
        return (_items[tokenId].price, _items[tokenId].owner, _items[tokenId].status);
    }
```

✓ The code meets the specification.

Formal Verification Request 19

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 125.67 ms

Line 91 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 101-116 in File MineralNFTMarket.sol

```
    function onERC721Received(address operator, address from, uint256 tokenId, bytes
        calldata data) external returns (bytes4) {
        require (_msgSender() == address(_nft), "msg.sender is not nft token address");
        require (_exists(tokenId) == false, "item with input tokenId is existing");
        uint256 price = data.toUint(0);
        require (0 < price, "input price is not valid");
        _items[tokenId] = Item({
            id: tokenId,
            price: price,
            owner: operator,
            status: uint8(ItemStatus.enable)
        });
        emit SellItem(_items[tokenId].owner, _items[tokenId].id, _items[tokenId].price);
        return _ERC721_RECEIVED;
    }
```

✓ The code meets the specification.

Formal Verification Request 20

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 3.86 ms

Line 92 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 101-116 in File MineralNFTMarket.sol

```
    function onERC721Received(address operator, address from, uint256 tokenId, bytes
        calldata data) external returns (bytes4) {
        require (_msgSender() == address(_nft), "msg.sender is not nft token address");
        require (_exists(tokenId) == false, "item with input tokenId is existing");
        uint256 price = data.toUint(0);
        require (0 < price, "input price is not valid");
        _items[tokenId] = Item({
            id: tokenId,
            price: price,
            owner: operator,
            status: uint8(ItemStatus.enable)
        });
        emit SellItem(_items[tokenId].owner, _items[tokenId].id, _items[tokenId].price);
        return _ERC721_RECEIVED;
    }
```

✓ The code meets the specification.

Formal Verification Request 21

Method will not encounter an assertion failure.

23, Dec 2019
4.03 ms

Line 93 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 101-116 in File MineralNFTMarket.sol

```
    function onERC721Received(address operator, address from, uint256 tokenId, bytes
        calldata data) external returns (bytes4) {
        require (_msgSender() == address(_nft), "msg.sender is not nft token address");
        require (_exists(tokenId) == false, "item with input tokenId is existing");
        uint256 price = data.toUint(0);
        require (0 < price, "input price is not valid");
        _items[tokenId] = Item({
            id: tokenId,
            price: price,
            owner: operator,
            status: uint8(ItemStatus.enable)
        });
        emit SellItem(_items[tokenId].owner, _items[tokenId].id, _items[tokenId].price);
        return _ERC721_RECEIVED;
    }
```

✓ The code meets the specification.

Formal Verification Request 22

onERC721Received

📅 23, Dec 2019
⏱ 4.03 ms

Line 94-100 in File MineralNFTMarket.sol

```
    /*@CTK onERC721Received
      @tag assume_completion
      @pre msg.sender == address(_nft)
      @pre _items[tokenId].price == 0 || (_items[tokenId].price != 0 && _items[tokenId].
          status != 0)
      // @pre 0 < price
      @post __return == 0x150b7a02
    */
```

Line 101-116 in File MineralNFTMarket.sol

```
    function onERC721Received(address operator, address from, uint256 tokenId, bytes
        calldata data) external returns (bytes4) {
        require (_msgSender() == address(_nft), "msg.sender is not nft token address");
        require (_exists(tokenId) == false, "item with input tokenId is existing");
        uint256 price = data.toUint(0);
        require (0 < price, "input price is not valid");
        _items[tokenId] = Item({
            id: tokenId,
            price: price,
            owner: operator,
            status: uint8(ItemStatus.enable)
        });
        emit SellItem(_items[tokenId].owner, _items[tokenId].id, _items[tokenId].price);
        return _ERC721_RECEIVED;
    }
```

✅ The code meets the specification.

Formal Verification Request 23

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 190.39 ms

Line 119 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 132-157 in File MineralNFTMarket.sol

```
    function onERC20Received(address from, uint256 amount, bytes memory data) public returns
         (bool) {
        require (_msgSender() == address(_mineral), "msg.sender is not mineral token address
            ");

        uint256 id = data.toUint(0);
        require (_exists(id), "item with input tokenId is existing");

        Item storage item = _items[id];

        require (item.price == amount, "input amount is not valid");
        require (from != item.owner, "input buyer is not valid");
        require (item.status == 0, "item is not available");
        //ctk start
        _takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner].add(amount);
        //ctk end
        _takeableMineral[item.owner] = _takeableMineral[item.owner].add(amount);
        _soldTokenIds[item.owner].push(id);
        item.status = uint8(ItemStatus.sold);
        _nft.safeTransferFrom(address(this), from, id);

        emit BuyItem(item.owner, from, id, amount);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 24

If method completes, integer overflow would not happen.

23, Dec 2019
17.7 ms

Line 120 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 132-157 in File MineralNFTMarket.sol

```
    function onERC20Received(address from, uint256 amount, bytes memory data) public returns
         (bool) {
        require (_msgSender() == address(_mineral), "msg.sender is not mineral token address
            ");

        uint256 id = data.toUint(0);
        require (_exists(id), "item with input tokenId is existing");

        Item storage item = _items[id];

        require (item.price == amount, "input amount is not valid");
        require (from != item.owner, "input buyer is not valid");
        require (item.status == 0, "item is not available");
        //ctk start
        _takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner].add(amount);
        //ctk end
        _takeableMineral[item.owner] = _takeableMineral[item.owner].add(amount);
        _soldTokenIds[item.owner].push(id);
        item.status = uint8(ItemStatus.sold);
        _nft.safeTransferFrom(address(this), from, id);

        emit BuyItem(item.owner, from, id, amount);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 25

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 12.02 ms

Line 121 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 132-157 in File MineralNFTMarket.sol

```
    function onERC20Received(address from, uint256 amount, bytes memory data) public returns
         (bool) {
        require (_msgSender() == address(_mineral), "msg.sender is not mineral token address
            ");

        uint256 id = data.toUint(0);
        require (_exists(id), "item with input tokenId is existing");

        Item storage item = _items[id];

        require (item.price == amount, "input amount is not valid");
        require (from != item.owner, "input buyer is not valid");
        require (item.status == 0, "item is not available");
        //ctk start
        _takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner].add(amount);
        //ctk end
        _takeableMineral[item.owner] = _takeableMineral[item.owner].add(amount);
        _soldTokenIds[item.owner].push(id);
        item.status = uint8(ItemStatus.sold);
        _nft.safeTransferFrom(address(this), from, id);

        emit BuyItem(item.owner, from, id, amount);
        return true;
    }
```

✅ The code meets the specification.

Formal Verification Request 26

onERC20Received

📅 23, Dec 2019
⏱ 0.57 ms

Line 122-131 in File MineralNFTMarket.sol

```
    /*@CTK onERC20Received
      @pre msg.sender == address(_mineral)
      @pre msg.sender == address(_nft)
      // @pre _items[id].price == 0 || (_items[id].price != 0 && _items[id].status != 0)
      // @pre _items[id].price == amount
      // @pre from != _items[id].owner
      // @pre _items[id].status == 0
      // @post __post._takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner
         ].add(amount)
    // @post __post._items[id].status == 1
   */
```

Line 132-157 in File MineralNFTMarket.sol

```
   function onERC20Received(address from, uint256 amount, bytes memory data) public returns
        (bool) {
      require (_msgSender() == address(_mineral), "msg.sender is not mineral token address
          ");

      uint256 id = data.toUint(0);
      require (_exists(id), "item with input tokenId is existing");

      Item storage item = _items[id];

      require (item.price == amount, "input amount is not valid");
      require (from != item.owner, "input buyer is not valid");
      require (item.status == 0, "item is not available");
      //ctk start
      _takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner].add(amount);
      //ctk end
      _takeableMineral[item.owner] = _takeableMineral[item.owner].add(amount);
      _soldTokenIds[item.owner].push(id);
      item.status = uint8(ItemStatus.sold);
      _nft.safeTransferFrom(address(this), from, id);

      emit BuyItem(item.owner, from, id, amount);
      return true;
   }
```

✓ The code meets the specification.

Formal Verification Request 27

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
83.91 ms

Line 159 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 167-181 in File MineralNFTMarket.sol

```
   function cancelItem(uint256 tokenId) external {
      require (_exists(tokenId), "item with input tokenId is existing");

      Item item = _items[tokenId];

      require (_msgSender() == item.owner, "msg.sender is not token owner");
      require (item.status != uint8(ItemStatus.canceled), "item is already canceled");

      item.status = uint8(ItemStatus.canceled);
      _nft.safeTransferFrom(address(this), _msgSender(), tokenId);

      emit CancelItem(_msgSender(), tokenId);
   }
```

✓ The code meets the specification.

Formal Verification Request 28

If method completes, integer overflow would not happen.

23, Dec 2019
3.35 ms

Line 160 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 167-181 in File MineralNFTMarket.sol

```
    function cancelItem(uint256 tokenId) external {
        require (_exists(tokenId), "item with input tokenId is existing");

        Item item = _items[tokenId];

        require (_msgSender() == item.owner, "msg.sender is not token owner");
        require (item.status != uint8(ItemStatus.canceled), "item is already canceled");

        item.status = uint8(ItemStatus.canceled);
        _nft.safeTransferFrom(address(this), _msgSender(), tokenId);

        emit CancelItem(_msgSender(), tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 29

Method will not encounter an assertion failure.

23, Dec 2019
3.47 ms

Line 161 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 167-181 in File MineralNFTMarket.sol

```
    function cancelItem(uint256 tokenId) external {
        require (_exists(tokenId), "item with input tokenId is existing");

        Item item = _items[tokenId];

        require (_msgSender() == item.owner, "msg.sender is not token owner");
        require (item.status != uint8(ItemStatus.canceled), "item is already canceled");

        item.status = uint8(ItemStatus.canceled);
        _nft.safeTransferFrom(address(this), _msgSender(), tokenId);

        emit CancelItem(_msgSender(), tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 30

cancelItem

23, Dec 2019
0.42 ms

Line 162-166 in File MineralNFTMarket.sol

```
    /*@CTK "cancelItem"
      @pre _items[tokenId].price != 0 && _items[tokenId].price == 0 || _items[tokenId].
          status == 0
      @pre msg.sender == _items[tokenId].owner
      @pre _items[tokenId].status != 2
     */
```

Line 167-181 in File MineralNFTMarket.sol

```
    function cancelItem(uint256 tokenId) external {
        require (_exists(tokenId), "item with input tokenId is existing");

        Item item = _items[tokenId];

        require (_msgSender() == item.owner, "msg.sender is not token owner");
        require (item.status != uint8(ItemStatus.canceled), "item is already canceled");

        item.status = uint8(ItemStatus.canceled);
        _nft.safeTransferFrom(address(this), _msgSender(), tokenId);

        emit CancelItem(_msgSender(), tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 31

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
106.32 ms

Line 183 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 192-195 in File MineralNFTMarket.sol

```
    function takeMineral() external {
        require (0 < _takeableMineral[_msgSender()], "There is no sender's mineral to be
            take");
        _takeMineral(_msgSender());
    }
```

✓ The code meets the specification.

Formal Verification Request 32

If method completes, integer overflow would not happen.

23, Dec 2019
0.93 ms

Line 184 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 192-195 in File MineralNFTMarket.sol

```
    function takeMineral() external {
       require (0 < _takeableMineral[_msgSender()], "There is no sender's mineral to be
           take");
       _takeMineral(_msgSender());
    }
```

✓ The code meets the specification.

Formal Verification Request 33

Method will not encounter an assertion failure.

23, Dec 2019
0.88 ms

Line 185 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 192-195 in File MineralNFTMarket.sol

```
    function takeMineral() external {
       require (0 < _takeableMineral[_msgSender()], "There is no sender's mineral to be
           take");
       _takeMineral(_msgSender());
    }
```

✓ The code meets the specification.

Formal Verification Request 34

takeMineral

23, Dec 2019
4.07 ms

Line 186-191 in File MineralNFTMarket.sol

```
    /*@CTK takeMineral
      @pre 0 < _takeableMineral[msg.sender]
      @pre _soldTokenIds[msg.sender].length > 0
      @post __post._takeableMineral[msg.sender] == 0
      @post __post._soldTokenIds[msg.sender].length == 0
    */
```

Line 192-195 in File MineralNFTMarket.sol

```
    function takeMineral() external {
        require (0 < _takeableMineral[_msgSender()], "There is no sender's mineral to be
            take");
        _takeMineral(_msgSender());
    }
```

✓ The code meets the specification.

Formal Verification Request 35

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.52 ms

Line 197 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 205-217 in File MineralNFTMarket.sol

```
    function _takeMineral(address addr) internal {
        require(_soldTokenIds[addr].length > 0, "There is no mineral to be take");

        uint256 amount = _takeableMineral[addr];
        uint256[] memory tokenIds = _soldTokenIds[addr];
        _takeableMineral[addr] = 0;
        _soldTokenIds[addr].length = 0;

        _mineral.safeTransfer(addr, amount);
        emit TakeMineral(addr, amount, tokenIds);
    }
```

✓ The code meets the specification.

Formal Verification Request 36

If method completes, integer overflow would not happen.

23, Dec 2019
0.48 ms

Line 198 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 205-217 in File MineralNFTMarket.sol

```
    function _takeMineral(address addr) internal {
        require(_soldTokenIds[addr].length > 0, "There is no mineral to be take");

        uint256 amount = _takeableMineral[addr];
        uint256[] memory tokenIds = _soldTokenIds[addr];
        _takeableMineral[addr] = 0;
        _soldTokenIds[addr].length = 0;

        _mineral.safeTransfer(addr, amount);
        emit TakeMineral(addr, amount, tokenIds);
    }
```

✓ The code meets the specification.

Formal Verification Request 37

Method will not encounter an assertion failure.

23, Dec 2019
0.47 ms

Line 199 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 205-217 in File MineralNFTMarket.sol

```
    function _takeMineral(address addr) internal {
        require(_soldTokenIds[addr].length > 0, "There is no mineral to be take");

        uint256 amount = _takeableMineral[addr];
        uint256[] memory tokenIds = _soldTokenIds[addr];
        _takeableMineral[addr] = 0;
        _soldTokenIds[addr].length = 0;

        _mineral.safeTransfer(addr, amount);
        emit TakeMineral(addr, amount, tokenIds);
    }
```

✓ The code meets the specification.

Formal Verification Request 38

_takeMineral

23, Dec 2019
2.39 ms

Line 200-204 in File MineralNFTMarket.sol

```
    /*@CTK _takeMineral
      @pre _soldTokenIds[addr].length > 0
      @post __post._takeableMineral[addr] == 0
      @post __post._soldTokenIds[addr].length == 0
    */
```

Line 205-217 in File MineralNFTMarket.sol

```
    function _takeMineral(address addr) internal {
        require(_soldTokenIds[addr].length > 0, "There is no mineral to be take");

        uint256 amount = _takeableMineral[addr];
        uint256[] memory tokenIds = _soldTokenIds[addr];
        _takeableMineral[addr] = 0;
        _soldTokenIds[addr].length = 0;

        _mineral.safeTransfer(addr, amount);
        emit TakeMineral(addr, amount, tokenIds);
    }
```

✓ The code meets the specification.

Formal Verification Request 39

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 6.3 ms

Line 219 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 225-227 in File MineralNFTMarket.sol

```
    function getSoldTokenIds(address addr) external view returns (uint256[] memory) {
        return _soldTokenIds[addr];
    }
```

✓ The code meets the specification.

Formal Verification Request 40

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.38 ms

Line 220 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 225-227 in File MineralNFTMarket.sol

```
    function getSoldTokenIds(address addr) external view returns (uint256[] memory) {
        return _soldTokenIds[addr];
    }
```

✓ The code meets the specification.

Formal Verification Request 41

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.37 ms

Line 221 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 225-227 in File MineralNFTMarket.sol

```
function getSoldTokenIds(address addr) external view returns (uint256[] memory) {
    return _soldTokenIds[addr];
}
```

✓ The code meets the specification.

Formal Verification Request 42

getSoldTokenIds

23, Dec 2019
0.39 ms

Line 222-224 in File MineralNFTMarket.sol

```
/*@CTK getSoldTokenIds
  @post !__reverted -> __return == _soldTokenIds[addr]
*/
```

Line 225-227 in File MineralNFTMarket.sol

```
function getSoldTokenIds(address addr) external view returns (uint256[] memory) {
    return _soldTokenIds[addr];
}
```

✓ The code meets the specification.

Formal Verification Request 43

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.65 ms

Line 228 in File MineralNFTMarket.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 231-233 in File MineralNFTMarket.sol

```
function setMineralNFTTokenContract(address addr) public onlyOwner {
    _nft = MineralNFT(addr);
}
```

✓ The code meets the specification.

Formal Verification Request 44

If method completes, integer overflow would not happen.

23, Dec 2019
0.64 ms

Line 229 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 231-233 in File MineralNFTMarket.sol

```
    function setMineralNFTTokenContract(address addr) public onlyOwner {
        _nft = MineralNFT(addr);
    }
```

✓ The code meets the specification.

Formal Verification Request 45

Method will not encounter an assertion failure.

23, Dec 2019
0.61 ms

Line 230 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 231-233 in File MineralNFTMarket.sol

```
    function setMineralNFTTokenContract(address addr) public onlyOwner {
        _nft = MineralNFT(addr);
    }
```

✓ The code meets the specification.

Formal Verification Request 46

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.63 ms

Line 234 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 237-239 in File MineralNFTMarket.sol

```
    function setMineralTokenContract(address addr) public onlyOwner {
        _mineral = IERC20(addr);
    }
```

✓ The code meets the specification.

Formal Verification Request 47

If method completes, integer overflow would not happen.

23, Dec 2019
0.63 ms

Line 235 in File MineralNFTMarket.sol

```
//@CTK NO_OVERFLOW
```

Line 237-239 in File MineralNFTMarket.sol

```
function setMineralTokenContract(address addr) public onlyOwner {
    _mineral = IERC20(addr);
}
```

✓ The code meets the specification.

Formal Verification Request 48

Method will not encounter an assertion failure.

23, Dec 2019
0.62 ms

Line 236 in File MineralNFTMarket.sol

```
//@CTK NO_ASF
```

Line 237-239 in File MineralNFTMarket.sol

```
function setMineralTokenContract(address addr) public onlyOwner {
    _mineral = IERC20(addr);
}
```

✓ The code meets the specification.

Formal Verification Request 49

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
48.5 ms

Line 242 in File MineralNFTMarket.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 249-251 in File MineralNFTMarket.sol

```
function getTakeableMineral(address addr) external view onlyOwner returns (uint256) {
    return _takeableMineral[addr];
}
```

✓ The code meets the specification.

Formal Verification Request 50

If method completes, integer overflow would not happen.

23, Dec 2019
0.64 ms

Line 243 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 249-251 in File MineralNFTMarket.sol

```
    function getTakeableMineral(address addr) external view onlyOwner returns (uint256) {
        return _takeableMineral[addr];
    }
```

✓ The code meets the specification.

Formal Verification Request 51

Method will not encounter an assertion failure.

23, Dec 2019
0.65 ms

Line 244 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 249-251 in File MineralNFTMarket.sol

```
    function getTakeableMineral(address addr) external view onlyOwner returns (uint256) {
        return _takeableMineral[addr];
    }
```

✓ The code meets the specification.

Formal Verification Request 52

getTakeableMineral

23, Dec 2019
0.7 ms

Line 245-248 in File MineralNFTMarket.sol

```
    /*@CTK getTakeableMineral
      @pre msg.sender == _owner
      @post !__reverted -> __return == _takeableMineral[addr]
    */
```

Line 249-251 in File MineralNFTMarket.sol

```
    function getTakeableMineral(address addr) external view onlyOwner returns (uint256) {
        return _takeableMineral[addr];
    }
```

✓ The code meets the specification.

Formal Verification Request 53

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 80.42 ms

Line 253 in File MineralNFTMarket.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 262-264 in File MineralNFTMarket.sol

```
    function takeMineralOwnerable(address addr) external onlyOwner {
        _takeMineral(addr);
    }
```

✓ The code meets the specification.

Formal Verification Request 54

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.97 ms

Line 254 in File MineralNFTMarket.sol

```
    //@CTK NO_OVERFLOW
```

Line 262-264 in File MineralNFTMarket.sol

```
    function takeMineralOwnerable(address addr) external onlyOwner {
        _takeMineral(addr);
    }
```

✓ The code meets the specification.

Formal Verification Request 55

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.91 ms

Line 255 in File MineralNFTMarket.sol

```
    //@CTK NO_ASF
```

Line 262-264 in File MineralNFTMarket.sol

```
    function takeMineralOwnerable(address addr) external onlyOwner {
        _takeMineral(addr);
    }
```

✓ The code meets the specification.

Formal Verification Request 56

takeMineralOwnerable

📅 23, Dec 2019
⏱ 3.84 ms

Line 256-261 in File MineralNFTMarket.sol

```
    /*@CTK takeMineralOwnerable
      @pre msg.sender == _owner
      @pre _soldTokenIds[addr].length > 0
      @post __post._takeableMineral[addr] == 0
      @post __post._soldTokenIds[addr].length == 0
    */
```

Line 262-264 in File MineralNFTMarket.sol

```
    function takeMineralOwnerable(address addr) external onlyOwner {
        _takeMineral(addr);
    }
```

✓ The code meets the specification.

Formal Verification Request 57

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 255.73 ms

Line 27 in File Mineral.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 35-57 in File Mineral.sol

```
    function lock(bytes32 _reason, uint256 _amount, uint256 _time)
        public
        returns (bool)
    {
        uint256 validUntil = now.add(_time); //solhint-disable-line

        // If tokens are already locked, then functions extendLock or
        // increaseLockAmount should be used to make any changes
        require(tokensLocked(_msgSender(), _reason) == 0, ALREADY_LOCKED);
        require(_amount != 0, AMOUNT_ZERO);

        if (locked[_msgSender()][_reason].amount == 0)
            lockReason[_msgSender()].push(_reason);

        transfer(address(this), _amount);

        locked[_msgSender()][_reason] = lockToken(_amount, validUntil, false);

        emit Locked(_msgSender(), _reason, _amount, validUntil);
        return true;
    }
```

The code meets the specification.

Formal Verification Request 58

Method will not encounter an assertion failure.

23, Dec 2019
13.61 ms

Line 28 in File Mineral.sol

```
28      //@CTK NO_ASF
```

Line 35-57 in File Mineral.sol

```
35      function lock(bytes32 _reason, uint256 _amount, uint256 _time)
36          public
37          returns (bool)
38      {
39          uint256 validUntil = now.add(_time); //solhint-disable-line
40
41          // If tokens are already locked, then functions extendLock or
42          // increaseLockAmount should be used to make any changes
43          require(tokensLocked(_msgSender(), _reason) == 0, ALREADY_LOCKED);
44          require(_amount != 0, AMOUNT_ZERO);
45
46          if (locked[_msgSender()][_reason].amount == 0)
47              lockReason[_msgSender()].push(_reason);
48
49          transfer(address(this), _amount);
50
51          locked[_msgSender()][_reason] = lockToken(_amount, validUntil, false);
52
53          emit Locked(_msgSender(), _reason, _amount, validUntil);
54          return true;
55      }
```

The code meets the specification.

Formal Verification Request 59

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
92.58 ms

Line 67 in File Mineral.sol

```
67       //@CTK NO_BUF_OVERFLOW
```

Line 75-95 in File Mineral.sol

```
75      function transferWithLock(address _to, bytes32 _reason, uint256 _amount, uint256 _time)
76          external
77          returns (bool)
78      {
79          uint256 validUntil = now.add(_time); //solhint-disable-line
```

```

    require(tokensLocked(_to, _reason) == 0, ALREADY_LOCKED);
    require(_amount != 0, AMOUNT_ZERO);

    if (locked[_to][_reason].amount == 0)
       lockReason[_to].push(_reason);

    transfer(address(this), _amount);

    locked[_to][_reason] = lockToken(_amount, validUntil, false);

    emit Locked(_to, _reason, _amount, validUntil);
    return true;
 }
```

✓ The code meets the specification.

Formal Verification Request 60

Method will not encounter an assertion failure.

23, Dec 2019
5.86 ms

Line 68 in File Mineral.sol

```
    //@CTK NO_ASF
```

Line 75-95 in File Mineral.sol

```
 function transferWithLock(address _to, bytes32 _reason, uint256 _amount, uint256 _time)
    external
    returns (bool)
 {
    uint256 validUntil = now.add(_time); //solhint-disable-line

    require(tokensLocked(_to, _reason) == 0, ALREADY_LOCKED);
    require(_amount != 0, AMOUNT_ZERO);

    if (locked[_to][_reason].amount == 0)
       lockReason[_to].push(_reason);

    transfer(address(this), _amount);

    locked[_to][_reason] = lockToken(_amount, validUntil, false);

    emit Locked(_to, _reason, _amount, validUntil);
    return true;
 }
```

✓ The code meets the specification.

Formal Verification Request 61

Buffer overflow / array index out of bound would never happen.

23, Dec 2019

⏱ 0.44 ms

Line 104 in File Mineral.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 111-118 in File Mineral.sol

```
    function tokensLocked(address _of, bytes32 _reason)
        public
        view
        returns (uint256 amount)
    {
        if (!locked[_of][_reason].claimed)
            amount = locked[_of][_reason].amount;
    }
```

✓ The code meets the specification.

Formal Verification Request 62

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.4 ms

Line 105 in File Mineral.sol

```
    //@CTK NO_OVERFLOW
```

Line 111-118 in File Mineral.sol

```
    function tokensLocked(address _of, bytes32 _reason)
        public
        view
        returns (uint256 amount)
    {
        if (!locked[_of][_reason].claimed)
            amount = locked[_of][_reason].amount;
    }
```

✓ The code meets the specification.

Formal Verification Request 63

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.39 ms

Line 106 in File Mineral.sol

```
    //@CTK NO_ASF
```

Line 111-118 in File Mineral.sol

```
111     function tokensLocked(address _of, bytes32 _reason)
112         public
113         view
114         returns (uint256 amount)
115     {
116         if (!locked[_of][_reason].claimed)
117             amount = locked[_of][_reason].amount;
118     }
```

✓ The code meets the specification.

Formal Verification Request 64

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
7.74 ms

Line 128 in File Mineral.sol

```
128      //@CTK NO_BUF_OVERFLOW
```

Line 135-142 in File Mineral.sol

```
135     function tokensLockedAtTime(address _of, bytes32 _reason, uint256 _time)
136         public
137         view
138         returns (uint256 amount)
139     {
140         if (locked[_of][_reason].validity > _time)
141             amount = locked[_of][_reason].amount;
142     }
```

✓ The code meets the specification.

Formal Verification Request 65

If method completes, integer overflow would not happen.

23, Dec 2019
0.44 ms

Line 129 in File Mineral.sol

```
129      //@CTK NO_OVERFLOW
```

Line 135-142 in File Mineral.sol

```
135     function tokensLockedAtTime(address _of, bytes32 _reason, uint256 _time)
136         public
137         view
138         returns (uint256 amount)
139     {
140         if (locked[_of][_reason].validity > _time)
141             amount = locked[_of][_reason].amount;
142     }
```

✓ The code meets the specification.

Formal Verification Request 66

Method will not encounter an assertion failure.

23, Dec 2019
0.4 ms

Line 130 in File Mineral.sol

```
    //@CTK NO_ASF
```

Line 135-142 in File Mineral.sol

```
    function tokensLockedAtTime(address _of, bytes32 _reason, uint256 _time)
        public
        view
        returns (uint256 amount)
    {
        if (locked[_of][_reason].validity > _time)
            amount = locked[_of][_reason].amount;
    }
```

✓ The code meets the specification.

Formal Verification Request 67

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
34.14 ms

Line 148 in File Mineral.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 154-165 in File Mineral.sol

```
    function totalBalanceOf(address _of)
        public
        view
        returns (uint256 amount)
    {
        amount = balanceOf(_of);
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            amount = amount.add(tokensLocked(_of, lockReason[_of][i]));
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 68

If method completes, integer overflow would not happen.

23, Dec 2019
0.54 ms

Line 149 in File Mineral.sol

```
    //@CTK NO_OVERFLOW
```

Line 154-165 in File Mineral.sol

```
    function totalBalanceOf(address _of)
        public
        view
        returns (uint256 amount)
    {
        amount = balanceOf(_of);
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            amount = amount.add(tokensLocked(_of, lockReason[_of][i]));
        }
    }
```

✅ The code meets the specification.

Formal Verification Request 69

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.5 ms

Line 150 in File Mineral.sol

```
    //@CTK NO_ASF
```

Line 154-165 in File Mineral.sol

```
    function totalBalanceOf(address _of)
        public
        view
        returns (uint256 amount)
    {
        amount = balanceOf(_of);
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            amount = amount.add(tokensLocked(_of, lockReason[_of][i]));
        }
    }
```

✅ The code meets the specification.

Formal Verification Request 70

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 163.18 ms

Line 172 in File Mineral.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 180-190 in File Mineral.sol

```
function extendLock(bytes32 _reason, uint256 _time)
    public
    returns (bool)
{
    require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);

    locked[_msgSender()][_reason].validity = locked[_msgSender()][_reason].validity.add(
        _time);

    emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
        _msgSender()][_reason].validity);
    return true;
}
```

✓ The code meets the specification.

Formal Verification Request 71

If method completes, integer overflow would not happen.

23, Dec 2019
3.48 ms

Line 173 in File Mineral.sol

```
    //@CTK NO_OVERFLOW
```

Line 180-190 in File Mineral.sol

```
function extendLock(bytes32 _reason, uint256 _time)
    public
    returns (bool)
{
    require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);

    locked[_msgSender()][_reason].validity = locked[_msgSender()][_reason].validity.add(
        _time);

    emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
        _msgSender()][_reason].validity);
    return true;
}
```

✓ The code meets the specification.

Formal Verification Request 72

Method will not encounter an assertion failure.

23, Dec 2019
1.42 ms

Line 174 in File Mineral.sol

```
    //@CTK NO_ASF
```

Line 180-190 in File Mineral.sol

```
function extendLock(bytes32 _reason, uint256 _time)
    public
    returns (bool)
{
    require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);

    locked[_msgSender()][_reason].validity = locked[_msgSender()][_reason].validity.add(
        _time);

    emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
        _msgSender()][_reason].validity);
    return true;
}
```

✓ The code meets the specification.

Formal Verification Request 73

extendLock

📅 23, Dec 2019
⏱ 19.36 ms

Line 175-179 in File Mineral.sol

```
/*@CTK extendLock
  @tag assume_completion
  @pre (!locked[msg.sender][_reason].claimed && locked[msg.sender][_reason].amount > 0)
  @post __post.locked[msg.sender][_reason].validity == locked[msg.sender][_reason].
      validity + _time
*/
```

Line 180-190 in File Mineral.sol

```
function extendLock(bytes32 _reason, uint256 _time)
    public
    returns (bool)
{
    require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);

    locked[_msgSender()][_reason].validity = locked[_msgSender()][_reason].validity.add(
        _time);

    emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
        _msgSender()][_reason].validity);
    return true;
}
```

✓ The code meets the specification.

Formal Verification Request 74

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 183.22 ms

Line 197 in File Mineral.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 205-218 in File Mineral.sol

```
    function increaseLockAmount(bytes32 _reason, uint256 _amount)
        public
        returns (bool)
    {
        require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
        transfer(address(this), _amount);

        locked[_msgSender()][_reason].amount = locked[_msgSender()][_reason].amount.add(
            _amount);

        emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
            _msgSender()][_reason].validity);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 75

If method completes, integer overflow would not happen.

23, Dec 2019
4.03 ms

Line 198 in File Mineral.sol

```
    //@CTK NO_OVERFLOW
```

Line 205-218 in File Mineral.sol

```
    function increaseLockAmount(bytes32 _reason, uint256 _amount)
        public
        returns (bool)
    {
        require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
        transfer(address(this), _amount);

        locked[_msgSender()][_reason].amount = locked[_msgSender()][_reason].amount.add(
            _amount);

        emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
            _msgSender()][_reason].validity);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 76

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 1.81 ms

Line 199 in File Mineral.sol

```
    //@CTK NO_ASF
```

Line 205-218 in File Mineral.sol

```
    function increaseLockAmount(bytes32 _reason, uint256 _amount)
        public
        returns (bool)
    {
        require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
        transfer(address(this), _amount);

        locked[_msgSender()][_reason].amount = locked[_msgSender()][_reason].amount.add(
            _amount);

        emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
            _msgSender()][_reason].validity);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 77

increaseLockAmount

📅 23, Dec 2019
⏱ 24.71 ms

Line 200-204 in File Mineral.sol

```
    /*@CTK increaseLockAmount
      @tag assume_completion
      @pre (!locked[msg.sender][_reason].claimed && locked[msg.sender][_reason].amount > 0)
      @post __post.locked[msg.sender][_reason].amount == locked[msg.sender][_reason].amount
          + _amount
    */
```

Line 205-218 in File Mineral.sol

```
    function increaseLockAmount(bytes32 _reason, uint256 _amount)
        public
        returns (bool)
    {
        require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
        transfer(address(this), _amount);

        locked[_msgSender()][_reason].amount = locked[_msgSender()][_reason].amount.add(
            _amount);

        emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
            _msgSender()][_reason].validity);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 78

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
10.68 ms

Line 225 in File Mineral.sol

```
     //@CTK NO_BUF_OVERFLOW
```

Line 231-238 in File Mineral.sol

```
    function tokensUnlockable(address _of, bytes32 _reason)
        public
        view
        returns (uint256 amount)
    {
        if (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) //solhint-
            disable-line
            amount = locked[_of][_reason].amount;
    }
```

✓ The code meets the specification.

Formal Verification Request 79

If method completes, integer overflow would not happen.

23, Dec 2019
0.92 ms

Line 226 in File Mineral.sol

```
     //@CTK NO_OVERFLOW
```

Line 231-238 in File Mineral.sol

```
    function tokensUnlockable(address _of, bytes32 _reason)
        public
        view
        returns (uint256 amount)
    {
        if (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) //solhint-
            disable-line
            amount = locked[_of][_reason].amount;
    }
```

✓ The code meets the specification.

Formal Verification Request 80

Method will not encounter an assertion failure.

23, Dec 2019
1.2 ms

Line 227 in File Mineral.sol

```
    //@CTK NO_ASF
```

Line 231-238 in File Mineral.sol

```
    function tokensUnlockable(address _of, bytes32 _reason)
        public
        view
        returns (uint256 amount)
    {
        if (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) //solhint-disable-line
            amount = locked[_of][_reason].amount;
    }
```

✓ The code meets the specification.

Formal Verification Request 81

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
10.4 ms

Line 244 in File Mineral.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 247-275 in File Mineral.sol

```
    function unlockAll(address _of)
        public
        returns (uint256 unlockableTokens)
    {
        uint256 lockedTokens;

        /*@CTK "loop_unlockAll"
          @inv i <= lockReason[_of].length
          @inv forall j: uint. (j >= 0 /\ j < i /\ (locked[_of][lockReason[_of][i]].validity
              <= now && !locked[_of][lockReason[_of][i]].claimed && locked[_of][lockReason[
              _of][i]].amount > 0)) -> locked[_of][lockReason[_of][i]].claimed
          @post i == lockReason[_of].length
          @post !__should_return
        */
        /*@CTK loop
          @inv true
        */
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            lockedTokens = tokensUnlockable(_of, lockReason[_of][i]);
            if (lockedTokens > 0) {
                unlockableTokens = unlockableTokens.add(lockedTokens);
```

```
266                 locked[_of][lockReason[_of][i]].claimed = true;
267                 emit Unlocked(_of, lockReason[_of][i], lockedTokens);
268             }
269         }
270 
271         if (unlockableTokens > 0)
272             this.transfer(_of, unlockableTokens);
273     }
```

✓ The code meets the specification.

Formal Verification Request 82

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.58 ms

Line 245 in File Mineral.sol

```
245       //@CTK NO_OVERFLOW
```

Line 247-275 in File Mineral.sol

```
247     function unlockAll(address _of)
248         public
249         returns (uint256 unlockableTokens)
250     {
251         uint256 lockedTokens;
252 
253         /*#CTK "loop_unlockAll"
254           @inv i <= lockReason[_of].length
255           @inv forall j: uint. (j >= 0 /\ j < i /\ (locked[_of][lockReason[_of][i]].validity
              <= now && !locked[_of][lockReason[_of][i]].claimed && locked[_of][lockReason[
              _of][i]].amount > 0)) -> locked[_of][lockReason[_of][i]].claimed
256           @post i == lockReason[_of].length
257           @post !__should_return
258         */
259         /*@CTK loop
260           @inv true
261         */
262         for (uint256 i = 0; i < lockReason[_of].length; i++) {
263             lockedTokens = tokensUnlockable(_of, lockReason[_of][i]);
264             if (lockedTokens > 0) {
265                 unlockableTokens = unlockableTokens.add(lockedTokens);
266                 locked[_of][lockReason[_of][i]].claimed = true;
267                 emit Unlocked(_of, lockReason[_of][i], lockedTokens);
268             }
269         }
270 
271         if (unlockableTokens > 0)
272             this.transfer(_of, unlockableTokens);
273     }
```

✓ The code meets the specification.

Formal Verification Request 83

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.51 ms

Line 246 in File Mineral.sol

```
    //@CTK NO_ASF
```

Line 247-275 in File Mineral.sol

```
    function unlockAll(address _of)
        public
        returns (uint256 unlockableTokens)
    {
        uint256 lockedTokens;

        /*#CTK "loop_unlockAll"
          @inv i <= lockReason[_of].length
          @inv forall j: uint. (j >= 0 /\ j < i /\ (locked[_of][lockReason[_of][i]].validity
              <= now && !locked[_of][lockReason[_of][i]].claimed && locked[_of][lockReason[
              _of][i]].amount > 0)) -> locked[_of][lockReason[_of][i]].claimed
          @post i == lockReason[_of].length
          @post !__should_return
        */
        /*@CTK loop
          @inv true
        */
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            lockedTokens = tokensUnlockable(_of, lockReason[_of][i]);
            if (lockedTokens > 0) {
                unlockableTokens = unlockableTokens.add(lockedTokens);
                locked[_of][lockReason[_of][i]].claimed = true;
                emit Unlocked(_of, lockReason[_of][i], lockedTokens);
            }
        }

        if (unlockableTokens > 0)
            this.transfer(_of, unlockableTokens);
    }
```

✓ The code meets the specification.

Formal Verification Request 84

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 52.85 ms

Line 282 in File Mineral.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 288-300 in File Mineral.sol

```
    function unlock(address _of, bytes32 _reason)
        public
        returns (uint256 unlocked)
    {
        unlocked = tokensUnlockable(_of, _reason);
        if (unlocked > 0) {
            locked[_of][_reason].claimed = true;
            emit Unlocked(_of, _reason, unlocked);
            this.transfer(_of, unlocked);
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 85

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.64 ms

Line 283 in File Mineral.sol

```
     //@CTK NO_OVERFLOW
```

Line 288-300 in File Mineral.sol

```
    function unlock(address _of, bytes32 _reason)
        public
        returns (uint256 unlocked)
    {
        unlocked = tokensUnlockable(_of, _reason);
        if (unlocked > 0) {
            locked[_of][_reason].claimed = true;
            emit Unlocked(_of, _reason, unlocked);
            this.transfer(_of, unlocked);
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 86

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.61 ms

Line 284 in File Mineral.sol

```
     //@CTK NO_ASF
```

Line 288-300 in File Mineral.sol

```
    function unlock(address _of, bytes32 _reason)
        public
        returns (uint256 unlocked)
    {
        unlocked = tokensUnlockable(_of, _reason);
        if (unlocked > 0) {
            locked[_of][_reason].claimed = true;
            emit Unlocked(_of, _reason, unlocked);
            this.transfer(_of, unlocked);
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 87

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
3.92 ms

Line 306 in File Mineral.sol

```
     //@CTK NO_BUF_OVERFLOW
```

Line 309-319 in File Mineral.sol

```
    function getUnlockableTokens(address _of)
        public
        view
        returns (uint256 unlockableTokens)
    {
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            unlockableTokens = unlockableTokens.add(tokensUnlockable(_of, lockReason[_of][i])
                );
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 88

If method completes, integer overflow would not happen.

23, Dec 2019
0.38 ms

Line 307 in File Mineral.sol

```
     //@CTK NO_OVERFLOW
```

Line 309-319 in File Mineral.sol

```
    function getUnlockableTokens(address _of)
        public
        view
        returns (uint256 unlockableTokens)
    {
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            unlockableTokens = unlockableTokens.add(tokensUnlockable(_of, lockReason[_of][i])
                );
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 89

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.37 ms

Line 308 in File Mineral.sol

```
     //@CTK NO_ASF
```

Line 309-319 in File Mineral.sol

```
    function getUnlockableTokens(address _of)
        public
        view
        returns (uint256 unlockableTokens)
    {
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            unlockableTokens = unlockableTokens.add(tokensUnlockable(_of, lockReason[_of][i])
                );
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 90

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 27.29 ms

Line 320 in File Mineral.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 322-339 in File Mineral.sol

```
    function getLockReasons(address _of, uint256 _start, uint256 _end)
        external
        view
        returns (bytes32[] memory reasons)
    {
        uint256 length = _end - _start;
        reasons = new bytes32[](length);
        /*@CTK loop_getLockReasons
          @inv i <= length
      @inv forall j: uint. (j >= 0 /\ j < i) -> reasons[j] == this.lockReason[_of][_start +
          j]
          @post i == length
          @post !__should_return
        */
        for (uint256 i = 0; i < length; i++) {
            reasons[i] = lockReason[_of][_start + i];
        }
        return reasons;
    }
```

✓ The code meets the specification.

Formal Verification Request 91

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.62 ms

Line 321 in File Mineral.sol

```
    //@CTK NO_ASF
```

Line 322-339 in File Mineral.sol

```
    function getLockReasons(address _of, uint256 _start, uint256 _end)
        external
        view
        returns (bytes32[] memory reasons)
    {
        uint256 length = _end - _start;
        reasons = new bytes32[](length);
        /*@CTK loop_getLockReasons
          @inv i <= length
      @inv forall j: uint. (j >= 0 /\ j < i) -> reasons[j] == this.lockReason[_of][_start +
          j]
          @post i == length
          @post !__should_return
        */
        for (uint256 i = 0; i < length; i++) {
            reasons[i] = lockReason[_of][_start + i];
        }
        return reasons;
    }
```

✓ The code meets the specification.

Formal Verification Request 92

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 6.49 ms

Line 340 in File Mineral.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 343-349 in File Mineral.sol

```
    function getLockReasonLength(address _of)
        external
        view
        returns (uint256 length)
    {
        return lockReason[_of].length;
    }
```

✓ The code meets the specification.

Formal Verification Request 93

If method completes, integer overflow would not happen.

23, Dec 2019
0.4 ms

Line 341 in File Mineral.sol

```
    //@CTK NO_OVERFLOW
```

Line 343-349 in File Mineral.sol

```
    function getLockReasonLength(address _of)
        external
        view
        returns (uint256 length)
    {
        return lockReason[_of].length;
    }
```

✓ The code meets the specification.

Formal Verification Request 94

Method will not encounter an assertion failure.

23, Dec 2019
0.38 ms

Line 342 in File Mineral.sol

```
    //@CTK NO_ASF
```

Line 343-349 in File Mineral.sol

```
    function getLockReasonLength(address _of)
        external
        view
        returns (uint256 length)
    {
        return lockReason[_of].length;
    }
```

✓ The code meets the specification.

Formal Verification Request 95

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 3.75 ms

Line 350 in File Mineral.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 353-360 in File Mineral.sol

```
    function safeTransfer(address _to, uint256 _amount, bytes calldata _data)
        external
    {
        require(transfer(_to, _amount), "ERC20: failed transfer");
        require(_checkOnERC20Received(_to, _amount, _data), "ERC20: transfer to non
            ERC20Receiver implementer");
    }
```

✅ The code meets the specification.

Formal Verification Request 96

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.38 ms

Line 351 in File Mineral.sol

```
    //@CTK NO_OVERFLOW
```

Line 353-360 in File Mineral.sol

```
    function safeTransfer(address _to, uint256 _amount, bytes calldata _data)
        external
    {
        require(transfer(_to, _amount), "ERC20: failed transfer");
        require(_checkOnERC20Received(_to, _amount, _data), "ERC20: transfer to non
            ERC20Receiver implementer");
    }
```

✅ The code meets the specification.

Formal Verification Request 97

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.38 ms

Line 352 in File Mineral.sol

```
    //@CTK NO_ASF
```

Line 353-360 in File Mineral.sol

```
353    function safeTransfer(address _to, uint256 _amount, bytes calldata _data)
354        external
355    {
356        require(transfer(_to, _amount), "ERC20: failed transfer");
357        require(_checkOnERC20Received(_to, _amount, _data), "ERC20: transfer to non
               ERC20Receiver implementer");
358    }
```

The code meets the specification.

Formal Verification Request 98

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
4.49 ms

Line 362 in File Mineral.sol

```
362    //@CTK NO_BUF_OVERFLOW
```

Line 363-374 in File Mineral.sol

```
363    function _checkOnERC20Received(address _to, uint256 _amount, bytes memory _data)
364        internal
365        returns (bool)
366    {
367        if (!_to.isContract()) {
368            return true;
369        }
370
371        return IERC20Receiver(_to).onERC20Received(_msgSender(), _amount, _data);
372    }
```

The code meets the specification.

Formal Verification Request 99

loop___Generated

23, Dec 2019
86.62 ms

(Loop) Line 259-261 in File Mineral.sol

```
259        /*@CTK loop
260          @inv true
261        */
```

(Loop) Line 259-269 in File Mineral.sol

```
259        /*@CTK loop
260          @inv true
261        */
262        for (uint256 i = 0; i < lockReason[_of].length; i++) {
263            lockedTokens = tokensUnlockable(_of, lockReason[_of][i]);
264            if (lockedTokens > 0) {
```

```
                unlockableTokens = unlockableTokens.add(lockedTokens);
                locked[_of][lockReason[_of][i]].claimed = true;
                emit Unlocked(_of, lockReason[_of][i], lockedTokens);
            }
        }
```

✓ The code meets the specification.

Formal Verification Request 100

loop__getLockReasons___Generated

📅 23, Dec 2019
⏱ 586.58 ms

(Loop) Line 329-334 in File Mineral.sol

```
        /*@CTK loop_getLockReasons
          @inv i <= length
      @inv forall j: uint. (j >= 0 /\ j < i) -> reasons[j] == this.lockReason[_of][_start +
          j]
          @post i == length
          @post !__should_return
        */
```

(Loop) Line 329-337 in File Mineral.sol

```
        /*@CTK loop_getLockReasons
          @inv i <= length
      @inv forall j: uint. (j >= 0 /\ j < i) -> reasons[j] == this.lockReason[_of][_start +
          j]
          @post i == length
          @post !__should_return
        */
        for (uint256 i = 0; i < length; i++) {
            reasons[i] = lockReason[_of][_start + i];
        }
```

✓ The code meets the specification.

Formal Verification Request 101

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 13.67 ms

Line 14 in File MineralNFT.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 20-22 in File MineralNFT.sol

```
    function _generateTokenId() internal returns (uint256) {
        return _finalTokenId++;
    }
```

✓ The code meets the specification.

Formal Verification Request 102

Method will not encounter an assertion failure.

23, Dec 2019
0.49 ms

Line 15 in File MineralNFT.sol

```
    //@CTK NO_ASF
```

Line 20-22 in File MineralNFT.sol

```
    function _generateTokenId() internal returns (uint256) {
        return _finalTokenId++;
    }
```

✓ The code meets the specification.

Formal Verification Request 103

_generateTokenId

23, Dec 2019
1.13 ms

Line 16-19 in File MineralNFT.sol

```
    /*@CTK _generateTokenId
      @post __post._finalTokenId == _finalTokenId + 1
      @post !__reverted -> __return == _finalTokenId
    */
```

Line 20-22 in File MineralNFT.sol

```
    function _generateTokenId() internal returns (uint256) {
        return _finalTokenId++;
    }
```

✓ The code meets the specification.

Formal Verification Request 104

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
388.82 ms

Line 24 in File MineralNFT.sol

```
  //@CTK NO_BUF_OVERFLOW
```

Line 29-34 in File MineralNFT.sol

```
    function createItem(address to, string calldata jsonUrl) external onlyOwner returns (
        uint256) {
        uint256 id = _generateTokenId();
        _mint(to, id);
        _setTokenURI(id, jsonUrl);
        return id;
    }
```

✓ The code meets the specification.

Formal Verification Request 105

Method will not encounter an assertion failure.

23, Dec 2019
19.82 ms

Line 25 in File MineralNFT.sol

```
   //@CTK NO_ASF
```

Line 29-34 in File MineralNFT.sol

```
    function createItem(address to, string calldata jsonUrl) external onlyOwner returns (
        uint256) {
        uint256 id = _generateTokenId();
        _mint(to, id);
        _setTokenURI(id, jsonUrl);
        return id;
    }
```

✓ The code meets the specification.

Formal Verification Request 106

createItem

23, Dec 2019
24.71 ms

Line 26-28 in File MineralNFT.sol

```
    /*@CTK createItem
      @post !__reverted -> __return == _finalTokenId
    */
```

Line 29-34 in File MineralNFT.sol

```
    function createItem(address to, string calldata jsonUrl) external onlyOwner returns (
        uint256) {
        uint256 id = _generateTokenId();
        _mint(to, id);
        _setTokenURI(id, jsonUrl);
        return id;
    }
```

✓ The code meets the specification.

Formal Verification Request 107

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 4.24 ms

Line 35 in File MineralNFT.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 38-43 in File MineralNFT.sol

```
    function burnItem(uint256 tokenId) external {
        require(_isApprovedOrOwner(_msgSender(), tokenId), "msg.sender is not token owner");
        _burn(_msgSender(), tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 108

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.42 ms

Line 36 in File MineralNFT.sol

```
    //@CTK NO_OVERFLOW
```

Line 38-43 in File MineralNFT.sol

```
    function burnItem(uint256 tokenId) external {
        require(_isApprovedOrOwner(_msgSender(), tokenId), "msg.sender is not token owner");
        _burn(_msgSender(), tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 109

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.44 ms

Line 37 in File MineralNFT.sol

```
    //@CTK NO_ASF
```

Line 38-43 in File MineralNFT.sol

```
    function burnItem(uint256 tokenId) external {
        require(_isApprovedOrOwner(_msgSender(), tokenId), "msg.sender is not token owner");
        _burn(_msgSender(), tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 110

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 5.66 ms

Line 46 in File ERC20.sol

```
    //@CTK NO_ASF
```

Line 49-51 in File ERC20.sol

```
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }
```

✓ The code meets the specification.

Formal Verification Request 111

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.4 ms

Line 47 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 49-51 in File ERC20.sol

```
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }
```

✓ The code meets the specification.

Formal Verification Request 112

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.36 ms

Line 48 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 49-51 in File ERC20.sol

```
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }
```

✓ The code meets the specification.

Formal Verification Request 113

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 6.02 ms

Line 56 in File ERC20.sol

```
    //@CTK NO_ASF
```

Line 59-61 in File ERC20.sol

```
   function balanceOf(address account) public view returns (uint256) {
       return _balances[account];
   }
```

✓ The code meets the specification.

Formal Verification Request 114

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.38 ms

Line 57 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 59-61 in File ERC20.sol

```
   function balanceOf(address account) public view returns (uint256) {
       return _balances[account];
   }
```

✓ The code meets the specification.

Formal Verification Request 115

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.36 ms

Line 58 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 59-61 in File ERC20.sol

```
   function balanceOf(address account) public view returns (uint256) {
       return _balances[account];
   }
```

✓ The code meets the specification.

Formal Verification Request 116

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 6.28 ms

Line 79 in File ERC20.sol

```
    //@CTK NO_ASF
```

Line 82-84 in File ERC20.sol

```
  function allowance(address owner, address spender) public view returns (uint256) {
      return _allowances[owner][spender];
  }
```

✓ The code meets the specification.

Formal Verification Request 117

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.39 ms

Line 80 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 82-84 in File ERC20.sol

```
  function allowance(address owner, address spender) public view returns (uint256) {
      return _allowances[owner][spender];
  }
```

✓ The code meets the specification.

Formal Verification Request 118

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.4 ms

Line 81 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 82-84 in File ERC20.sol

```
  function allowance(address owner, address spender) public view returns (uint256) {
      return _allowances[owner][spender];
  }
```

✓ The code meets the specification.

Formal Verification Request 119

Method will not encounter an assertion failure.

23, Dec 2019
88.32 ms

Line 93 in File ERC20.sol

```
    //@CTK NO_ASF
```

Line 96-99 in File ERC20.sol

```
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 120

If method completes, integer overflow would not happen.

23, Dec 2019
0.76 ms

Line 94 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 96-99 in File ERC20.sol

```
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 121

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.68 ms

Line 95 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 96-99 in File ERC20.sol

```
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 122

Method will not encounter an assertion failure.

23, Dec 2019
99.21 ms

Line 131 in File ERC20.sol

```
    //@CTK NO_ASF
```

Line 134-137 in File ERC20.sol

```
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 123

If method completes, integer overflow would not happen.

23, Dec 2019
3.23 ms

Line 132 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 134-137 in File ERC20.sol

```
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 124

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
1.15 ms

Line 133 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 134-137 in File ERC20.sol

```
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 125

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 72.44 ms

Line 190 in File ERC20.sol

```
    //@CTK NO_ASF
```

Line 193-199 in File ERC20.sol

```
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 126

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 6.43 ms

Line 191 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 193-199 in File ERC20.sol

```
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 127

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 3.26 ms

Line 192 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 193-199 in File ERC20.sol

```
function _mint(address account, uint256 amount) internal {
    require(account != address(0), "ERC20: mint to the zero address");

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}
```

✓ The code meets the specification.

Formal Verification Request 128

Method will not encounter an assertion failure.

23, Dec 2019
0.57 ms

Line 233 in File ERC20.sol

```
    //@CTK NO_ASF
```

Line 236-242 in File ERC20.sol

```
function _approve(address owner, address spender, uint256 amount) internal {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}
```

✓ The code meets the specification.

Formal Verification Request 129

If method completes, integer overflow would not happen.

23, Dec 2019
0.52 ms

Line 234 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 236-242 in File ERC20.sol

```
function _approve(address owner, address spender, uint256 amount) internal {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}
```

✓ The code meets the specification.

Formal Verification Request 130

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.52 ms

Line 235 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 236-242 in File ERC20.sol

```
    function _approve(address owner, address spender, uint256 amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

✅ The code meets the specification.

Formal Verification Request 131

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 65.53 ms

Line 37 in File ERC721Enumerable.sol

```
    //@CTK NO_ASF
```

Line 40-43 in File ERC721Enumerable.sol

```
    constructor () public {
        // register the supported interface to conform to ERC721Enumerable via ERC165
        _registerInterface(_INTERFACE_ID_ERC721_ENUMERABLE);
    }
```

✅ The code meets the specification.

Formal Verification Request 132

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.74 ms

Line 38 in File ERC721Enumerable.sol

```
    //@CTK NO_OVERFLOW
```

Line 40-43 in File ERC721Enumerable.sol

```
    constructor () public {
        // register the supported interface to conform to ERC721Enumerable via ERC165
        _registerInterface(_INTERFACE_ID_ERC721_ENUMERABLE);
    }
```

✓ The code meets the specification.

Formal Verification Request 133

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.69 ms

Line 39 in File ERC721Enumerable.sol

```
39      //@CTK NO_BUF_OVERFLOW
```

Line 40-43 in File ERC721Enumerable.sol

```
40    constructor () public {
41        // register the supported interface to conform to ERC721Enumerable via ERC165
42        _registerInterface(_INTERFACE_ID_ERC721_ENUMERABLE);
43    }
```

✓ The code meets the specification.

Formal Verification Request 134

Method will not encounter an assertion failure.

23, Dec 2019
7.49 ms

Line 61 in File ERC721Enumerable.sol

```
61      //@CTK NO_ASF
```

Line 64-66 in File ERC721Enumerable.sol

```
64    function totalSupply() public view returns (uint256) {
65        return _allTokens.length;
66    }
```

✓ The code meets the specification.

Formal Verification Request 135

If method completes, integer overflow would not happen.

23, Dec 2019
0.39 ms

Line 62 in File ERC721Enumerable.sol

```
62      //@CTK NO_OVERFLOW
```

Line 64-66 in File ERC721Enumerable.sol

```
    function totalSupply() public view returns (uint256) {
        return _allTokens.length;
    }
```

✓ The code meets the specification.

Formal Verification Request 136

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.4 ms

Line 63 in File ERC721Enumerable.sol

```
     //@CTK NO_BUF_OVERFLOW
```

Line 64-66 in File ERC721Enumerable.sol

```
    function totalSupply() public view returns (uint256) {
        return _allTokens.length;
    }
```

✓ The code meets the specification.

Formal Verification Request 137

Method will not encounter an assertion failure.

23, Dec 2019
40.38 ms

Line 74 in File ERC721Enumerable.sol

```
     //@CTK NO_ASF
```

Line 77-80 in File ERC721Enumerable.sol

```
    function tokenByIndex(uint256 index) public view returns (uint256) {
        require(index < totalSupply(), "ERC721Enumerable: global index out of bounds");
        return _allTokens[index];
    }
```

✓ The code meets the specification.

Formal Verification Request 138

If method completes, integer overflow would not happen.

23, Dec 2019
0.64 ms

Line 75 in File ERC721Enumerable.sol

```
    //@CTK NO_OVERFLOW
```

Line 77-80 in File ERC721Enumerable.sol

```
function tokenByIndex(uint256 index) public view returns (uint256) {
    require(index < totalSupply(), "ERC721Enumerable: global index out of bounds");
    return _allTokens[index];
}
```

✓ The code meets the specification.

Formal Verification Request 139

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
1.5 ms

Line 76 in File ERC721Enumerable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 77-80 in File ERC721Enumerable.sol

```
function tokenByIndex(uint256 index) public view returns (uint256) {
    require(index < totalSupply(), "ERC721Enumerable: global index out of bounds");
    return _allTokens[index];
}
```

✓ The code meets the specification.

Formal Verification Request 140

Method will not encounter an assertion failure.

23, Dec 2019
273.39 ms

Line 103 in File ERC721Enumerable.sol

```
    //@CTK NO_ASF
```

Line 105-111 in File ERC721Enumerable.sol

```
function _mint(address to, uint256 tokenId) internal {
    super._mint(to, tokenId);

    _addTokenToOwnerEnumeration(to, tokenId);

    _addTokenToAllTokensEnumeration(tokenId);
}
```

✓ The code meets the specification.

Formal Verification Request 141

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 8.61 ms

Line 104 in File ERC721Enumerable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 105-111 in File ERC721Enumerable.sol

```
    function _mint(address to, uint256 tokenId) internal {
        super._mint(to, tokenId);

        _addTokenToOwnerEnumeration(to, tokenId);

        _addTokenToAllTokensEnumeration(tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 142

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 6.48 ms

Line 136 in File ERC721Enumerable.sol

```
    //@CTK NO_ASF
```

Line 139-141 in File ERC721Enumerable.sol

```
    function _tokensOfOwner(address owner) internal view returns (uint256[] storage) {
        return _ownedTokens[owner];
    }
```

✓ The code meets the specification.

Formal Verification Request 143

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.38 ms

Line 137 in File ERC721Enumerable.sol

```
    //@CTK NO_OVERFLOW
```

Line 139-141 in File ERC721Enumerable.sol

```
    function _tokensOfOwner(address owner) internal view returns (uint256[] storage) {
        return _ownedTokens[owner];
    }
```

✓ The code meets the specification.

Formal Verification Request 144

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.36 ms

Line 138 in File ERC721Enumerable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 139-141 in File ERC721Enumerable.sol

```
    function _tokensOfOwner(address owner) internal view returns (uint256[] storage) {
        return _ownedTokens[owner];
    }
```

✓ The code meets the specification.

Formal Verification Request 145

Method will not encounter an assertion failure.

23, Dec 2019
0.44 ms

Line 148 in File ERC721Enumerable.sol

```
    //@CTK NO_ASF
```

Line 150-153 in File ERC721Enumerable.sol

```
    function _addTokenToOwnerEnumeration(address to, uint256 tokenId) private {
        _ownedTokensIndex[tokenId] = _ownedTokens[to].length;
        _ownedTokens[to].push(tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 146

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.47 ms

Line 149 in File ERC721Enumerable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 150-153 in File ERC721Enumerable.sol

```
    function _addTokenToOwnerEnumeration(address to, uint256 tokenId) private {
        _ownedTokensIndex[tokenId] = _ownedTokens[to].length;
        _ownedTokens[to].push(tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 147

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.43 ms

Line 159 in File ERC721Enumerable.sol

```
    //@CTK NO_ASF
```

Line 161-164 in File ERC721Enumerable.sol

```
   function _addTokenToAllTokensEnumeration(uint256 tokenId) private {
       _allTokensIndex[tokenId] = _allTokens.length;
       _allTokens.push(tokenId);
   }
```

✓ The code meets the specification.

Formal Verification Request 148

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.45 ms

Line 160 in File ERC721Enumerable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 161-164 in File ERC721Enumerable.sol

```
   function _addTokenToAllTokensEnumeration(uint256 tokenId) private {
       _allTokensIndex[tokenId] = _allTokens.length;
       _allTokens.push(tokenId);
   }
```

✓ The code meets the specification.

Formal Verification Request 149

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 75.01 ms

Line 33 in File ERC721Metadata.sol

```
    //@CTK NO_ASF
```

Line 36-42 in File ERC721Metadata.sol

```
   constructor (string memory name, string memory symbol) public {
       _name = name;
       _symbol = symbol;

       // register the supported interfaces to conform to ERC721 via ERC165
       _registerInterface(_INTERFACE_ID_ERC721_METADATA);
   }
```

✓ The code meets the specification.

Formal Verification Request 150

If method completes, integer overflow would not happen.

23, Dec 2019
0.99 ms

Line 34 in File ERC721Metadata.sol

```
    //@CTK NO_OVERFLOW
```

Line 36-42 in File ERC721Metadata.sol

```
    constructor (string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;

        // register the supported interfaces to conform to ERC721 via ERC165
        _registerInterface(_INTERFACE_ID_ERC721_METADATA);
    }
```

✓ The code meets the specification.

Formal Verification Request 151

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.82 ms

Line 35 in File ERC721Metadata.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 36-42 in File ERC721Metadata.sol

```
    constructor (string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;

        // register the supported interfaces to conform to ERC721 via ERC165
        _registerInterface(_INTERFACE_ID_ERC721_METADATA);
    }
```

✓ The code meets the specification.

Formal Verification Request 152

Method will not encounter an assertion failure.

23, Dec 2019
6.01 ms

Line 48 in File ERC721Metadata.sol

```
    //@CTK NO_ASF
```

Line 51-53 in File ERC721Metadata.sol

```
    function name() external view returns (string memory) {
        return _name;
    }
```

✓ The code meets the specification.

Formal Verification Request 153

If method completes, integer overflow would not happen.

23, Dec 2019
0.44 ms

Line 49 in File ERC721Metadata.sol

```
    //@CTK NO_OVERFLOW
```

Line 51-53 in File ERC721Metadata.sol

```
    function name() external view returns (string memory) {
        return _name;
    }
```

✓ The code meets the specification.

Formal Verification Request 154

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.39 ms

Line 50 in File ERC721Metadata.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 51-53 in File ERC721Metadata.sol

```
    function name() external view returns (string memory) {
        return _name;
    }
```

✓ The code meets the specification.

Formal Verification Request 155

Method will not encounter an assertion failure.

23, Dec 2019
6.47 ms

Line 59 in File ERC721Metadata.sol

```
    //@CTK NO_ASF
```

Line 62-64 in File ERC721Metadata.sol

```
    function symbol() external view returns (string memory) {
        return _symbol;
    }
```

✓ The code meets the specification.

Formal Verification Request 156

If method completes, integer overflow would not happen.

23, Dec 2019
0.43 ms

Line 60 in File ERC721Metadata.sol

```
    //@CTK NO_OVERFLOW
```

Line 62-64 in File ERC721Metadata.sol

```
    function symbol() external view returns (string memory) {
        return _symbol;
    }
```

✓ The code meets the specification.

Formal Verification Request 157

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.45 ms

Line 61 in File ERC721Metadata.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 62-64 in File ERC721Metadata.sol

```
    function symbol() external view returns (string memory) {
        return _symbol;
    }
```

✓ The code meets the specification.

Formal Verification Request 158

Method will not encounter an assertion failure.

23, Dec 2019
49.59 ms

Line 74 in File ERC721Metadata.sol

```
    //@CTK NO_ASF
```

Line 77-91 in File ERC721Metadata.sol

```
    function tokenURI(uint256 tokenId) external view returns (string memory) {
        require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");

        string memory _tokenURI = _tokenURIs[tokenId];

        // Even if there is a base URI, it is only appended to non-empty token-specific URIs
        if (bytes(_tokenURI).length == 0) {
            return "";
        } else {
            // abi.encodePacked is being used to concatenate strings
            return string(abi.encodePacked(_baseURI, _tokenURI));
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 159

If method completes, integer overflow would not happen.

23, Dec 2019
0.85 ms

Line 75 in File ERC721Metadata.sol

```
    //@CTK NO_OVERFLOW
```

Line 77-91 in File ERC721Metadata.sol

```
    function tokenURI(uint256 tokenId) external view returns (string memory) {
        require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");

        string memory _tokenURI = _tokenURIs[tokenId];

        // Even if there is a base URI, it is only appended to non-empty token-specific URIs
        if (bytes(_tokenURI).length == 0) {
            return "";
        } else {
            // abi.encodePacked is being used to concatenate strings
            return string(abi.encodePacked(_baseURI, _tokenURI));
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 160

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.62 ms

Line 76 in File ERC721Metadata.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 77-91 in File ERC721Metadata.sol

```
  function tokenURI(uint256 tokenId) external view returns (string memory) {
    require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");

    string memory _tokenURI = _tokenURIs[tokenId];

    // Even if there is a base URI, it is only appended to non-empty token-specific URIs
    if (bytes(_tokenURI).length == 0) {
      return "";
    } else {
      // abi.encodePacked is being used to concatenate strings
      return string(abi.encodePacked(_baseURI, _tokenURI));
    }
  }
```

✓ The code meets the specification.

Formal Verification Request 161

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 5.79 ms

Line 98 in File ERC721Metadata.sol

```
  //@CTK NO_ASF
```

Line 101-103 in File ERC721Metadata.sol

```
  function baseURI() external view returns (string memory) {
    return _baseURI;
  }
```

✓ The code meets the specification.

Formal Verification Request 162

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.43 ms

Line 99 in File ERC721Metadata.sol

```
  //@CTK NO_OVERFLOW
```

Line 101-103 in File ERC721Metadata.sol

```
  function baseURI() external view returns (string memory) {
    return _baseURI;
  }
```

✓ The code meets the specification.

Formal Verification Request 163

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.37 ms

Line 100 in File ERC721Metadata.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 101-103 in File ERC721Metadata.sol

```
    function baseURI() external view returns (string memory) {
        return _baseURI;
    }
```

✓ The code meets the specification.

Formal Verification Request 164

Method will not encounter an assertion failure.

23, Dec 2019
38.57 ms

Line 114 in File ERC721Metadata.sol

```
     //@CTK NO_ASF
```

Line 117-120 in File ERC721Metadata.sol

```
    function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal {
        require(_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
        _tokenURIs[tokenId] = _tokenURI;
    }
```

✓ The code meets the specification.

Formal Verification Request 165

If method completes, integer overflow would not happen.

23, Dec 2019
0.57 ms

Line 115 in File ERC721Metadata.sol

```
     //@CTK NO_OVERFLOW
```

Line 117-120 in File ERC721Metadata.sol

```
    function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal {
        require(_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
        _tokenURIs[tokenId] = _tokenURI;
    }
```

✓ The code meets the specification.

Formal Verification Request 166

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.61 ms

Line 116 in File ERC721Metadata.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 117-120 in File ERC721Metadata.sol

```
   function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal {
      require(_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
      _tokenURIs[tokenId] = _tokenURI;
   }
```

✓ The code meets the specification.

Formal Verification Request 167

Method will not encounter an assertion failure.

23, Dec 2019
8.25 ms

Line 128 in File ERC721Metadata.sol

```
    //@CTK NO_ASF
```

Line 131-133 in File ERC721Metadata.sol

```
   function _setBaseURI(string memory uri) internal {
      _baseURI = uri;
   }
```

✓ The code meets the specification.

Formal Verification Request 168

If method completes, integer overflow would not happen.

23, Dec 2019
0.38 ms

Line 129 in File ERC721Metadata.sol

```
    //@CTK NO_OVERFLOW
```

Line 131-133 in File ERC721Metadata.sol

```
   function _setBaseURI(string memory uri) internal {
      _baseURI = uri;
   }
```

✓ The code meets the specification.

Formal Verification Request 169

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.37 ms

Line 130 in File ERC721Metadata.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 131-133 in File ERC721Metadata.sol

```
    function _setBaseURI(string memory uri) internal {
        _baseURI = uri;
    }
```

✓ The code meets the specification.

Formal Verification Request 170

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 12.9 ms

Line 142 in File ERC721Metadata.sol

```
    //@CTK NO_ASF
```

Line 145-154 in File ERC721Metadata.sol

```
    function _burn(address owner, uint256 tokenId) internal {
        super._burn(owner, tokenId);

        // Clear metadata (if any)
        if (bytes(_tokenURIs[tokenId]).length != 0) {
            delete _tokenURIs[tokenId];
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 171

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.4 ms

Line 143 in File ERC721Metadata.sol

```
    //@CTK NO_OVERFLOW
```

Line 145-154 in File ERC721Metadata.sol

```
    function _burn(address owner, uint256 tokenId) internal {
        super._burn(owner, tokenId);

        // Clear metadata (if any)
        if (bytes(_tokenURIs[tokenId]).length != 0) {
            delete _tokenURIs[tokenId];
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 172

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.4 ms

Line 144 in File ERC721Metadata.sol

```
     //@CTK NO_BUF_OVERFLOW
```

Line 145-154 in File ERC721Metadata.sol

```
    function _burn(address owner, uint256 tokenId) internal {
        super._burn(owner, tokenId);

        // Clear metadata (if any)
        if (bytes(_tokenURIs[tokenId]).length != 0) {
            delete _tokenURIs[tokenId];
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 173

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 61.14 ms

Line 51 in File ERC721.sol

```
    //@CTK NO_ASF
```

Line 54-57 in File ERC721.sol

```
    constructor () public {
        // register the supported interfaces to conform to ERC721 via ERC165
        _registerInterface(_INTERFACE_ID_ERC721);
    }
```

✓ The code meets the specification.

Formal Verification Request 174

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.62 ms

Line 52 in File ERC721.sol

```
    //@CTK NO_OVERFLOW
```

Line 54-57 in File ERC721.sol

```
    constructor () public {
        // register the supported interfaces to conform to ERC721 via ERC165
        _registerInterface(_INTERFACE_ID_ERC721);
    }
```

✓ The code meets the specification.

Formal Verification Request 175

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.59 ms

Line 53 in File ERC721.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 54-57 in File ERC721.sol

```
    constructor () public {
        // register the supported interfaces to conform to ERC721 via ERC165
        _registerInterface(_INTERFACE_ID_ERC721);
    }
```

✓ The code meets the specification.

Formal Verification Request 176

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 44.66 ms

Line 64 in File ERC721.sol

```
     //@CTK NO_ASF
```

Line 67-71 in File ERC721.sol

```
    function balanceOf(address owner) public view returns (uint256) {
        require(owner != address(0), "ERC721: balance query for the zero address");

        return _ownedTokensCount[owner].current();
    }
```

✓ The code meets the specification.

Formal Verification Request 177

If method completes, integer overflow would not happen.

23, Dec 2019
0.6 ms

Line 65 in File ERC721.sol

```
    //@CTK NO_OVERFLOW
```

Line 67-71 in File ERC721.sol

```
   function balanceOf(address owner) public view returns (uint256) {
      require(owner != address(0), "ERC721: balance query for the zero address");

      return _ownedTokensCount[owner].current();
   }
```

✓ The code meets the specification.

Formal Verification Request 178

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.68 ms

Line 66 in File ERC721.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 67-71 in File ERC721.sol

```
   function balanceOf(address owner) public view returns (uint256) {
      require(owner != address(0), "ERC721: balance query for the zero address");

      return _ownedTokensCount[owner].current();
   }
```

✓ The code meets the specification.

Formal Verification Request 179

Method will not encounter an assertion failure.

23, Dec 2019
21.51 ms

Line 78 in File ERC721.sol

```
    //@CTK NO_ASF
```

Line 81-86 in File ERC721.sol

```
81      function ownerOf(uint256 tokenId) public view returns (address) {
82          address owner = _tokenOwner[tokenId];
83          require(owner != address(0), "ERC721: owner query for nonexistent token");
84
85          return owner;
86      }
```

The code meets the specification.

Formal Verification Request 180

If method completes, integer overflow would not happen.

23, Dec 2019
0.48 ms

Line 79 in File ERC721.sol

```
79       //@CTK NO_OVERFLOW
```

Line 81-86 in File ERC721.sol

```
81      function ownerOf(uint256 tokenId) public view returns (address) {
82          address owner = _tokenOwner[tokenId];
83          require(owner != address(0), "ERC721: owner query for nonexistent token");
84
85          return owner;
86      }
```

The code meets the specification.

Formal Verification Request 181

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.47 ms

Line 80 in File ERC721.sol

```
80       //@CTK NO_BUF_OVERFLOW
```

Line 81-86 in File ERC721.sol

```
81      function ownerOf(uint256 tokenId) public view returns (address) {
82          address owner = _tokenOwner[tokenId];
83          require(owner != address(0), "ERC721: owner query for nonexistent token");
84
85          return owner;
86      }
```

The code meets the specification.

Formal Verification Request 182

Method will not encounter an assertion failure.

23, Dec 2019
126.24 ms

Line 96 in File ERC721.sol

```
    //@CTK NO_ASF
```

Line 99-109 in File ERC721.sol

```
    function approve(address to, uint256 tokenId) public {
        address owner = ownerOf(tokenId);
        require(to != owner, "ERC721: approval to current owner");

        require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()),
            "ERC721: approve caller is not owner nor approved for all"
        );

        _tokenApprovals[tokenId] = to;
        emit Approval(owner, to, tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 183

If method completes, integer overflow would not happen.

23, Dec 2019
3.4 ms

Line 97 in File ERC721.sol

```
    //@CTK NO_OVERFLOW
```

Line 99-109 in File ERC721.sol

```
    function approve(address to, uint256 tokenId) public {
        address owner = ownerOf(tokenId);
        require(to != owner, "ERC721: approval to current owner");

        require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()),
            "ERC721: approve caller is not owner nor approved for all"
        );

        _tokenApprovals[tokenId] = to;
        emit Approval(owner, to, tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 184

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 3.49 ms

Line 98 in File ERC721.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 99-109 in File ERC721.sol

```
    function approve(address to, uint256 tokenId) public {
        address owner = ownerOf(tokenId);
        require(to != owner, "ERC721: approval to current owner");

        require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()),
            "ERC721: approve caller is not owner nor approved for all"
        );

        _tokenApprovals[tokenId] = to;
        emit Approval(owner, to, tokenId);
    }
```

✓ The code meets the specification.

Formal Verification Request 185

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 41.18 ms

Line 117 in File ERC721.sol

```
    //@CTK NO_ASF
```

Line 120-124 in File ERC721.sol

```
    function getApproved(uint256 tokenId) public view returns (address) {
        require(_exists(tokenId), "ERC721: approved query for nonexistent token");

        return _tokenApprovals[tokenId];
    }
```

✓ The code meets the specification.

Formal Verification Request 186

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.59 ms

Line 118 in File ERC721.sol

```
    //@CTK NO_OVERFLOW
```

Line 120-124 in File ERC721.sol

```
    function getApproved(uint256 tokenId) public view returns (address) {
        require(_exists(tokenId), "ERC721: approved query for nonexistent token");

        return _tokenApprovals[tokenId];
    }
```

✓ The code meets the specification.

Formal Verification Request 187

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.53 ms

Line 119 in File ERC721.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 120-124 in File ERC721.sol

```
    function getApproved(uint256 tokenId) public view returns (address) {
        require(_exists(tokenId), "ERC721: approved query for nonexistent token");

        return _tokenApprovals[tokenId];
    }
```

✓ The code meets the specification.

Formal Verification Request 188

Method will not encounter an assertion failure.

23, Dec 2019
60.89 ms

Line 132 in File ERC721.sol

```
    //@CTK NO_ASF
```

Line 135-140 in File ERC721.sol

```
    function setApprovalForAll(address to, bool approved) public {
        require(to != _msgSender(), "ERC721: approve to caller");

        _operatorApprovals[_msgSender()][to] = approved;
        emit ApprovalForAll(_msgSender(), to, approved);
    }
```

✓ The code meets the specification.

Formal Verification Request 189

If method completes, integer overflow would not happen.

23, Dec 2019
0.85 ms

Line 133 in File ERC721.sol

```
    //@CTK NO_OVERFLOW
```

Line 135-140 in File ERC721.sol

```
    function setApprovalForAll(address to, bool approved) public {
        require(to != _msgSender(), "ERC721: approve to caller");

        _operatorApprovals[_msgSender()][to] = approved;
        emit ApprovalForAll(_msgSender(), to, approved);
    }
```

✓ The code meets the specification.

Formal Verification Request 190

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.78 ms

Line 134 in File ERC721.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 135-140 in File ERC721.sol

```
    function setApprovalForAll(address to, bool approved) public {
        require(to != _msgSender(), "ERC721: approve to caller");

        _operatorApprovals[_msgSender()][to] = approved;
        emit ApprovalForAll(_msgSender(), to, approved);
    }
```

✓ The code meets the specification.

Formal Verification Request 191

Method will not encounter an assertion failure.

23, Dec 2019
0.46 ms

Line 148 in File ERC721.sol

```
    //@CTK NO_ASF
```

Line 151-153 in File ERC721.sol

```
    function isApprovedForAll(address owner, address operator) public view returns (bool) {
        return _operatorApprovals[owner][operator];
    }
```

✓ The code meets the specification.

Formal Verification Request 192

If method completes, integer overflow would not happen.

23, Dec 2019
0.48 ms

Line 149 in File ERC721.sol

```
149     //@CTK NO_OVERFLOW
```

Line 151-153 in File ERC721.sol

```
151     function isApprovedForAll(address owner, address operator) public view returns (bool) {
152         return _operatorApprovals[owner][operator];
153     }
```

✓ The code meets the specification.

Formal Verification Request 193

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.66 ms

Line 150 in File ERC721.sol

```
150     //@CTK NO_BUF_OVERFLOW
```

Line 151-153 in File ERC721.sol

```
151     function isApprovedForAll(address owner, address operator) public view returns (bool) {
152         return _operatorApprovals[owner][operator];
153     }
```

✓ The code meets the specification.

Formal Verification Request 194

Method will not encounter an assertion failure.

23, Dec 2019
0.69 ms

Line 228 in File ERC721.sol

```
228     //@CTK NO_ASF
```

Line 231-234 in File ERC721.sol

```
    function _exists(uint256 tokenId) internal view returns (bool) {
        address owner = _tokenOwner[tokenId];
        return owner != address(0);
    }
```

✓ The code meets the specification.

Formal Verification Request 195

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.42 ms

Line 229 in File ERC721.sol

```
     //@CTK NO_OVERFLOW
```

Line 231-234 in File ERC721.sol

```
    function _exists(uint256 tokenId) internal view returns (bool) {
        address owner = _tokenOwner[tokenId];
        return owner != address(0);
    }
```

✓ The code meets the specification.

Formal Verification Request 196

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.38 ms

Line 230 in File ERC721.sol

```
     //@CTK NO_BUF_OVERFLOW
```

Line 231-234 in File ERC721.sol

```
    function _exists(uint256 tokenId) internal view returns (bool) {
        address owner = _tokenOwner[tokenId];
        return owner != address(0);
    }
```

✓ The code meets the specification.

Formal Verification Request 197

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 128.92 ms

Line 243 in File ERC721.sol

```
    //@CTK NO_ASF
```

Line 246-250 in File ERC721.sol

```
  function _isApprovedOrOwner(address spender, uint256 tokenId) internal view returns (
      bool) {
     require(_exists(tokenId), "ERC721: operator query for nonexistent token");
     address owner = ownerOf(tokenId);
     return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(
         owner, spender));
  }
```

✓ The code meets the specification.

Formal Verification Request 198

If method completes, integer overflow would not happen.

23, Dec 2019
5.64 ms

Line 244 in File ERC721.sol

```
    //@CTK NO_OVERFLOW
```

Line 246-250 in File ERC721.sol

```
  function _isApprovedOrOwner(address spender, uint256 tokenId) internal view returns (
      bool) {
     require(_exists(tokenId), "ERC721: operator query for nonexistent token");
     address owner = ownerOf(tokenId);
     return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(
         owner, spender));
  }
```

✓ The code meets the specification.

Formal Verification Request 199

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
4.6 ms

Line 245 in File ERC721.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 246-250 in File ERC721.sol

```
  function _isApprovedOrOwner(address spender, uint256 tokenId) internal view returns (
      bool) {
     require(_exists(tokenId), "ERC721: operator query for nonexistent token");
     address owner = ownerOf(tokenId);
     return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(
         owner, spender));
  }
```

✓ The code meets the specification.

Formal Verification Request 200

Method will not encounter an assertion failure.

23, Dec 2019
75.23 ms

Line 290 in File ERC721.sol

```
    //@CTK NO_ASF
```

Line 292-300 in File ERC721.sol

```
   function _mint(address to, uint256 tokenId) internal {
      require(to != address(0), "ERC721: mint to the zero address");
      require(!_exists(tokenId), "ERC721: token already minted");

      _tokenOwner[tokenId] = to;
      _ownedTokensCount[to].increment();

      emit Transfer(address(0), to, tokenId);
   }
```

✓ The code meets the specification.

Formal Verification Request 201

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
3.56 ms

Line 291 in File ERC721.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 292-300 in File ERC721.sol

```
   function _mint(address to, uint256 tokenId) internal {
      require(to != address(0), "ERC721: mint to the zero address");
      require(!_exists(tokenId), "ERC721: token already minted");

      _tokenOwner[tokenId] = to;
      _ownedTokensCount[to].increment();

      emit Transfer(address(0), to, tokenId);
   }
```

✓ The code meets the specification.

Formal Verification Request 202

Method will not encounter an assertion failure.

23, Dec 2019
11.2 ms

Line 378 in File ERC721.sol

```
    //@CTK NO_ASF
```

Line 381-385 in File ERC721.sol

```
    function _clearApproval(uint256 tokenId) private {
        if (_tokenApprovals[tokenId] != address(0)) {
            _tokenApprovals[tokenId] = address(0);
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 203

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.66 ms

Line 379 in File ERC721.sol

```
    //@CTK NO_OVERFLOW
```

Line 381-385 in File ERC721.sol

```
    function _clearApproval(uint256 tokenId) private {
        if (_tokenApprovals[tokenId] != address(0)) {
            _tokenApprovals[tokenId] = address(0);
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 204

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.52 ms

Line 380 in File ERC721.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 381-385 in File ERC721.sol

```
    function _clearApproval(uint256 tokenId) private {
        if (_tokenApprovals[tokenId] != address(0)) {
            _tokenApprovals[tokenId] = address(0);
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 205

Method will not encounter an assertion failure.

23, Dec 2019
51.21 ms

Line 21 in File ERC165.sol

```
    //@CTK NO_ASF
```

Line 24-28 in File ERC165.sol

```
    constructor () internal {
        // Derived contracts need only register support for their own interfaces,
        // we register support for ERC165 itself here
        _registerInterface(_INTERFACE_ID_ERC165);
    }
```

✓ The code meets the specification.

Formal Verification Request 206

If method completes, integer overflow would not happen.

23, Dec 2019
0.57 ms

Line 22 in File ERC165.sol

```
    //@CTK NO_OVERFLOW
```

Line 24-28 in File ERC165.sol

```
    constructor () internal {
        // Derived contracts need only register support for their own interfaces,
        // we register support for ERC165 itself here
        _registerInterface(_INTERFACE_ID_ERC165);
    }
```

✓ The code meets the specification.

Formal Verification Request 207

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.57 ms

Line 23 in File ERC165.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 24-28 in File ERC165.sol

```
    constructor () internal {
        // Derived contracts need only register support for their own interfaces,
        // we register support for ERC165 itself here
        _registerInterface(_INTERFACE_ID_ERC165);
    }
```

✓ The code meets the specification.

Formal Verification Request 208

Method will not encounter an assertion failure.

23, Dec 2019
6.0 ms

Line 35 in File ERC165.sol

```
    //@CTK NO_ASF
```

Line 38-40 in File ERC165.sol

```
    function supportsInterface(bytes4 interfaceId) external view returns (bool) {
        return _supportedInterfaces[interfaceId];
    }
```

✓ The code meets the specification.

Formal Verification Request 209

If method completes, integer overflow would not happen.

23, Dec 2019
0.44 ms

Line 36 in File ERC165.sol

```
    //@CTK NO_OVERFLOW
```

Line 38-40 in File ERC165.sol

```
    function supportsInterface(bytes4 interfaceId) external view returns (bool) {
        return _supportedInterfaces[interfaceId];
    }
```

✓ The code meets the specification.

Formal Verification Request 210

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.37 ms

Line 37 in File ERC165.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 38-40 in File ERC165.sol

```
    function supportsInterface(bytes4 interfaceId) external view returns (bool) {
        return _supportedInterfaces[interfaceId];
    }
```

✓ The code meets the specification.

Formal Verification Request 211

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.52 ms

Line 53 in File ERC165.sol

```
    //@CTK NO_ASF
```

Line 56-59 in File ERC165.sol

```
   function _registerInterface(bytes4 interfaceId) internal {
      require(interfaceId != 0xffffffff, "ERC165: invalid interface id");
      _supportedInterfaces[interfaceId] = true;
   }
```

✓ The code meets the specification.

Formal Verification Request 212

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.44 ms

Line 54 in File ERC165.sol

```
    //@CTK NO_OVERFLOW
```

Line 56-59 in File ERC165.sol

```
   function _registerInterface(bytes4 interfaceId) internal {
      require(interfaceId != 0xffffffff, "ERC165: invalid interface id");
      _supportedInterfaces[interfaceId] = true;
   }
```

✓ The code meets the specification.

Formal Verification Request 213

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.44 ms

Line 55 in File ERC165.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 56-59 in File ERC165.sol

```
   function _registerInterface(bytes4 interfaceId) internal {
      require(interfaceId != 0xffffffff, "ERC165: invalid interface id");
      _supportedInterfaces[interfaceId] = true;
   }
```

✓ The code meets the specification.

Formal Verification Request 214

Method will not encounter an assertion failure.

23, Dec 2019
3.61 ms

Line 17 in File ERC20Burnable.sol

```
    //@CTK NO_ASF
```

Line 20-24 in File ERC20Burnable.sol

```
   function burn(uint256 amount) public {
       _burn(_msgSender(), amount);
   }
```

✓ The code meets the specification.

Formal Verification Request 215

If method completes, integer overflow would not happen.

23, Dec 2019
0.36 ms

Line 18 in File ERC20Burnable.sol

```
    //@CTK NO_OVERFLOW
```

Line 20-24 in File ERC20Burnable.sol

```
   function burn(uint256 amount) public {
       _burn(_msgSender(), amount);
   }
```

✓ The code meets the specification.

Formal Verification Request 216

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.34 ms

Line 19 in File ERC20Burnable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 20-24 in File ERC20Burnable.sol

```
   function burn(uint256 amount) public {
       _burn(_msgSender(), amount);
   }
```

✓ The code meets the specification.

Formal Verification Request 217

Method will not encounter an assertion failure.

23, Dec 2019
3.47 ms

Line 29 in File ERC20Burnable.sol

```
    //@CTK NO_ASF
```

Line 32-36 in File ERC20Burnable.sol

```
    function burnFrom(address account, uint256 amount) public {
        _burnFrom(account, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 218

If method completes, integer overflow would not happen.

23, Dec 2019
0.36 ms

Line 30 in File ERC20Burnable.sol

```
    //@CTK NO_OVERFLOW
```

Line 32-36 in File ERC20Burnable.sol

```
    function burnFrom(address account, uint256 amount) public {
        _burnFrom(account, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 219

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.37 ms

Line 31 in File ERC20Burnable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 32-36 in File ERC20Burnable.sol

```
    function burnFrom(address account, uint256 amount) public {
        _burnFrom(account, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 220

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 3.5 ms

Line 16 in File Context.sol

```
    //@CTK NO_ASF
```

Line 19 in File Context.sol

```
    constructor () internal { }
```

✓ The code meets the specification.

Formal Verification Request 221

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.35 ms

Line 17 in File Context.sol

```
    //@CTK NO_OVERFLOW
```

Line 19 in File Context.sol

```
    constructor () internal { }
```

✓ The code meets the specification.

Formal Verification Request 222

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.33 ms

Line 18 in File Context.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 19 in File Context.sol

```
    constructor () internal { }
```

✓ The code meets the specification.

Formal Verification Request 223

Method will not encounter an assertion failure.

23, Dec 2019
5.28 ms

Line 21 in File Context.sol

```
    //@CTK NO_ASF
```

Line 24-26 in File Context.sol

```
    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }
```

✓ The code meets the specification.

Formal Verification Request 224

If method completes, integer overflow would not happen.

23, Dec 2019
0.39 ms

Line 22 in File Context.sol

```
    //@CTK NO_OVERFLOW
```

Line 24-26 in File Context.sol

```
    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }
```

✓ The code meets the specification.

Formal Verification Request 225

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.46 ms

Line 23 in File Context.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 24-26 in File Context.sol

```
    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }
```

✓ The code meets the specification.

Formal Verification Request 226

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 4.72 ms

Line 25 in File Counters.sol

```
    //@CTK NO_ASF
```

Line 28-30 in File Counters.sol

```
    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }
```

✓ The code meets the specification.

Formal Verification Request 227

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.38 ms

Line 26 in File Counters.sol

```
    //@CTK NO_OVERFLOW
```

Line 28-30 in File Counters.sol

```
    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }
```

✓ The code meets the specification.

Formal Verification Request 228

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 0.39 ms

Line 27 in File Counters.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 28-30 in File Counters.sol

```
    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }
```

✓ The code meets the specification.

Formal Verification Request 229

Method will not encounter an assertion failure.

23, Dec 2019
5.44 ms

Line 31 in File Counters.sol

```
//@CTK NO_ASF
```

Line 33-36 in File Counters.sol

```
function increment(Counter storage counter) internal {
    // The {SafeMath} overflow check can be skipped here, see the comment at the top
    counter._value += 1;
}
```

✓ The code meets the specification.

Formal Verification Request 230

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
0.38 ms

Line 32 in File Counters.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 33-36 in File Counters.sol

```
function increment(Counter storage counter) internal {
    // The {SafeMath} overflow check can be skipped here, see the comment at the top
    counter._value += 1;
}
```

✓ The code meets the specification.

Formal Verification Request 231

Method will not encounter an assertion failure.

04, Dec 2019
5.3 ms

Line 5 in File Ownable.sol

```
//@CTK NO_ASF
```

Line 8-10 in File Ownable.sol

```
constructor() public {
    owner = msg.sender;
}
```

✓ The code meets the specification.

Formal Verification Request 232

If method completes, integer overflow would not happen.

📅 04, Dec 2019
⏱ 0.41 ms

Line 6 in File Ownable.sol

```
    //@CTK NO_OVERFLOW
```

Line 8-10 in File Ownable.sol

```
    constructor() public {
        owner = msg.sender;
    }
```

✓ The code meets the specification.

Formal Verification Request 233

Buffer overflow / array index out of bound would never happen.

📅 04, Dec 2019
⏱ 0.36 ms

Line 7 in File Ownable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 8-10 in File Ownable.sol

```
    constructor() public {
        owner = msg.sender;
    }
```

✓ The code meets the specification.

Formal Verification Request 234

Method will not encounter an assertion failure.

📅 04, Dec 2019
⏱ 17.54 ms

Line 16 in File Ownable.sol

```
    //@CTK NO_ASF
```

Line 19-22 in File Ownable.sol

```
    function transferOwnership(address newOwner) public onlyOwner {
        if (newOwner != address(0))
            owner = newOwner;
    }
```

✓ The code meets the specification.

Formal Verification Request 235

If method completes, integer overflow would not happen.

📅 04, Dec 2019
⏱ 0.47 ms

Line 17 in File Ownable.sol

```
    //@CTK NO_OVERFLOW
```

Line 19-22 in File Ownable.sol

```
    function transferOwnership(address newOwner) public onlyOwner {
        if (newOwner != address(0))
            owner = newOwner;
    }
```

✓ The code meets the specification.

Formal Verification Request 236

Buffer overflow / array index out of bound would never happen.

📅 04, Dec 2019
⏱ 0.48 ms

Line 18 in File Ownable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 19-22 in File Ownable.sol

```
    function transferOwnership(address newOwner) public onlyOwner {
        if (newOwner != address(0))
            owner = newOwner;
    }
```

✓ The code meets the specification.

Formal Verification Request 237

isContract

📅 04, Dec 2019
⏱ 6.21 ms

Line 18-20 in File Address.sol

```
    /*@CTK isContract
      @post !__reverted -> __return == (account != msg.sender)
    */
```

Line 24-40 in File Address.sol

```
    function isContract(address account) internal view returns (bool) {
        return (account != msg.sender);
        /*
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
            returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0
            xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
        */
    }
```

✓ The code meets the specification.

Formal Verification Request 238

Buffer overflow / array index out of bound would never happen.

04, Dec 2019
0.4 ms

Line 21 in File Address.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 24-40 in File Address.sol

```
    function isContract(address account) internal view returns (bool) {
        return (account != msg.sender);
        /*
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
            returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0
            xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
        */
    }
```

✓ The code meets the specification.

Formal Verification Request 239

If method completes, integer overflow would not happen.

📅 04, Dec 2019
⏱ 0.4 ms

Line 22 in File Address.sol

```
    //@CTK NO_OVERFLOW
```

Line 24-40 in File Address.sol

```
    function isContract(address account) internal view returns (bool) {
        return (account != msg.sender);
        /*
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
            returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0
            xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
        */
    }
```

✅ The code meets the specification.

Formal Verification Request 240

Method will not encounter an assertion failure.

📅 04, Dec 2019
⏱ 0.38 ms

Line 23 in File Address.sol

```
    //@CTK NO_ASF
```

Line 24-40 in File Address.sol

```
    function isContract(address account) internal view returns (bool) {
        return (account != msg.sender);
        /*
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
            returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0
            xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
        */
    }
```

✓ The code meets the specification.

Formal Verification Request 241

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
7.45 ms

Line 13 in File BytesLib.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 16-92 in File BytesLib.sol

```
    function concat(
        bytes memory _preBytes,
        bytes memory _postBytes
    )
        internal
        pure
        returns (bytes memory)
    {
        bytes memory tempBytes;

        assembly {
            // Get a location of some free memory and store it in tempBytes as
            // Solidity does for memory variables.
            tempBytes := mload(0x40)

            // Store the length of the first bytes array at the beginning of
            // the memory for tempBytes.
            let length := mload(_preBytes)
            mstore(tempBytes, length)

            // Maintain a memory counter for the current write location in the
            // temp bytes array by adding the 32 bytes for the array length to
            // the starting location.
            let mc := add(tempBytes, 0x20)
            // Stop copying when the memory counter reaches the length of the
            // first bytes array.
            let end := add(mc, length)

            for {
                // Initialize a copy counter to the start of the _preBytes data,
                // 32 bytes into its memory.
                let cc := add(_preBytes, 0x20)
            } lt(mc, end) {
                // Increase both counters by 32 bytes each iteration.
                mc := add(mc, 0x20)
                cc := add(cc, 0x20)
            } {
                // Write the _preBytes data into the tempBytes memory 32 bytes
                // at a time.
                mstore(mc, mload(cc))
            }

            // Add the length of _postBytes to the current length of tempBytes
            // and store it as the new length in the first 32 bytes of the
            // tempBytes memory.
            length := mload(_postBytes)
            mstore(tempBytes, add(length, mload(tempBytes)))

            // Move the memory counter back from a multiple of 0x20 to the
            // actual end of the _preBytes data.
            mc := end
            // Stop copying when the memory counter reaches the new combined
            // length of the arrays.
            end := add(mc, length)

            for {
                let cc := add(_postBytes, 0x20)
            } lt(mc, end) {
                mc := add(mc, 0x20)
                cc := add(cc, 0x20)
            } {
                mstore(mc, mload(cc))
            }

            // Update the free-memory pointer by padding our last write location
            // to 32 bytes: add 31 bytes to the end of tempBytes to move to the
            // next 32 byte block, then round down to the nearest multiple of
            // 32. If the sum of the length of the two arrays is zero then add
            // one before rounding down to leave a blank 32 bytes (the length block with 0).
            mstore(0x40, and(
              add(add(end, iszero(add(length, mload(_preBytes)))), 31),
              not(31) // Round down to the nearest 32 bytes.
            ))
        }

        return tempBytes;
    }
```

✓ The code meets the specification.

Formal Verification Request 242

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.41 ms

Line 14 in File BytesLib.sol

```
14      //@CTK NO_OVERFLOW
```

Line 16-92 in File BytesLib.sol

```
    function concat(
        bytes memory _preBytes,
        bytes memory _postBytes
    )
        internal
        pure
        returns (bytes memory)
    {
        bytes memory tempBytes;

        assembly {
            // Get a location of some free memory and store it in tempBytes as
            // Solidity does for memory variables.
            tempBytes := mload(0x40)

            // Store the length of the first bytes array at the beginning of
            // the memory for tempBytes.
            let length := mload(_preBytes)
            mstore(tempBytes, length)

            // Maintain a memory counter for the current write location in the
            // temp bytes array by adding the 32 bytes for the array length to
            // the starting location.
            let mc := add(tempBytes, 0x20)
            // Stop copying when the memory counter reaches the length of the
            // first bytes array.
            let end := add(mc, length)

            for {
                // Initialize a copy counter to the start of the _preBytes data,
                // 32 bytes into its memory.
                let cc := add(_preBytes, 0x20)
            } lt(mc, end) {
                // Increase both counters by 32 bytes each iteration.
                mc := add(mc, 0x20)
                cc := add(cc, 0x20)
            } {
                // Write the _preBytes data into the tempBytes memory 32 bytes
                // at a time.
                mstore(mc, mload(cc))
            }

            // Add the length of _postBytes to the current length of tempBytes
            // and store it as the new length in the first 32 bytes of the
            // tempBytes memory.
            length := mload(_postBytes)
            mstore(tempBytes, add(length, mload(tempBytes)))

            // Move the memory counter back from a multiple of 0x20 to the
            // actual end of the _preBytes data.
            mc := end
            // Stop copying when the memory counter reaches the new combined
            // length of the arrays.
            end := add(mc, length)

            for {
                let cc := add(_postBytes, 0x20)
            } lt(mc, end) {
                mc := add(mc, 0x20)
                cc := add(cc, 0x20)
            } {
                mstore(mc, mload(cc))
            }

            // Update the free-memory pointer by padding our last write location
            // to 32 bytes: add 31 bytes to the end of tempBytes to move to the
            // next 32 byte block, then round down to the nearest multiple of
            // 32. If the sum of the length of the two arrays is zero then add
            // one before rounding down to leave a blank 32 bytes (the length block with 0).
            mstore(0x40, and(
              add(add(end, iszero(add(length, mload(_preBytes)))), 31),
              not(31) // Round down to the nearest 32 bytes.
            ))
        }

        return tempBytes;
    }
```

✓ The code meets the specification.

Formal Verification Request 243

Method will not encounter an assertion failure.

23, Dec 2019
0.39 ms

Line 15 in File BytesLib.sol

```
    //@CTK NO_ASF
```

Line 16-92 in File BytesLib.sol

```
    function concat(
        bytes memory _preBytes,
        bytes memory _postBytes
    )
        internal
        pure
        returns (bytes memory)
    {
        bytes memory tempBytes;

        assembly {
            // Get a location of some free memory and store it in tempBytes as
            // Solidity does for memory variables.
            tempBytes := mload(0x40)

            // Store the length of the first bytes array at the beginning of
            // the memory for tempBytes.
            let length := mload(_preBytes)
            mstore(tempBytes, length)

            // Maintain a memory counter for the current write location in the
            // temp bytes array by adding the 32 bytes for the array length to
            // the starting location.
            let mc := add(tempBytes, 0x20)
            // Stop copying when the memory counter reaches the length of the
            // first bytes array.
            let end := add(mc, length)

            for {
                // Initialize a copy counter to the start of the _preBytes data,
                // 32 bytes into its memory.
                let cc := add(_preBytes, 0x20)
            } lt(mc, end) {
                // Increase both counters by 32 bytes each iteration.
                mc := add(mc, 0x20)
                cc := add(cc, 0x20)
            } {
                // Write the _preBytes data into the tempBytes memory 32 bytes
                // at a time.
                mstore(mc, mload(cc))
            }

            // Add the length of _postBytes to the current length of tempBytes
            // and store it as the new length in the first 32 bytes of the
            // tempBytes memory.
            length := mload(_postBytes)
            mstore(tempBytes, add(length, mload(tempBytes)))

            // Move the memory counter back from a multiple of 0x20 to the
            // actual end of the _preBytes data.
            mc := end
            // Stop copying when the memory counter reaches the new combined
            // length of the arrays.
            end := add(mc, length)

            for {
                let cc := add(_postBytes, 0x20)
            } lt(mc, end) {
                mc := add(mc, 0x20)
                cc := add(cc, 0x20)
            } {
                mstore(mc, mload(cc))
            }

            // Update the free-memory pointer by padding our last write location
            // to 32 bytes: add 31 bytes to the end of tempBytes to move to the
            // next 32 byte block, then round down to the nearest multiple of
            // 32. If the sum of the length of the two arrays is zero then add
            // one before rounding down to leave a blank 32 bytes (the length block with 0).
            mstore(0x40, and(
              add(add(end, iszero(add(length, mload(_preBytes)))), 31),
              not(31) // Round down to the nearest 32 bytes.
            ))
        }

        return tempBytes;
    }
```

✓ The code meets the specification.

Formal Verification Request 244

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 4.0 ms

Line 93 in File BytesLib.sol

```
93      //@CTK NO_BUF_OVERFLOW
```

Line 96-231 in File BytesLib.sol

```
96      function concatStorage(bytes storage _preBytes, bytes memory _postBytes) internal {
97          assembly {
98              // Read the first 32 bytes of _preBytes storage, which is the length
99              // of the array. (We don't need to use the offset into the slot
100             // because arrays use the entire slot.)
101             let fslot := sload(_preBytes_slot)
102             // Arrays of 31 bytes or less have an even value in their slot,
103             // while longer arrays have an odd value. The actual length is
104             // the slot divided by two for odd values, and the lowest order
105             // byte divided by two for even values.
106             // If the slot is even, bitwise and the slot with 255 and divide by
107             // two to get the length. If the slot is odd, bitwise and the slot
108             // with -1 and divide by two.
109             let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
110             let mlength := mload(_postBytes)
111             let newlength := add(slength, mlength)
112             // slength can contain both the length and contents of the array
113             // if length < 32 bytes so let's prepare for that
114             // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-state-
                    variables-in-storage
115             switch add(lt(slength, 32), lt(newlength, 32))
116             case 2 {
117                 // Since the new array still fits in the slot, we just need to
118                 // update the contents of the slot.
119                 // uint256(bytes_storage) = uint256(bytes_storage) + uint256(bytes_memory) +
                        new_length
120                 sstore(
121                     _preBytes_slot,
122                     // all the modifications to the slot are inside this
123                     // next block
124                     add(
125                         // we can just add to the slot contents because the
126                         // bytes we want to change are the LSBs
127                         fslot,
128                         add(
129                             mul(
130                                 div(
131                                     // load the bytes from memory
132                                     mload(add(_postBytes, 0x20)),
133                                     // zero all bytes to the right
134                                     exp(0x100, sub(32, mlength))
135                                 ),
136                                 // and now shift left the number of bytes to
137                                 // leave space for the length in the slot
138                                 exp(0x100, sub(32, newlength))
139                             ),
                            // increase length by the double of the memory
                            // bytes length
                            mul(mlength, 2)
                        )
                    )
                )
            }
            case 1 {
                // The stored value fits in the slot, but the combined value
                // will exceed it.
                // get the keccak hash to get the contents of the array
                mstore(0x0, _preBytes_slot)
                let sc := add(keccak256(0x0, 0x20), div(slength, 32))

                // save new length
                sstore(_preBytes_slot, add(mul(newlength, 2), 1))

                // The contents of the _postBytes array start 32 bytes into
                // the structure. Our first read should obtain the `submod`
                // bytes that can fit into the unused space in the last word
                // of the stored array. To get this, we read 32 bytes starting
                // from `submod`, so the data we read overlaps with the array
                // contents by `submod` bytes. Masking the lowest-order
                // `submod` bytes allows us to add that value directly to the
                // stored value.

                let submod := sub(32, slength)
                let mc := add(_postBytes, submod)
                let end := add(_postBytes, mlength)
                let mask := sub(exp(0x100, submod), 1)

                sstore(
                    sc,
                    add(
                        and(
                            fslot,
                            0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff00
                        ),
                        and(mload(mc), mask)
                    )
                )

                for {
                    mc := add(mc, 0x20)
                    sc := add(sc, 1)
                } lt(mc, end) {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } {
                    sstore(sc, mload(mc))
                }

                mask := exp(0x100, sub(mc, end))

                sstore(sc, mul(div(mload(mc), mask), mask))
            }
            default {
                // get the keccak hash to get the contents of the array
                mstore(0x0, _preBytes_slot)
                // Start copying to the last used word of the stored array.
                let sc := add(keccak256(0x0, 0x20), div(slength, 32))

                // save new length
                sstore(_preBytes_slot, add(mul(newlength, 2), 1))

                // Copy over the first `submod` bytes of the new data as in
                // case 1 above.
                let slengthmod := mod(slength, 32)
                let mlengthmod := mod(mlength, 32)
                let submod := sub(32, slengthmod)
                let mc := add(_postBytes, submod)
                let end := add(_postBytes, mlength)
                let mask := sub(exp(0x100, submod), 1)

                sstore(sc, add(sload(sc), and(mload(mc), mask)))

                for {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } lt(mc, end) {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } {
                    sstore(sc, mload(mc))
                }

                mask := exp(0x100, sub(mc, end))

                sstore(sc, mul(div(mload(mc), mask), mask))
            }
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 245

If method completes, integer overflow would not happen.

23, Dec 2019
0.37 ms

Line 94 in File BytesLib.sol

```
    //@CTK NO_OVERFLOW
```

Line 96-231 in File BytesLib.sol

```
    function concatStorage(bytes storage _preBytes, bytes memory _postBytes) internal {
        assembly {
            // Read the first 32 bytes of _preBytes storage, which is the length
            // of the array. (We don't need to use the offset into the slot
            // because arrays use the entire slot.)
            let fslot := sload(_preBytes_slot)
            // Arrays of 31 bytes or less have an even value in their slot,
            // while longer arrays have an odd value. The actual length is
            // the slot divided by two for odd values, and the lowest order
            // byte divided by two for even values.
            // If the slot is even, bitwise and the slot with 255 and divide by
            // two to get the length. If the slot is odd, bitwise and the slot
            // with -1 and divide by two.
            let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
            let mlength := mload(_postBytes)
            let newlength := add(slength, mlength)
            // slength can contain both the length and contents of the array
            // if length < 32 bytes so let's prepare for that
            // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-state-
                variables-in-storage
            switch add(lt(slength, 32), lt(newlength, 32))
            case 2 {
                // Since the new array still fits in the slot, we just need to
                // update the contents of the slot.
                // uint256(bytes_storage) = uint256(bytes_storage) + uint256(bytes_memory) +
                    new_length
                sstore(
                    _preBytes_slot,
                    // all the modifications to the slot are inside this
                    // next block
                    add(
                        // we can just add to the slot contents because the
                        // bytes we want to change are the LSBs
                        fslot,
                        add(
                            mul(
                                div(
                                    // load the bytes from memory
                                    mload(add(_postBytes, 0x20)),
                                    // zero all bytes to the right
                                    exp(0x100, sub(32, mlength))
                                ),
                                // and now shift left the number of bytes to
                                // leave space for the length in the slot
                                exp(0x100, sub(32, newlength))
                            ),
                            // increase length by the double of the memory
                            // bytes length
                            mul(mlength, 2)
                        )
                    )
                )
            }
            case 1 {
                // The stored value fits in the slot, but the combined value
                // will exceed it.
                // get the keccak hash to get the contents of the array
                mstore(0x0, _preBytes_slot)
                let sc := add(keccak256(0x0, 0x20), div(slength, 32))

                // save new length
                sstore(_preBytes_slot, add(mul(newlength, 2), 1))

                // The contents of the _postBytes array start 32 bytes into
                // the structure. Our first read should obtain the `submod`
                // bytes that can fit into the unused space in the last word
                // of the stored array. To get this, we read 32 bytes starting
                // from `submod`, so the data we read overlaps with the array
                // contents by `submod` bytes. Masking the lowest-order
                // `submod` bytes allows us to add that value directly to the
                // stored value.

                let submod := sub(32, slength)
                let mc := add(_postBytes, submod)
                let end := add(_postBytes, mlength)
                let mask := sub(exp(0x100, submod), 1)

                sstore(
                    sc,
                    add(
                        and(
                            fslot,
                            0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff00
                        ),
                        and(mload(mc), mask)
                    )
                )

                for {
                    mc := add(mc, 0x20)
                    sc := add(sc, 1)
                } lt(mc, end) {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } {
                    sstore(sc, mload(mc))
                }

                mask := exp(0x100, sub(mc, end))

                sstore(sc, mul(div(mload(mc), mask), mask))
            }
            default {
                // get the keccak hash to get the contents of the array
                mstore(0x0, _preBytes_slot)
                // Start copying to the last used word of the stored array.
                let sc := add(keccak256(0x0, 0x20), div(slength, 32))

                // save new length
                sstore(_preBytes_slot, add(mul(newlength, 2), 1))

                // Copy over the first `submod` bytes of the new data as in
                // case 1 above.
                let slengthmod := mod(slength, 32)
                let mlengthmod := mod(mlength, 32)
                let submod := sub(32, slengthmod)
                let mc := add(_postBytes, submod)
                let end := add(_postBytes, mlength)
                let mask := sub(exp(0x100, submod), 1)

                sstore(sc, add(sload(sc), and(mload(mc), mask)))

                for {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } lt(mc, end) {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } {
                    sstore(sc, mload(mc))
                }

                mask := exp(0x100, sub(mc, end))

                sstore(sc, mul(div(mload(mc), mask), mask))
            }
         }
     }
```

✓ The code meets the specification.

Formal Verification Request 246

Method will not encounter an assertion failure.

23, Dec 2019
0.35 ms

Line 95 in File BytesLib.sol

```
//@CTK NO_ASF
```

Line 96-231 in File BytesLib.sol

```
function concatStorage(bytes storage _preBytes, bytes memory _postBytes) internal {
    assembly {
        // Read the first 32 bytes of _preBytes storage, which is the length
        // of the array. (We don't need to use the offset into the slot
        // because arrays use the entire slot.)
        let fslot := sload(_preBytes_slot)
        // Arrays of 31 bytes or less have an even value in their slot,
        // while longer arrays have an odd value. The actual length is
        // the slot divided by two for odd values, and the lowest order
        // byte divided by two for even values.
        // If the slot is even, bitwise and the slot with 255 and divide by
        // two to get the length. If the slot is odd, bitwise and the slot
        // with -1 and divide by two.
        let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
        let mlength := mload(_postBytes)
        let newlength := add(slength, mlength)
        // slength can contain both the length and contents of the array
        // if length < 32 bytes so let's prepare for that
        // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-state-
            variables-in-storage
        switch add(lt(slength, 32), lt(newlength, 32))
        case 2 {
            // Since the new array still fits in the slot, we just need to
            // update the contents of the slot.
            // uint256(bytes_storage) = uint256(bytes_storage) + uint256(bytes_memory) +
                new_length
            sstore(
                _preBytes_slot,
                    // all the modifications to the slot are inside this
                    // next block
                    add(
                        // we can just add to the slot contents because the
                        // bytes we want to change are the LSBs
                        fslot,
                        add(
                            mul(
                                div(
                                    // load the bytes from memory
                                    mload(add(_postBytes, 0x20)),
                                    // zero all bytes to the right
                                    exp(0x100, sub(32, mlength))
                                ),
                                // and now shift left the number of bytes to
                                // leave space for the length in the slot
                                exp(0x100, sub(32, newlength))
                            ),
                            // increase length by the double of the memory
                            // bytes length
                            mul(mlength, 2)
                        )
                    )
                )
            }
            case 1 {
                // The stored value fits in the slot, but the combined value
                // will exceed it.
                // get the keccak hash to get the contents of the array
                mstore(0x0, _preBytes_slot)
                let sc := add(keccak256(0x0, 0x20), div(slength, 32))

                // save new length
                sstore(_preBytes_slot, add(mul(newlength, 2), 1))

                // The contents of the _postBytes array start 32 bytes into
                // the structure. Our first read should obtain the `submod`
                // bytes that can fit into the unused space in the last word
                // of the stored array. To get this, we read 32 bytes starting
                // from `submod`, so the data we read overlaps with the array
                // contents by `submod` bytes. Masking the lowest-order
                // `submod` bytes allows us to add that value directly to the
                // stored value.

                let submod := sub(32, slength)
                let mc := add(_postBytes, submod)
                let end := add(_postBytes, mlength)
                let mask := sub(exp(0x100, submod), 1)

                sstore(
                    sc,
                    add(
                        and(
                            fslot,
                            0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff00
                        ),
                        and(mload(mc), mask)
                    )
                )

                for {
                    mc := add(mc, 0x20)
                    sc := add(sc, 1)
                } lt(mc, end) {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } {
                    sstore(sc, mload(mc))
                }

                mask := exp(0x100, sub(mc, end))

                sstore(sc, mul(div(mload(mc), mask), mask))
            }
            default {
                // get the keccak hash to get the contents of the array
                mstore(0x0, _preBytes_slot)
                // Start copying to the last used word of the stored array.
                let sc := add(keccak256(0x0, 0x20), div(slength, 32))

                // save new length
                sstore(_preBytes_slot, add(mul(newlength, 2), 1))

                // Copy over the first `submod` bytes of the new data as in
                // case 1 above.
                let slengthmod := mod(slength, 32)
                let mlengthmod := mod(mlength, 32)
                let submod := sub(32, slengthmod)
                let mc := add(_postBytes, submod)
                let end := add(_postBytes, mlength)
                let mask := sub(exp(0x100, submod), 1)

                sstore(sc, add(sload(sc), and(mload(mc), mask)))

                for {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } lt(mc, end) {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } {
                    sstore(sc, mload(mc))
                }

                mask := exp(0x100, sub(mc, end))

                sstore(sc, mul(div(mload(mc), mask), mask))
            }
        }
    }
```

✓ The code meets the specification.

Formal Verification Request 247

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
20.97 ms

Line 232 in File BytesLib.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 234-297 in File BytesLib.sol

```
function slice(
    bytes memory _bytes,
    uint _start,
    uint _length
)
    internal
    pure
    returns (bytes memory)
{
    require(_bytes.length >= (_start + _length), "_bytes.length >= (_start + _length)");

    bytes memory tempBytes;

    assembly {
        switch iszero(_length)
        case 0 {
            // Get a location of some free memory and store it in tempBytes as
            // Solidity does for memory variables.
            tempBytes := mload(0x40)

            // The first word of the slice result is potentially a partial
            // word read from the original array. To read it, we calculate
            // the length of that partial word and start copying that many
            // bytes into the array. The first word we copy will start with
            // data we don't care about, but the last `lengthmod` bytes will
            // land at the beginning of the contents of the new array. When
            // we're done copying, we overwrite the full first word with
            // the actual length of the slice.
            let lengthmod := and(_length, 31)

            // The multiplication in the next line is necessary
            // because when slicing multiples of 32 bytes (lengthmod == 0)
            // the following copy loop was copying the origin's length
            // and then ending prematurely not copying everything it should.
            let mc := add(add(tempBytes, lengthmod), mul(0x20, iszero(lengthmod)))
            let end := add(mc, _length)

            for {
                // The multiplication in the next line has the same exact purpose
                // as the one above.
                let cc := add(add(add(_bytes, lengthmod), mul(0x20, iszero(lengthmod))),
                    _start)
            } lt(mc, end) {
                mc := add(mc, 0x20)
                cc := add(cc, 0x20)
            } {
                mstore(mc, mload(cc))
            }

            mstore(tempBytes, _length)

            //update free-memory pointer
            //allocating the array padded to 32 bytes like the compiler does now
            mstore(0x40, and(add(mc, 31), not(31)))
        }
        //if we want a zero-length slice let's just return a zero-length array
        default {
            tempBytes := mload(0x40)

            mstore(0x40, add(tempBytes, 0x20))
        }
    }

    return tempBytes;
}
```

✓ The code meets the specification.

Formal Verification Request 248

Method will not encounter an assertion failure.

23, Dec 2019
0.74 ms

Line 233 in File BytesLib.sol

```
//@CTK NO_ASF
```

Line 234-297 in File BytesLib.sol

```
function slice(
    bytes memory _bytes,
    uint _start,
    uint _length
)
    internal
    pure
    returns (bytes memory)
{
    require(_bytes.length >= (_start + _length), "_bytes.length >= (_start + _length)");

    bytes memory tempBytes;

    assembly {
        switch iszero(_length)
        case 0 {
            // Get a location of some free memory and store it in tempBytes as
            // Solidity does for memory variables.
            tempBytes := mload(0x40)

            // The first word of the slice result is potentially a partial
            // word read from the original array. To read it, we calculate
            // the length of that partial word and start copying that many
```

```
                // bytes into the array. The first word we copy will start with
                // data we don't care about, but the last `lengthmod` bytes will
                // land at the beginning of the contents of the new array. When
                // we're done copying, we overwrite the full first word with
                // the actual length of the slice.
                let lengthmod := and(_length, 31)

                // The multiplication in the next line is necessary
                // because when slicing multiples of 32 bytes (lengthmod == 0)
                // the following copy loop was copying the origin's length
                // and then ending prematurely not copying everything it should.
                let mc := add(add(tempBytes, lengthmod), mul(0x20, iszero(lengthmod)))
                let end := add(mc, _length)

                for {
                    // The multiplication in the next line has the same exact purpose
                    // as the one above.
                    let cc := add(add(add(_bytes, lengthmod), mul(0x20, iszero(lengthmod))),
                        _start)
                } lt(mc, end) {
                    mc := add(mc, 0x20)
                    cc := add(cc, 0x20)
                } {
                    mstore(mc, mload(cc))
                }

                mstore(tempBytes, _length)

                //update free-memory pointer
                //allocating the array padded to 32 bytes like the compiler does now
                mstore(0x40, and(add(mc, 31), not(31)))
            }
            //if we want a zero-length slice let's just return a zero-length array
            default {
                tempBytes := mload(0x40)

                mstore(0x40, add(tempBytes, 0x20))
            }
        }

        return tempBytes;
    }
```

✓ The code meets the specification.

Formal Verification Request 249

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
20.46 ms

Line 298 in File BytesLib.sol

```
298     //@CTK NO_BUF_OVERFLOW
```

Line 300-309 in File BytesLib.sol

```
    function toAddress(bytes memory _bytes, uint _start) internal pure returns (address) {
        require(_bytes.length >= (_start + 20), "_bytes.length >= (_start + 20)");
        address tempAddress;

        assembly {
            tempAddress := div(mload(add(add(_bytes, 0x20), _start)), 0
                x1000000000000000000000000)
        }

        return tempAddress;
    }
```

✓ The code meets the specification.

Formal Verification Request 250

Method will not encounter an assertion failure.

23, Dec 2019
0.59 ms

Line 299 in File BytesLib.sol

```
    //@CTK NO_ASF
```

Line 300-309 in File BytesLib.sol

```
    function toAddress(bytes memory _bytes, uint _start) internal pure returns (address) {
        require(_bytes.length >= (_start + 20), "_bytes.length >= (_start + 20)");
        address tempAddress;

        assembly {
            tempAddress := div(mload(add(add(_bytes, 0x20), _start)), 0
                x1000000000000000000000000)
        }

        return tempAddress;
    }
```

✓ The code meets the specification.

Formal Verification Request 251

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
21.18 ms

Line 310 in File BytesLib.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 312-321 in File BytesLib.sol

```
    function toUint8(bytes memory _bytes, uint _start) internal pure returns (uint8) {
        require(_bytes.length >= (_start + 1), "_bytes.length >= (_start + 1)");
        uint8 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x1), _start))
        }

        return tempUint;
    }
```

✓ The code meets the specification.

Formal Verification Request 252

Method will not encounter an assertion failure.

23, Dec 2019
0.62 ms

Line 311 in File BytesLib.sol

```
    //@CTK NO_ASF
```

Line 312-321 in File BytesLib.sol

```
    function toUint8(bytes memory _bytes, uint _start) internal pure returns (uint8) {
        require(_bytes.length >= (_start + 1), "_bytes.length >= (_start + 1)");
        uint8 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x1), _start))
        }

        return tempUint;
    }
```

✓ The code meets the specification.

Formal Verification Request 253

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
21.51 ms

Line 322 in File BytesLib.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 324-333 in File BytesLib.sol

```
    function toUint16(bytes memory _bytes, uint _start) internal pure returns (uint16) {
        require(_bytes.length >= (_start + 2), "_bytes.length >= (_start + 2)");
        uint16 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x2), _start))
        }

        return tempUint;
    }
```

✓ The code meets the specification.

Formal Verification Request 254

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.53 ms

Line 323 in File BytesLib.sol

```
    //@CTK NO_ASF
```

Line 324-333 in File BytesLib.sol

```
    function toUint16(bytes memory _bytes, uint _start) internal pure returns (uint16) {
        require(_bytes.length >= (_start + 2), "_bytes.length >= (_start + 2)");
        uint16 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x2), _start))
        }

        return tempUint;
    }
```

✓ The code meets the specification.

Formal Verification Request 255

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 20.84 ms

Line 334 in File BytesLib.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 336-345 in File BytesLib.sol

```
    function toUint32(bytes memory _bytes, uint _start) internal pure returns (uint32) {
        require(_bytes.length >= (_start + 4), "_bytes.length >= (_start + 4)");
        uint32 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x4), _start))
        }

        return tempUint;
    }
```

✓ The code meets the specification.

Formal Verification Request 256

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.79 ms

Line 335 in File BytesLib.sol

```
//@CTK NO_ASF
```

Line 336-345 in File BytesLib.sol

```
function toUint32(bytes memory _bytes, uint _start) internal pure returns (uint32) {
    require(_bytes.length >= (_start + 4), "_bytes.length >= (_start + 4)");
    uint32 tempUint;

    assembly {
        tempUint := mload(add(add(_bytes, 0x4), _start))
    }

    return tempUint;
}
```

✓ The code meets the specification.

Formal Verification Request 257

Buffer overflow / array index out of bound would never happen.

📅 23, Dec 2019
⏱ 19.03 ms

Line 346 in File BytesLib.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 348-357 in File BytesLib.sol

```
function toUint64(bytes memory _bytes, uint _start) internal pure returns (uint64) {
    require(_bytes.length >= (_start + 8), "_bytes.length >= (_start + 8)");
    uint64 tempUint;

    assembly {
        tempUint := mload(add(add(_bytes, 0x8), _start))
    }

    return tempUint;
}
```

✓ The code meets the specification.

Formal Verification Request 258

Method will not encounter an assertion failure.

📅 23, Dec 2019
⏱ 0.55 ms

Line 347 in File BytesLib.sol

```
    //@CTK NO_ASF
```

Line 348-357 in File BytesLib.sol

```
    function toUint64(bytes memory _bytes, uint _start) internal pure returns (uint64) {
        require(_bytes.length >= (_start + 8), "_bytes.length >= (_start + 8)");
        uint64 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x8), _start))
        }

        return tempUint;
    }
```

✓ The code meets the specification.

Formal Verification Request 259

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
23.43 ms

Line 358 in File BytesLib.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 360-369 in File BytesLib.sol

```
    function toUint96(bytes memory _bytes, uint _start) internal pure returns (uint96) {
        require(_bytes.length >= (_start + 12), "_bytes.length >= (_start + 12)");
        uint96 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0xc), _start))
        }

        return tempUint;
    }
```

✓ The code meets the specification.

Formal Verification Request 260

Method will not encounter an assertion failure.

23, Dec 2019
0.53 ms

Line 359 in File BytesLib.sol

```
    //@CTK NO_ASF
```

Line 360-369 in File BytesLib.sol

```
function toUint96(bytes memory _bytes, uint _start) internal pure returns (uint96) {
    require(_bytes.length >= (_start + 12), "_bytes.length >= (_start + 12)");
    uint96 tempUint;

    assembly {
        tempUint := mload(add(add(_bytes, 0xc), _start))
    }

    return tempUint;
}
```

✓ The code meets the specification.

Formal Verification Request 261

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
21.54 ms

Line 370 in File BytesLib.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 372-381 in File BytesLib.sol

```
function toUint128(bytes memory _bytes, uint _start) internal pure returns (uint128) {
    require(_bytes.length >= (_start + 16), "_bytes.length >= (_start + 16)");
    uint128 tempUint;

    assembly {
        tempUint := mload(add(add(_bytes, 0x10), _start))
    }

    return tempUint;
}
```

✓ The code meets the specification.

Formal Verification Request 262

Method will not encounter an assertion failure.

23, Dec 2019
0.54 ms

Line 371 in File BytesLib.sol

```
//@CTK NO_ASF
```

Line 372-381 in File BytesLib.sol

```
function toUint128(bytes memory _bytes, uint _start) internal pure returns (uint128) {
    require(_bytes.length >= (_start + 16), "_bytes.length >= (_start + 16)");
    uint128 tempUint;

    assembly {
        tempUint := mload(add(add(_bytes, 0x10), _start))
    }

    return tempUint;
}
```

✓ The code meets the specification.

Formal Verification Request 263

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
22.57 ms

Line 382 in File BytesLib.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 384-393 in File BytesLib.sol

```
    function toUint(bytes memory _bytes, uint _start) internal pure returns (uint256) {
        require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
        uint256 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x20), _start))
        }

        return tempUint;
    }
```

✓ The code meets the specification.

Formal Verification Request 264

Method will not encounter an assertion failure.

23, Dec 2019
0.7 ms

Line 383 in File BytesLib.sol

```
    //@CTK NO_ASF
```

Line 384-393 in File BytesLib.sol

```
    function toUint(bytes memory _bytes, uint _start) internal pure returns (uint256) {
        require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
        uint256 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x20), _start))
        }

        return tempUint;
    }
```

✓ The code meets the specification.

Formal Verification Request 265

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
22.1 ms

Line 394 in File BytesLib.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 396-405 in File BytesLib.sol

```
    function toBytes32(bytes memory _bytes, uint _start) internal pure returns (bytes32) {
        require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
        bytes32 tempBytes32;

        assembly {
            tempBytes32 := mload(add(add(_bytes, 0x20), _start))
        }

        return tempBytes32;
    }
```

✓ The code meets the specification.

Formal Verification Request 266

Method will not encounter an assertion failure.

23, Dec 2019
0.59 ms

Line 395 in File BytesLib.sol

```
    //@CTK NO_ASF
```

Line 396-405 in File BytesLib.sol

```
    function toBytes32(bytes memory _bytes, uint _start) internal pure returns (bytes32) {
        require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
        bytes32 tempBytes32;

        assembly {
            tempBytes32 := mload(add(add(_bytes, 0x20), _start))
        }

        return tempBytes32;
    }
```

✓ The code meets the specification.

Formal Verification Request 267

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
7.55 ms

Line 406 in File BytesLib.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 409-450 in File BytesLib.sol

```
function equal(bytes memory _preBytes, bytes memory _postBytes) internal pure returns (
    bool) {
    bool success = true;

    assembly {
        let length := mload(_preBytes)

        // if lengths don't match the arrays are not equal
        switch eq(length, mload(_postBytes))
        case 1 {
            // cb is a circuit breaker in the for loop since there's
            //  no said feature for inline assembly loops
            // cb = 1 - don't breaker
            // cb = 0 - break
            let cb := 1

            let mc := add(_preBytes, 0x20)
            let end := add(mc, length)

            for {
                let cc := add(_postBytes, 0x20)
            // the next line is the loop condition:
            // while(uint(mc < end) + cb == 2)
            } eq(add(lt(mc, end), cb), 2) {
                mc := add(mc, 0x20)
                cc := add(cc, 0x20)
            } {
                // if any of these checks fails then arrays are not equal
                if iszero(eq(mload(mc), mload(cc))) {
                    // unsuccess:
                    success := 0
                    cb := 0
                }
            }
        }
        default {
            // unsuccess:
            success := 0
        }
    }

    return success;
}
```

The code meets the specification.

Formal Verification Request 268

If method completes, integer overflow would not happen.

23, Dec 2019
0.43 ms

Line 407 in File BytesLib.sol

```
    //@CTK NO_OVERFLOW
```

Line 409-450 in File BytesLib.sol

```
    function equal(bytes memory _preBytes, bytes memory _postBytes) internal pure returns (
        bool) {
        bool success = true;

        assembly {
            let length := mload(_preBytes)

            // if lengths don't match the arrays are not equal
            switch eq(length, mload(_postBytes))
            case 1 {
                // cb is a circuit breaker in the for loop since there's
                //  no said feature for inline assembly loops
                // cb = 1 - don't breaker
                // cb = 0 - break
                let cb := 1

                let mc := add(_preBytes, 0x20)
                let end := add(mc, length)

                for {
                    let cc := add(_postBytes, 0x20)
                // the next line is the loop condition:
                // while(uint(mc < end) + cb == 2)
                } eq(add(lt(mc, end), cb), 2) {
                    mc := add(mc, 0x20)
                    cc := add(cc, 0x20)
                } {
                    // if any of these checks fails then arrays are not equal
                    if iszero(eq(mload(mc), mload(cc))) {
                        // unsuccess:
                        success := 0
                        cb := 0
                    }
                }
            }
            default {
                // unsuccess:
                success := 0
            }
        }

        return success;
    }
```

✓ The code meets the specification.

Formal Verification Request 269

Method will not encounter an assertion failure.

23, Dec 2019
0.45 ms

Line 408 in File BytesLib.sol

```
//@CTK NO_ASF
```

Line 409-450 in File BytesLib.sol

```
function equal(bytes memory _preBytes, bytes memory _postBytes) internal pure returns (
    bool) {
    bool success = true;

    assembly {
        let length := mload(_preBytes)

        // if lengths don't match the arrays are not equal
        switch eq(length, mload(_postBytes))
        case 1 {
            // cb is a circuit breaker in the for loop since there's
            //  no said feature for inline assembly loops
            // cb = 1 - don't breaker
            // cb = 0 - break
            let cb := 1

            let mc := add(_preBytes, 0x20)
            let end := add(mc, length)

            for {
                let cc := add(_postBytes, 0x20)
            // the next line is the loop condition:
            // while(uint(mc < end) + cb == 2)
            } eq(add(lt(mc, end), cb), 2) {
                mc := add(mc, 0x20)
                cc := add(cc, 0x20)
            } {
                // if any of these checks fails then arrays are not equal
                if iszero(eq(mload(mc), mload(cc))) {
                    // unsuccess:
                    success := 0
                    cb := 0
                }
            }
        }
        default {
            // unsuccess:
            success := 0
        }
    }

    return success;
}
```

✓ The code meets the specification.

Formal Verification Request 270

Buffer overflow / array index out of bound would never happen.

23, Dec 2019
6.82 ms

Line 451 in File BytesLib.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 454-524 in File BytesLib.sol

```
    function equalStorage(
        bytes storage _preBytes,
        bytes memory _postBytes
    )
        internal
        view
        returns (bool)
    {
        bool success = true;

        assembly {
            // we know _preBytes_offset is 0
            let fslot := sload(_preBytes_slot)
            // Decode the length of the stored array like in concatStorage().
            let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
            let mlength := mload(_postBytes)

            // if lengths don't match the arrays are not equal
            switch eq(slength, mlength)
            case 1 {
                // slength can contain both the length and contents of the array
                // if length < 32 bytes so let's prepare for that
                // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-
                    state-variables-in-storage
                if iszero(iszero(slength)) {
                    switch lt(slength, 32)
                    case 1 {
                        // blank the last byte which is the length
                        fslot := mul(div(fslot, 0x100), 0x100)

                        if iszero(eq(fslot, mload(add(_postBytes, 0x20)))) {
                            // unsuccess:
                            success := 0
                        }
                    }
                    default {
                        // cb is a circuit breaker in the for loop since there's
                        //  no said feature for inline assembly loops
                        // cb = 1 - don't breaker
                        // cb = 0 - break
                        let cb := 1

                        // get the keccak hash to get the contents of the array
                        mstore(0x0, _preBytes_slot)
                        let sc := keccak256(0x0, 0x20)

                        let mc := add(_postBytes, 0x20)
                        let end := add(mc, mlength)

                        // the next line is the loop condition:
                        // while(uint(mc < end) + cb == 2)
                        for {} eq(add(lt(mc, end), cb), 2) {
                            sc := add(sc, 1)
                            mc := add(mc, 0x20)
                        } {
                            if iszero(eq(sload(sc), mload(mc))) {
                                // unsuccess:
                                success := 0
                                cb := 0
                            }
                        }
                    }
                }
            }
            default {
                // unsuccess:
                success := 0
            }
        }

        return success;
    }
```

✅ The code meets the specification.

Formal Verification Request 271

If method completes, integer overflow would not happen.

📅 23, Dec 2019
⏱ 0.48 ms

Line 452 in File BytesLib.sol

```
    //@CTK NO_OVERFLOW
```

Line 454-524 in File BytesLib.sol

```
    function equalStorage(
        bytes storage _preBytes,
        bytes memory _postBytes
    )
        internal
        view
        returns (bool)
    {
        bool success = true;

        assembly {
            // we know _preBytes_offset is 0
            let fslot := sload(_preBytes_slot)
            // Decode the length of the stored array like in concatStorage().
            let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
            let mlength := mload(_postBytes)

            // if lengths don't match the arrays are not equal
            switch eq(slength, mlength)
            case 1 {
                // slength can contain both the length and contents of the array
                // if length < 32 bytes so let's prepare for that
                // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-
                    state-variables-in-storage
                if iszero(iszero(slength)) {
                    switch lt(slength, 32)
                    case 1 {
                        // blank the last byte which is the length
                        fslot := mul(div(fslot, 0x100), 0x100)

                        if iszero(eq(fslot, mload(add(_postBytes, 0x20)))) {
                            // unsuccess:
                            success := 0
                        }
                    }
                    default {
                        // cb is a circuit breaker in the for loop since there's
                        //  no said feature for inline assembly loops
                        // cb = 1 - don't breaker
                        // cb = 0 - break
                        let cb := 1

                        // get the keccak hash to get the contents of the array
                        mstore(0x0, _preBytes_slot)
                        let sc := keccak256(0x0, 0x20)

                        let mc := add(_postBytes, 0x20)
                        let end := add(mc, mlength)

                        // the next line is the loop condition:
                        // while(uint(mc < end) + cb == 2)
                        for {} eq(add(lt(mc, end), cb), 2) {
                            sc := add(sc, 1)
                            mc := add(mc, 0x20)
                        } {
                            if iszero(eq(sload(sc), mload(mc))) {
                                // unsuccess:
                                success := 0
                                cb := 0
                            }
                        }
                    }
                }
            }
            default {
                // unsuccess:
                success := 0
            }
        }

        return success;
    }
```

✓ The code meets the specification.

Formal Verification Request 272

Method will not encounter an assertion failure.

23, Dec 2019
0.4 ms

Line 453 in File BytesLib.sol

```
//@CTK NO_ASF
```

Line 454-524 in File BytesLib.sol

```
function equalStorage(
    bytes storage _preBytes,
    bytes memory _postBytes
)
    internal
    view
    returns (bool)
{
    bool success = true;

    assembly {
        // we know _preBytes_offset is 0
        let fslot := sload(_preBytes_slot)
        // Decode the length of the stored array like in concatStorage().
        let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
        let mlength := mload(_postBytes)

        // if lengths don't match the arrays are not equal
        switch eq(slength, mlength)
        case 1 {
            // slength can contain both the length and contents of the array
            // if length < 32 bytes so let's prepare for that
            // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-
                state-variables-in-storage
            if iszero(iszero(slength)) {
                switch lt(slength, 32)
                case 1 {
                    // blank the last byte which is the length
                    fslot := mul(div(fslot, 0x100), 0x100)

                    if iszero(eq(fslot, mload(add(_postBytes, 0x20)))) {
                        // unsuccess:
                        success := 0
                    }
                }
                default {
                    // cb is a circuit breaker in the for loop since there's
                    //  no said feature for inline assembly loops
                    // cb = 1 - don't breaker
                    // cb = 0 - break
                    let cb := 1

                    // get the keccak hash to get the contents of the array
                    mstore(0x0, _preBytes_slot)
                    let sc := keccak256(0x0, 0x20)

```

```
                        let mc := add(_postBytes, 0x20)
                        let end := add(mc, mlength)

                        // the next line is the loop condition:
                        // while(uint(mc < end) + cb == 2)
                        for {} eq(add(lt(mc, end), cb), 2) {
                            sc := add(sc, 1)
                            mc := add(mc, 0x20)
                        } {
                            if iszero(eq(sload(sc), mload(mc))) {
                                // unsuccess:
                                success := 0
                                cb := 0
                            }
                        }
                    }
                }
            }
            default {
                // unsuccess:
                success := 0
            }
        }

        return success;
    }
```

✓ The code meets the specification.

Formal Verification Request 273

SafeMath add

📅 23, Dec 2019
⏱ 20.68 ms

Line 26-32 in File SafeMath.sol

```
    /*@CTK "SafeMath add"
     @post (a + b < a || a + b < b) == __reverted
     @post !__reverted -> __return == a + b
     @post !__reverted -> !__has_overflow
     @post !__reverted -> !__has_assertion_failure
     @post !(__has_buf_overflow)
    */
```

Line 33-38 in File SafeMath.sol

```
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");

      return c;
  }
```

✓ The code meets the specification.

Formal Verification Request 274

SafeMath sub

📅 23, Dec 2019
⏱ 15.96 ms

Line 64-70 in File SafeMath.sol

```
    /*@CTK "SafeMath sub"
     @post (a < b) == __reverted
     @post !__reverted -> __return == a - b
     @post !__reverted -> !__has_overflow
     @post !__reverted -> !__has_assertion_failure
     @post !(__has_buf_overflow)
    */
```

Line 71-76 in File SafeMath.sol

```
   function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
       uint256) {
       require(b <= a, errorMessage);
       uint256 c = a - b;

       return c;
   }
```

✓ The code meets the specification.

Formal Verification Request 275

SafeMath mul

📅 23, Dec 2019
⏱ 221.26 ms

Line 87-93 in File SafeMath.sol

```
    /*@CTK "SafeMath mul"
     @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
     @post !__reverted -> __return == a * b
     @post !__reverted == !__has_overflow
     @post !__reverted -> !__has_assertion_failure
     @post !(__has_buf_overflow)
    */
```

Line 94-106 in File SafeMath.sol

```
   function mul(uint256 a, uint256 b) internal pure returns (uint256) {
       // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
       // benefit is lost if 'b' is also tested.
       // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
       if (a == 0) {
          return 0;
       }

       uint256 c = a * b;
       require(c / a == b, "SafeMath: multiplication overflow");

```

```
        return c;
    }
```

✓ The code meets the specification.

Formal Verification Request 276

SafeMath div

23, Dec 2019
16.85 ms

Line 136-142 in File SafeMath.sol

```
     /*@CTK "SafeMath div"
      @post (b <= 0) == __reverted
      @post !__reverted -> __return == a / b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
    */
```

Line 143-150 in File SafeMath.sol

```
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
        uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }
```

✓ The code meets the specification.

Formal Verification Request 277

SafeMath mod

23, Dec 2019
14.05 ms

Line 180-186 in File SafeMath.sol

```
     /*@CTK "SafeMath mod"
      @post (b == 0) == __reverted
      @post !__reverted -> __return == a % b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
    */
```

Line 187-190 in File SafeMath.sol

```
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
        uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
```

✓ The code meets the specification.

Source Code with CertiK Labels

File MineralNFTMarket.sol

```solidity
pragma solidity ^0.5.13;

import "./GSN/Context.sol";
import "./token/MineralNFT.sol";
import "./token/Mineral.sol";
import "./token/ERC/SafeERC20.sol";
import "./token/ERC/IERC721Receiver.sol";
import "./token/ERC/IERC20Receiver.sol";
import "./utils/Ownable.sol";
import "./utils/BytesLib.sol";
import "./math/SafeMath.sol";

contract MineralNFTMarket is Context, IERC721Receiver, IERC20Receiver, Ownable {
    using SafeMath for uint256;
    using BytesLib for bytes;
    using SafeERC20 for IERC20;

    enum ItemStatus { enable, sold, canceled }

    struct Item {
        uint256 id;
        uint256 price;
        address owner;
        uint8 status; // 0 : enable, 1 : sold, 2 : cancel
    }

    bytes4 private constant _ERC721_RECEIVED = 0x150b7a02;

    event SellItem(address owner, uint256 id, uint256 price);
    event BuyItem(address seller, address buyer, uint256 id, uint256 price);
    event CancelItem(address owner, uint256 id);
    event TakeMineral(address owner, uint256 mineral, uint256[] ids);

    mapping(uint256 => Item) private _items;
    mapping(address => uint256[]) private _soldTokenIds;
    mapping(address => uint256) private _takeableMineral;

    MineralNFT public _nft;
    IERC20 public _mineral;

    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    constructor(address nft, address mineral) public {
        setMineralNFTTokenContract(nft);
        setMineralTokenContract(mineral);
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    /*@CTK exists
      @post _items[id].price == 0 -> __return == false
      @post _items[id].price != 0 -> __return == (_items[id].status == 0)
    */
    function exists(uint id) external view returns (bool) {
        return _exists(id);
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    /*@CTK _exists
      @post _items[id].price == 0 -> __return == false
      @post _items[id].price != 0 -> __return == (_items[id].status == 0)
    */
    function _exists(uint id) internal view returns (bool) {
        if (_items[id].price == 0)
            return false;

        return _items[id].status == uint8(ItemStatus.enable);
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    /*@CTK getTakeableMineral
      @post !__reverted -> __return == _takeableMineral[msg.sender]
    */
    function getTakeableMineral() external view returns (uint256) {
        return _takeableMineral[_msgSender()];
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    /*#CTK getItemInfo
      @post !__reverted -> __return == (_items[tokenId].price, _items[tokenId].owner, _items
          [tokenId].status)
    */
    function getItemInfo(uint256 tokenId) external view returns (uint256 price, address
        owner, uint8 status) {
        return (_items[tokenId].price, _items[tokenId].owner, _items[tokenId].status);
    }

    // sel
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    /*@CTK onERC721Received
      @tag assume_completion
      @pre msg.sender == address(_nft)
      @pre _items[tokenId].price == 0 || (_items[tokenId].price != 0 && _items[tokenId].
          status != 0)
      // @pre 0 < price
      @post __return == 0x150b7a02
    */
    function onERC721Received(address operator, address from, uint256 tokenId, bytes
        calldata data) external returns (bytes4) {
        require (_msgSender() == address(_nft), "msg.sender is not nft token address");
        require (_exists(tokenId) == false, "item with input tokenId is existing");
        uint256 price = data.toUint(0);
        require (0 < price, "input price is not valid");
        _items[tokenId] = Item({
            id: tokenId,
            price: price,
            owner: operator,
            status: uint8(ItemStatus.enable)
        });
        emit SellItem(_items[tokenId].owner, _items[tokenId].id, _items[tokenId].price);
        return _ERC721_RECEIVED;
    }

    // buy
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    /*@CTK onERC20Received
      @pre msg.sender == address(_mineral)
      @pre msg.sender == address(_nft)
      // @pre _items[id].price == 0 || (_items[id].price != 0 && _items[id].status != 0)
      // @pre _items[id].price == amount
      // @pre from != _items[id].owner
      // @pre _items[id].status == 0
      // @post __post._takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner
          ].add(amount)
      // @post __post._items[id].status == 1
    */
    function onERC20Received(address from, uint256 amount, bytes memory data) public returns
         (bool) {
        require (_msgSender() == address(_mineral), "msg.sender is not mineral token address
            ");

        uint256 id = data.toUint(0);
        require (_exists(id), "item with input tokenId is existing");

        Item storage item = _items[id];

        require (item.price == amount, "input amount is not valid");
        require (from != item.owner, "input buyer is not valid");
        require (item.status == 0, "item is not available");
        //ctk start
        _takeableMineral[_items[id].owner] = _takeableMineral[_items[id].owner].add(amount);
        //ctk end
        _takeableMineral[item.owner] = _takeableMineral[item.owner].add(amount);
        _soldTokenIds[item.owner].push(id);
        item.status = uint8(ItemStatus.sold);
        _nft.safeTransferFrom(address(this), from, id);

        emit BuyItem(item.owner, from, id, amount);
        return true;
    }

     //@CTK NO_BUF_OVERFLOW
     //@CTK NO_OVERFLOW
     //@CTK NO_ASF
     /*@CTK "cancelItem"
       @pre _items[tokenId].price != 0 && _items[tokenId].price == 0 || _items[tokenId].
           status == 0
       @pre msg.sender == _items[tokenId].owner
       @pre _items[tokenId].status != 2
     */
    function cancelItem(uint256 tokenId) external {
        require (_exists(tokenId), "item with input tokenId is existing");

        Item item = _items[tokenId];

        require (_msgSender() == item.owner, "msg.sender is not token owner");
        require (item.status != uint8(ItemStatus.canceled), "item is already canceled");

        item.status = uint8(ItemStatus.canceled);
        _nft.safeTransferFrom(address(this), _msgSender(), tokenId);

        emit CancelItem(_msgSender(), tokenId);
    }

    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    /*@CTK takeMineral
      @pre 0 < _takeableMineral[msg.sender]
      @pre _soldTokenIds[msg.sender].length > 0
      @post __post._takeableMineral[msg.sender] == 0
      @post __post._soldTokenIds[msg.sender].length == 0
    */
    function takeMineral() external {
        require (0 < _takeableMineral[_msgSender()], "There is no sender's mineral to be
            take");
        _takeMineral(_msgSender());
    }

    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    /*@CTK _takeMineral
      @pre _soldTokenIds[addr].length > 0
      @post __post._takeableMineral[addr] == 0
      @post __post._soldTokenIds[addr].length == 0
    */
    function _takeMineral(address addr) internal {
        require(_soldTokenIds[addr].length > 0, "There is no mineral to be take");

        uint256 amount = _takeableMineral[addr];
        uint256[] memory tokenIds = _soldTokenIds[addr];
        _takeableMineral[addr] = 0;
        _soldTokenIds[addr].length = 0;

        _mineral.safeTransfer(addr, amount);
        emit TakeMineral(addr, amount, tokenIds);
    }

    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    /*@CTK getSoldTokenIds
      @post !__reverted -> __return == _soldTokenIds[addr]
    */
    function getSoldTokenIds(address addr) external view returns (uint256[] memory) {
        return _soldTokenIds[addr];
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    function setMineralNFTTokenContract(address addr) public onlyOwner {
        _nft = MineralNFT(addr);
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    function setMineralTokenContract(address addr) public onlyOwner {
        _mineral = IERC20(addr);
    }

    // reset contract
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    /*@CTK getTakeableMineral
      @pre msg.sender == _owner
      @post !__reverted -> __return == _takeableMineral[addr]
    */
    function getTakeableMineral(address addr) external view onlyOwner returns (uint256) {
        return _takeableMineral[addr];
    }

    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    /*@CTK takeMineralOwnerable
      @pre msg.sender == _owner
      @pre _soldTokenIds[addr].length > 0
      @post __post._takeableMineral[addr] == 0
      @post __post._soldTokenIds[addr].length == 0
    */
    function takeMineralOwnerable(address addr) external onlyOwner {
        _takeMineral(addr);
    }
}
```

File token/Mineral.sol

```
pragma solidity ^0.5.13;

import "./ERC/ERC20Burnable.sol";
import "./ERC/ERC1132.sol";

contract Mineral is ERC1132, ERC20Burnable {
    string internal constant ALREADY_LOCKED = 'Tokens already locked';
    string internal constant NOT_LOCKED = 'No tokens locked';
    string internal constant AMOUNT_ZERO = 'Amount can not be 0';

    string public name = "Mineral";
    string public symbol = "MNR";
    uint public decimals = 6;
    uint public INITIAL_SUPPLY = (10 ** 10) * (10 ** decimals);

    constructor() public {
        _mint(_msgSender(), INITIAL_SUPPLY);
    }

    /**
```

```
     * @dev Locks a specified amount of tokens against an address,
     *     for a specified reason and time
     * @param _reason The reason to lock tokens
     * @param _amount Number of tokens to be locked
     * @param _time Lock time in seconds
     */
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*#CTK lock
      @tag assume_completion
      @pre _amount != 0
      @pre locked[msg.sender][_reason].claimed || (!locked[msg.sender][_reason].claimed &&
          locked[msg.sender][_reason].amount == 0)
      @post locked[msg.sender][_reason].amount == 0 -> __post.lockReason[msg.sender].length
          == lockReason[msg.sender].length + 1
    */
    function lock(bytes32 _reason, uint256 _amount, uint256 _time)
        public
        returns (bool)
    {
        uint256 validUntil = now.add(_time); //solhint-disable-line

        // If tokens are already locked, then functions extendLock or
        // increaseLockAmount should be used to make any changes
        require(tokensLocked(_msgSender(), _reason) == 0, ALREADY_LOCKED);
        require(_amount != 0, AMOUNT_ZERO);

        if (locked[_msgSender()][_reason].amount == 0)
            lockReason[_msgSender()].push(_reason);

        transfer(address(this), _amount);

        locked[_msgSender()][_reason] = lockToken(_amount, validUntil, false);

        emit Locked(_msgSender(), _reason, _amount, validUntil);
        return true;
    }

    /**
     * @dev Transfers and Locks a specified amount of tokens,
     *     for a specified reason and time
     * @param _to adress to which tokens are to be transfered
     * @param _reason The reason to lock tokens
     * @param _amount Number of tokens to be transfered and locked
     * @param _time Lock time in seconds
     */
     //@CTK NO_BUF_OVERFLOW
     //@CTK NO_ASF
     /*#CTK transferWithLock
      @tag assume_completion
      @pre locked[_to][_reason].claimed || (!locked[_to][_reason].claimed && locked[_to][
          _reason].amount == 0)
      @pre _amount != 0
      @post locked[_to][_reason].amount == 0 -> __post.lockReason[_to].length == lockReason[
          _to].length + 1
     */
    function transferWithLock(address _to, bytes32 _reason, uint256 _amount, uint256 _time)
        external
        returns (bool)
    {
        uint256 validUntil = now.add(_time); //solhint-disable-line

        require(tokensLocked(_to, _reason) == 0, ALREADY_LOCKED);
        require(_amount != 0, AMOUNT_ZERO);

        if (locked[_to][_reason].amount == 0)
            lockReason[_to].push(_reason);

        transfer(address(this), _amount);

        locked[_to][_reason] = lockToken(_amount, validUntil, false);

        emit Locked(_to, _reason, _amount, validUntil);
        return true;
    }

    /**
     * @dev Returns tokens locked for a specified address for a
     *      specified reason
     *
     * @param _of The address whose tokens are locked
     * @param _reason The reason to query the lock tokens for
     */
     //@CTK NO_BUF_OVERFLOW
     //@CTK NO_OVERFLOW
     //@CTK NO_ASF
    /*#CTK tokensLocked
      @tag assume_completion
      @post !locked[_of][_reason].claimed -> __return == locked[_of][_reason].amount
    */
    function tokensLocked(address _of, bytes32 _reason)
        public
        view
        returns (uint256 amount)
    {
        if (!locked[_of][_reason].claimed)
            amount = locked[_of][_reason].amount;
    }

    /**
     * @dev Returns tokens locked for a specified address for a
     *      specified reason at a specific time
     *
     * @param _of The address whose tokens are locked
     * @param _reason The reason to query the lock tokens for
     * @param _time The timestamp to query the lock tokens for
     */
     //@CTK NO_BUF_OVERFLOW
     //@CTK NO_OVERFLOW
     //@CTK NO_ASF
    /*#CTK tokensLockedAtTime
      @tag assume_completion
      @post locked[_of][_reason].validity > _time -> __return == locked[_of][_reason].amount
    */
    function tokensLockedAtTime(address _of, bytes32 _reason, uint256 _time)
        public
        view
        returns (uint256 amount)
    {
        if (locked[_of][_reason].validity > _time)
            amount = locked[_of][_reason].amount;
    }

    /**
     * @dev Returns total tokens held by an address (locked + transferable)
     * @param _of The address to query the total balance of
     */
     //@CTK NO_BUF_OVERFLOW
     //@CTK NO_OVERFLOW
     //@CTK NO_ASF
    /*#CTK totalBalanceOf
      @post !__reverted -> __return == balances[_of]
    */
    function totalBalanceOf(address _of)
        public
        view
        returns (uint256 amount)
    {
        amount = balanceOf(_of);
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            amount = amount.add(tokensLocked(_of, lockReason[_of][i]));
        }
    }

    /**
     * @dev Extends lock for a specified reason and time
     * @param _reason The reason to lock tokens
     * @param _time Lock extension time in seconds
     */
     //@CTK NO_BUF_OVERFLOW
     //@CTK NO_OVERFLOW
     //@CTK NO_ASF
    /*@CTK extendLock
      @tag assume_completion
      @pre (!locked[msg.sender][_reason].claimed && locked[msg.sender][_reason].amount > 0)
      @post __post.locked[msg.sender][_reason].validity == locked[msg.sender][_reason].
          validity + _time
    */
    function extendLock(bytes32 _reason, uint256 _time)
        public
        returns (bool)
    {
        require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);

        locked[_msgSender()][_reason].validity = locked[_msgSender()][_reason].validity.add(
            _time);

        emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
            _msgSender()][_reason].validity);
        return true;
    }

    /**
     * @dev Increase number of tokens locked for a specified reason
     * @param _reason The reason to lock tokens
     * @param _amount Number of tokens to be increased
     */
     //@CTK NO_BUF_OVERFLOW
     //@CTK NO_OVERFLOW
     //@CTK NO_ASF
    /*@CTK increaseLockAmount
      @tag assume_completion
      @pre (!locked[msg.sender][_reason].claimed && locked[msg.sender][_reason].amount > 0)
      @post __post.locked[msg.sender][_reason].amount == locked[msg.sender][_reason].amount
          + _amount
    */
    function increaseLockAmount(bytes32 _reason, uint256 _amount)
        public
        returns (bool)
    {
        require(tokensLocked(_msgSender(), _reason) > 0, NOT_LOCKED);
        transfer(address(this), _amount);

        locked[_msgSender()][_reason].amount = locked[_msgSender()][_reason].amount.add(
            _amount);

        emit Locked(_msgSender(), _reason, locked[_msgSender()][_reason].amount, locked[
            _msgSender()][_reason].validity);
        return true;
    }

    /**
     * @dev Returns unlockable tokens for a specified address for a specified reason
     * @param _of The address to query the the unlockable token count of
     * @param _reason The reason to query the unlockable tokens for
     */
     //@CTK NO_BUF_OVERFLOW
     //@CTK NO_OVERFLOW
     //@CTK NO_ASF
    /*#CTK tokensUnlockable
      @post (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) ->
          __return == locked[_of][_reason].amount
    */
    function tokensUnlockable(address _of, bytes32 _reason)
        public
        view
        returns (uint256 amount)
    {
        if (locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed) //solhint-
            disable-line
            amount = locked[_of][_reason].amount;
    }

    /**
     * @dev Unlocks the unlockable tokens of a specified address
     * @param _of Address of user, claiming back unlockable tokens
     */
     //@CTK NO_BUF_OVERFLOW
     //@CTK NO_OVERFLOW
     //@CTK NO_ASF
    function unlockAll(address _of)
        public
        returns (uint256 unlockableTokens)
    {
        uint256 lockedTokens;

        /*#CTK "loop_unlockAll"
          @inv i <= lockReason[_of].length
          @inv forall j: uint. (j >= 0 /\ j < i /\ (locked[_of][lockReason[_of][i]].validity
              <= now && !locked[_of][lockReason[_of][i]].claimed && locked[_of][lockReason[
              _of][i]].amount > 0)) -> locked[_of][lockReason[_of][i]].claimed
          @post i == lockReason[_of].length
          @post !__should_return
        */
        /*@CTK loop
          @inv true
        */
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            lockedTokens = tokensUnlockable(_of, lockReason[_of][i]);
            if (lockedTokens > 0) {
                unlockableTokens = unlockableTokens.add(lockedTokens);
                locked[_of][lockReason[_of][i]].claimed = true;
                emit Unlocked(_of, lockReason[_of][i], lockedTokens);
            }
        }

        if (unlockableTokens > 0)
            this.transfer(_of, unlockableTokens);
    }

    /**
     * @dev Unlock once
     * @param _of Address of user, claiming back unlockable tokens
     * @param _reason Once reason
     */
     //@CTK NO_BUF_OVERFLOW
     //@CTK NO_OVERFLOW
     //@CTK NO_ASF
    /*#CTK unlock
      @post locked[_of][_reason].validity <= now && !locked[_of][_reason].claimed && locked[
          _of][_reason].amount > 0 -> __post.locked[_of][_reason].claimed
    */
    function unlock(address _of, bytes32 _reason)
        public
        returns (uint256 unlocked)
    {
        unlocked = tokensUnlockable(_of, _reason);
        if (unlocked > 0) {
            locked[_of][_reason].claimed = true;
            emit Unlocked(_of, _reason, unlocked);
            this.transfer(_of, unlocked);
        }
    }

    /**
     * @dev Gets the unlockable tokens of a specified address
     * @param _of The address to query the the unlockable token count of
     */
     //@CTK NO_BUF_OVERFLOW
     //@CTK NO_OVERFLOW
     //@CTK NO_ASF
    function getUnlockableTokens(address _of)
        public
        view
        returns (uint256 unlockableTokens)
    {
        for (uint256 i = 0; i < lockReason[_of].length; i++) {
            unlockableTokens = unlockableTokens.add(tokensUnlockable(_of, lockReason[_of][i])
                );
        }
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function getLockReasons(address _of, uint256 _start, uint256 _end)
        external
        view
        returns (bytes32[] memory reasons)
    {
        uint256 length = _end - _start;
        reasons = new bytes32[](length);
        /*@CTK loop_getLockReasons
          @inv i <= length
      @inv forall j: uint. (j >= 0 /\ j < i) -> reasons[j] == this.lockReason[_of][_start +
          j]
          @post i == length
          @post !__should_return
        */
        for (uint256 i = 0; i < length; i++) {
            reasons[i] = lockReason[_of][_start + i];
        }
        return reasons;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    function getLockReasonLength(address _of)
        external
        view
        returns (uint256 length)
    {
        return lockReason[_of].length;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    function safeTransfer(address _to, uint256 _amount, bytes calldata _data)
        external
    {
        require(transfer(_to, _amount), "ERC20: failed transfer");
        require(_checkOnERC20Received(_to, _amount, _data), "ERC20: transfer to non
            ERC20Receiver implementer");
    }

    //@CTK NO_BUF_OVERFLOW
    function _checkOnERC20Received(address _to, uint256 _amount, bytes memory _data)
        internal
        returns (bool)
    {
        if (!_to.isContract()) {
            return true;
        }

        return IERC20Receiver(_to).onERC20Received(_msgSender(), _amount, _data);
    }
}
```

File token/MineralNFT.sol

```
pragma solidity ^0.5.13;

import "./ERC/ERC721Full.sol";
import "../utils/Ownable.sol";
import "../math/SafeMath.sol";

contract MineralNFT is ERC721Full, Ownable {
    using SafeMath for uint256;

    uint256 private _finalTokenId = 0;

    constructor (string memory name, string memory symbol) ERC721Full(name, symbol) public {
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK _generateTokenId
      @post __post._finalTokenId == _finalTokenId + 1
      @post !__reverted -> __return == _finalTokenId
    */
    function _generateTokenId() internal returns (uint256) {
        return _finalTokenId++;
    }

   //@CTK NO_BUF_OVERFLOW
   //@CTK NO_ASF
    /*@CTK createItem
      @post !__reverted -> __return == _finalTokenId
    */
    function createItem(address to, string calldata jsonUrl) external onlyOwner returns (
        uint256) {
        uint256 id = _generateTokenId();
        _mint(to, id);
        _setTokenURI(id, jsonUrl);
        return id;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    function burnItem(uint256 tokenId) external {
        require(_isApprovedOrOwner(_msgSender(), tokenId), "msg.sender is not token owner");
        _burn(_msgSender(), tokenId);
    }
}
```

File token/ERC/ERC20.sol

```
pragma solidity ^0.5.13;

import "../../GSN/Context.sol";
import "./IERC20.sol";
import "./IERC20Receiver.sol";
import "../../math/SafeMath.sol";
import "../../utils/Address.sol";

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20Mintable}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;
    using Address for address;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    /**
     * @dev See {IERC20-totalSupply}.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for `sender`'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public returns
        (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20:
            transfer amount exceeds allowance"));
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (
        bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(
            subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds
            balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements
     *
     * - `to` cannot be the zero address.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds
            balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
     *
     * This is internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function _approve(address owner, address spender, uint256 amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See {_burn} and {_approve}.
     */
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "
            ERC20: burn amount exceeds balance"));
    }
}
```

File token/ERC/ERC721Enumerable.sol

```
pragma solidity ^0.5.13;

import "../../GSN/Context.sol";
import "./IERC721Enumerable.sol";
import "./ERC721.sol";
import "./ERC165.sol";

/**
 * @title ERC-721 Non-Fungible Token with optional enumeration extension logic
 * @dev See https://eips.ethereum.org/EIPS/eip-721
 */
contract ERC721Enumerable is Context, ERC165, ERC721, IERC721Enumerable {
    // Mapping from owner to list of owned token IDs
    mapping(address => uint256[]) private _ownedTokens;

    // Mapping from token ID to index of the owner tokens list
    mapping(uint256 => uint256) private _ownedTokensIndex;

    // Array with all token ids, used for enumeration
    uint256[] private _allTokens;

    // Mapping from token id to position in the allTokens array
    mapping(uint256 => uint256) private _allTokensIndex;

    /*
     *     bytes4(keccak256('totalSupply()')) == 0x18160ddd
     *     bytes4(keccak256('tokenOfOwnerByIndex(address,uint256)')) == 0x2f745c59
     *     bytes4(keccak256('tokenByIndex(uint256)')) == 0x4f6ccce7
     *
     *     => 0x18160ddd ^ 0x2f745c59 ^ 0x4f6ccce7 == 0x780e9d63
     */
    bytes4 private constant _INTERFACE_ID_ERC721_ENUMERABLE = 0x780e9d63;

    /**
     * @dev Constructor function.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    constructor () public {
        // register the supported interface to conform to ERC721Enumerable via ERC165
        _registerInterface(_INTERFACE_ID_ERC721_ENUMERABLE);
    }

    /**
     * @dev Gets the token ID at a given index of the tokens list of the requested owner.
     * @param owner address owning the tokens list to be accessed
     * @param index uint256 representing the index to be accessed of the requested tokens
         list
     * @return uint256 token ID at the given index of the tokens list owned by the requested
          address
     */

    function tokenOfOwnerByIndex(address owner, uint256 index) public view returns (uint256)
         {
        require(index < balanceOf(owner), "ERC721Enumerable: owner index out of bounds");
        return _ownedTokens[owner][index];
    }

    /**
     * @dev Gets the total amount of tokens stored by the contract.
     * @return uint256 representing the total amount of tokens
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function totalSupply() public view returns (uint256) {
        return _allTokens.length;
    }

    /**
     * @dev Gets the token ID at a given index of all the tokens in this contract
     * Reverts if the index is greater or equal to the total number of tokens.
     * @param index uint256 representing the index to be accessed of the tokens list
     * @return uint256 token ID at the given index of the tokens list
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function tokenByIndex(uint256 index) public view returns (uint256) {
        require(index < totalSupply(), "ERC721Enumerable: global index out of bounds");
        return _allTokens[index];
    }

    /**
     * @dev Internal function to transfer ownership of a given token ID to another address.
     * As opposed to transferFrom, this imposes no restrictions on msg.sender.
     * @param from current owner of the token
     * @param to address to receive the ownership of the given token ID
     * @param tokenId uint256 ID of the token to be transferred
     */
    function _transferFrom(address from, address to, uint256 tokenId) internal {
        super._transferFrom(from, to, tokenId);

        _removeTokenFromOwnerEnumeration(from, tokenId);

        _addTokenToOwnerEnumeration(to, tokenId);
    }

    /**
     * @dev Internal function to mint a new token.
     * Reverts if the given token ID already exists.
     * @param to address the beneficiary that will own the minted token
     * @param tokenId uint256 ID of the token to be minted
     */
     //@CTK NO_ASF
     //@CTK NO_BUF_OVERFLOW
    function _mint(address to, uint256 tokenId) internal {
        super._mint(to, tokenId);

        _addTokenToOwnerEnumeration(to, tokenId);

        _addTokenToAllTokensEnumeration(tokenId);
    }

    /**
     * @dev Internal function to burn a specific token.
     * Reverts if the token does not exist.
     * Deprecated, use {ERC721-_burn} instead.
     * @param owner owner of the token to burn
     * @param tokenId uint256 ID of the token being burned
     */

    function _burn(address owner, uint256 tokenId) internal {
        super._burn(owner, tokenId);

        _removeTokenFromOwnerEnumeration(owner, tokenId);
        // Since tokenId will be deleted, we can clear its slot in _ownedTokensIndex to
            trigger a gas refund
        _ownedTokensIndex[tokenId] = 0;

        _removeTokenFromAllTokensEnumeration(tokenId);
    }

    /**
     * @dev Gets the list of token IDs of the requested owner.
     * @param owner address owning the tokens
     * @return uint256[] List of token IDs owned by the requested address
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function _tokensOfOwner(address owner) internal view returns (uint256[] storage) {
        return _ownedTokens[owner];
    }

    /**
```

```
     * @dev Private function to add a token to this extension's ownership-tracking data
         structures.
     * @param to address representing the new owner of the given token ID
     * @param tokenId uint256 ID of the token to be added to the tokens list of the given
         address
     */
     //@CTK NO_ASF
     //@CTK NO_BUF_OVERFLOW
    function _addTokenToOwnerEnumeration(address to, uint256 tokenId) private {
        _ownedTokensIndex[tokenId] = _ownedTokens[to].length;
        _ownedTokens[to].push(tokenId);
    }

    /**
     * @dev Private function to add a token to this extension's token tracking data
         structures.
     * @param tokenId uint256 ID of the token to be added to the tokens list
     */
     //@CTK NO_ASF
     //@CTK NO_BUF_OVERFLOW
    function _addTokenToAllTokensEnumeration(uint256 tokenId) private {
        _allTokensIndex[tokenId] = _allTokens.length;
        _allTokens.push(tokenId);
    }

    /**
     * @dev Private function to remove a token from this extension's ownership-tracking data
          structures. Note that
     * while the token is not assigned a new owner, the `_ownedTokensIndex` mapping is _not_
          updated: this allows for
     * gas optimizations e.g. when performing a transfer operation (avoiding double writes).
     * This has O(1) time complexity, but alters the order of the _ownedTokens array.
     * @param from address representing the previous owner of the given token ID
     * @param tokenId uint256 ID of the token to be removed from the tokens list of the
         given address
     */

    function _removeTokenFromOwnerEnumeration(address from, uint256 tokenId) private {
        // To prevent a gap in from's tokens array, we store the last token in the index of
            the token to delete, and
        // then delete the last slot (swap and pop).

        uint256 lastTokenIndex = _ownedTokens[from].length.sub(1);
        uint256 tokenIndex = _ownedTokensIndex[tokenId];

        // When the token to delete is the last token, the swap operation is unnecessary
        if (tokenIndex != lastTokenIndex) {
            uint256 lastTokenId = _ownedTokens[from][lastTokenIndex];

            _ownedTokens[from][tokenIndex] = lastTokenId; // Move the last token to the slot
                of the to-delete token
            _ownedTokensIndex[lastTokenId] = tokenIndex; // Update the moved token's index
        }

        // This also deletes the contents at the last position of the array
        _ownedTokens[from].length--;

        // Note that _ownedTokensIndex[tokenId] hasn't been cleared: it still points to the
```

```
          old slot (now occupied by
        // lastTokenId, or just over the end of the array if the token was the last one).
    }

    /**
     * @dev Private function to remove a token from this extension's token tracking data
         structures.
     * This has O(1) time complexity, but alters the order of the _allTokens array.
     * @param tokenId uint256 ID of the token to be removed from the tokens list
     */

    function _removeTokenFromAllTokensEnumeration(uint256 tokenId) private {
        // To prevent a gap in the tokens array, we store the last token in the index of the
            token to delete, and
        // then delete the last slot (swap and pop).

        uint256 lastTokenIndex = _allTokens.length.sub(1);
        uint256 tokenIndex = _allTokensIndex[tokenId];

        // When the token to delete is the last token, the swap operation is unnecessary.
           However, since this occurs so
        // rarely (when the last minted token is burnt) that we still do the swap here to
           avoid the gas cost of adding
        // an 'if' statement (like in _removeTokenFromOwnerEnumeration)
        uint256 lastTokenId = _allTokens[lastTokenIndex];

        _allTokens[tokenIndex] = lastTokenId; // Move the last token to the slot of the to-
           delete token
        _allTokensIndex[lastTokenId] = tokenIndex; // Update the moved token's index

        // This also deletes the contents at the last position of the array
        _allTokens.length--;
        _allTokensIndex[tokenId] = 0;
    }
}
```

File token/ERC/ERC721Metadata.sol

```
pragma solidity ^0.5.13;

import "../../GSN/Context.sol";
import "./ERC721.sol";
import "./IERC721Metadata.sol";
import "./ERC165.sol";

contract ERC721Metadata is Context, ERC165, ERC721, IERC721Metadata {
    // Token name
    string private _name;

    // Token symbol
    string private _symbol;

    // Base URI
    string private _baseURI;

    // Optional mapping for token URIs
    mapping(uint256 => string) private _tokenURIs;

    /*
```

```
     *    bytes4(keccak256('name()')) == 0x06fdde03
     *    bytes4(keccak256('symbol()')) == 0x95d89b41
     *    bytes4(keccak256('tokenURI(uint256)')) == 0xc87b56dd
     *
     *    => 0x06fdde03 ^ 0x95d89b41 ^ 0xc87b56dd == 0x5b5e139f
     */
    bytes4 private constant _INTERFACE_ID_ERC721_METADATA = 0x5b5e139f;

    /**
     * @dev Constructor function
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    constructor (string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;

        // register the supported interfaces to conform to ERC721 via ERC165
        _registerInterface(_INTERFACE_ID_ERC721_METADATA);
    }

    /**
     * @dev Gets the token name.
     * @return string representing the token name
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function name() external view returns (string memory) {
        return _name;
    }

    /**
     * @dev Gets the token symbol.
     * @return string representing the token symbol
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function symbol() external view returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the URI for a given token ID. May return an empty string.
     *
     * If the token's URI is non-empty and a base URI was set (via
     * {_setBaseURI}), it will be added to the token ID's URI as a prefix.
     *
     * Reverts if the token ID does not exist.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function tokenURI(uint256 tokenId) external view returns (string memory) {
        require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");

        string memory _tokenURI = _tokenURIs[tokenId];

        // Even if there is a base URI, it is only appended to non-empty token-specific URIs
        if (bytes(_tokenURI).length == 0) {
            return "";
        } else {
            // abi.encodePacked is being used to concatenate strings
            return string(abi.encodePacked(_baseURI, _tokenURI));
        }
    }

    /**
    * @dev Returns the base URI set via {_setBaseURI}. This will be
    * automatically added as a preffix in {tokenURI} to each token's URI, when
    * they are non-empty.
    */
    //@CTK NO_ASF
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    function baseURI() external view returns (string memory) {
        return _baseURI;
    }

    /**
     * @dev Internal function to set the token URI for a given token.
     *
     * Reverts if the token ID does not exist.
     *
     * TIP: if all token IDs share a prefix (e.g. if your URIs look like
     * `http://api.myproject.com/token/<id>`), use {_setBaseURI} to store
     * it and save gas.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function _setTokenURI(uint256 tokenId, string memory _tokenURI) internal {
        require(_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
        _tokenURIs[tokenId] = _tokenURI;
    }

    /**
     * @dev Internal function to set the base URI for all token IDs. It is
     * automatically added as a prefix to the value returned in {tokenURI}.
     *
     * _Available since v2.5.0._
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function _setBaseURI(string memory uri) internal {
        _baseURI = uri;
    }

    /**
     * @dev Internal function to burn a specific token.
     * Reverts if the token does not exist.
     * Deprecated, use _burn(uint256) instead.
     * @param owner owner of the token to burn
     * @param tokenId uint256 ID of the token being burned by the msg.sender
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function _burn(address owner, uint256 tokenId) internal {
        super._burn(owner, tokenId);

        // Clear metadata (if any)
        if (bytes(_tokenURIs[tokenId]).length != 0) {
            delete _tokenURIs[tokenId];
        }
    }
}
```

File token/ERC/ERC721.sol

```
pragma solidity ^0.5.13;

import "../../GSN/Context.sol";
import "./IERC721.sol";
import "./IERC721Receiver.sol";
import "./ERC165.sol";
import "../../math/SafeMath.sol";
import "../../utils/Address.sol";
import "../../drafts/Counters.sol";

/**
 * @title ERC721 Non-Fungible Token Standard basic implementation
 * @dev see https://eips.ethereum.org/EIPS/eip-721
 */
contract ERC721 is Context, ERC165, IERC721 {
    using SafeMath for uint256;
    using Address for address;
    using Counters for Counters.Counter;

    // Equals to `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
    // which can be also obtained as `IERC721Receiver(0).onERC721Received.selector`
    bytes4 private constant _ERC721_RECEIVED = 0x150b7a02;

    // Mapping from token ID to owner
    mapping (uint256 => address) private _tokenOwner;

    // Mapping from token ID to approved address
    mapping (uint256 => address) private _tokenApprovals;

    // Mapping from owner to number of owned token
    mapping (address => Counters.Counter) private _ownedTokensCount;

    // Mapping from owner to operator approvals
    mapping (address => mapping (address => bool)) private _operatorApprovals;

    /*
     *     bytes4(keccak256('balanceOf(address)')) == 0x70a08231
     *     bytes4(keccak256('ownerOf(uint256)')) == 0x6352211e
     *     bytes4(keccak256('approve(address,uint256)')) == 0x095ea7b3
     *     bytes4(keccak256('getApproved(uint256)')) == 0x081812fc
     *     bytes4(keccak256('setApprovalForAll(address,bool)')) == 0xa22cb465
     *     bytes4(keccak256('isApprovedForAll(address,address)')) == 0xe985e9c5
     *    bytes4(keccak256('transferFrom(address,address,uint256)')) == 0x23b872dd
     *    bytes4(keccak256('safeTransferFrom(address,address,uint256)')) == 0x42842e0e
     *    bytes4(keccak256('safeTransferFrom(address,address,uint256,bytes)')) == 0xb88d4fde
     *
     *    => 0x70a08231 ^ 0x6352211e ^ 0x095ea7b3 ^ 0x081812fc ^
     *       0xa22cb465 ^ 0xe985e9c ^ 0x23b872dd ^ 0x42842e0e ^ 0xb88d4fde == 0x80ac58cd
     */
    bytes4 private constant _INTERFACE_ID_ERC721 = 0x80ac58cd;
    //@CTK NO_ASF
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    constructor () public {
       // register the supported interfaces to conform to ERC721 via ERC165
       _registerInterface(_INTERFACE_ID_ERC721);
    }

    /**
     * @dev Gets the balance of the specified address.
     * @param owner address to query the balance of
     * @return uint256 representing the amount owned by the passed address
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function balanceOf(address owner) public view returns (uint256) {
       require(owner != address(0), "ERC721: balance query for the zero address");

       return _ownedTokensCount[owner].current();
    }

    /**
     * @dev Gets the owner of the specified token ID.
     * @param tokenId uint256 ID of the token to query the owner of
     * @return address currently marked as the owner of the given token ID
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function ownerOf(uint256 tokenId) public view returns (address) {
       address owner = _tokenOwner[tokenId];
       require(owner != address(0), "ERC721: owner query for nonexistent token");

       return owner;
    }

    /**
     * @dev Approves another address to transfer the given token ID
     * The zero address indicates there is no approved address.
     * There can only be one approved address per token at a given time.
     * Can only be called by the token owner or an approved operator.
     * @param to address to be approved for the given token ID
     * @param tokenId uint256 ID of the token to be approved
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function approve(address to, uint256 tokenId) public {
       address owner = ownerOf(tokenId);
```

```
        require(to != owner, "ERC721: approval to current owner");

        require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()),
            "ERC721: approve caller is not owner nor approved for all"
        );

        _tokenApprovals[tokenId] = to;
        emit Approval(owner, to, tokenId);
    }

    /**
     * @dev Gets the approved address for a token ID, or zero if no address set
     * Reverts if the token ID does not exist.
     * @param tokenId uint256 ID of the token to query the approval of
     * @return address currently approved for the given token ID
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function getApproved(uint256 tokenId) public view returns (address) {
        require(_exists(tokenId), "ERC721: approved query for nonexistent token");

        return _tokenApprovals[tokenId];
    }

    /**
     * @dev Sets or unsets the approval of a given operator
     * An operator is allowed to transfer all tokens of the sender on their behalf.
     * @param to operator address to set the approval
     * @param approved representing the status of the approval to be set
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function setApprovalForAll(address to, bool approved) public {
        require(to != _msgSender(), "ERC721: approve to caller");

        _operatorApprovals[_msgSender()][to] = approved;
        emit ApprovalForAll(_msgSender(), to, approved);
    }

    /**
     * @dev Tells whether an operator is approved by a given owner.
     * @param owner owner address which you want to query the approval of
     * @param operator operator address which you want to query the approval of
     * @return bool whether the given operator is approved by the given owner
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function isApprovedForAll(address owner, address operator) public view returns (bool) {
        return _operatorApprovals[owner][operator];
    }

    /**
     * @dev Transfers the ownership of a given token ID to another address.
     * Usage of this method is discouraged, use {safeTransferFrom} whenever possible.
     * Requires the msg.sender to be the owner, approved, or operator.
     * @param from current owner of the token
     * @param to address to receive the ownership of the given token ID
     * @param tokenId uint256 ID of the token to be transferred
     */

    function transferFrom(address from, address to, uint256 tokenId) public {
        //solhint-disable-next-line max-line-length
        require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not
            owner nor approved");

        _transferFrom(from, to, tokenId);
    }

    /**
     * @dev Safely transfers the ownership of a given token ID to another address
     * If the target address is a contract, it must implement {IERC721Receiver-
         onERC721Received},
     * which is called upon a safe transfer, and return the magic value
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
     * the transfer is reverted.
     * Requires the msg.sender to be the owner, approved, or operator
     * @param from current owner of the token
     * @param to address to receive the ownership of the given token ID
     * @param tokenId uint256 ID of the token to be transferred
     */

    function safeTransferFrom(address from, address to, uint256 tokenId) public {
        safeTransferFrom(from, to, tokenId, "");
    }

    /**
     * @dev Safely transfers the ownership of a given token ID to another address
     * If the target address is a contract, it must implement {IERC721Receiver-
         onERC721Received},
     * which is called upon a safe transfer, and return the magic value
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
     * the transfer is reverted.
     * Requires the _msgSender() to be the owner, approved, or operator
     * @param from current owner of the token
     * @param to address to receive the ownership of the given token ID
     * @param tokenId uint256 ID of the token to be transferred
     * @param _data bytes data to send along with a safe transfer check
     */

    function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data)
         public {
        require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not
            owner nor approved");
        _safeTransferFrom(from, to, tokenId, _data);
    }

    /**
     * @dev Safely transfers the ownership of a given token ID to another address
     * If the target address is a contract, it must implement `onERC721Received`,
     * which is called upon a safe transfer, and return the magic value
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
     * the transfer is reverted.
     * Requires the msg.sender to be the owner, approved, or operator
     * @param from current owner of the token
     * @param to address to receive the ownership of the given token ID
     * @param tokenId uint256 ID of the token to be transferred
     * @param _data bytes data to send along with a safe transfer check
     */

    function _safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data
        ) internal {
        _transferFrom(from, to, tokenId);
        require(_checkOnERC721Received(from, to, tokenId, _data), "ERC721: transfer to non
            ERC721Receiver implementer");
    }

    /**
     * @dev Returns whether the specified token exists.
     * @param tokenId uint256 ID of the token to query the existence of
     * @return bool whether the token exists
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function _exists(uint256 tokenId) internal view returns (bool) {
        address owner = _tokenOwner[tokenId];
        return owner != address(0);
    }

    /**
     * @dev Returns whether the given spender can transfer a given token ID.
     * @param spender address of the spender to query
     * @param tokenId uint256 ID of the token to be transferred
     * @return bool whether the msg.sender is approved for the given token ID,
     * is an operator of the owner, or is the owner of the token
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function _isApprovedOrOwner(address spender, uint256 tokenId) internal view returns (
        bool) {
        require(_exists(tokenId), "ERC721: operator query for nonexistent token");
        address owner = ownerOf(tokenId);
        return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(
            owner, spender));
    }

    /**
     * @dev Internal function to safely mint a new token.
     * Reverts if the given token ID already exists.
     * If the target address is a contract, it must implement `onERC721Received`,
     * which is called upon a safe transfer, and return the magic value
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
     * the transfer is reverted.
     * @param to The address that will own the minted token
     * @param tokenId uint256 ID of the token to be minted
     */

    function _safeMint(address to, uint256 tokenId) internal {
        _safeMint(to, tokenId, "");
    }

    /**
     * @dev Internal function to safely mint a new token.
     * Reverts if the given token ID already exists.
     * If the target address is a contract, it must implement `onERC721Received`,
     * which is called upon a safe transfer, and return the magic value
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
     * the transfer is reverted.
     * @param to The address that will own the minted token
     * @param tokenId uint256 ID of the token to be minted
     * @param _data bytes data to send along with a safe transfer check
     */

    function _safeMint(address to, uint256 tokenId, bytes memory _data) internal {
        _mint(to, tokenId);
        require(_checkOnERC721Received(address(0), to, tokenId, _data), "ERC721: transfer to
             non ERC721Receiver implementer");
    }

    /**
     * @dev Internal function to mint a new token.
     * Reverts if the given token ID already exists.
     * @param to The address that will own the minted token
     * @param tokenId uint256 ID of the token to be minted
     */
     //@CTK NO_ASF
     //@CTK NO_BUF_OVERFLOW
    function _mint(address to, uint256 tokenId) internal {
        require(to != address(0), "ERC721: mint to the zero address");
        require(!_exists(tokenId), "ERC721: token already minted");

        _tokenOwner[tokenId] = to;
        _ownedTokensCount[to].increment();

        emit Transfer(address(0), to, tokenId);
    }

    /**
     * @dev Internal function to burn a specific token.
     * Reverts if the token does not exist.
     * Deprecated, use {_burn} instead.
     * @param owner owner of the token to burn
     * @param tokenId uint256 ID of the token being burned
     */

    function _burn(address owner, uint256 tokenId) internal {
        require(ownerOf(tokenId) == owner, "ERC721: burn of token that is not own");

        _clearApproval(tokenId);

        _ownedTokensCount[owner].decrement();
        _tokenOwner[tokenId] = address(0);

        emit Transfer(owner, address(0), tokenId);
    }

    /**
     * @dev Internal function to burn a specific token.
     * Reverts if the token does not exist.
     * @param tokenId uint256 ID of the token being burned
     */

    function _burn(uint256 tokenId) internal {
        _burn(ownerOf(tokenId), tokenId);
    }

    /**
     * @dev Internal function to transfer ownership of a given token ID to another address.
     * As opposed to {transferFrom}, this imposes no restrictions on msg.sender.
     * @param from current owner of the token
     * @param to address to receive the ownership of the given token ID
     * @param tokenId uint256 ID of the token to be transferred
     */
    function _transferFrom(address from, address to, uint256 tokenId) internal {
        require(ownerOf(tokenId) == from, "ERC721: transfer of token that is not own");
        require(to != address(0), "ERC721: transfer to the zero address");

        _clearApproval(tokenId);

        _ownedTokensCount[from].decrement();
        _ownedTokensCount[to].increment();

        _tokenOwner[tokenId] = to;

        emit Transfer(from, to, tokenId);
    }

    /**
     * @dev Internal function to invoke {IERC721Receiver-onERC721Received} on a target
         address.
     * The call is not executed if the target address is not a contract.
     *
     * This is an internal detail of the `ERC721` contract and its use is deprecated.
     * @param from address representing the previous owner of the given token ID
     * @param to target address that will receive the tokens
     * @param tokenId uint256 ID of the token to be transferred
     * @param _data bytes optional data to send along with the call
     * @return bool whether the call correctly returned the expected magic value
     */

    function _checkOnERC721Received(address from, address to, uint256 tokenId, bytes memory
         _data)
        internal returns (bool)
    {
        if (!to.isContract()) {
            return true;
        }
        bytes4 retval = IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId,
             _data);
        return (retval == _ERC721_RECEIVED);
    }

    /**
     * @dev Private function to clear current approval of a given token ID.
     * @param tokenId uint256 ID of the token to be transferred
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function _clearApproval(uint256 tokenId) private {
        if (_tokenApprovals[tokenId] != address(0)) {
            _tokenApprovals[tokenId] = address(0);
        }
    }
}
```

File token/ERC/ERC165.sol

```
pragma solidity ^0.5.13;

import "./IERC165.sol";

/**
 * @dev Implementation of the {IERC165} interface.
 *
 * Contracts may inherit from this and call {_registerInterface} to declare
 * their support of an interface.
 */
contract ERC165 is IERC165 {
    /*
     * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
     */
    bytes4 private constant _INTERFACE_ID_ERC165 = 0x01ffc9a7;

    /**
     * @dev Mapping of interface ids to whether or not it's supported.
     */
    mapping(bytes4 => bool) private _supportedInterfaces;
    //@CTK NO_ASF
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    constructor () internal {
        // Derived contracts need only register support for their own interfaces,
        // we register support for ERC165 itself here
        _registerInterface(_INTERFACE_ID_ERC165);
    }

    /**
     * @dev See {IERC165-supportsInterface}.
     *
     * Time complexity O(1), guaranteed to always use less than 30 000 gas.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function supportsInterface(bytes4 interfaceId) external view returns (bool) {
        return _supportedInterfaces[interfaceId];
    }

    /**
     * @dev Registers the contract as an implementer of the interface defined by
     * `interfaceId`. Support of the actual ERC165 interface is automatic and
     * registering its interface id is not required.
     *
     * See {IERC165-supportsInterface}.
     *
     * Requirements:
     *
     * - `interfaceId` cannot be the ERC165 invalid interface (`0xffffffff`).
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function _registerInterface(bytes4 interfaceId) internal {
        require(interfaceId != 0xffffffff, "ERC165: invalid interface id");
        _supportedInterfaces[interfaceId] = true;
    }
}
```

File token/ERC/ERC20Burnable.sol

```
pragma solidity ^0.5.13;

import "../../GSN/Context.sol";
import "./ERC20.sol";

/**
 * @dev Extension of {ERC20} that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
contract ERC20Burnable is Context, ERC20 {
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function burn(uint256 amount) public {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev See {ERC20-_burnFrom}.
     */
     //@CTK NO_ASF
     //@CTK NO_OVERFLOW
     //@CTK NO_BUF_OVERFLOW
    function burnFrom(address account, uint256 amount) public {
        _burnFrom(account, amount);
    }
}
```

File GSN/Context.sol

```
pragma solidity ^0.5.13;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    //@CTK NO_ASF
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks
    //@CTK NO_ASF
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://
            github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}
```

File drafts/Counters.sol

```
pragma solidity ^0.5.13;

import "../math/SafeMath.sol";

/**
 * @title Counters
 * @author Matt Condon (@shrugs)
 * @dev Provides counters that can only be incremented or decremented by one. This can be
     used e.g. to track the number
 * of elements in a mapping, issuing ERC721 ids, or counting request ids.
 *
 * Include with `using Counters for Counters.Counter;`
 * Since it is not possible to overflow a 256 bit integer with increments of one, `
     increment` can skip the {SafeMath}
 * overflow check, thereby saving gas. This does assume however correct usage, in that the
     underlying `_value` is never
 * directly accessed.
 */
library Counters {
    using SafeMath for uint256;

    struct Counter {
        // This variable should never be directly accessed by users of the library:
            interactions must be restricted to
        // the library's function. As of Solidity v0.5.2, this cannot be enforced, though
            there is a proposal to add
        // this feature: see https://github.com/ethereum/solidity/issues/4637
        uint256 _value; // default: 0
    }
    //@CTK NO_ASF
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }
    //@CTK NO_ASF
    //@CTK NO_BUF_OVERFLOW
    function increment(Counter storage counter) internal {
        // The {SafeMath} overflow check can be skipped here, see the comment at the top
        counter._value += 1;
    }
    function decrement(Counter storage counter) internal {
        counter._value = counter._value.sub(1);
    }
}
```

File utils/Ownable.sol

```
pragma solidity ^0.5.13;

contract Ownable {
    address public owner;
    //@CTK NO_ASF
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    constructor() public {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require (msg.sender == owner, "only Onwer");
        _;
    }
    //@CTK NO_ASF
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    function transferOwnership(address newOwner) public onlyOwner {
        if (newOwner != address(0))
            owner = newOwner;
    }
}
```

File utils/Address.sol

```
pragma solidity ^0.5.13;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * IMPORTANT: It is unsafe to assume that an address for which this
     * function returns false is an externally-owned account (EOA) and not a
     * contract.
     */
    /*@CTK isContract
      @post !__reverted -> __return == (account != msg.sender)
    */
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    function isContract(address account) internal view returns (bool) {
        return (account != msg.sender);
        /*
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
            returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0
            xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
        */
    }
}
```

File utils/BytesLib.sol

```
/*
 * @title Solidity Bytes Arrays Utils
 * @author Goncalo Sa <goncalo.sa@consensys.net>
 *
 * @dev Bytes tightly packed arrays utility library for ethereum contracts written in
     Solidity.
 *      The library lets you concatenate, slice and type cast bytes arrays both in memory
     and storage.
 */

pragma solidity ^0.5.13;


library BytesLib {
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    function concat(
        bytes memory _preBytes,
        bytes memory _postBytes
    )
        internal
        pure
        returns (bytes memory)
    {
        bytes memory tempBytes;

        assembly {
            // Get a location of some free memory and store it in tempBytes as
            // Solidity does for memory variables.
            tempBytes := mload(0x40)

            // Store the length of the first bytes array at the beginning of
            // the memory for tempBytes.
            let length := mload(_preBytes)
            mstore(tempBytes, length)

            // Maintain a memory counter for the current write location in the
            // temp bytes array by adding the 32 bytes for the array length to
            // the starting location.
            let mc := add(tempBytes, 0x20)
            // Stop copying when the memory counter reaches the length of the
            // first bytes array.
            let end := add(mc, length)

            for {
                // Initialize a copy counter to the start of the _preBytes data,
                // 32 bytes into its memory.
                let cc := add(_preBytes, 0x20)
            } lt(mc, end) {
                // Increase both counters by 32 bytes each iteration.
                mc := add(mc, 0x20)
                cc := add(cc, 0x20)
            } {
                // Write the _preBytes data into the tempBytes memory 32 bytes
                // at a time.
                mstore(mc, mload(cc))
            }

            // Add the length of _postBytes to the current length of tempBytes
            // and store it as the new length in the first 32 bytes of the
            // tempBytes memory.
            length := mload(_postBytes)
            mstore(tempBytes, add(length, mload(tempBytes)))

            // Move the memory counter back from a multiple of 0x20 to the
            // actual end of the _preBytes data.
            mc := end
            // Stop copying when the memory counter reaches the new combined
            // length of the arrays.
            end := add(mc, length)

            for {
                let cc := add(_postBytes, 0x20)
            } lt(mc, end) {
                mc := add(mc, 0x20)
                cc := add(cc, 0x20)
            } {
                mstore(mc, mload(cc))
            }

            // Update the free-memory pointer by padding our last write location
            // to 32 bytes: add 31 bytes to the end of tempBytes to move to the
            // next 32 byte block, then round down to the nearest multiple of
            // 32. If the sum of the length of the two arrays is zero then add
            // one before rounding down to leave a blank 32 bytes (the length block with 0).
            mstore(0x40, and(
              add(add(end, iszero(add(length, mload(_preBytes)))), 31),
                not(31) // Round down to the nearest 32 bytes.
            ))
        }

        return tempBytes;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    function concatStorage(bytes storage _preBytes, bytes memory _postBytes) internal {
        assembly {
            // Read the first 32 bytes of _preBytes storage, which is the length
            // of the array. (We don't need to use the offset into the slot
            // because arrays use the entire slot.)
            let fslot := sload(_preBytes_slot)
            // Arrays of 31 bytes or less have an even value in their slot,
            // while longer arrays have an odd value. The actual length is
            // the slot divided by two for odd values, and the lowest order
            // byte divided by two for even values.
            // If the slot is even, bitwise and the slot with 255 and divide by
            // two to get the length. If the slot is odd, bitwise and the slot
            // with -1 and divide by two.
            let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
            let mlength := mload(_postBytes)
            let newlength := add(slength, mlength)
            // slength can contain both the length and contents of the array
            // if length < 32 bytes so let's prepare for that
            // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-state-
                variables-in-storage
            switch add(lt(slength, 32), lt(newlength, 32))
            case 2 {
                // Since the new array still fits in the slot, we just need to
                // update the contents of the slot.
                // uint256(bytes_storage) = uint256(bytes_storage) + uint256(bytes_memory) +
                    new_length
                sstore(
                    _preBytes_slot,
                    // all the modifications to the slot are inside this
                    // next block
                    add(
                        // we can just add to the slot contents because the
                        // bytes we want to change are the LSBs
                        fslot,
                        add(
                            mul(
                                div(
                                    // load the bytes from memory
                                    mload(add(_postBytes, 0x20)),
                                    // zero all bytes to the right
                                    exp(0x100, sub(32, mlength))
                                ),
                                // and now shift left the number of bytes to
                                // leave space for the length in the slot
                                exp(0x100, sub(32, newlength))
                            ),
                            // increase length by the double of the memory
                            // bytes length
                            mul(mlength, 2)
                        )
                    )
                )
            }
            case 1 {
                // The stored value fits in the slot, but the combined value
                // will exceed it.
                // get the keccak hash to get the contents of the array
                mstore(0x0, _preBytes_slot)
                let sc := add(keccak256(0x0, 0x20), div(slength, 32))

                // save new length
                sstore(_preBytes_slot, add(mul(newlength, 2), 1))

                // The contents of the _postBytes array start 32 bytes into
                // the structure. Our first read should obtain the `submod`
                // bytes that can fit into the unused space in the last word
                // of the stored array. To get this, we read 32 bytes starting
                // from `submod`, so the data we read overlaps with the array
                // contents by `submod` bytes. Masking the lowest-order
                // `submod` bytes allows us to add that value directly to the
                // stored value.

                let submod := sub(32, slength)
                let mc := add(_postBytes, submod)
                let end := add(_postBytes, mlength)
                let mask := sub(exp(0x100, submod), 1)

                sstore(
                    sc,
                    add(
                        and(
                            fslot,
                            0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff00
                        ),
                        and(mload(mc), mask)
                    )
                )

                for {
                    mc := add(mc, 0x20)
                    sc := add(sc, 1)
                } lt(mc, end) {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } {
                    sstore(sc, mload(mc))
                }

                mask := exp(0x100, sub(mc, end))

                sstore(sc, mul(div(mload(mc), mask), mask))
            }
            default {
                // get the keccak hash to get the contents of the array
                mstore(0x0, _preBytes_slot)
                // Start copying to the last used word of the stored array.
                let sc := add(keccak256(0x0, 0x20), div(slength, 32))

                // save new length
                sstore(_preBytes_slot, add(mul(newlength, 2), 1))

                // Copy over the first `submod` bytes of the new data as in
                // case 1 above.
                let slengthmod := mod(slength, 32)
                let mlengthmod := mod(mlength, 32)
                let submod := sub(32, slengthmod)
                let mc := add(_postBytes, submod)
                let end := add(_postBytes, mlength)
                let mask := sub(exp(0x100, submod), 1)

                sstore(sc, add(sload(sc), and(mload(mc), mask)))

                for {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } lt(mc, end) {
                    sc := add(sc, 1)
                    mc := add(mc, 0x20)
                } {
                    sstore(sc, mload(mc))
                }

                mask := exp(0x100, sub(mc, end))

                sstore(sc, mul(div(mload(mc), mask), mask))
            }
        }
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function slice(
        bytes memory _bytes,
        uint _start,
        uint _length
    )
        internal
        pure
        returns (bytes memory)
    {
        require(_bytes.length >= (_start + _length), "_bytes.length >= (_start + _length)");

        bytes memory tempBytes;

        assembly {
            switch iszero(_length)
            case 0 {
                // Get a location of some free memory and store it in tempBytes as
                // Solidity does for memory variables.
                tempBytes := mload(0x40)

                // The first word of the slice result is potentially a partial
                // word read from the original array. To read it, we calculate
                // the length of that partial word and start copying that many
                // bytes into the array. The first word we copy will start with
                // data we don't care about, but the last `lengthmod` bytes will
                // land at the beginning of the contents of the new array. When
                // we're done copying, we overwrite the full first word with
                // the actual length of the slice.
                let lengthmod := and(_length, 31)

                // The multiplication in the next line is necessary
                // because when slicing multiples of 32 bytes (lengthmod == 0)
                // the following copy loop was copying the origin's length
                // and then ending prematurely not copying everything it should.
                let mc := add(add(tempBytes, lengthmod), mul(0x20, iszero(lengthmod)))
                let end := add(mc, _length)

                for {
                    // The multiplication in the next line has the same exact purpose
                    // as the one above.
                    let cc := add(add(add(_bytes, lengthmod), mul(0x20, iszero(lengthmod))),
                        _start)
                } lt(mc, end) {
                    mc := add(mc, 0x20)
                    cc := add(cc, 0x20)
                } {
                    mstore(mc, mload(cc))
                }

                mstore(tempBytes, _length)

                //update free-memory pointer
                //allocating the array padded to 32 bytes like the compiler does now
                mstore(0x40, and(add(mc, 31), not(31)))
            }
            //if we want a zero-length slice let's just return a zero-length array
            default {
                tempBytes := mload(0x40)

                mstore(0x40, add(tempBytes, 0x20))
            }
        }

        return tempBytes;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function toAddress(bytes memory _bytes, uint _start) internal pure returns (address) {
        require(_bytes.length >= (_start + 20), "_bytes.length >= (_start + 20)");
        address tempAddress;

        assembly {
            tempAddress := div(mload(add(add(_bytes, 0x20), _start)), 0
                x1000000000000000000000000)
        }

        return tempAddress;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function toUint8(bytes memory _bytes, uint _start) internal pure returns (uint8) {
        require(_bytes.length >= (_start + 1), "_bytes.length >= (_start + 1)");
        uint8 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x1), _start))
        }

        return tempUint;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function toUint16(bytes memory _bytes, uint _start) internal pure returns (uint16) {
        require(_bytes.length >= (_start + 2), "_bytes.length >= (_start + 2)");
        uint16 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x2), _start))
        }

        return tempUint;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function toUint32(bytes memory _bytes, uint _start) internal pure returns (uint32) {
        require(_bytes.length >= (_start + 4), "_bytes.length >= (_start + 4)");
        uint32 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x4), _start))
        }

        return tempUint;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function toUint64(bytes memory _bytes, uint _start) internal pure returns (uint64) {
        require(_bytes.length >= (_start + 8), "_bytes.length >= (_start + 8)");
        uint64 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x8), _start))
        }

        return tempUint;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function toUint96(bytes memory _bytes, uint _start) internal pure returns (uint96) {
        require(_bytes.length >= (_start + 12), "_bytes.length >= (_start + 12)");
        uint96 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0xc), _start))
        }

        return tempUint;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function toUint128(bytes memory _bytes, uint _start) internal pure returns (uint128) {
        require(_bytes.length >= (_start + 16), "_bytes.length >= (_start + 16)");
        uint128 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x10), _start))
        }

        return tempUint;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function toUint(bytes memory _bytes, uint _start) internal pure returns (uint256) {
        require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
        uint256 tempUint;

        assembly {
            tempUint := mload(add(add(_bytes, 0x20), _start))
        }

        return tempUint;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function toBytes32(bytes memory _bytes, uint _start) internal pure returns (bytes32) {
        require(_bytes.length >= (_start + 32), "_bytes.length >= (_start + 32)");
        bytes32 tempBytes32;

        assembly {
            tempBytes32 := mload(add(add(_bytes, 0x20), _start))
        }

        return tempBytes32;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    function equal(bytes memory _preBytes, bytes memory _postBytes) internal pure returns (
        bool) {
        bool success = true;

        assembly {
            let length := mload(_preBytes)

            // if lengths don't match the arrays are not equal
            switch eq(length, mload(_postBytes))
            case 1 {
                // cb is a circuit breaker in the for loop since there's
                //  no said feature for inline assembly loops
                // cb = 1 - don't breaker
                // cb = 0 - break
                let cb := 1

                let mc := add(_preBytes, 0x20)
                let end := add(mc, length)

                for {
                    let cc := add(_postBytes, 0x20)
                // the next line is the loop condition:
```

```
                // while(uint(mc < end) + cb == 2)
                } eq(add(lt(mc, end), cb), 2) {
                    mc := add(mc, 0x20)
                    cc := add(cc, 0x20)
                } {
                    // if any of these checks fails then arrays are not equal
                    if iszero(eq(mload(mc), mload(cc))) {
                        // unsuccess:
                        success := 0
                        cb := 0
                    }
                }
            }
            default {
                // unsuccess:
                success := 0
            }
        }

        return success;
    }
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_OVERFLOW
    //@CTK NO_ASF
    function equalStorage(
        bytes storage _preBytes,
        bytes memory _postBytes
    )
        internal
        view
        returns (bool)
    {
        bool success = true;

        assembly {
            // we know _preBytes_offset is 0
            let fslot := sload(_preBytes_slot)
            // Decode the length of the stored array like in concatStorage().
            let slength := div(and(fslot, sub(mul(0x100, iszero(and(fslot, 1))), 1)), 2)
            let mlength := mload(_postBytes)

            // if lengths don't match the arrays are not equal
            switch eq(slength, mlength)
            case 1 {
                // slength can contain both the length and contents of the array
                // if length < 32 bytes so let's prepare for that
                // v. http://solidity.readthedocs.io/en/latest/miscellaneous.html#layout-of-
                    state-variables-in-storage
                if iszero(iszero(slength)) {
                    switch lt(slength, 32)
                    case 1 {
                        // blank the last byte which is the length
                        fslot := mul(div(fslot, 0x100), 0x100)

                        if iszero(eq(fslot, mload(add(_postBytes, 0x20)))) {
                            // unsuccess:
                            success := 0
                        }
                    }
                    default {
                        // cb is a circuit breaker in the for loop since there's
                        //  no said feature for inline assembly loops
                        // cb = 1 - don't breaker
                        // cb = 0 - break
                        let cb := 1

                        // get the keccak hash to get the contents of the array
                        mstore(0x0, _preBytes_slot)
                        let sc := keccak256(0x0, 0x20)

                        let mc := add(_postBytes, 0x20)
                        let end := add(mc, mlength)

                        // the next line is the loop condition:
                        // while(uint(mc < end) + cb == 2)
                        for {} eq(add(lt(mc, end), cb), 2) {
                            sc := add(sc, 1)
                            mc := add(mc, 0x20)
                        } {
                            if iszero(eq(sload(sc), mload(mc))) {
                                // unsuccess:
                                success := 0
                                cb := 0
                            }
                        }
                    }
                }
            }
            default {
                // unsuccess:
                success := 0
            }
        }

        return success;
    }
  }
```

File math/SafeMath.sol

```
pragma solidity ^0.5.13;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
```

```
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
     /*@CTK "SafeMath add"
      @post (a + b < a || a + b < b) == __reverted
      @post !__reverted -> __return == a + b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message
         on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
     /*@CTK "SafeMath sub"
      @post (a < b) == __reverted
      @post !__reverted -> __return == a - b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
        uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
     /*@CTK "SafeMath mul"
      @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
      @post !__reverted -> __return == a * b
      @post !__reverted == !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom
         message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
     /*@CTK "SafeMath div"
      @post (b <= 0) == __reverted
      @post !__reverted -> __return == a / b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
    */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
        uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
         modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
         modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
     /*@CTK "SafeMath mod"
      @post (b == 0) == __reverted
      @post !__reverted -> __return == a % b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
    */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (
        uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}
```