Sapienza
Università di Roma

Department of Computer Science

# Fuzz Testing with Echidna

Security in Software Application

**Professors:**
Daniele Friolo
Edoardo Persichetti

**Students:**
Andrea Di Marco
dimarco.1835169@studenti.uniroma1.it

Academic Year 2023/2024

## Contents

## 1 Introduction

During the course of this assignment I was tasked with checking a set of properties on the `Taxpayer.sol` smart contract and subsequently fixing the contract to enforce said properties.

The tools used for developement are:

- **Remix IDE**: for writing and running Solidity code in safe environments, before deployment of the contracts.
- **Echidna**: fuzz testing tool made for Solidity smart contracts.

Fuzz testing allowed to define the requested properties as invariants that must remain true as Echidna stressed the system by calling the provided API with a umber of different input values. The source code can be found here.

## 2 API

The contract *Fuzzer*, inside `fuzz_testing.sol`, provided to Echidna contained a set of functions tailor made to stress the system and challenge the properties that needed to be checked.

### 2.1 Environment

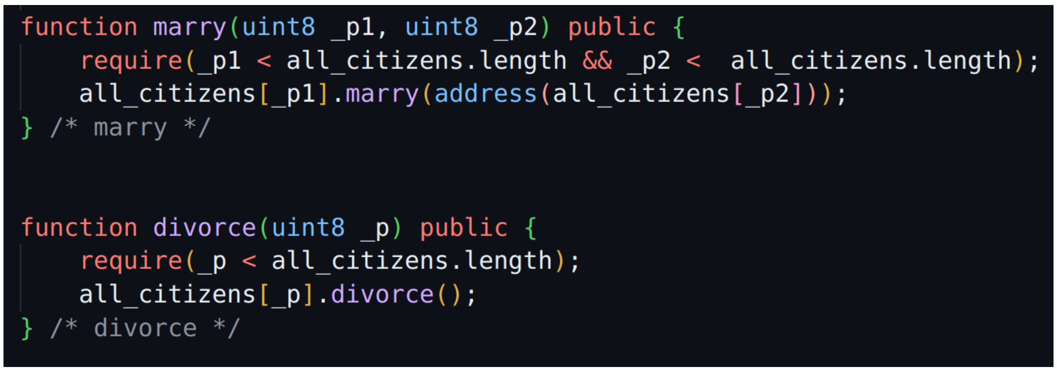
The environment generated for testing is a dynamic list of Taxpayer objects, called `all_citizens`. This was done since some properties might hold for a certain number of objects but then be violated as more objects interact.

The Taxpayer objects are then referenced by index by the other functions. The following tests have three Taxpayer objects interacting with each other.

### 2.2 Marriage & Divorce

The function $Marry(p_1, p_2)$ takes as input two indices and marries the person with index $p_1$ to the person with index $p_2$ using the Taxpayer's function for marriage.

The function $Divorce(p)$ takes as input one index and calls the function `divorce` for the person with index $p$.



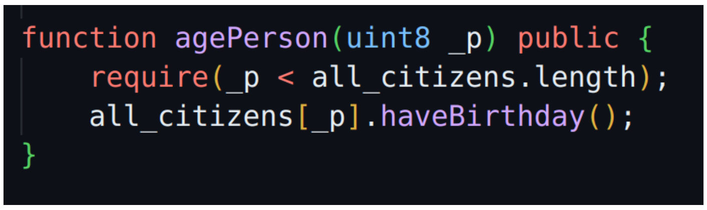
```
function marry(uint8 _p1, uint8 _p2) public {
    require(_p1 < all_citizens.length && _p2 <  all_citizens.length);
    all_citizens[_p1].marry(address(all_citizens[_p2]));
} /* marry */


function divorce(uint8 _p) public {
    require(_p < all_citizens.length);
    all_citizens[_p].divorce();
} /* divorce */
```

Figure 1: Marry and Divorce functions for the API

### 2.3 Ageing

The $agePerson(p)$ function takes as input one index and calls the function `haveBirthday` for the Taxpayer object with index $p$.



```
function agePerson(uint8 _p) public {
    require(_p < all_citizens.length);
    all_citizens[_p].haveBirthday();
}
```

Figure 2: Ageing function for the API

## 3 Properties

For testing the properties with Echidna both the default mode and the assertion mode where used.

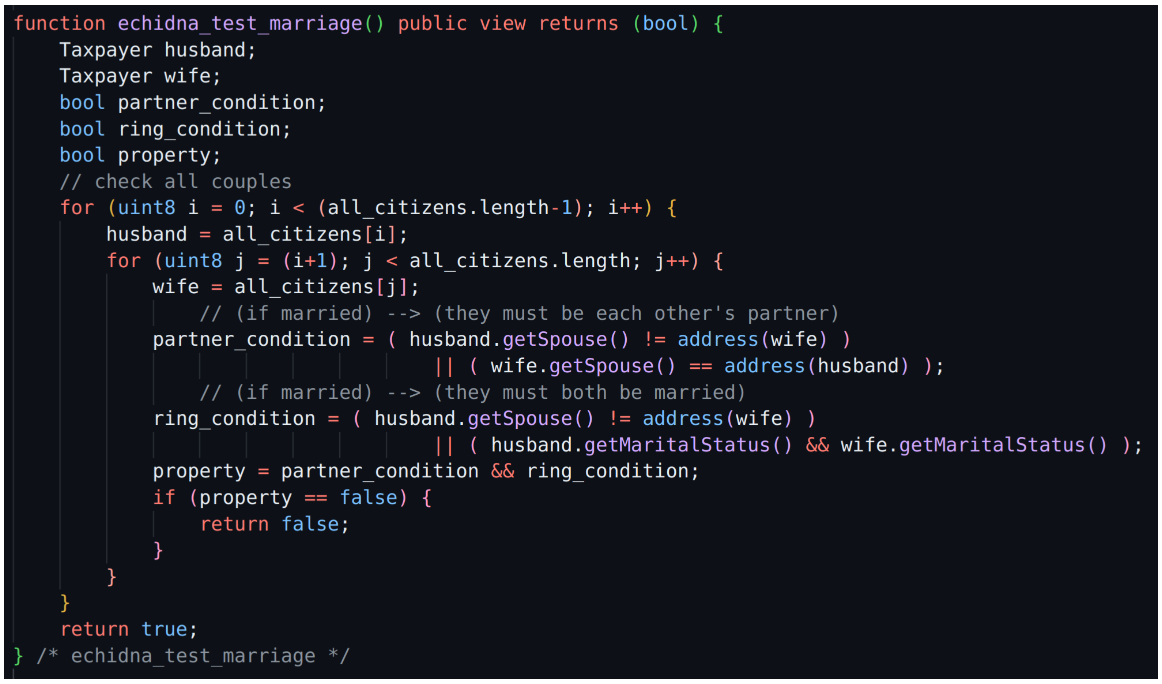
- `default`: The properties are expressed as Solidity functions with no arguments that return a boolean value.
  - `true`: the property has *not* been violated.
  - `false`: the property has been violated.
- `assertion`: The properties are expressed as Solidity functions with possible arguments. These functions contain assertion statements that must not be violated for the property to be satisfied.

### 3.1 Task 1: Marriage Property

The first property to check is the Marriage property stating that for every Taxpayer object $x$ married to another Taxpayer object $y$, then $y$ must also be married and to $x$.

This property is inspected by the `echidna_test_marriage` function, that scans every pair of Taxpayer objects and then checks two implications:

- `partner_condition`: tests if the `husband` and `wife` objects have each other as their partner.
- `ring_condition`: tests if they are both wearing the engagement ring, meaning they both have the `isMarried` parameter as `true`.



```
function echidna_test_marriage() public view returns (bool) {
    Taxpayer husband;
    Taxpayer wife;
    bool partner_condition;
    bool ring_condition;
    bool property;
    // check all couples
    for (uint8 i = 0; i < (all_citizens.length-1); i++) {
        husband = all_citizens[i];
        for (uint8 j = (i+1); j < all_citizens.length; j++) {
            wife = all_citizens[j];
                // (if married) --> (they must be each other's partner)
            partner_condition = ( husband.getSpouse() != address(wife) )
                                || ( wife.getSpouse() == address(husband) );
                // (if married) --> (they must both be married)
            ring_condition = ( husband.getSpouse() != address(wife) )
                                || ( husband.getMaritalStatus() && wife.getMaritalStatus() );
            property = partner_condition && ring_condition;
            if (property == false) {
                return false;
            }
        }
    }
    return true;
} /* echidna_test_marriage */
```

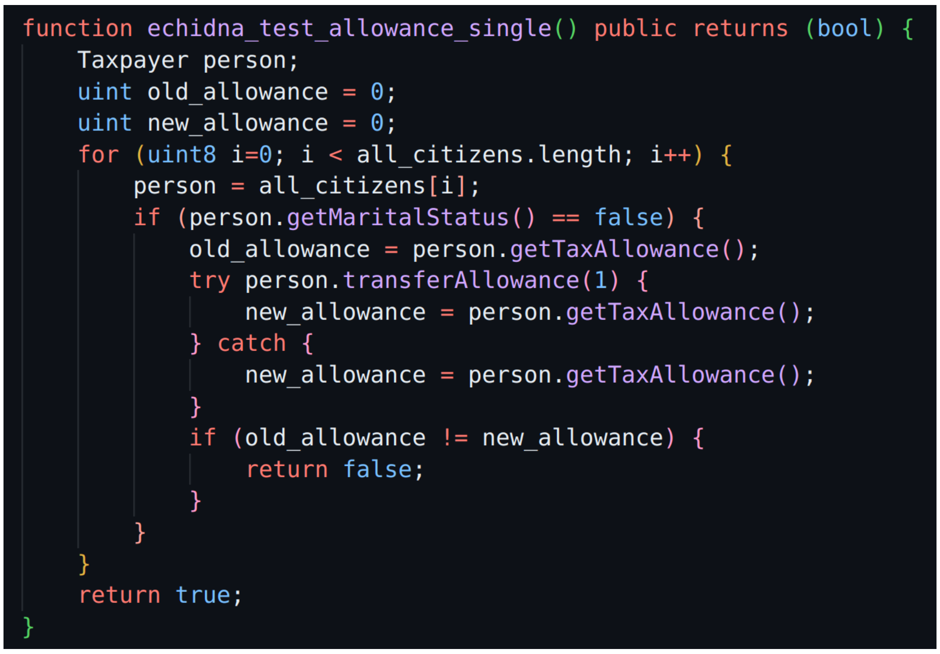
Figure 3: Solidity function for the marriage property

## 3.2 Task 2: Tax Allowance Property

The Tax Allowance property required more than one function to be properly modelled.

### 3.2.1 Allowance for Single Taxpayers

If a Taxpayer object is not married then it can't transfer its allowance to anyone. This functions checks all Taxpayer objects that are not married, stores their tax allowance and tries to call the `transferAllowance` function, if the value has changed the property has been violated. The try-catch statement is there to prevent Echidna from considering the reverted transaction of the `transferAllowance` function as a failed test, since aborting allowance transfers of single Taxpayers is the intended behavior.



```
function echidna_test_allowance_single() public returns (bool) {
    Taxpayer person;
    uint old_allowance = 0;
    uint new_allowance = 0;
    for (uint8 i=0; i < all_citizens.length; i++) {
        person = all_citizens[i];
        if (person.getMaritalStatus() == false) {
            old_allowance = person.getTaxAllowance();
            try person.transferAllowance(1) {
                new_allowance = person.getTaxAllowance();
            } catch {
                new_allowance = person.getTaxAllowance();
            }
            if (old_allowance != new_allowance) {
                return false;
            }
        }
    }
    return true;
}
```

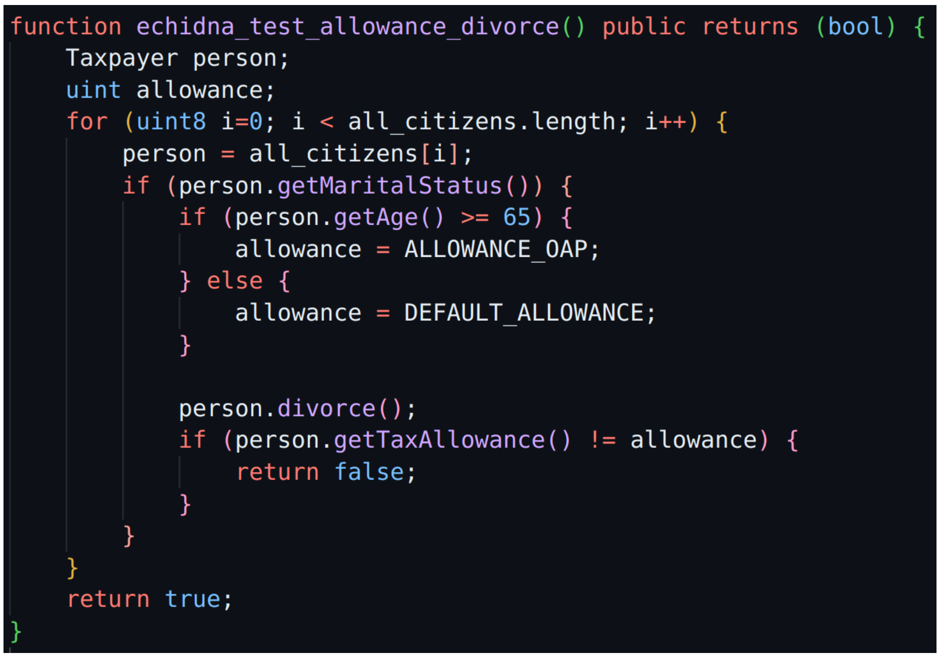
Figure 4: Solidity function for the Allowance of single Taxpayers

### 3.2.2 Allowance after Divorce

After a Taxpayer object divorces their tax allowance must return to the default amount which is:

- 5000 for Taxpayers younger than 65
- 7000 for Taxpayers older than 64

The function scans every Taxpayer object and, in an act of cruelty, forcefully divorces the married objects to then check the new tax allowance value.



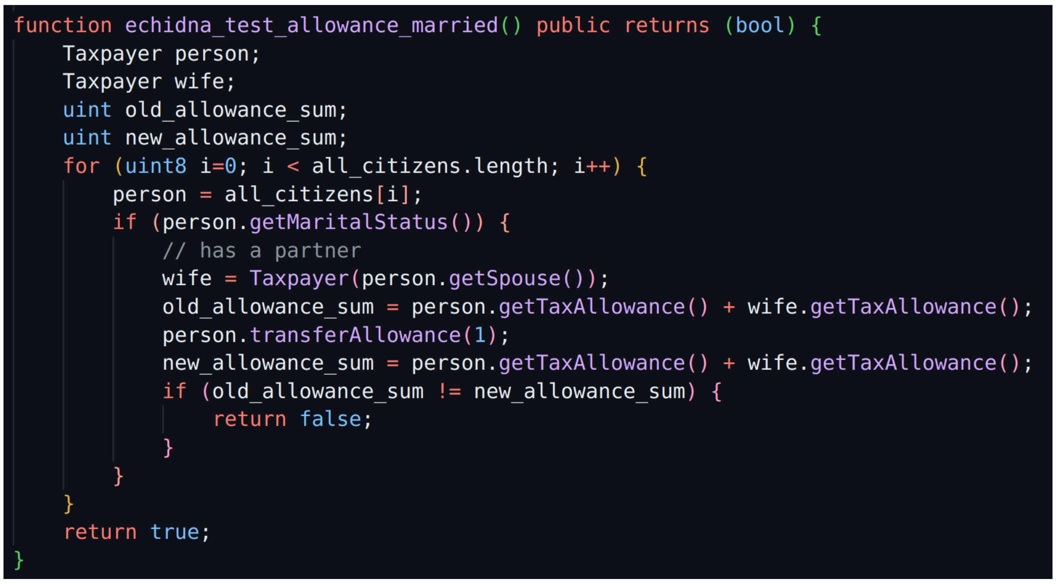
```
function echidna_test_allowance_divorce() public returns (bool) {
    Taxpayer person;
    uint allowance;
    for (uint8 i=0; i < all_citizens.length; i++) {
        person = all_citizens[i];
        if (person.getMaritalStatus()) {
            if (person.getAge() >= 65) {
                allowance = ALLOWANCE_OAP;
            } else {
                allowance = DEFAULT_ALLOWANCE;
            }

            person.divorce();
            if (person.getTaxAllowance() != allowance) {
                return false;
            }
        }
    }
    return true;
}
```

Figure 5: Solidity function for the Allowance of divorced Taxpayers

### 3.2.3 Allowance during Marriage

During marriage, two Taxpayer objects can transfer their tax allowance to their partners as long as the sum of both partner's tax allowance remains the same. The function checks every married citizen's tax allowance before and after transferring a unit of tax allowance to their partners.



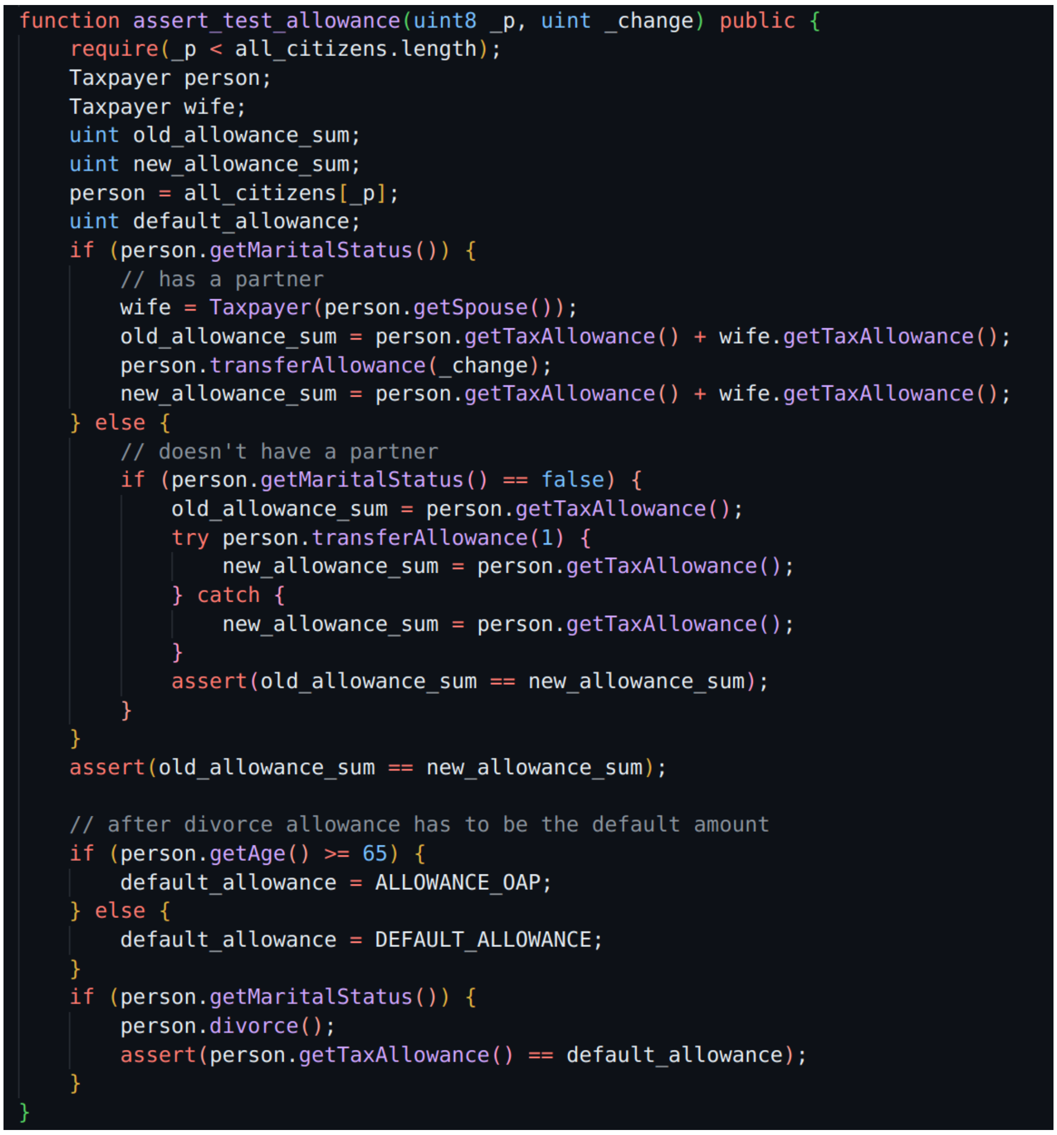
```
function echidna_test_allowance_married() public returns (bool) {
    Taxpayer person;
    Taxpayer wife;
    uint old_allowance_sum;
    uint new_allowance_sum;
    for (uint8 i=0; i < all_citizens.length; i++) {
        person = all_citizens[i];
        if (person.getMaritalStatus()) {
            // has a partner
            wife = Taxpayer(person.getSpouse());
            old_allowance_sum = person.getTaxAllowance() + wife.getTaxAllowance();
            person.transferAllowance(1);
            new_allowance_sum = person.getTaxAllowance() + wife.getTaxAllowance();
            if (old_allowance_sum != new_allowance_sum) {
                return false;
            }
        }
    }
    return true;
}
```

Figure 6: Solidity function for the Allowance of married Taxpayers

## 3.3 Assertion Mode

For good measure another function has been made containing all three allowance sub-properties but using the `assert` keyword instead, this allows the testing function to handle inputs by Echidna when executed in assertion mode.
The inputs are:

- `_p`: the Taxpayer object's index.
- `_change`: how many tax allowance units to transfer.



```
function assert_test_allowance(uint8 _p, uint _change) public {
    require(_p < all_citizens.length);
    Taxpayer person;
    Taxpayer wife;
    uint old_allowance_sum;
    uint new_allowance_sum;
    person = all_citizens[_p];
    uint default_allowance;
    if (person.getMaritalStatus()) {
        // has a partner
        wife = Taxpayer(person.getSpouse());
        old_allowance_sum = person.getTaxAllowance() + wife.getTaxAllowance();
        person.transferAllowance(_change);
        new_allowance_sum = person.getTaxAllowance() + wife.getTaxAllowance();
    } else {
        // doesn't have a partner
        if (person.getMaritalStatus() == false) {
            old_allowance_sum = person.getTaxAllowance();
            try person.transferAllowance(1) {
                new_allowance_sum = person.getTaxAllowance();
            } catch {
                new_allowance_sum = person.getTaxAllowance();
            }
            assert(old_allowance_sum == new_allowance_sum);
        }
    }
    assert(old_allowance_sum == new_allowance_sum);

    // after divorce allowance has to be the default amount
    if (person.getAge() >= 65) {
        default_allowance = ALLOWANCE_OAP;
    } else {
        default_allowance = DEFAULT_ALLOWANCE;
    }
    if (person.getMaritalStatus()) {
        person.divorce();
        assert(person.getTaxAllowance() == default_allowance);
    }
}
```

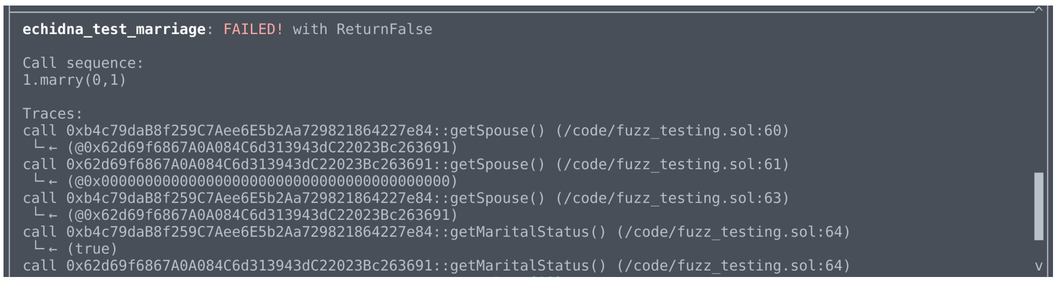
Figure 7: Allowance property for assertion mode.

## 4 Enforcing Properties

This section will discuss the fixes added to the `Taxpayer.sol` file in order to prevent violations of the properties. Said changes can be found in the `Taxpayer_fixed.sol` file.

### 4.1 Mutual Marriage

The original function for marriage only set the object's local variables without updating the object's partner's parameters.



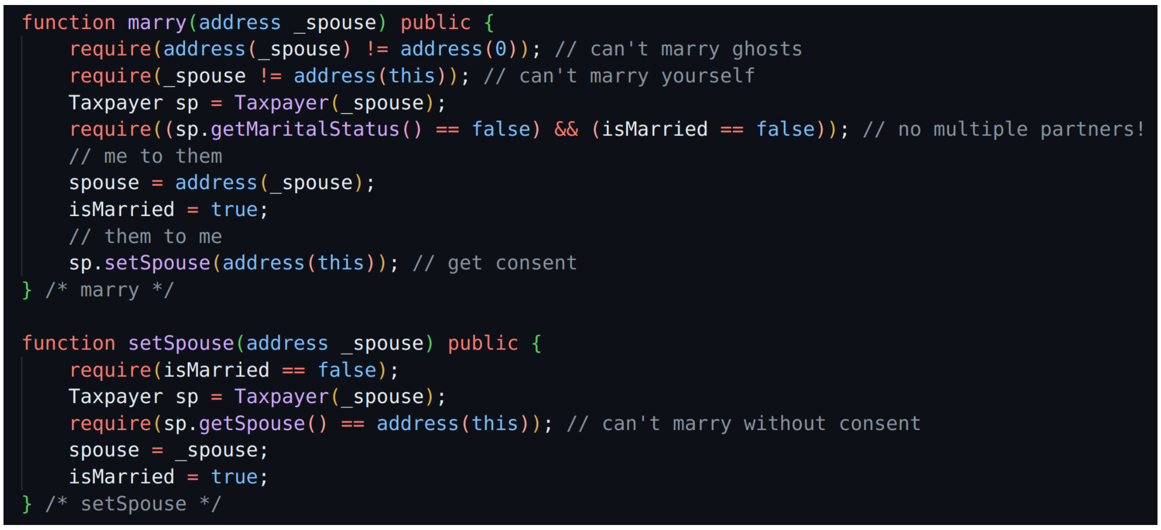
```
echidna_test_marriage: FAILED! with ReturnFalse

Call sequence:
1.marry(0,1)

Traces:
call 0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84::getSpouse() (/code/fuzz_testing.sol:60)
 └← (@0x62d69f6867A0A084C6d313943dC22023Bc263691)
call 0x62d69f6867A0A084C6d313943dC22023Bc263691::getSpouse() (/code/fuzz_testing.sol:61)
 └← (@0x0000000000000000000000000000000000000000)
call 0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84::getSpouse() (/code/fuzz_testing.sol:63)
 └← (@0x62d69f6867A0A084C6d313943dC22023Bc263691)
call 0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84::getMaritalStatus() (/code/fuzz_testing.sol:64)
 └← (true)
call 0x62d69f6867A0A084C6d313943dC22023Bc263691::getMaritalStatus() (/code/fuzz_testing.sol:64)
```

Figure 8: Marriage property is violated.

For this reason this property was not enforced, and tests failed. To enforce the property the function now also calls the partner's `setSpouse` function and adequate `require` conditions have been written to prevent the misuse of the `setSpouse` function as it will only be possible to call it from inside another object's `marry` function.



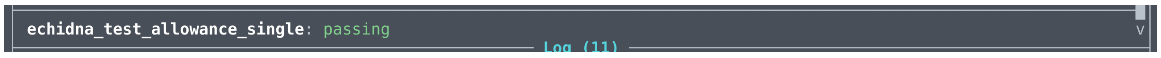
```
function marry(address _spouse) public {
    require(address(_spouse) != address(0)); // can't marry ghosts
    require(_spouse != address(this)); // can't marry yourself
    Taxpayer sp = Taxpayer(_spouse);
    require((sp.getMaritalStatus() == false) && (isMarried == false)); // no multiple partners!
    // me to them
    spouse = address(_spouse);
    isMarried = true;
    // them to me
    sp.setSpouse(address(this)); // get consent
} /* marry */

function setSpouse(address _spouse) public {
    require(isMarried == false);
    Taxpayer sp = Taxpayer(_spouse);
    require(sp.getSpouse() == address(this)); // can't marry without consent
    spouse = _spouse;
    isMarried = true;
} /* setSpouse */
```

Figure 9: Fixed `marry` and `setSpouse` functions

### 4.2 Allowance for Single Taxpayers

This property was already satisfied by the original script as the instruction "`Taxpayer sp = Taxpayer(address(spouse))`" reverted every call to the `transferAllowance` function made by single Taxpayer objects since the `spouse` attribute is set to `address(0)` and there is no Taxpayer object with such address.

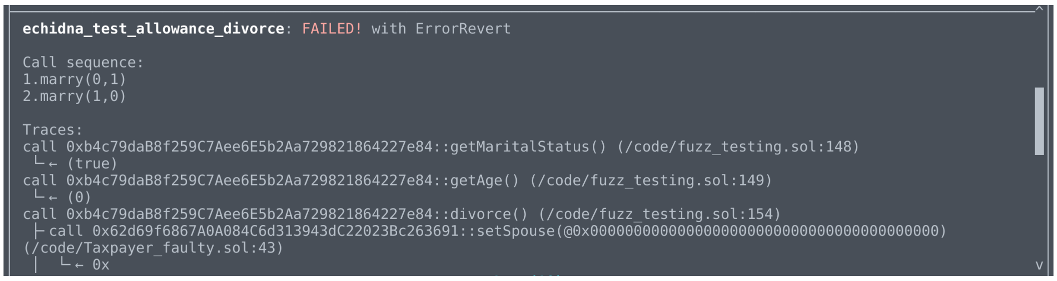


```
echidna_test_allowance_single: passing
                              Log (11)
```

Figure 10: Tax Allowance for Singles is satisfied!

## 4.3 Allowance after Divorce

In the original code allowance was not reset after divorce, this leads to the property being violated.



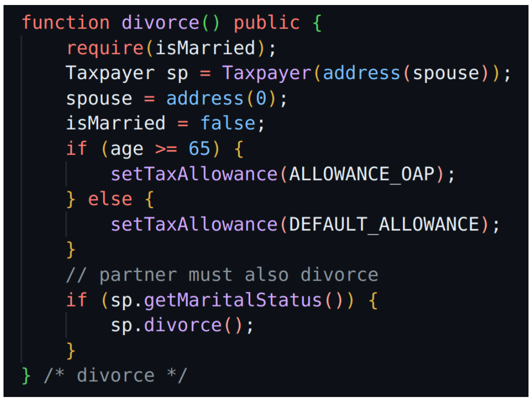
```
echidna_test_allowance_divorce: FAILED! with ErrorRevert

Call sequence:
1.marry(0,1)
2.marry(1,0)

Traces:
call 0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84::getMaritalStatus() (/code/fuzz_testing.sol:148)
 └← (true)
call 0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84::getAge() (/code/fuzz_testing.sol:149)
 └← (0)
call 0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84::divorce() (/code/fuzz_testing.sol:154)
 ├call 0x62d69f6867A0A084C6d313943dC22023Bc263691::setSpouse(@0x0000000000000000000000000000000000000000)
(/code/Taxpayer_faulty.sol:43)
 │  └← 0x
```

Figure 11: Tax Allowance after Divorce is violated

To enforce the property it is sufficient to reset the Taxpayer object's tax allowance after the divorce and then call again the `divorce` function for the ex-partner Taxpayer object. Adequate require conditions have been added to prevent endless loops.

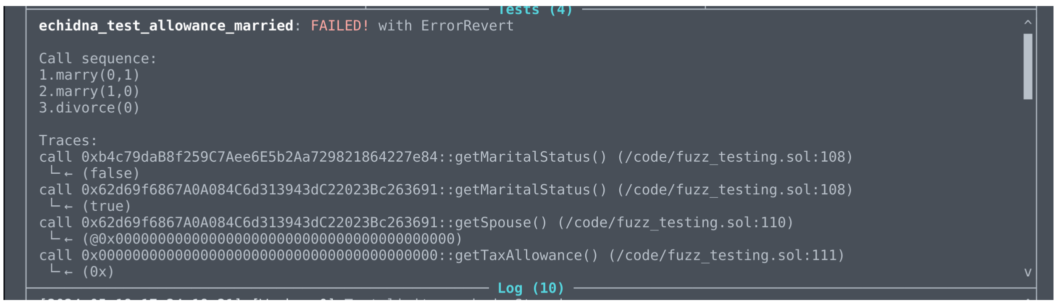


```
function divorce() public {
    require(isMarried);
    Taxpayer sp = Taxpayer(address(spouse));
    spouse = address(0);
    isMarried = false;
    if (age >= 65) {
        setTaxAllowance(ALLOWANCE_OAP);
    } else {
        setTaxAllowance(DEFAULT_ALLOWANCE);
    }
    // partner must also divorce
    if (sp.getMaritalStatus()) {
        sp.divorce();
    }
} /* divorce */
```

Figure 12: Fixed divorce function

## 4.4 Allowance for Married couples

The original code fails the tests.



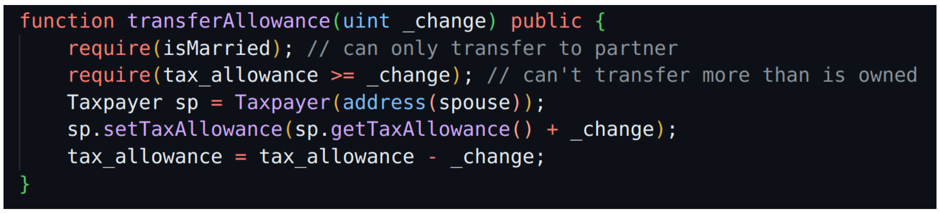
```
Tests (4)
echidna_test_allowance_married: FAILED! with ErrorRevert

Call sequence:
1.marry(0,1)
2.marry(1,0)
3.divorce(0)

Traces:
call 0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84::getMaritalStatus() (/code/fuzz_testing.sol:108)
 └← (false)
call 0x62d69f6867A0A084C6d313943dC22023Bc263691::getMaritalStatus() (/code/fuzz_testing.sol:108)
 └← (true)
call 0x62d69f6867A0A084C6d313943dC22023Bc263691::getSpouse() (/code/fuzz_testing.sol:110)
 └← (@0x0000000000000000000000000000000000000000)
call 0x0000000000000000000000000000000000000000::getTaxAllowance() (/code/fuzz_testing.sol:111)
 └← (0x)
Log (10)
```

Figure 13

Adding extra `require` statements to the `transferAllowance` function fixes the issue.

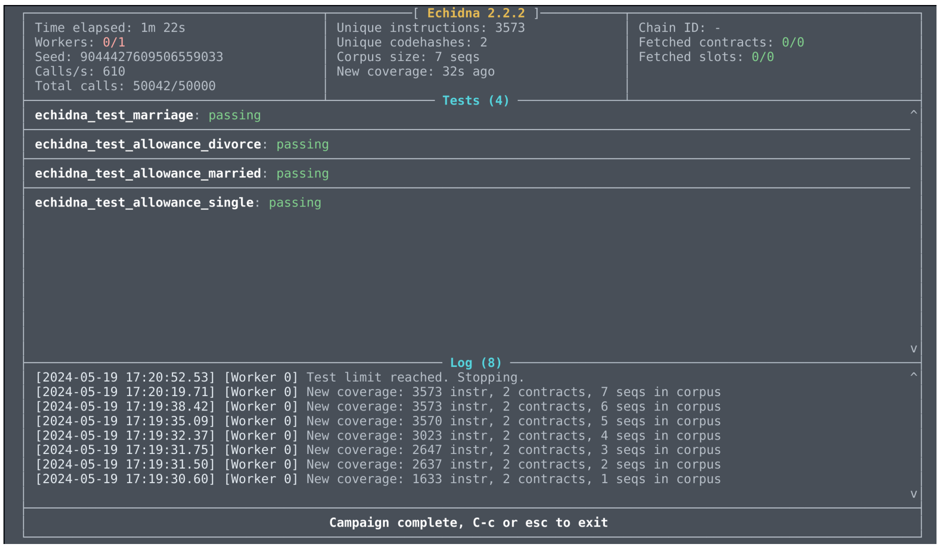


```
function transferAllowance(uint _change) public {
    require(isMarried); // can only transfer to partner
    require(tax_allowance >= _change); // can't transfer more than is owned
    Taxpayer sp = Taxpayer(address(spouse));
    sp.setTaxAllowance(sp.getTaxAllowance() + _change);
    tax_allowance = tax_allowance - _change;
}
```

Figure 14: Fixed Transfer Allowance function

## 5 Conclusions

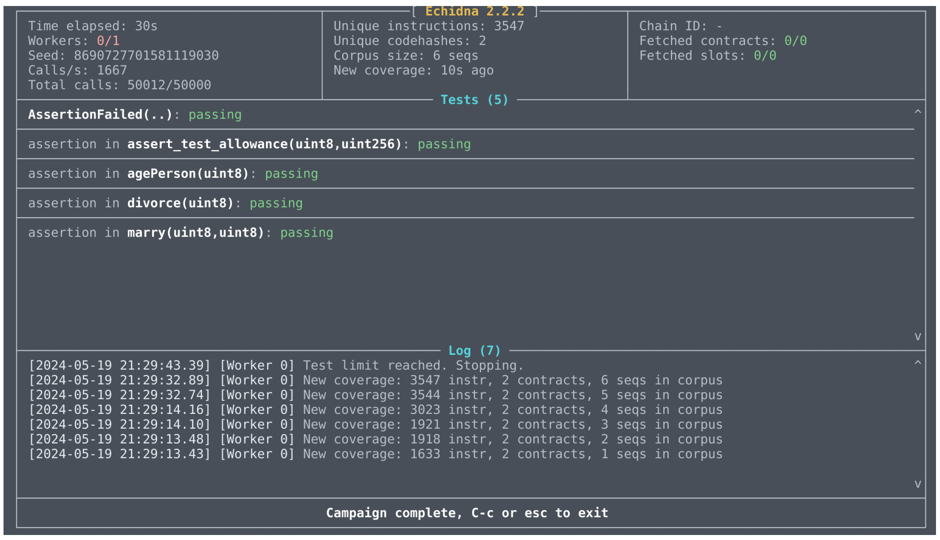
The proposed fixes to `Taxpayer.sol` correctly enforce all the mentioned properties. All fixes can be seen in the `Taxpayer_fixed.sol` file.



```
[ Echidna 2.2.2 ]
Time elapsed: 1m 22s          Unique instructions: 3573     Chain ID: -
Workers: 0/1                  Unique codehashes: 2          Fetched contracts: 0/0
Seed: 9044427609506559033     Corpus size: 7 seqs           Fetched slots: 0/0
Calls/s: 610                  New coverage: 32s ago
Total calls: 50042/50000
Tests (4)
echidna_test_marriage: passing
echidna_test_allowance_divorce: passing
echidna_test_allowance_married: passing
echidna_test_allowance_single: passing
Log (8)
[2024-05-19 17:20:52.53] [Worker 0] Test limit reached. Stopping.
[2024-05-19 17:20:19.71] [Worker 0] New coverage: 3573 instr, 2 contracts, 7 seqs in corpus
[2024-05-19 17:19:38.42] [Worker 0] New coverage: 3573 instr, 2 contracts, 6 seqs in corpus
[2024-05-19 17:19:35.09] [Worker 0] New coverage: 3570 instr, 2 contracts, 5 seqs in corpus
[2024-05-19 17:19:32.37] [Worker 0] New coverage: 3023 instr, 2 contracts, 4 seqs in corpus
[2024-05-19 17:19:31.75] [Worker 0] New coverage: 2647 instr, 2 contracts, 3 seqs in corpus
[2024-05-19 17:19:31.50] [Worker 0] New coverage: 2637 instr, 2 contracts, 2 seqs in corpus
[2024-05-19 17:19:30.60] [Worker 0] New coverage: 1633 instr, 2 contracts, 1 seqs in corpus
Campaign complete, C-c or esc to exit
```

Figure 15: All properties are satisfied!

Running tests in assertion mode also shows that the properties are satisfied.



```
[ Echidna 2.2.2 ]
Time elapsed: 30s                     Unique instructions: 3547        Chain ID: -
Workers: 0/1                          Unique codehashes: 2             Fetched contracts: 0/0
Seed: 8690727701581119030             Corpus size: 6 seqs              Fetched slots: 0/0
Calls/s: 1667                         New coverage: 10s ago
Total calls: 50012/50000
                                      Tests (5)
AssertionFailed(..): passing
assertion in assert_test_allowance(uint8,uint256): passing
assertion in agePerson(uint8): passing
assertion in divorce(uint8): passing
assertion in marry(uint8,uint8): passing
                                      Log (7)
[2024-05-19 21:29:43.39] [Worker 0] Test limit reached. Stopping.
[2024-05-19 21:29:32.89] [Worker 0] New coverage: 3547 instr, 2 contracts, 6 seqs in corpus
[2024-05-19 21:29:32.74] [Worker 0] New coverage: 3544 instr, 2 contracts, 5 seqs in corpus
[2024-05-19 21:29:14.16] [Worker 0] New coverage: 3023 instr, 2 contracts, 4 seqs in corpus
[2024-05-19 21:29:14.10] [Worker 0] New coverage: 1921 instr, 2 contracts, 3 seqs in corpus
[2024-05-19 21:29:13.48] [Worker 0] New coverage: 1918 instr, 2 contracts, 2 seqs in corpus
[2024-05-19 21:29:13.43] [Worker 0] New coverage: 1633 instr, 2 contracts, 1 seqs in corpus
                        Campaign complete, C-c or esc to exit
```

Figure 16: All properties are satisfied even in assertion mode