Case studies showing that Mpro does not work properly

Mythril:

a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts. It is a pure symbolic execution tool that executes all possible transaction sequences within a given depth.

https://github.com/ConsenSys/mythril

Mpro:

a tool built on top of Mythril to detect depth-n vulnerabilities. It prunes sequences based on data dependency.

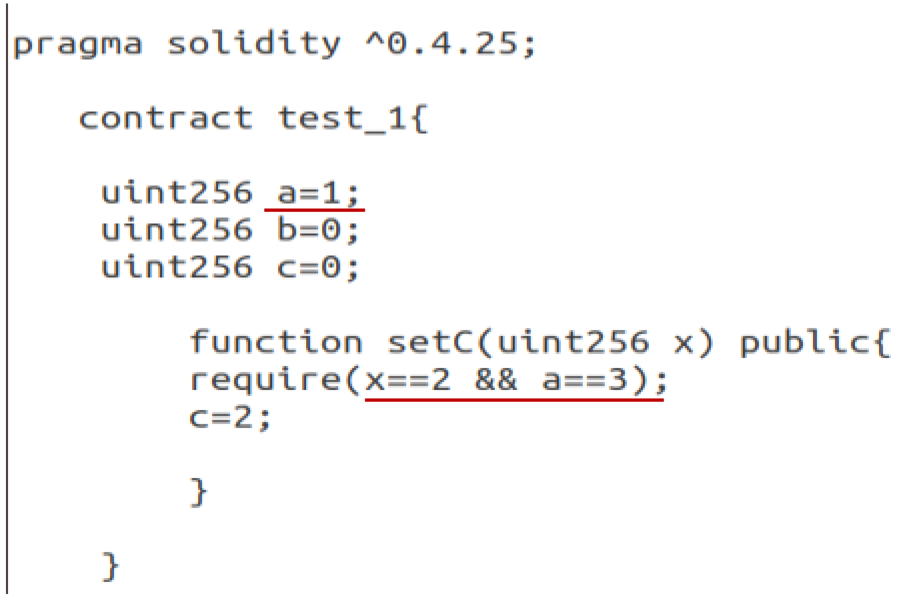
https://www.aminer.cn/pub/5e4a65023a55acdb0462ef3e/mpro-combining-static-and-symbolic-analysis-for-scalable-testing-of-smart-contr

https://github.com/QuanZhang-William/M-Pro

The case studies are to prove that:

a) Mpro generates a larger number of states than Mythril
b) Mpro executes infeasible paths
c) Mpro executes on infeasible states

Case study 1



```
pragma solidity ^0.4.25;

   contract test_1{

    uint256 a=1;
    uint256 b=0;
    uint256 c=0;

        function setC(uint256 x) public{
        require(x==2 && a==3);
        c=2;

        }

    }
```

Ran Mpro and Mythril on the test contract on the left by setting the depth limit to **1**, other parameters default values.

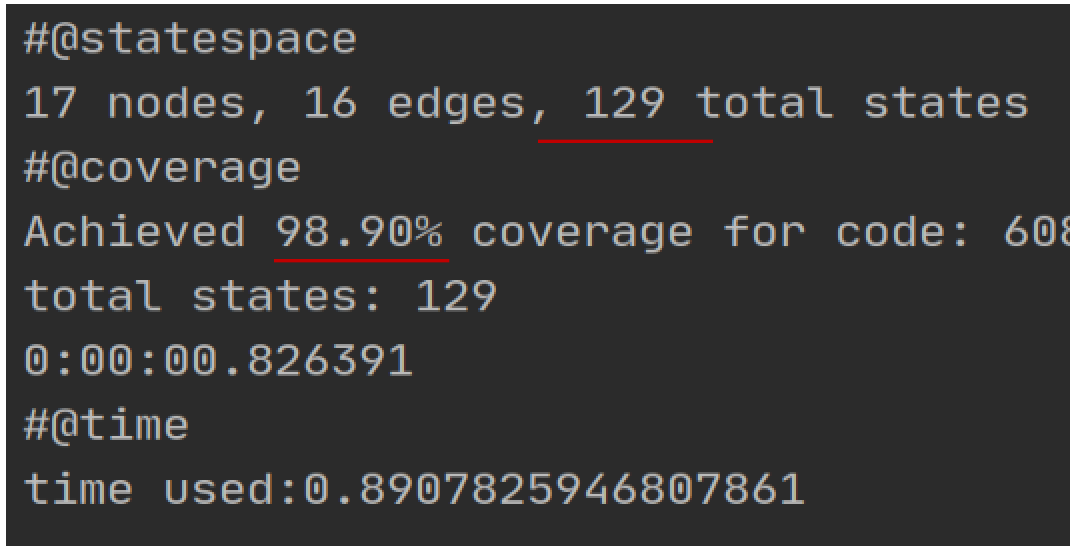
Results:

- Mpro generates 129 states whileas Mythril 118 states.
- Mpro has coverage 98.90% while Mythril only 86.81%.

This means that Mpro passes the require statement to execute c=2. However, the conditions in the require statement is false.

Mpro

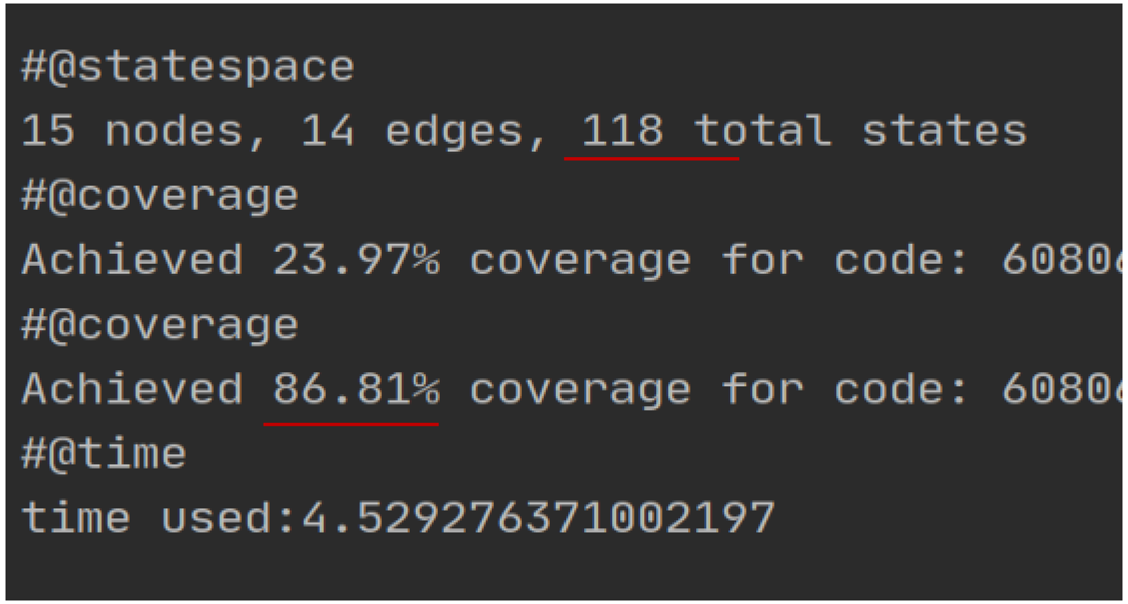
Mpro does not know that **a==3 is false**



```
#@statespace
17 nodes, 16 edges, 129 total states
#@coverage
Achieved 98.90% coverage for code: 608
total states: 129
0:00:00.826391
#@time
time used:0.8907825946807861
```

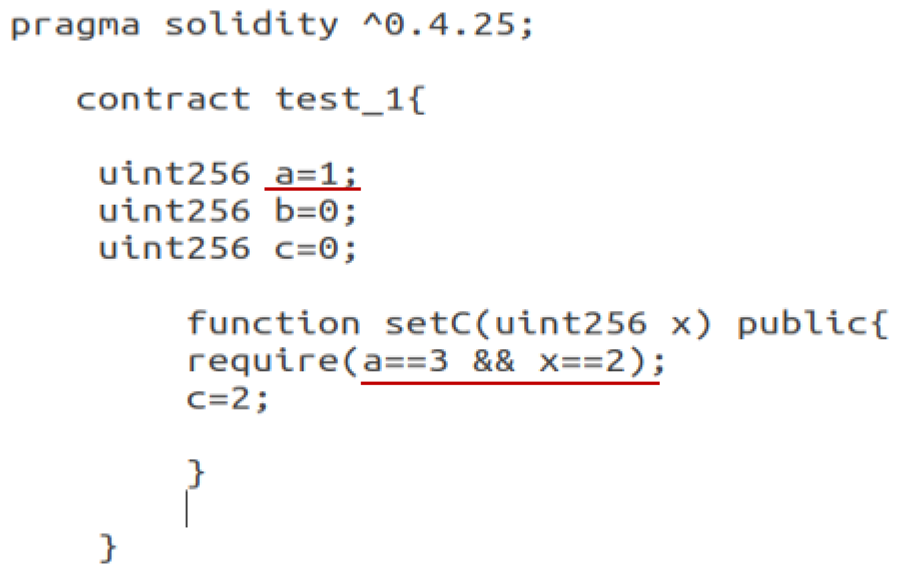
Mythril

Mythril knows that **a==3 is false**



```
#@statespace
15 nodes, 14 edges, 118 total states
#@coverage
Achieved 23.97% coverage for code: 6080
#@coverage
Achieved 86.81% coverage for code: 6080
#@time
time used:4.529276371002197
```

Case study 2



```
pragma solidity ^0.4.25;

    contract test_1{

     uint256 a=1;
     uint256 b=0;
     uint256 c=0;

         function setC(uint256 x) public{
         require(a==3 && x==2);
         c=2;

         }

     }
```

Change the order of the conditions in the require statement in function setC()

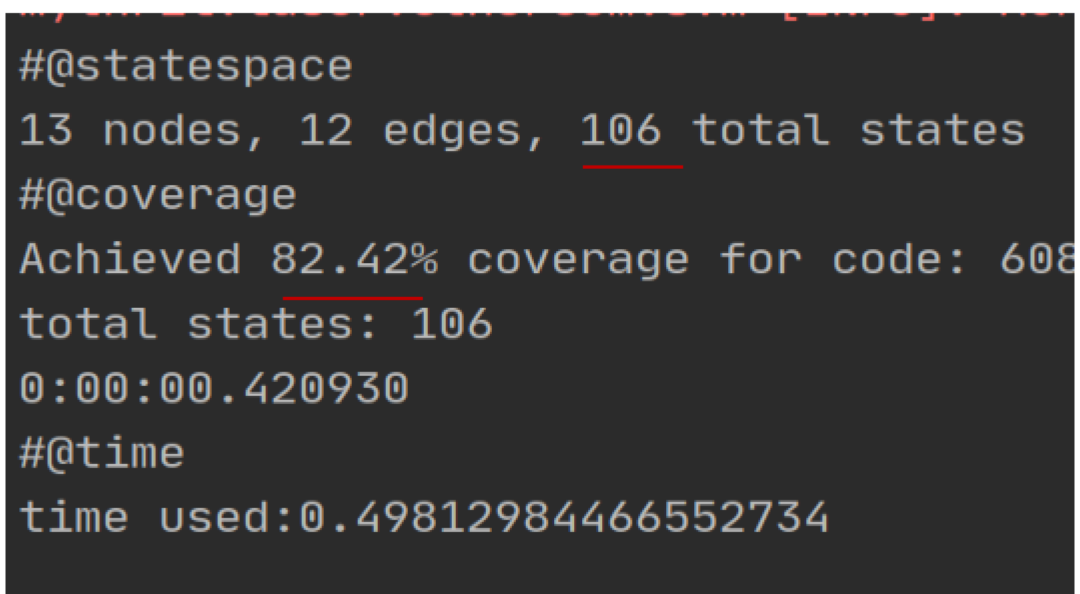
Ran Mpro and Mythril on the test contract on the left by setting the depth limit to **1**, other parameters default values.

Results:

both Mpro and Mythril generate the same number of states and have the same coverage. So, in this case, Mpro does know that the conditions in the requirement is false.

Mpro

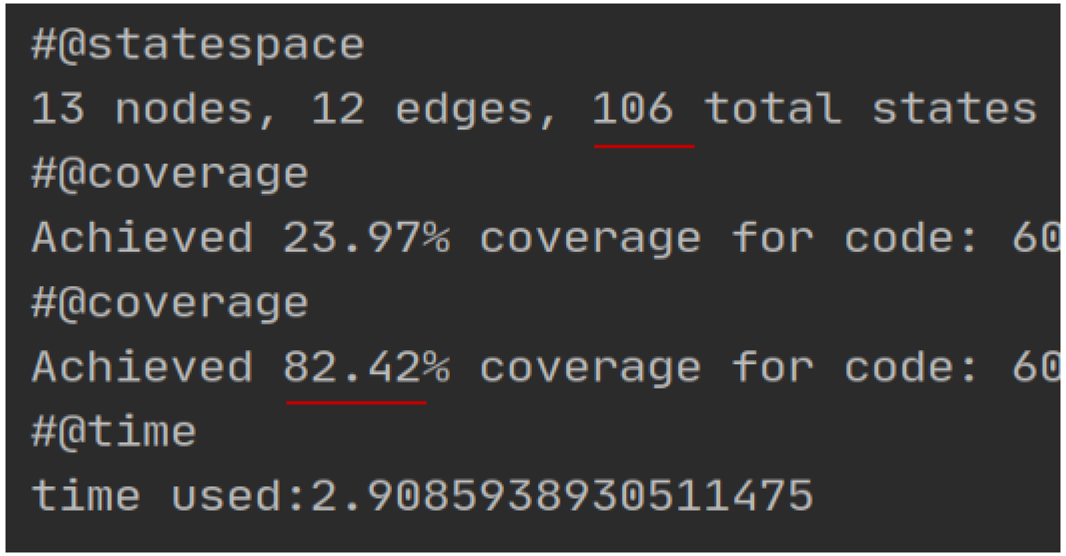
Mpro knows that **a==3 is false**



```
#@statespace
13 nodes, 12 edges, 106 total states
#@coverage
Achieved 82.42% coverage for code: 608
total states: 106
0:00:00.420930
#@time
time used:0.49812984466552734
```

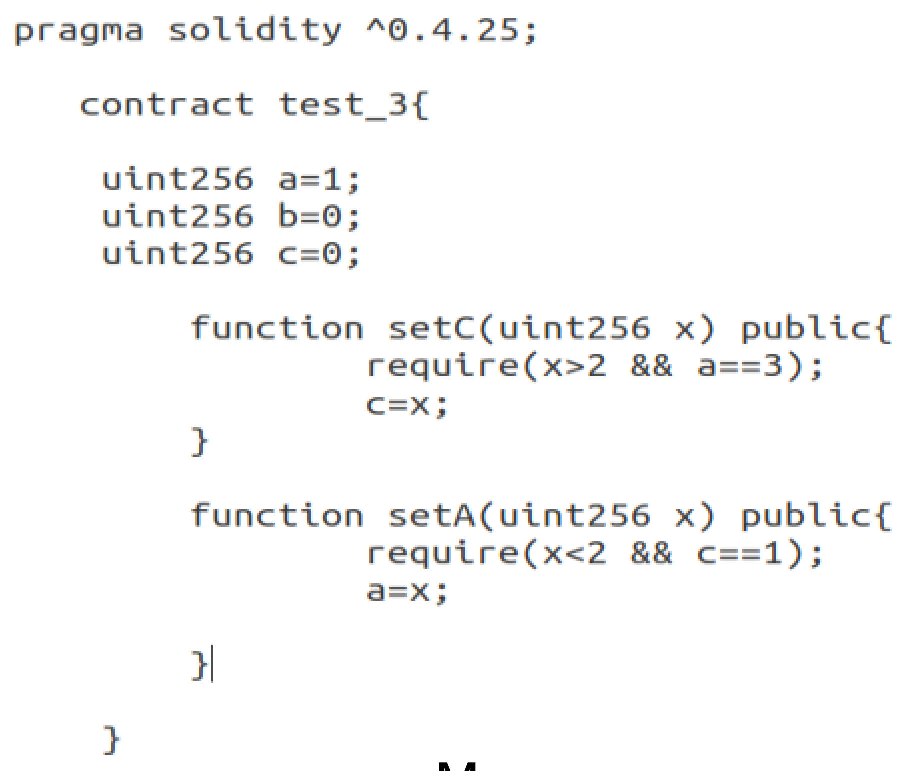
Mythril

Mythril knows that **a==3 is false**



```
#@statespace
13 nodes, 12 edges, 106 total states
#@coverage
Achieved 23.97% coverage for code: 60
#@coverage
Achieved 82.42% coverage for code: 60
#@time
time used:2.9085938930511475
```

Case study 3



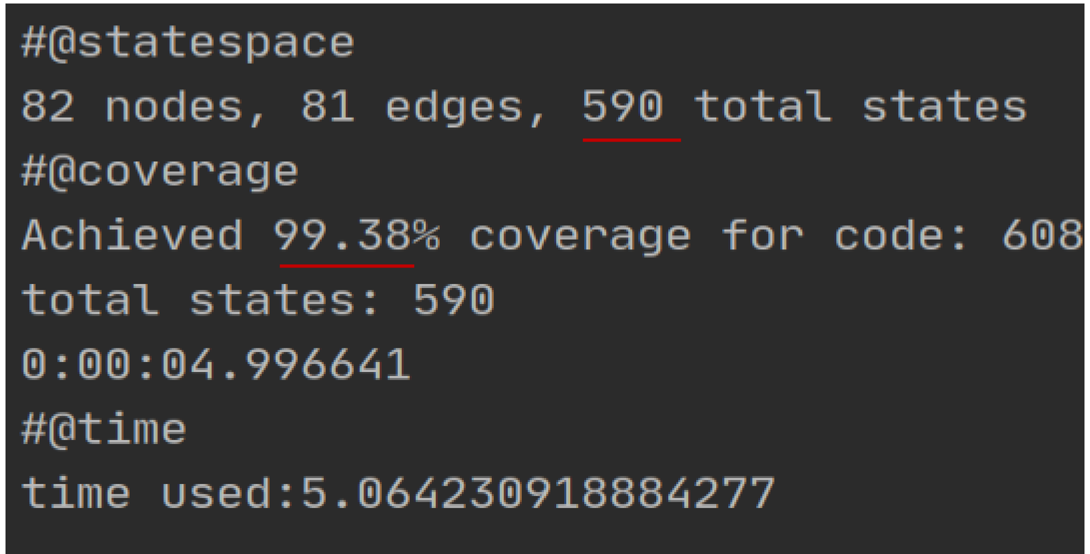
```
pragma solidity ^0.4.25;

   contract test_3{

    uint256 a=1;
    uint256 b=0;
    uint256 c=0;

        function setC(uint256 x) public{
                require(x>2 && a==3);
                c=x;
        }

        function setA(uint256 x) public{
                require(x<2 && c==1);
                a=x;

        }

    }
```

Ran Mpro and Mythril on the test contract on the left by setting the depth limit to **2**, other parameters default values.

Results:

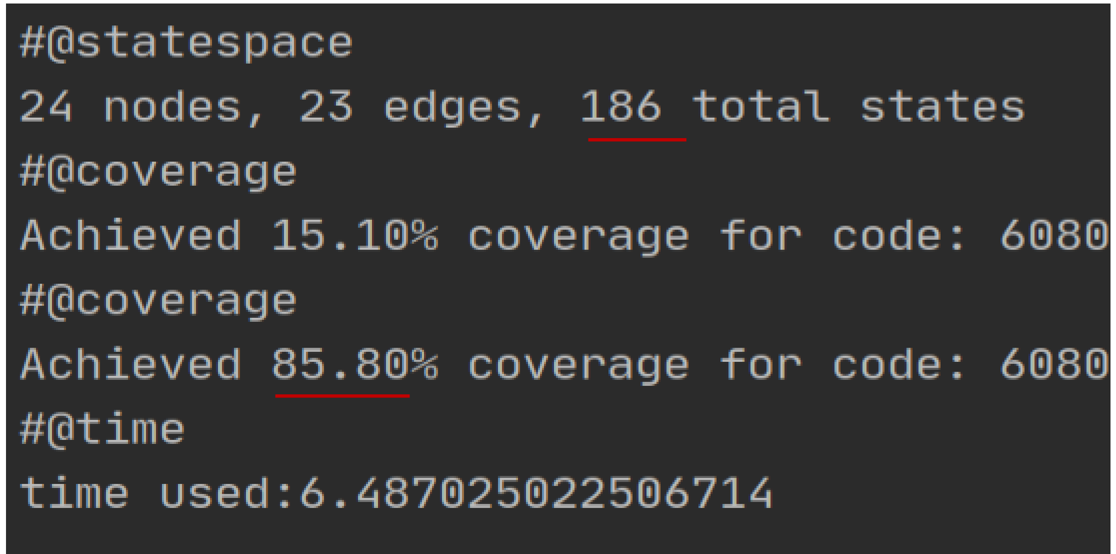
Mpro generates 590 states while Mythril 186 states.
Mpro gets 99.38% coverage while Mythril 85.80%.

Mpro



```
#@statespace
82 nodes, 81 edges, 590 total states
#@coverage
Achieved 99.38% coverage for code: 608
total states: 590
0:00:04.996641
#@time
time used:5.064230918884277
```

Mthril



```
#@statespace
24 nodes, 23 edges, 186 total states
#@coverage
Achieved 15.10% coverage for code: 6080
#@coverage
Achieved 85.80% coverage for code: 6080
#@time
time used:6.487025022506714
```

Case study 3

Mpro generates 590 states while Mythril only produces 186 states.

This is because Mpro executes sequences up to depth 2 whereas Mythril only executes sequences up to depth 1 (The execution traces of both tools are shown in the next slide.)

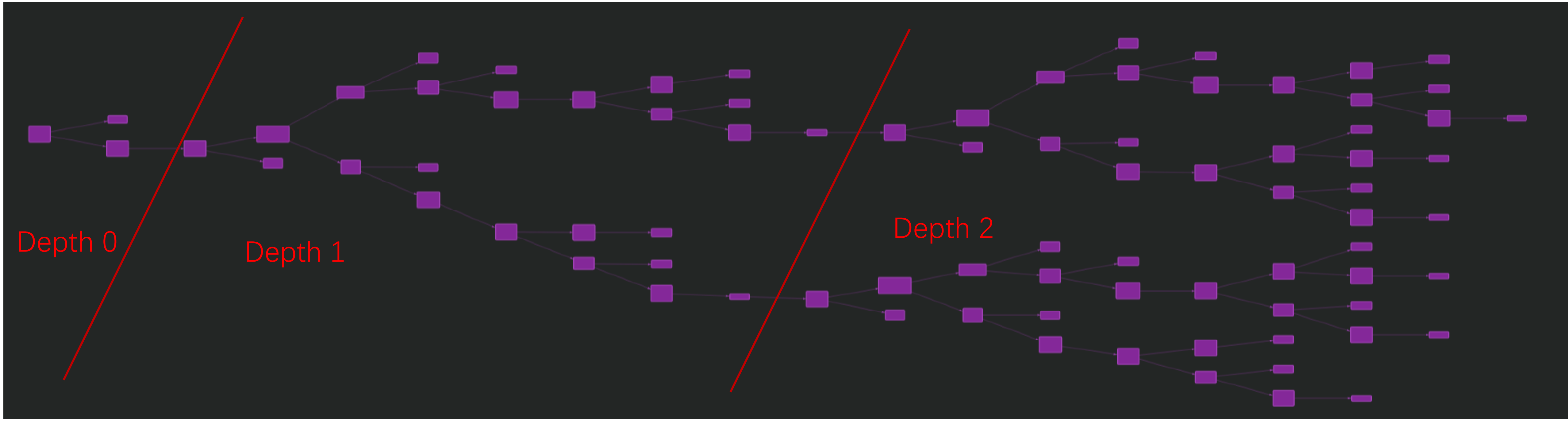
The reason why Mythril does not execute sequences up to depth 2 is that there are no feasible states generated at the end of depth 1. The execution of all paths reverts. And this can be verified as true because we can see that the conditions in both functions are false.

Mpro, anyhow, generates two states at the end of depth 1 (which are infeasible). Then Mpro continues to execute the next transaction on these infeasible states.

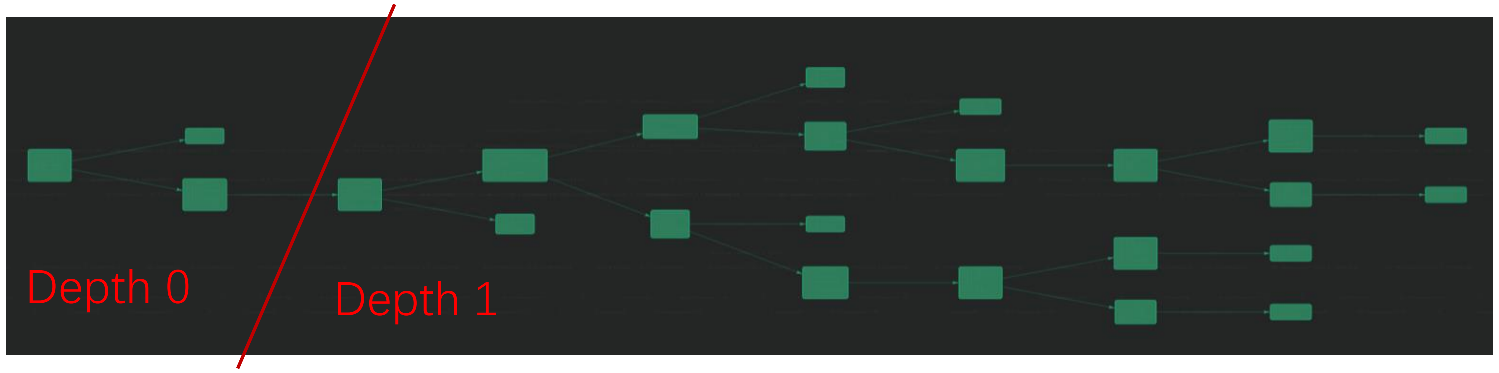
Mpro gets 99.38% coverage while Mythril 85.80%.

As Mpro executes all paths in the contract, so the coverage is 99.38%. Note that it is impossible to get 100% coverage as there is some code appended by compiler.

Execution trace of Mpro on contract test_3



Execution trace of Mythril on contract test_3



Discussion:

Mpro claims that it can prune sequences, so the number of states generated is supposed to be lower than that generated by Mythril. Nevertheless, case studies 1 and 3 show that this is not true.

Mpro executes infeasible paths. At depth 1, Mpro uses the original execution engine of Mythril. So, by setting the depth to 1, the performances of Mpro and Mythril are supposed to be exactly the same. However, case study 1 suggest that this is not true. But what is strange is that when changing the order of the conditions in the requirement, Mpro then has the same performance as Mythril (see case study 2). We think that Mpro does not apply the execution engine of Mythril as Mythril does.

Mpro executes on infeasible states. As shown in the third case study, Mpro generates two states at the end of depth 1, which are infeasible. But anyhow, Mpro does not check them, and then continue the execution on them in the next depth.

Based on the results of the above three case studies, we can say, Mpro does not work properly.