Analysis of the Solidity Compiler for Smart Contract Redundancy Detection

Jonas Gebele, January 11, 2020, Final Presentation Bachelor Thesis

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

Outline

1. Motivation and Background Information
2. Problem Statement
3. Research Questions
   3.1. What are the internal workings of the bytecode-optimizers in the Solidity compiler?
   3.2. How does enabling the optimization in the compiler-instruction modify the bytecode in general?
   3.3. How many bytecodes and therefore smart-contracts are redundant regarding their functionality due to different or missing optimization?
4. Conclusion and Future Work

Motivation and Background Information

Solidity Code
(sourceFile.sol)

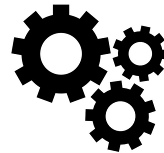
```
pragma solidity 0.5.8;

contract ExampleContract {
    uint256 number = 1;
}
```

```
0x6080604052600160005534801560145760
0080fd5b5060358060226000396000f3fe6080604052600080fdfea16562
7a7a723058204e048d6cab20eb0d9f95671510277b55a61a582250e04db7f658
7a1bebc134d20029
```

EVM (Deployment)
Bytecode

solc - Solidity Compiler



```
$ solc --optimize --bin sourceFile.sol
```

Motivation and Background Information

Contract creation transaction

```
> src =  web3.eth.accounts[0];
> ourContractDeploymentBytecode = "0x6080604052600160005534801560014…"
```

```
> web3.eth.sendTransaction ({
      from: src,
      data: ourContractDeploymentBytecode,
      gas: 113558,
      gasPrice: 200000000000
})
```

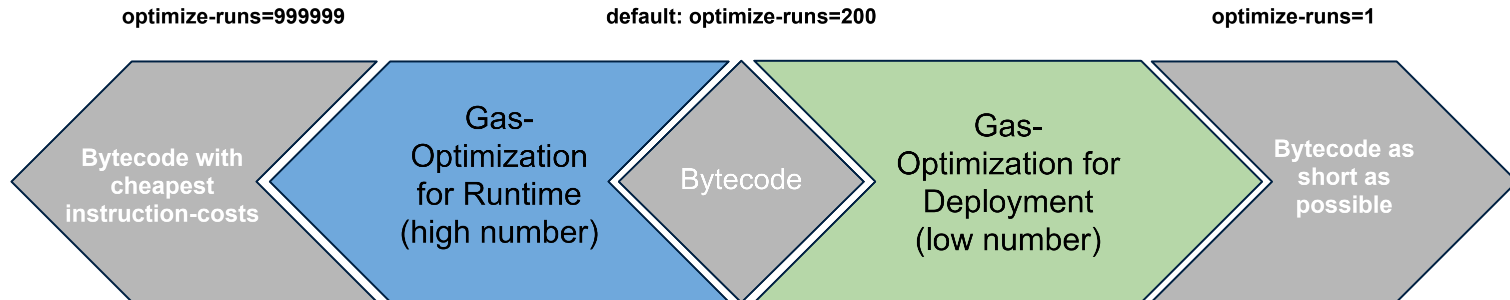
Deployment workflow of a smart contract

Motivation and Background Information

```
pragma solidity 0.5.8;

contract ExampleContract {
    uint256 number = 1;
}
```

```
0x6080604052600160005534801560145760
57600080fd5b506035806022600039 60
```



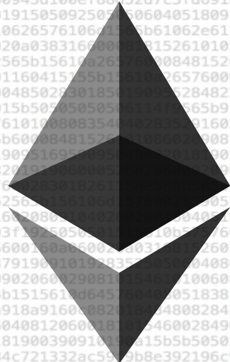
Outline

1. Motivation and Background Information
2. Problem Statement
3. Research Questions
   3.1. What are the internal workings of the bytecode-optimizers in the Solidity compiler?
   3.2. How does enabling the optimization in the compiler-instruction modify the bytecode in general?
   3.3. How many bytecodes and therefore smart-contracts are redundant regarding their functionality due to different or missing optimization?
4. Conclusion and Future Work

Problem Statement

Many studies in bytecode-analysis work with sets of unique smart contracts

Missing inclusion of the optimization process of the compiler



How many EVM bytecodes are redundant due to different or missing optimization?

Outline

1. Motivation and Background Information
2. Problem Statement
3. Research Questions
   3.1. What are the internal workings of the bytecode-optimizers in the Solidity compiler?
   3.2. How does enabling the optimization in the compiler-instruction modify the bytecode in general?
   3.3. How many bytecodes and therefore smart-contracts are redundant regarding their functionality due to different or missing optimization?
4. Conclusion and Future Work

Research Questions

R1: What are the internal workings of the bytecode-optimizers in the Solidity compiler?

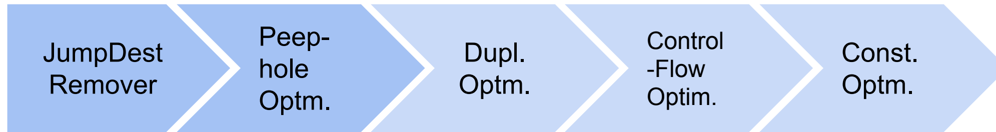
R2: How does enabling the optimization in the compiler-instruction modify the bytecode in general?

R3: How many bytecodes and therefore smart-contracts are redundant regarding their functionality due to different or missing optimization?

What are the internal workings of the bytecode-optimizers in the Solidity compiler?

Research question 1

RQ1.1 Of which sub-optimizers does the bytecode-optimizer of the Solidity compiler consist and what are their functionalities?



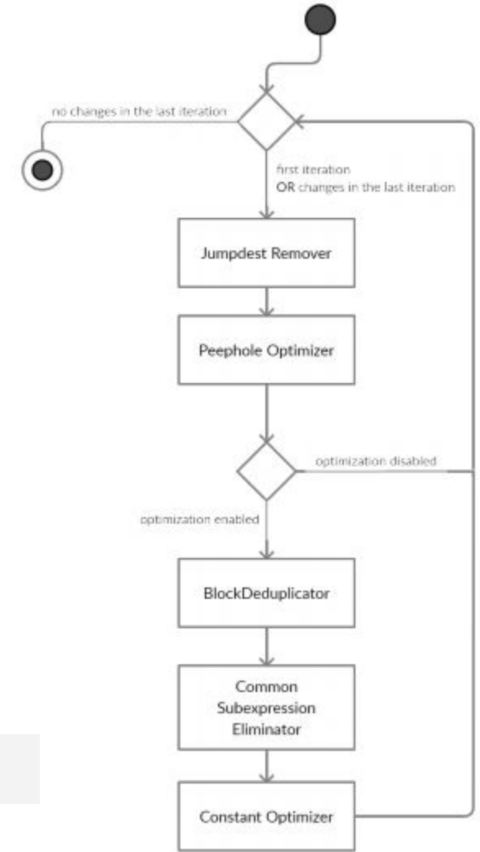
RQ1.2 How do the compilation-parameters affect the optimization-process of the compiler and the resulting bytecode?

```
pragma solidity 0.5.8;

contract ExampleContract {
    uint256 number = 1;
}
```

```
0x6080604052600160005534801560145
7600080fd5b50603580602260003960
00f3fe6080604052600080fdfea16562
7a7a723058204e048d6cab20eb0d9f95
671510277b55a61a582250e04db7f658
7a1bebc134d20029
```

```
$ solc --optimize-runs=200  --bin sourceFile.solv
```



How does enabling the optimization in the compiler-instruction modify the bytecode in general?

Research question 2

RQ2.1 Which bytecode-sections get optimized in what way of the compilation-process?

Deployment-Bytecode

```
608060405234801561001057600080fd5b506040516020806102178339810160409081529051 60
00818155338152600160205291909120556101d1806100466000396000f300
```

Runtime-Bytecode

```
6080604052600436106100565763ffffffff7c0100000000000000000000000000000000000000
00000000000000000060003504166318160ddd811461005b57806370a0823114610082578063a9
059cbb146100b0575b600080fd5b34801561006757600080fd5b506100706100f5565b60408051
918252519081900360200190f35b34801561008e57600080fd5b5061007073ffffffffffffffff
ffffffffffffffffffffffff600435166100fb565b3480156100bc57600080fd5b506100e173ff
ffffffffffffffffffffffffffffffffffffff6004351660243561012356 5b604080519115158 2
52519081900360200190f35b60005490565b73ffffffffffffffffffffffffffffffffffffffff
166000908152600160205260409020549056 5b600073ffffffffffffffffffffffffffffffffff
ffffff83161515610147 57600080fd5b33600090815260016020526040902054821115610163 57
600080fd5b503360009081526001602081905260408083208054859003905573ffffffffffffff
ffffffffffffffffffffffffff85168352909120805483019055929150505600a1
```

Meta-data Hash

```
65627a7a72305820a5d999f4459642872a29be93a490575d345e40fc91a7cccb2cf29c88bcdaf3
be0029
```

How does enabling the optimization in the compiler-instruction modify the bytecode in general?

Research question 2

RQ2.2 Which opcode-patterns and bytecode-methods can be simplified through setting optimization in the compiler-instruction?

```
CALLER
PUSH20
0xffffffffffffffffffffffffffffffffffffffff
AND
```

```
CALLER
PUSH1 0x01
PUSH1 0xA0
PUSH1 0x02
EXP
SUB
AND
```

Dynamic Calculations on the stack instead of hardcoding constants

```
608060405234801561001057600080fd5b506040516020806102178339810160409081529051 60
00818155338152600160205291909120556101d18061004660003960 00f300
```

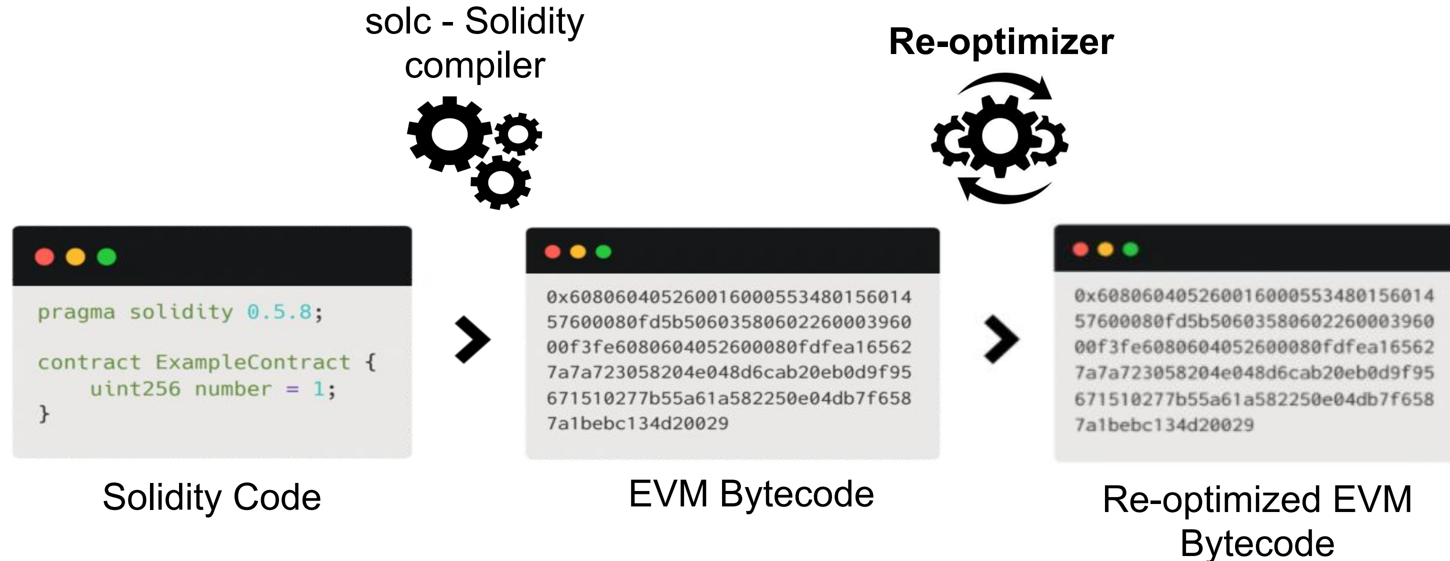
```
608060405260043610610056576 3ffffffff7c0100000000000000000000000000000000000000
000000000000000000006000350416631816 0ddd811461005b57806370a0823114610082578063a9
059cbb146100b0575b600080fd5b348015610067576 00080fd5b506100706100f5565b60408051
918252519081900360200190f35b3480156100 8e57600080fd5b5061007073ffffffffffffffff
ffffffffffffffffffffffff600435166100fb565b3480156100bc57600080fd5b506100e173ff
ffffffffffffffffffffffffffffffffffffff6004351660243561012356 5b6040805191151582
5251908190036020 0190f35b60005490565b73ffffffffffffffffffffffffffffffffffffffff
1660009081526001602052604090205490 565b600073ffffffffffffffffffffffffffffffffff
ffffff831615156101475760 0080fd5b33600090815260016020526040902054821115610163 57
600080fd5b503360009081526001602081905260408083208054859003905573ffffffffffffff
ffffffffffffffffffffffffff85168352909120805483019055929150505600a1
```

```
65627a7a7230582 0a5d999f4459642872a29be93a490575d345e40fc91a7cccb2cf29c88bcdaf3
be0029
```

How many bytecodes and therefore smart-contracts are redundant regarding their functionality due to different or missing optimization?

Research question 3

RQ3.1 What design could a re-optimizer have and what restrictions on the re-optimization are there?



How many bytecodes and therefore smart-contracts are redundant regarding their functionality due to different or missing optimization?

Research question 3

RQ3.1 What is a possible design of such a re-optimizer?

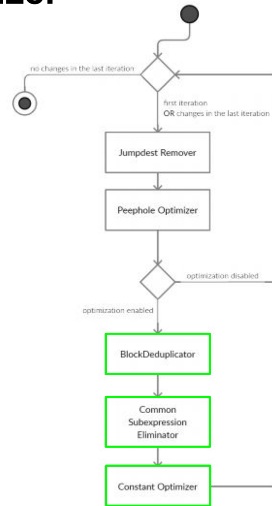
Re-optimizer

```
0x6080604052600160005534801560145
7600080fd5b50603580602260003960
00f3fe6080604052600080fdfea16562
7a7a723058204e048d6cab20eb0d9f95
671510277b55a61a582250e04db7f658
7a1bebc134d20029
```

EVM Bytecode

```
{
  "m_items": [
    0: {solidity::evmasm::AssemblyItem},
    1: {solidity::evmasm::AssemblyItem},
    2: {solidity::evmasm::AssemblyItem},
    3: {solidity::evmasm::AssemblyItem},
    4: {solidity::evmasm::AssemblyItem},
    5: {solidity::evmasm::AssemblyItem},
    ...
  ]
}
```

```
{
  "0": {
    m_modifierDepth = 0,
    m_type = solidity::evmasm::Push,
    m_instruction = -3,
    m_data = { m_data },
    m_location = { m_location },
    m_jumpType = solidity::evmasm::AssemblyItem::JumpType::Ordinary,
    ...
  }
}
```



```
0x6080604052600160005534801560145
7600080fd5b50603580602260003960
00f3fe6080604052600080fdfea16562
7a7a723058204e048d6cab20eb0d9f95
671510277b55a61a582250e04db7f658
7a1bebc134d20029
```

Re-optimized EVM Bytecode

How many bytecodes and therefore smart-contracts are redundant regarding their functionality due to different or missing optimization?

Research question 3

RQ3.2 Which restrictions does such an re-optimization have?

Technical restrictions

```
0x6080604052600160005534801560145
7600080fd5b5060358060226000396
00f3fe6080604052600080fdfea16562
7a7a723058204e048d6cab20eb0d9f95
671510277b55a61a582250e04db7f658
7a1bebc134d20029
```

EVM Bytecode

```
{
  "m_items": [
    0: {solidity::evmasm::AssemblyItem},
    1: {solidity::evmasm::AssemblyItem},
    2: {solidity::evmasm::AssemblyItem},
    3: {solidity::evmasm::AssemblyItem},
    4: {solidity::evmasm::AssemblyItem},
    5: {solidity::evmasm::AssemblyItem},
    ...
  ]
}
```

```
{
  "0": {
    m_modifierDepth = 0,
    m_type = solidity::evmasm::Push,
    m_instruction = -3,
    m_data = { m_data },
    m_location = { m_location },
    m_jumpType = solidity::evmasm::AssemblyItem::JumpType::Ordinary,
    ...
  }
}
```

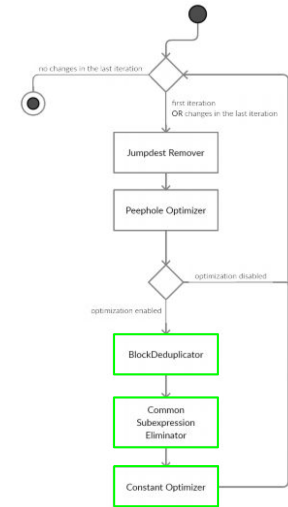
Conceptual restrictions

Yul Optimizer

Bugfixes and modifications in the Solidity compiler

Runtime bytecodes

Standard optimization



Outline

1. Motivation and Background Information
2. Problem Statement
3. Research Questions
    3.1. What are the internal workings of the bytecode-optimizers in the Solidity compiler?
    3.2. How does enabling the optimization in the compiler-instruction modify the bytecode in general?
    3.3. How many bytecodes and therefore smart-contracts are redundant regarding their functionality due to different or missing optimization?
4. Conclusion and Future Work

Conclusion and Future Work

Conclusion

First comprehensive description of the inner workings of the Solidity optimizer

Insights are already outdated with the introduction of the Yul-optimizer

- ➢ Analysis of deployed smart-contracts in the past

Compilation-parameters affect the optimization and the resulting bytecode

Possible design of an bytecode re-optimizer

Conclusion and Future Work

Future Work

Actual implementation of such an re-optimizer

- ➢ use the re-optimization in combination with statistical methods to determine the redundancy of smart-contracts deployed on Ethereum

Integration of the Yul-optimizer

Jonas Gebele

jonas.gebele@in.tum.de

Technische Universität München
Faculty of Informatics
Chair of Software Engineering for
Business Information Systems

Boltzmannstraße 3
85748 Garching bei München