



一种基于语言特性分析和交易 重放的智能合约测试方法

毕业论文答辩

姓名: 王子彦

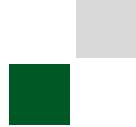
专业: 电子信息 (软件工程)

学号: 20215133





目录



1 / 研究背景

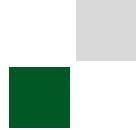
2 / Solidity语言特性实证研究

3 / 智能合约交易重放测试方法

4 / 总结与展望



目录



-
- 1 / 研究背景
 - 2 / Solidity语言特性实证研究
 - 3 / 智能合约交易重放测试方法
 - 4 / 总结与展望
-

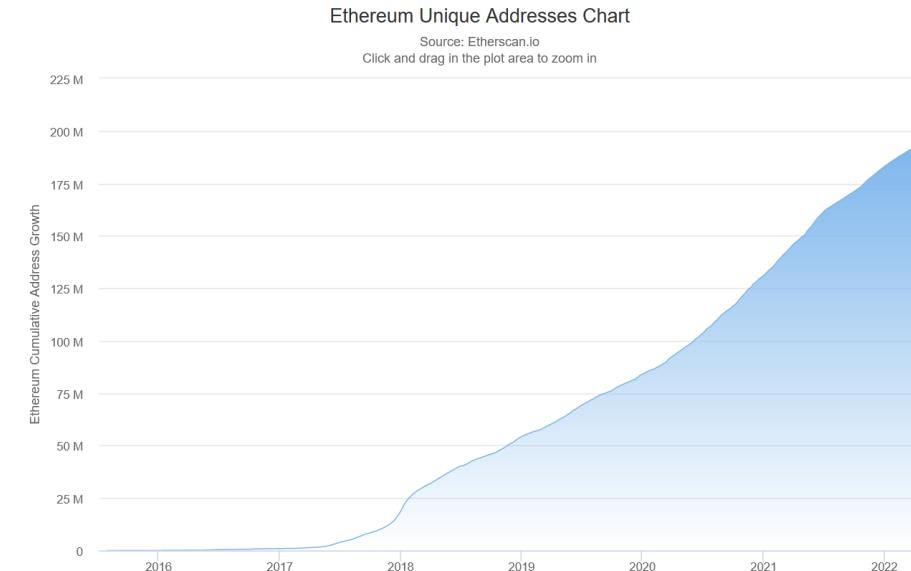


区块链市场快速发展

- 以太坊区块链上每日执行超过100万笔交易，已有接近2亿个地址
- 2021-2026年中国区块链市场规模年复合增速达73%¹
- 2026年中国市场规模将达163亿美元，未来20年有望达万亿级别¹



以太坊每日交易数量趋势图²



以太坊地址数量趋势图²

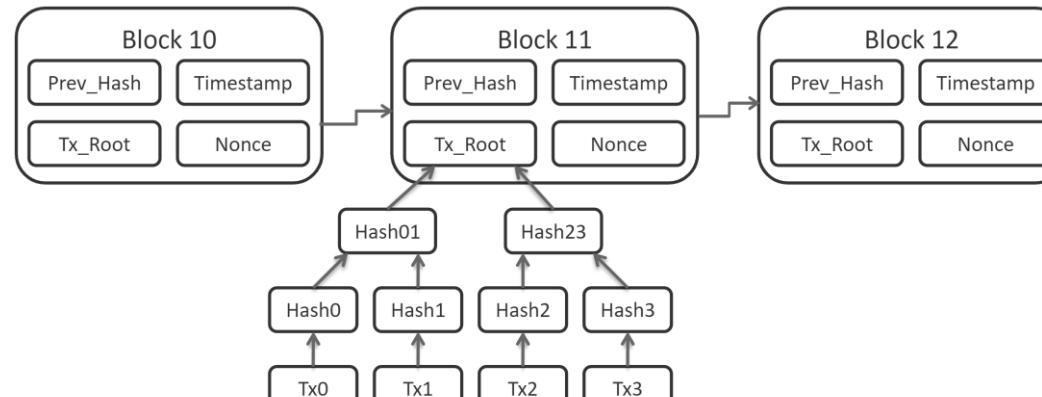
¹ "2021年中国区块链行业市场现状与发展前景分析区块链加速渗透各" Accessed May 8, 2022. <https://www.qianzhan.com/analyst/detail/220/210304-fc47dc0e.html>.

² "Etherscan." Accessed May 8, 2022. <https://etherscan.io>.



研究背景

- 区块链¹
 - 去中心化的分布式数据库，解决交易信任问题
 - 一种数据结构，将多个区块通过密码学算法安全地连接在一起
 - 只可新增数据，不可修改或删除
 - 交易：导致区块链数据库状态改变的一次操作，如添加一条记录或者一笔在两个账户之间的转账操作
- 智能合约
 - 在区块链上分布式运行的程序，不可篡改、不可停止



区块链数据结构示例图²

¹ "区块链发展研究报告2020." Accessed May 8, 2022. <https://static.aminer.cn/misc/pdf/blockchain20.pdf>.

² "Blockchain - Wikipedia." Accessed May 8, 2022. <https://en.wikipedia.org/wiki/Blockchain>.



问题提出

- 智能合约开发者在编程语言的使用和程序测试方面遇到问题
- 部分语言特性不安全、难以测试



问题提出

- 智能合约开发者在编程语言的使用和程序测试方面遇到问题
- **部分语言特性不安全**、难以测试
 - Solidity是以太坊区块链的主要编程语言，用于编写智能合约
 - 诞生于2015年的新语言，目前已进行96次版本迭代
 - 语言特性频繁更新，新的语言特性有时引入安全问题
 - 目前已知Solidity编译器中存在至少50个安全相关的bug，其中14个严重性评级为“中等”或“高”¹
 - 开发者对新特性的了解有限，容易写出有bug的代码

¹ "List of Known Bugs — Solidity 0.8.13 documentation." Accessed May 8, 2022. <https://docs.soliditylang.org/en/v0.8.13/bugs.html>.



问题提出

- 智能合约开发者在编程语言的使用和程序测试方面遇到问题
- 部分语言特性不安全、难以测试
 - 智能合约一旦部署，就不可再修改其代码
 - 需要对其进行完备的测试，但难以构建测试环境，也难以定位缺陷
 - 模拟链上的真实环境，需重放所有交易，交易执行易失败抛出异常
 - 发生异常时，基础的错误信息不足以将错误定位到具体代码位置



主要工作

- 开展针对Solidity语言特性的实证研究  SolEngine
 - 总结41个通用语言特性，分为6类
 - 设计可扩展的特性静态分析工具SolEngine
 - 针对关键特性进行了案例研究，总结使用模式
- 设计并行交易重放测试方法GethReplayer  GethReplayer
 - 对源代码进行语言特性分析，提示可能存在的缺陷
 - 复用区块链上已有的交易数据进行测试
 - 动态替换智能合约的字节码、监测环境信息来检查合约的正确性



研究现状

- 语言特性的实证研究

语言特性实证研究相关工作表

主要研究内容	期刊 / 会议	作者	面向语言	分析多种特性	分析特性产生的安全漏洞	总结使用模式
开发者使用语言特性的 方式	OOPSLA 2020	Coblenz等 ^[1]	Obsidian	✓	✓	✗
	OOPSLA 2019	Mastrangelo等 ^[2]	Java	✗	✗	✓
语言特性对 开发者效率 的影响	ICSE 2016	Uesbeck等 ^[3]	C++	✗	✗	✗
	SANER 2021	Peng等 ^[4]	Python	✓	✗	✓
	SANER 2016	Rebouças等 ^[5]	Swift	✓	✗	✗
	SANER 2016	Mazinanian等 ^[6]	CSS	✓	✗	✓

- 联系：实证研究均面向开发者，考虑了语言特性对开发者的影响
- 区别：相关工作很少考虑到特性产生的安全漏洞，本文进行了安全分析



研究现状

- 智能合约的程序测试

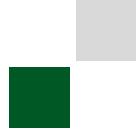
智能合约程序测试相关工作表

类型	方法	期刊 / 会议	作者	方法名称	包含静态分析	支持气体消耗量优化场景
测试环境构建		USENIX 2021	Kim等	Substate Replayer	×	×
		CORR 2019	Hartel等 ^[7]	ContractVis	×	×
测试用例生成	模糊测试	ISSTA 2021	Groce等 ^[8]	Echidna-parade	√	-
		ISSTA 2020	Grieco等 ^[9]	Echidna	√	-
		APSEC 2018	Chan等 ^[10]	Fuse	×	-
	变异测试	ASE 2019	Li等 ^[11]	MuSC	×	-
		AST 2021	Barboni等 ^[12]	SuMo	×	-
		APSEC 2019	Akca等 ^[13]	SolAnalyser	√	-

- 联系：目的均为帮助开发者找到代码中的缺陷
- 区别：本文充分模拟真实环境，在测试中结合静态分析，并支持气体消耗量优化场景



目录



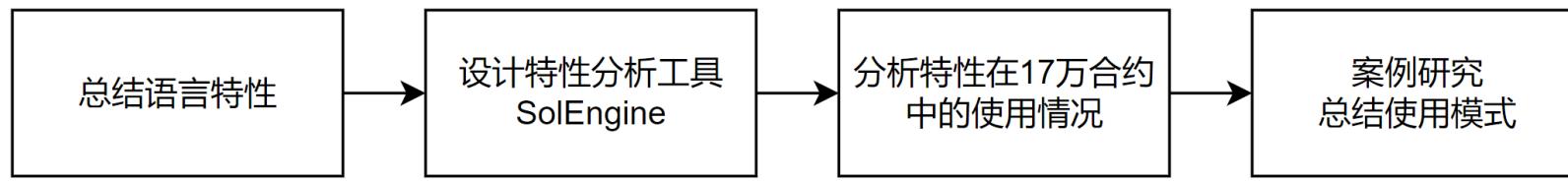
-
- 1 / 研究背景
 - 2 / Solidity语言特性实证研究
 - 3 / 智能合约交易重放测试方法
 - 4 / 总结与展望
-



研究问题

- **RQ1:** Solidity语言特性在开源智能合约中的分布情况如何?
- **RQ2:** 为什么开发者更频繁地使用一些语言特性，它们是如何被使用的?

Solidity语言特性实证研究



Solidity语言特性实证研究流程图

- 总结41个通用语言特性，分为6类
- 设计可扩展的语言特性静态分析工具SolEngine
- 分析特性在172645个开源智能合约中的使用情况
- 针对关键特性进行了案例研究，总结使用模式



41个语言特性，分为6类

- 函数
- 控制流
- 面向对象编程
- 数据结构
- 特殊机制
- 智能合约领域
- 专用特性

Solidity语言特性表			
Category	Language Feature	Category	Language Feature
A. Function	Returning Multiple Value	D. Data Structure	Array
	Recursion [12]		Struct
	First-Class Function [13]		Nested Array Or Struct
	Constant Function		Enum
	Function Modifier		Constant State Variable
	Named Call	E. Special Mechanism	SMT Checker
	Free Function		Inline Assembly [16]
	Return Variable		Unicode Literal
	Function Overloading		Hexadecimal Literal
	Loop [14]		Ether Unit
B. Control Flow [7]	Assert		Time Unit
	Exception Handling		NatSpec Documentation
	Single Inheritance	F. Smart Contract Domain-Specific Features	Fallback Function
	Multiple Inheritance		Receive Ether Function
	Virtual Method Lookup [15]		High-Level Cross-Contract Function Invocation
C. Object-Oriented Programming	Function Overriding		Low-Level Cross-Contract Function Invocation
	Function Modifier Overriding		Transfer
	Abstract Contract		Creating Contract Via new
	Interface		Event
	Visibility		Manual Gas Control
	Library		



特性举例：备用函数（智能合约领域专用特性）

```
1 contract MyContract {  
2     function func1() {  
3     }  
4     function func2() {  
5     }  
6     function() {  
7         // fallback function  
8     }  
9 }
```

备用函数的示例代码



特性举例：备用函数（智能合约领域专用特性）

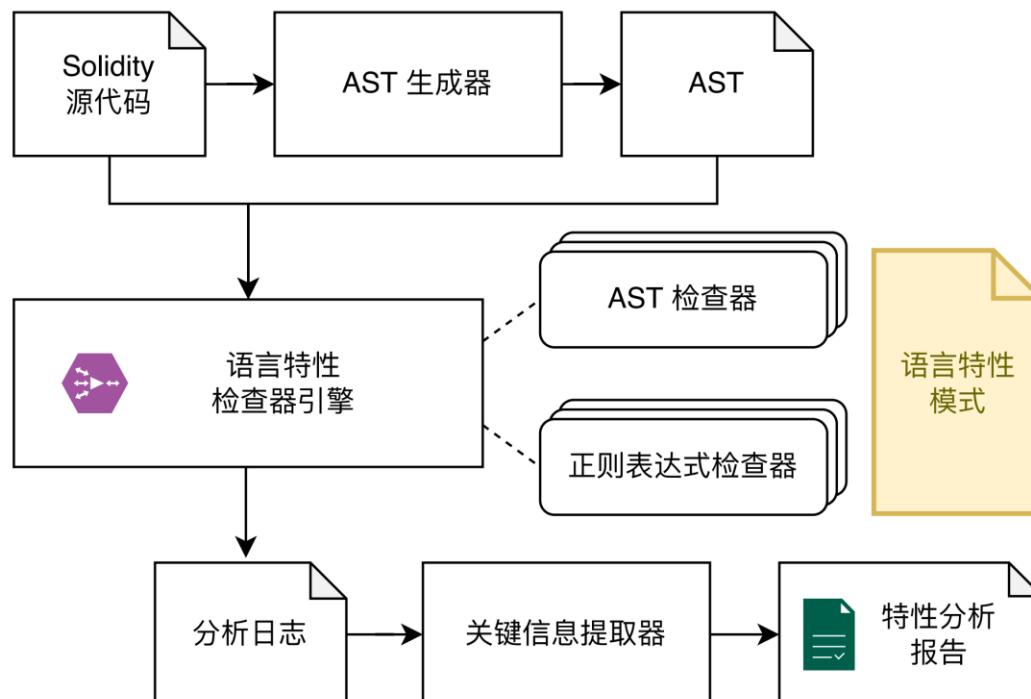
```
1 contract MyContract {  
2     function func1() {  
3     }  
4     function func2() {  
5     }  
6     function() {  
7         // fallback function  
8     }  
9 }
```

备用函数的示例代码



可扩展的语言特性静态分析工具SolEngine

- 将Solidity源代码转换为抽象语法树（AST）
- 运行多个语言特性检查器
- 生成特性分析报告



solengine.github.io

可从此处获取源码

SolEngine架构图



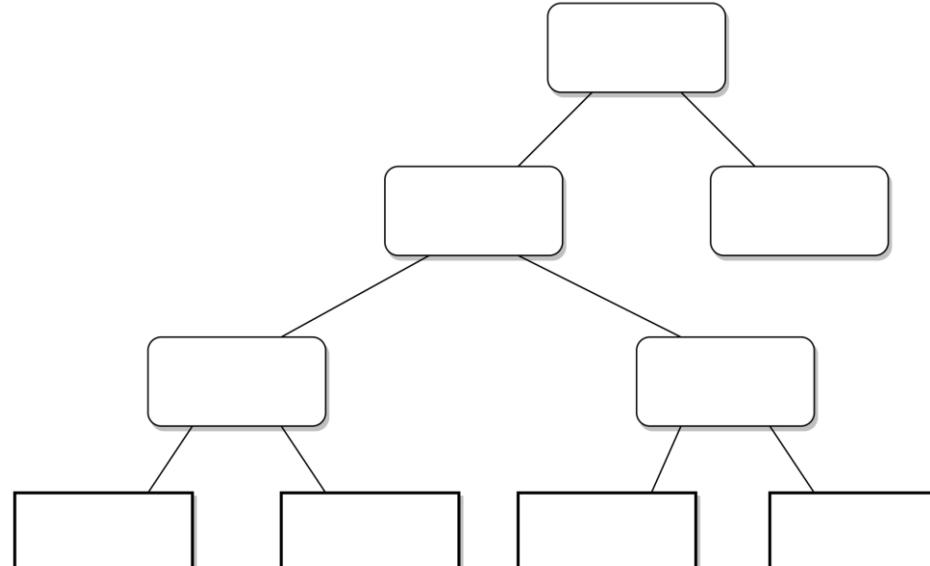
语言特性检查算法

```
check(node, visitor)
    if entry flag of node is in visitor
        visit(visitor, flag)

    for each child of node
        check(child, visitor)

    if exit flag of node is in visitor
        visit(visitor, flag)
```

检查算法伪代码



抽象语法树 (AST) 示意图



语言特性检查算法

```
check(node, visitor)
    if entry flag of node is in visitor
        visit(visitor, flag)

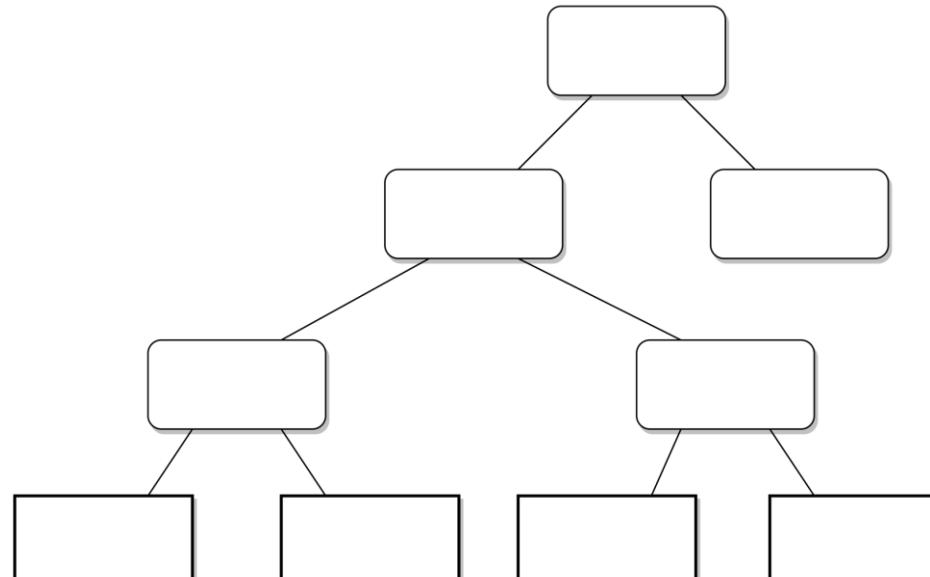
    for each child of node
        check(child, visitor)

    if exit flag of node is in visitor
        visit(visitor, flag)
```

检查算法伪代码

```
1 address addr = ...;
2 addr.transfer();
```

转账特性示例代码



抽象语法树 (AST) 示意图



语言特性检查算法

```
check(node, visitor)
    if entry flag of node is in visitor
        visit(visitor, flag)

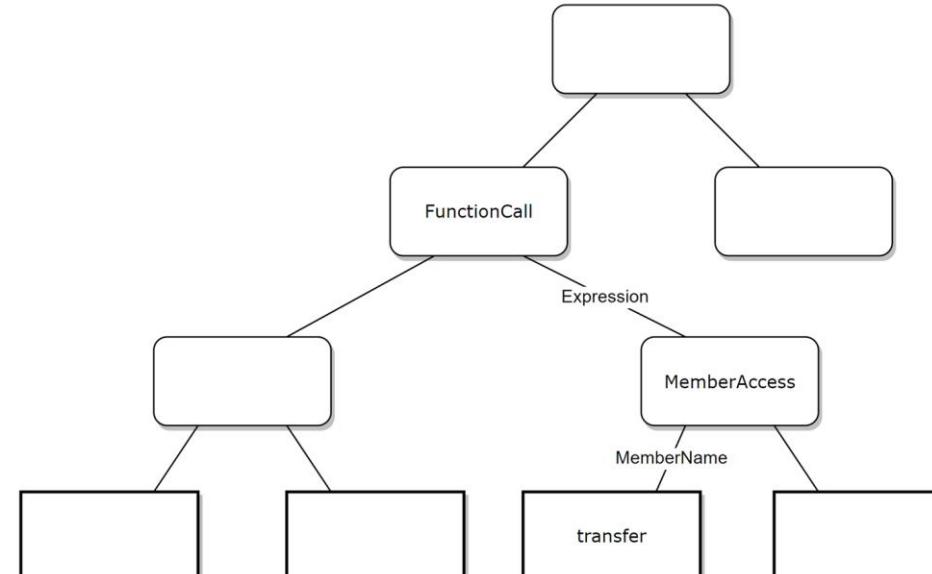
    for each child of node
        check(child, visitor)

    if exit flag of node is in visitor
        visit(visitor, flag)
```

检查算法伪代码

```
1 address addr = ...;
2 addr.transfer();
```

转账特性示例代码



抽象语法树 (AST) 示意图



语言特性检查算法

```
check(node, visitor)
    if entry flag of node is in visitor
        visit(visitor, flag)
```

```
    for each child of node
        check(child, visitor)
```

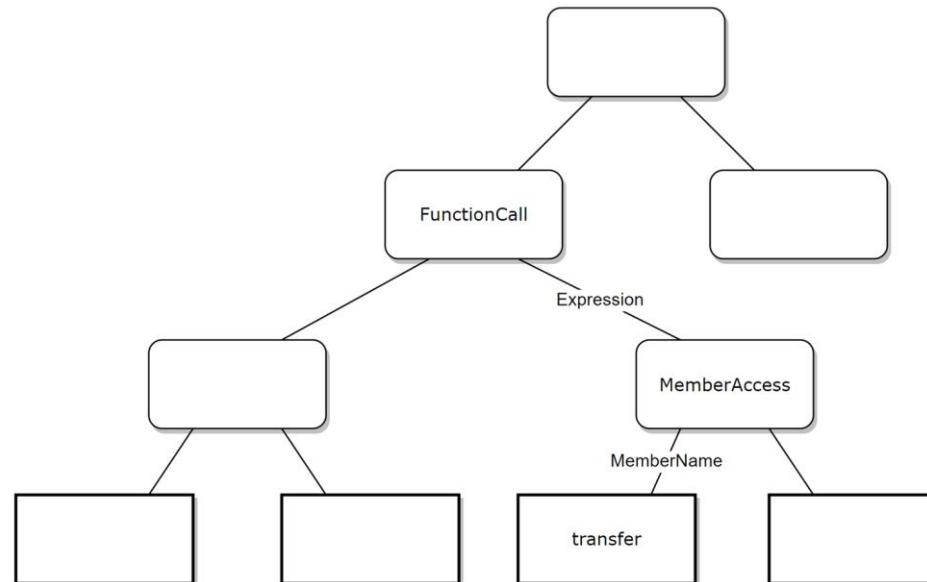
```
    if exit flag of node is in visitor
        visit(visitor, flag)
```

检查算法伪代码

```
FunctionCall {
    MemberAccess {
        (FunctionCall.Expression is MemberAccess and MemberAccess.MemberName
         is "transfer")
    }
}
```

```
1 address addr = ...;
2 addr.transfer();
```

转账特性示例代码



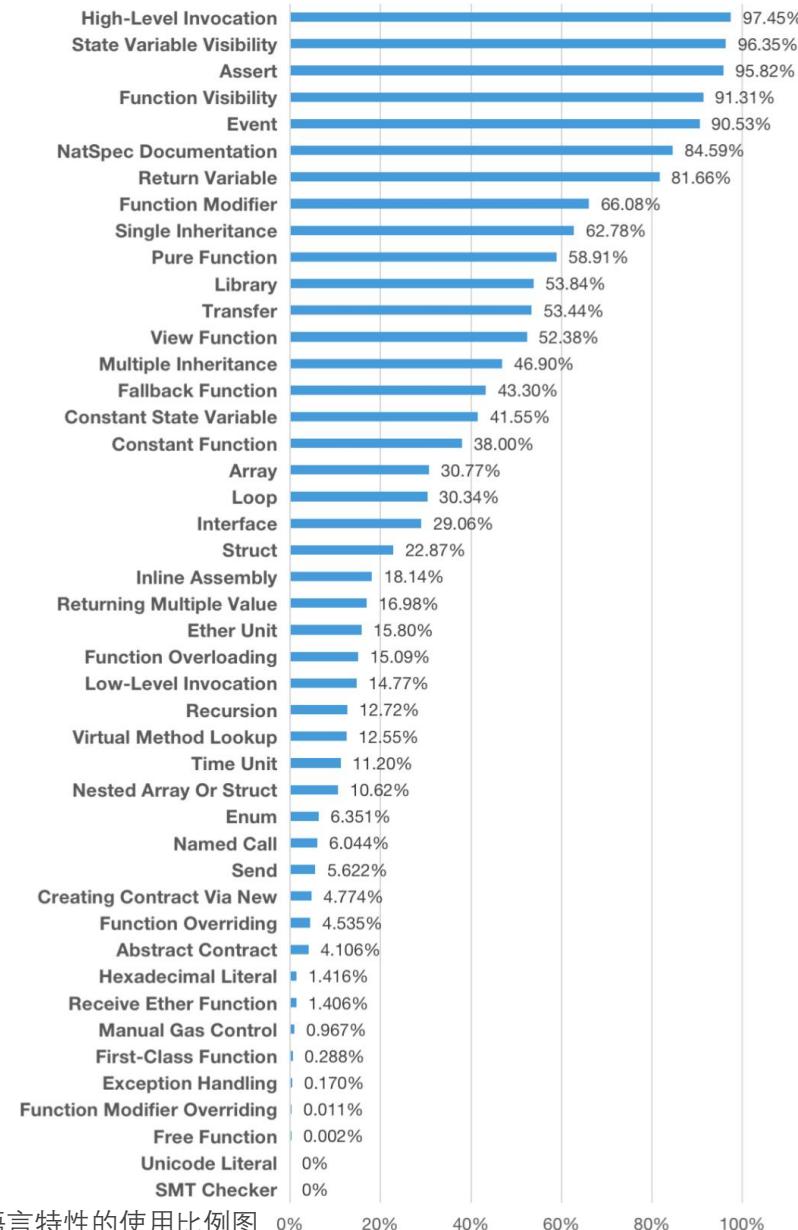
抽象语法树 (AST) 示意图

转账特性的AST检查器伪代码 (已简化)



语言特性的使用分布

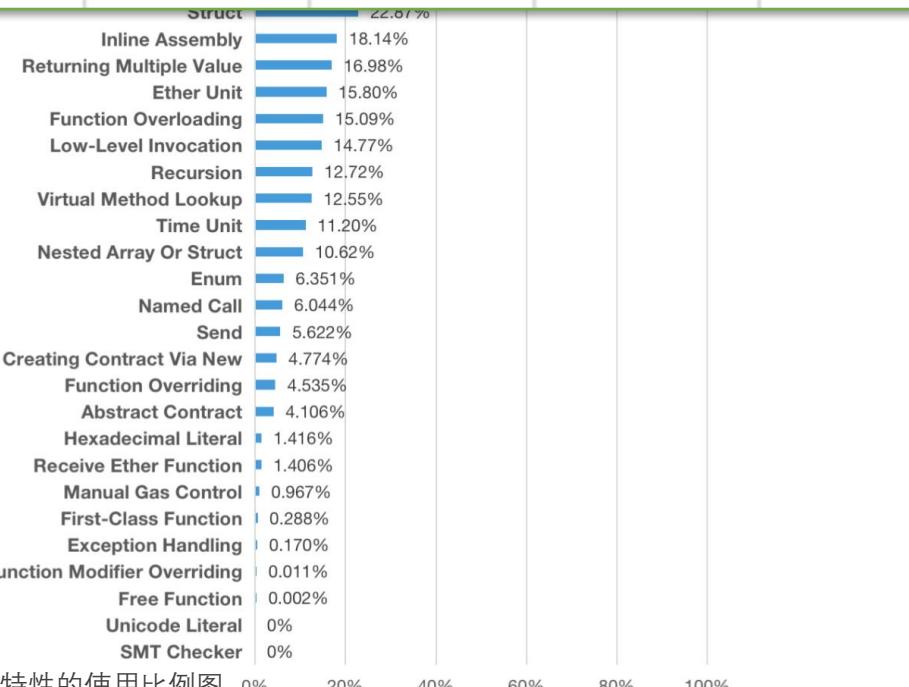
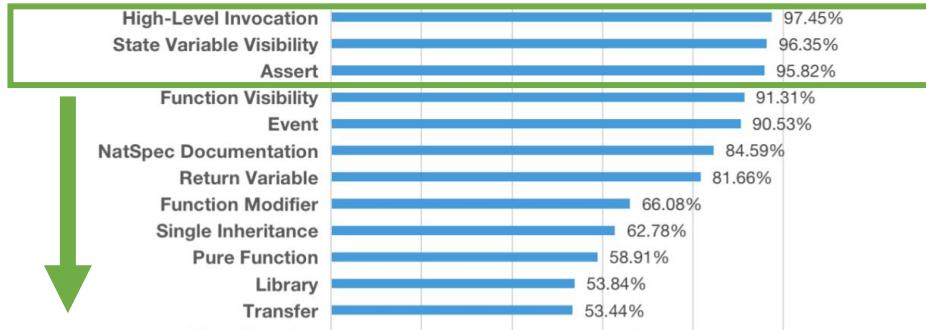
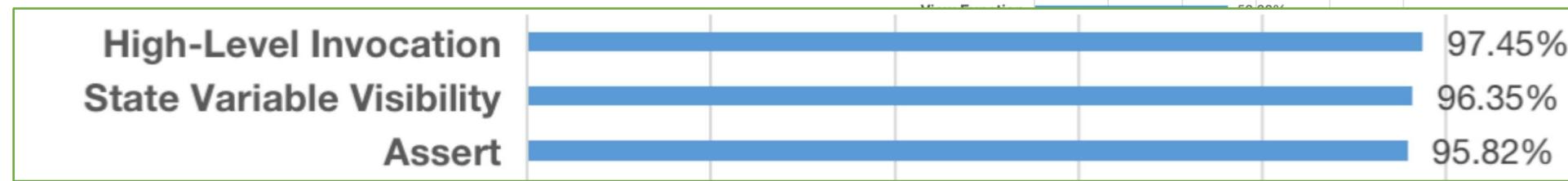
- 使用率：使用某种特性的智能合约占数据集中所有合约的比例





语言特性的使用分布

- 使用率：使用某种特性的智能合约占数据集中所有合约的比例

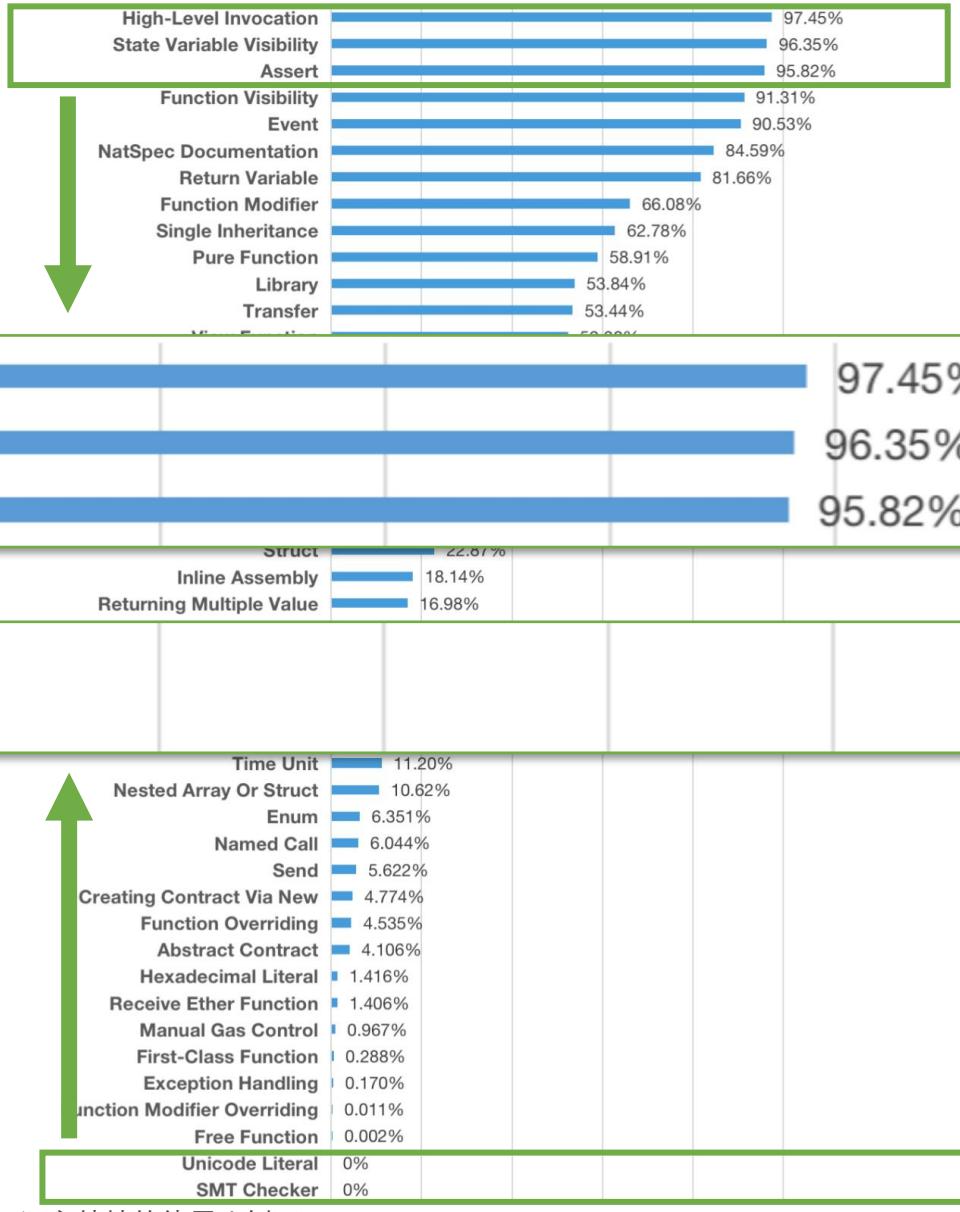
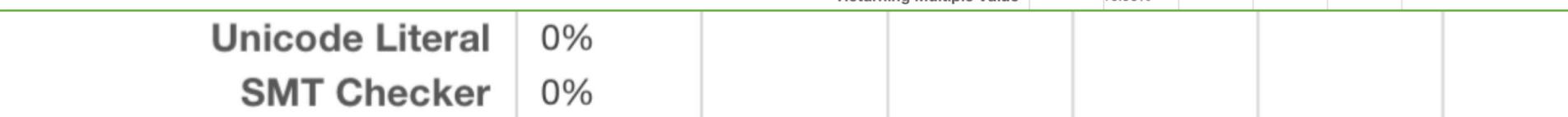
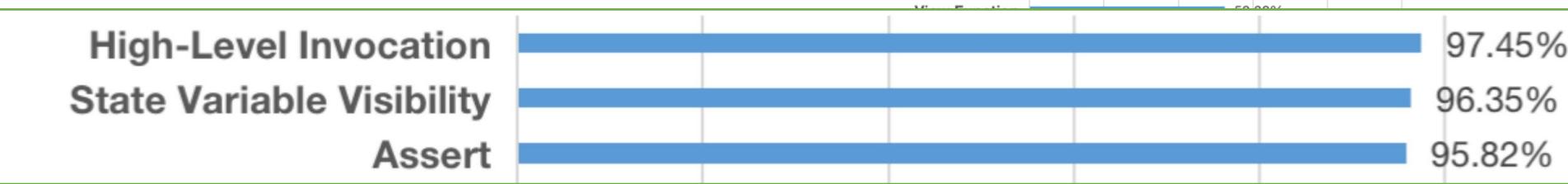


语言特性的使用比例图



语言特性的使用分布

- 使用率：使用某种特性的智能合约占数据集中所有合约的比例



语言特性的使用比例图

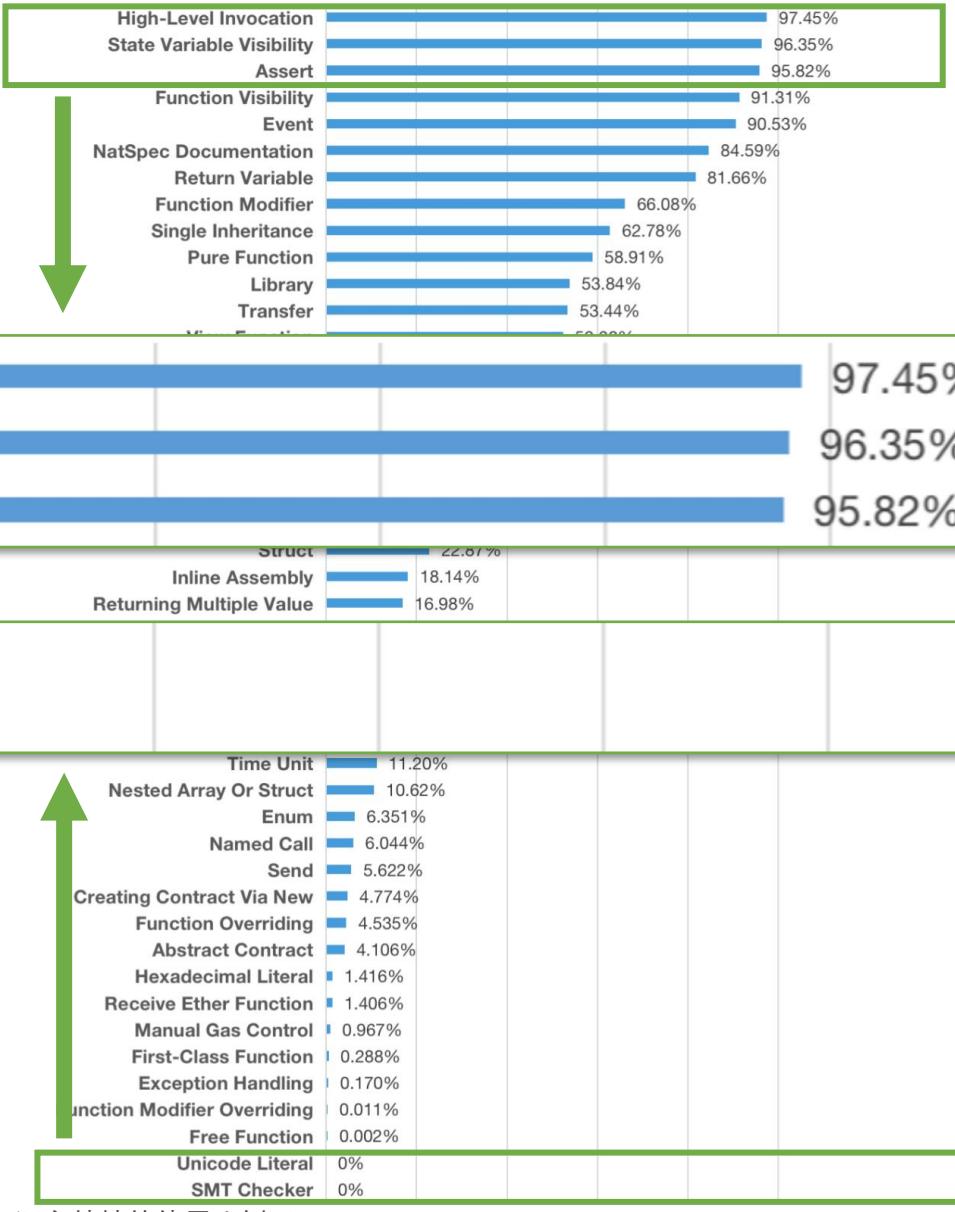


语言特性的使用分布

- 使用率：使用某种特性的智能合约占数据集中所有合约的比例



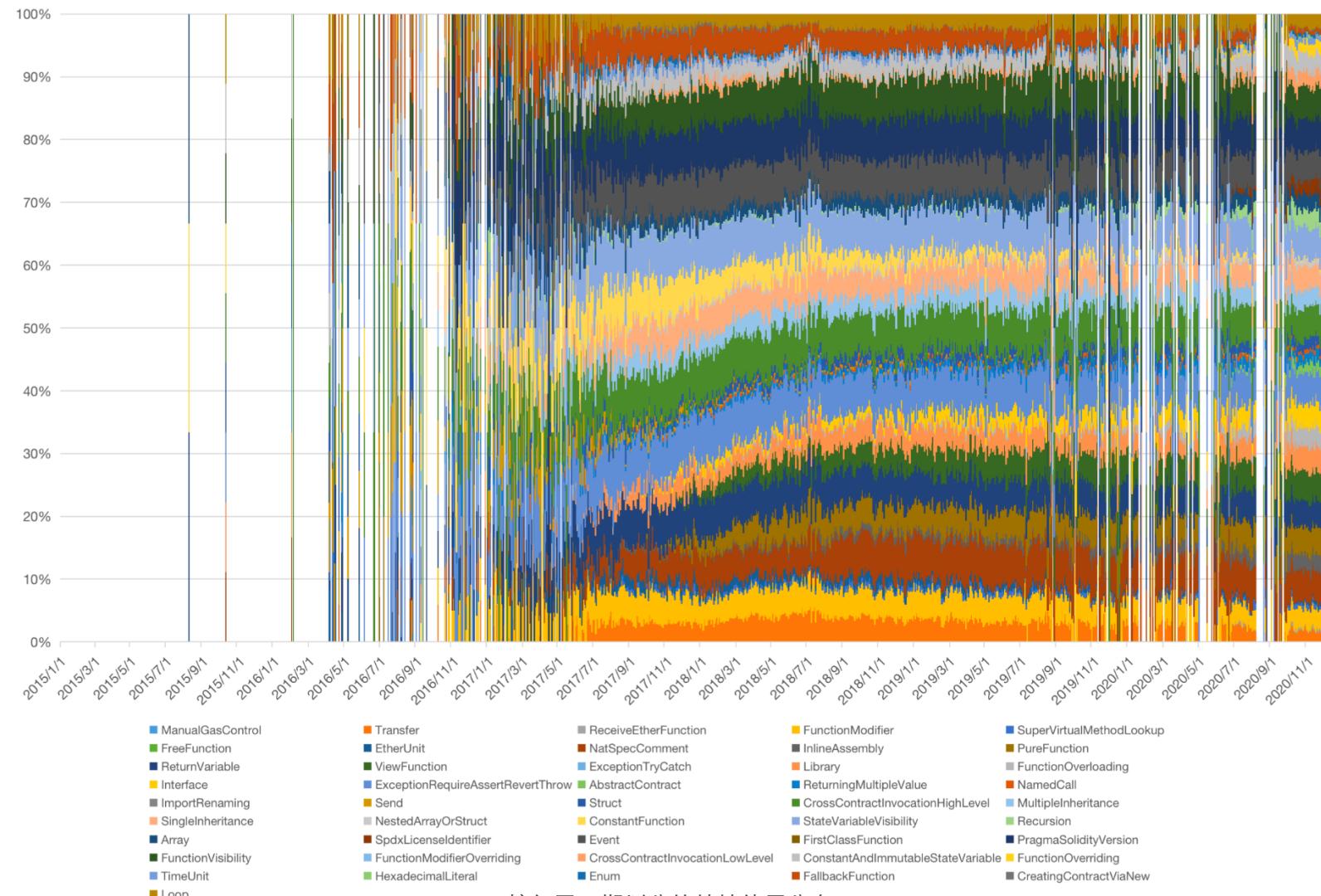
结论：高级跨合约函数调用、可见性、断言、事件和NatSpec文档是最流行的特性，而SMT检查器、Unicode字符串、自由函数和函数修饰符覆盖是最不流行的特性。



语言特性的使用比例图

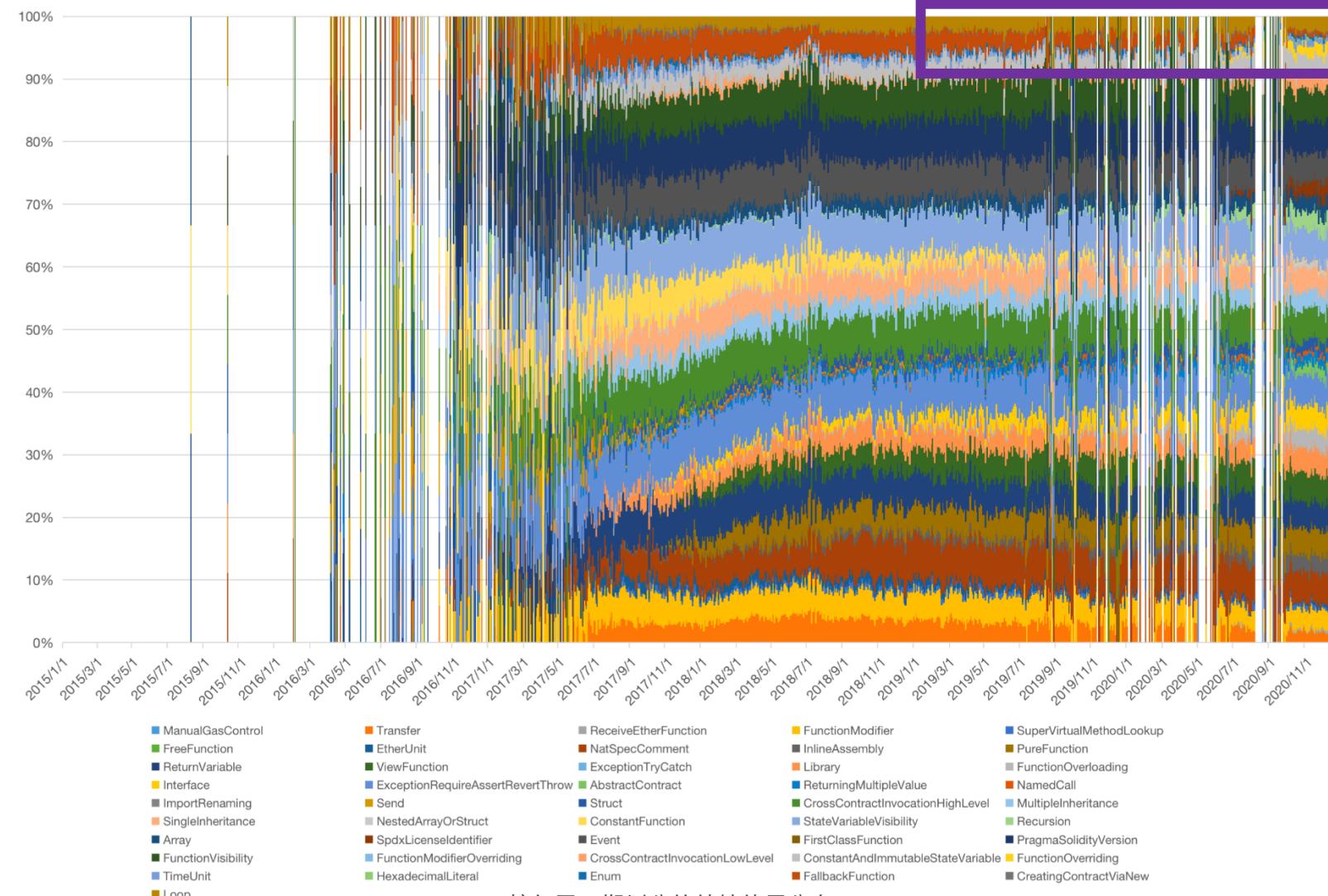


按部署日期划分的特性使用分布



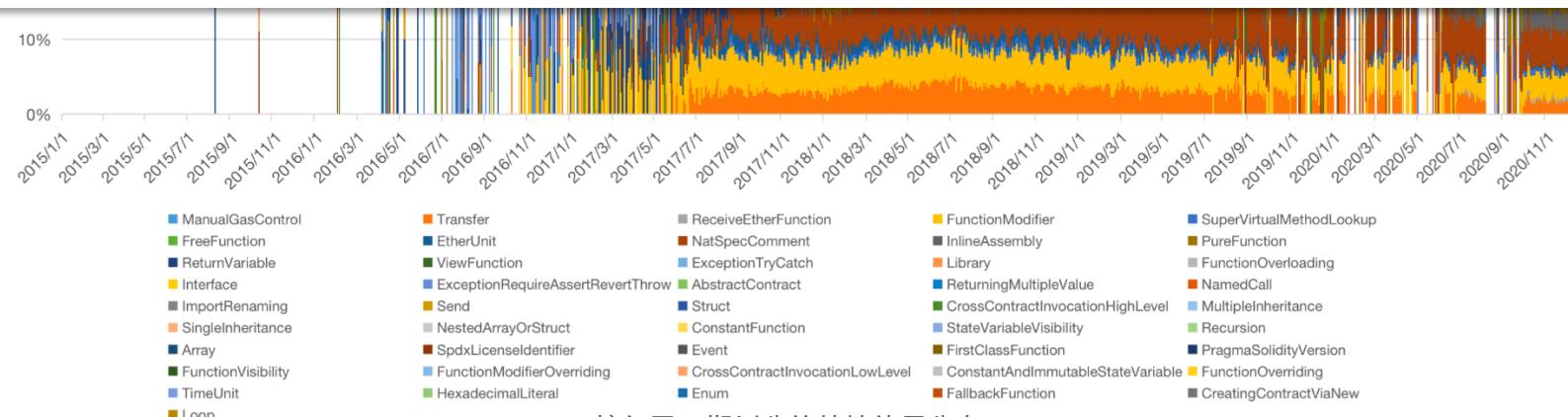
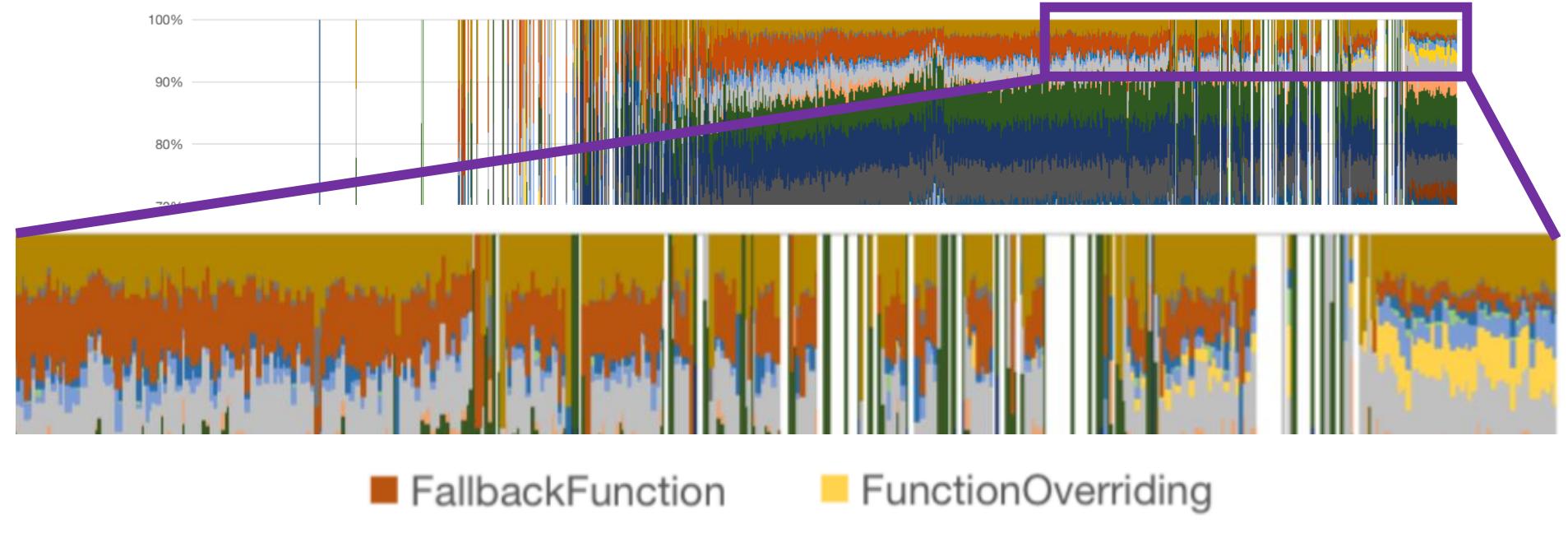


按部署日期划分的特性使用分布



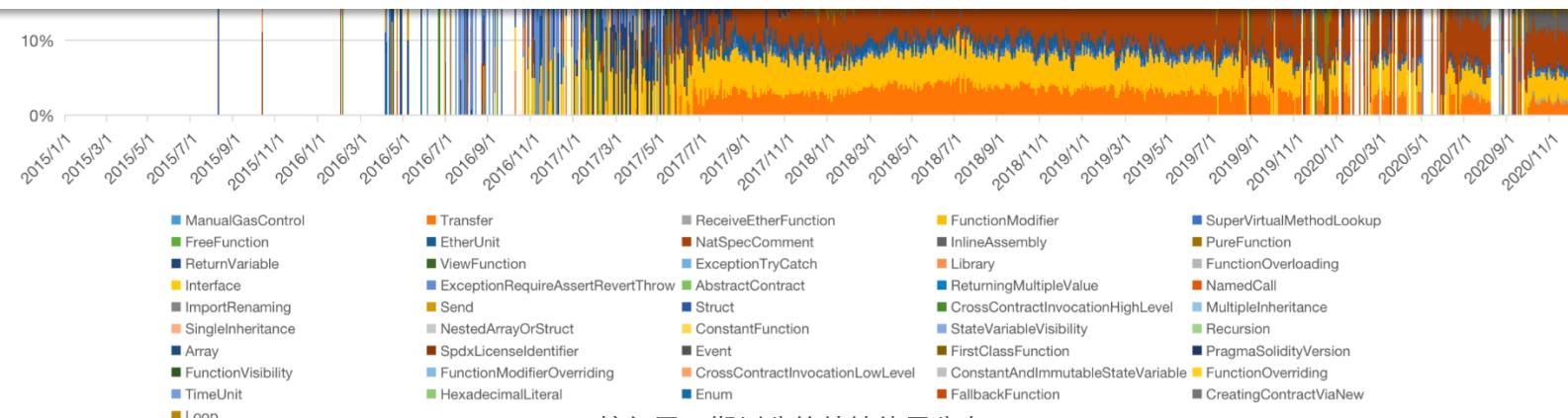
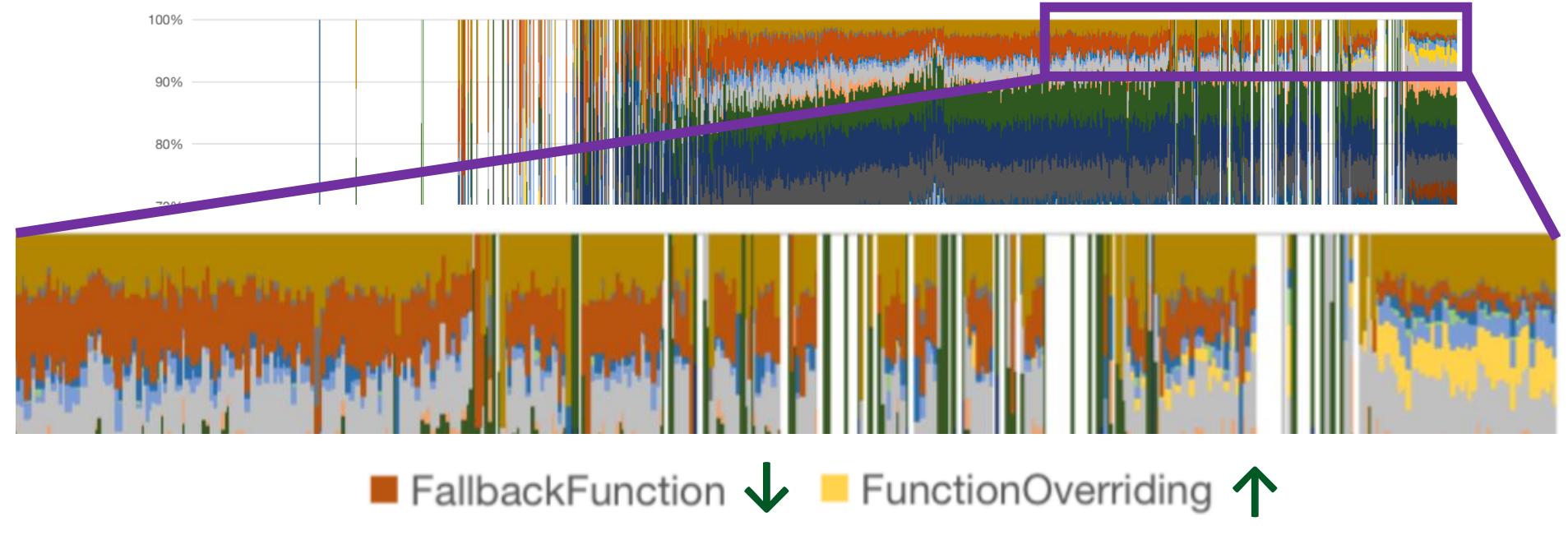


按部署日期划分的特性使用分布





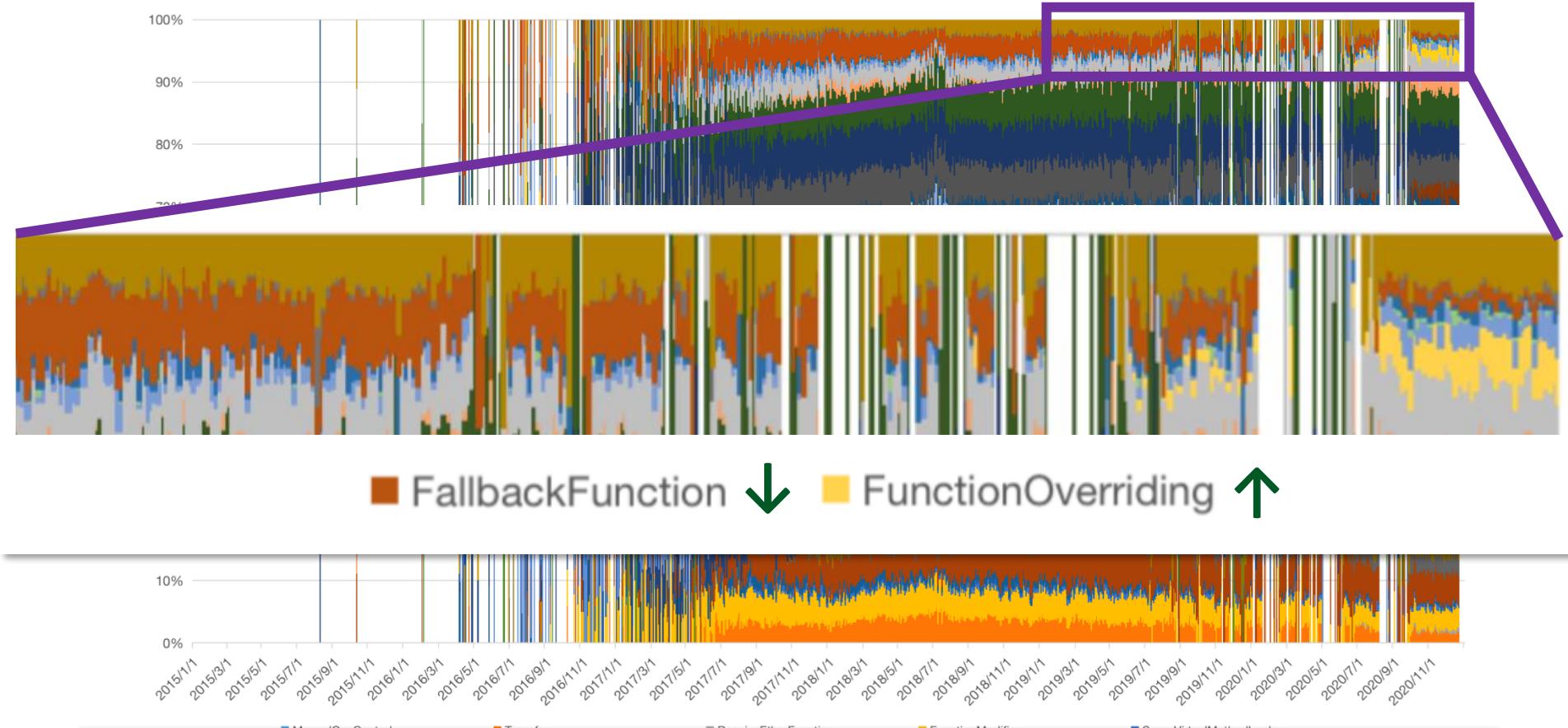
按部署日期划分的特性使用分布



按部署日期划分的特性使用分布图



按部署日期划分的特性使用分布



结论：随着时间的推移，开发人员倾向于使用构建大型复杂合约所必需的特性，并避免使用备用函数等容易出错的特性。



| 案例分析

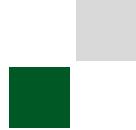
- 针对3个关键特性找出4个案例和使用模式
- 包括断言、循环、函数修饰符特性

```
function internalForward() internal {
    for (uint256 i = 0; i < receivers.length; i++) {
        Receiver storage receiver = receivers[i];
        receiver.receiver.transfer(value);
    }
}
```

循环特性：不安全的循环示例代码



目录

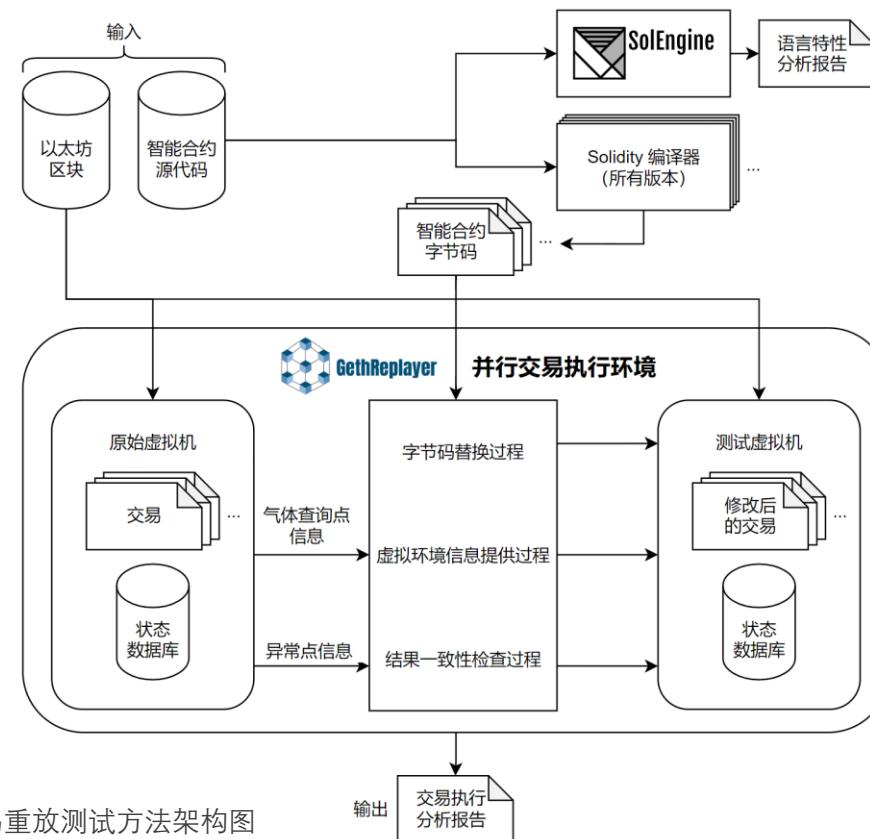


-
- 1 / 研究背景
 - 2 / Solidity语言特性实证研究
 - 3 / 智能合约交易重放测试方法
 - 4 / 总结与展望
-



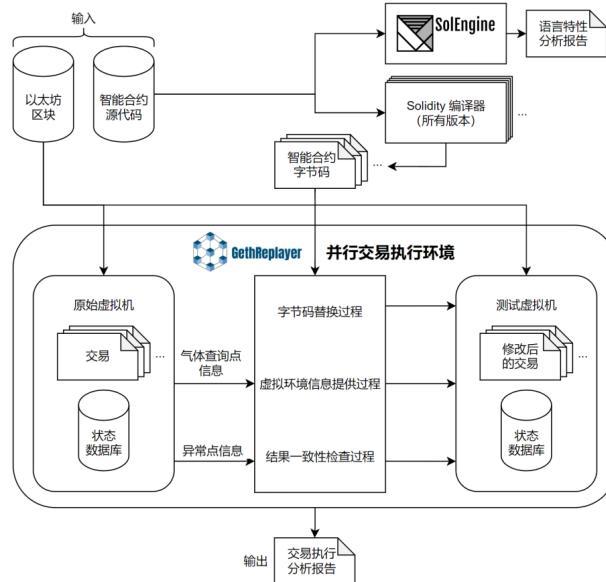
并行交易重放测试方法

- 用SolEngine对源代码进行语言特性分析，提示可能存在的缺陷
- 并行重放区块链交易，对测试合约分别运行其新旧版本
 - 监测并同步环境信息，确保合约持续运行
 - 检查运行结果的一致性，并定位错误





重放测试算法



并行交易重放测试方法架构图

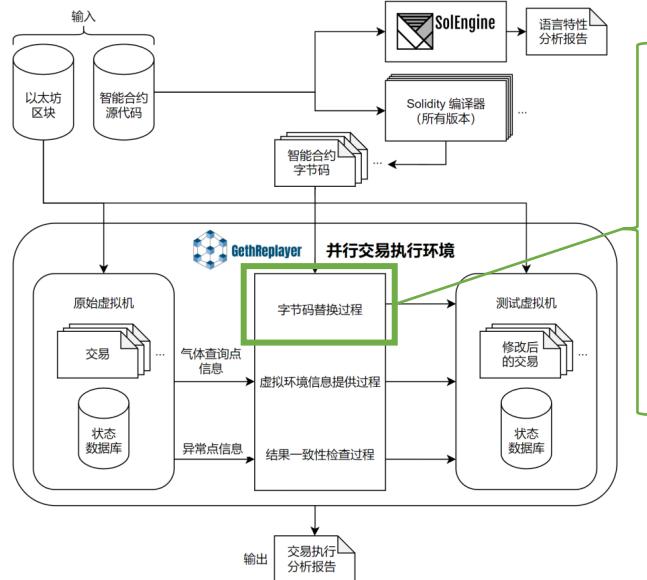
输入：以太坊区块列表Blocks，被测智能合约的源代码列表Contracts
输出：交易信息列表Results

```
1 将Contracts的地址记录为受监控的合约地址列表Addresses
2 从Etherscan平台上根据Addresses获取所有合约的编译器版本号、部署时的构造函数参数，得到数据集EtherscanData
3 计算修改后的合约字节码字典BytecodeMap
4 初始化原始虚拟机Machine0和测试虚拟机Machine1
5 初始化交易信息列表Results为空
6 对Blocks中的每一个以太坊区块Block {
7   对Block中的每一个交易Transaction {
8     如果Transaction是部署合约的交易，且部署的合约Contract的地址Address在Addresses中 {
9       在Machine0中执行Transaction
10      从BytecodeMap中检索键为Address的项，取其值Bytecode
11      从EtherscanData中找到Contract的构造函数参数Argument
12      将Bytecode和Argument组合成FullBytecode
13      把Transaction中的合约字节码替换为FullBytecode，得到ManipulatedTransaction
14      在Machine1中执行ManipulatedTransaction
15    }
16    否则 {
17      在Machine0和Machine1中分别执行Transaction
18      如果Transaction涉及到Addresses {
19        记录交易执行结果Info1，记录并对齐气体查询点信息Info2、异常点信息Info3
20        把Info1、Info2、Info3存入Results
21      }
22    }
23  }
24 }
```

重放测试算法伪代码（已简化）



重放测试算法



并行交易重放测试方法架构图

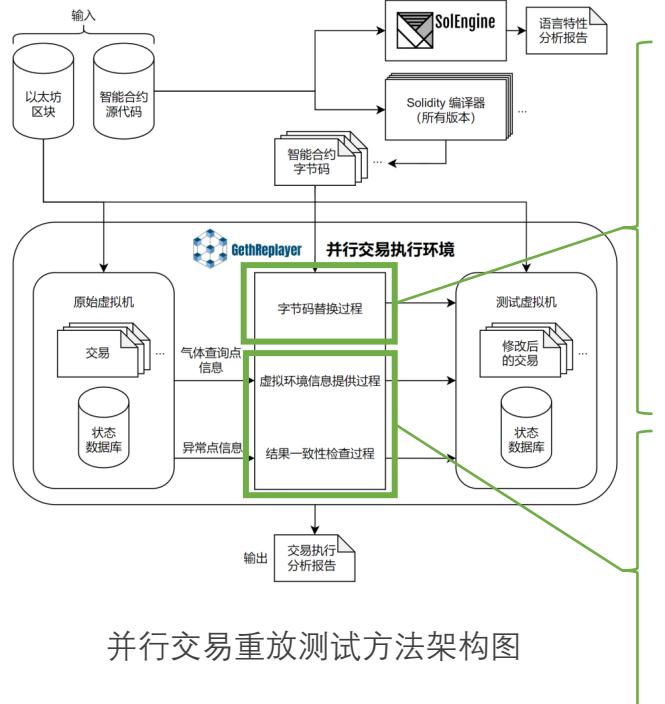
输入：以太坊区块列表Blocks，被测智能合约的源代码列表Contracts
输出：交易信息列表Results

```
1 将Contracts的地址记录为受监控的合约地址列表Addresses
2 从Etherscan平台上根据Addresses获取所有合约的编译器版本号、部署时的构造函数参数，得到数据集EtherscanData
3 计算修改后的合约字节码字典BytecodeMap
4 初始化原始虚拟机Machine0和测试虚拟机Machine1
5 初始化交易信息列表Results为空
6 对Blocks中的每一个以太坊区块Block {
7   对Block中的每一个交易Transaction {
8     如果Transaction是部署合约的交易，且部署的合约Contract的地址Address在Addresses中 {
9       在Machine0中执行Transaction
10      从BytecodeMap中检索键为Address的项，取其值Bytecode
11      从EtherscanData中找到Contract的构造函数参数Argument
12      将Bytecode和Argument组合成FullBytecode
13      把Transaction中的合约字节码替换为FullBytecode，得到ManipulatedTransaction
14      在Machine1中执行ManipulatedTransaction
15    }
16    否则 {
17      在Machine0和Machine1中分别执行Transaction
18      如果Transaction涉及到Addresses {
19        记录交易执行结果Info1，记录并对齐气体查询点信息Info2、异常点信息Info3
20        把Info1、Info2、Info3存入Results
21      }
22    }
23  }
24 }
```

重放测试算法伪代码（已简化）



重放测试算法



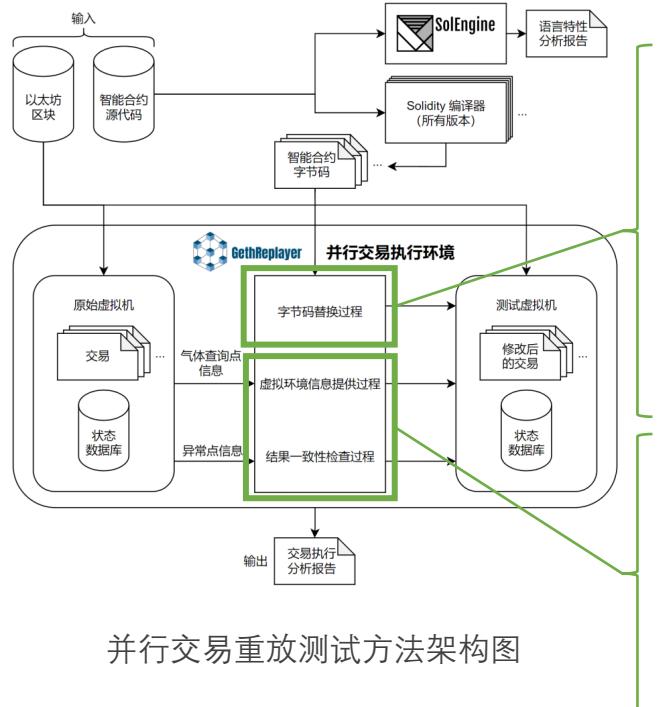
输入：以太坊区块列表Blocks，被测智能合约的源代码列表Contracts
输出：交易信息列表Results

```
1 将Contracts的地址记录为受监控的合约地址列表Addresses
2 从Etherscan平台上根据Addresses获取所有合约的编译器版本号、部署时的构
造函数参数，得到数据集EtherscanData
3 计算修改后的合约字节码字典BytecodeMap
4 初始化原始虚拟机Machine0和测试虚拟机Machine1
5 初始化交易信息列表Results为空
6 对Blocks中的每一个以太坊区块Block {
7     对Block中的每一个交易Transaction {
8         如果Transaction是部署合约的交易，且部署的合约Contract的地
址Address在Addresses中 {
9             在Machine0中执行Transaction
10            从BytecodeMap中检索键为Address的项，取其值Bytecode
11            从EtherscanData中找到Contract的构造函数参数Argument
12            将Bytecode和Argument组合成FullBytecode
13            把Transaction中的合约字节码替换为FullBytecode，得到
ManipulatedTransaction
14            在Machine1中执行ManipulatedTransaction
15        }
16        否则 {
17            在Machine0和Machine1中分别执行Transaction
18            如果Transaction涉及到Addresses {
19                记录交易执行结果Info1，记录并对齐气体查询点信息
Info2、异常点信息Info3
20                把Info1、Info2、Info3存入Results
21            }
22        }
23    }
24 }
```

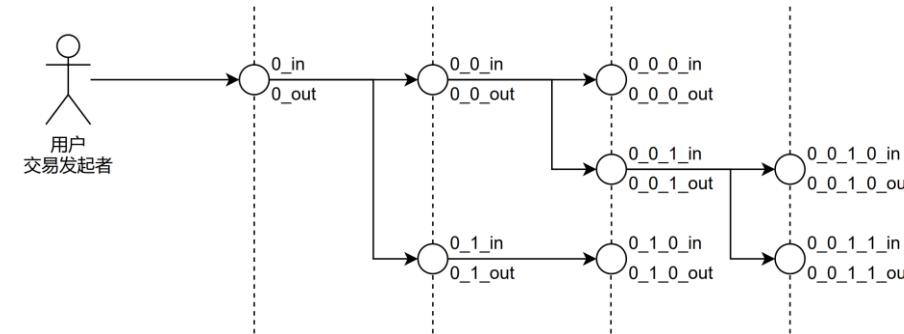
重放测试算法伪代码（已简化）



重放测试算法



并行交易重放测试方法架构图



函数调用树、函数调用链索引示例图

```
7 对Block中的每一个交易 Transaction {
8   如果Transaction是部署合约的交易，且部署的合约Contract的地
址Address在Addresses中 {
9     在Machine0中执行Transaction
10    从BytecodeMap中检索键为Address的项，取其值Bytecode
11    从EtherscanData中找到Contract的构造函数参数Argument
12    将Bytecode和Argument组合成FullBytecode
13    把Transaction中的合约字节码替换为FullBytecode，得到
ManipulatedTransaction
14    在Machine1中执行ManipulatedTransaction
15  }
16  否则 {
17    在Machine0和Machine1中分别执行Transaction
18    如果Transaction涉及到Addresses {
19      记录交易执行结果Info1，记录并对齐气体查询点信息
Info2、异常点信息Info3
20      把Info1、Info2、Info3存入Results
21    }
22  }
23 }
24 }
```

重放测试算法伪代码（已简化）

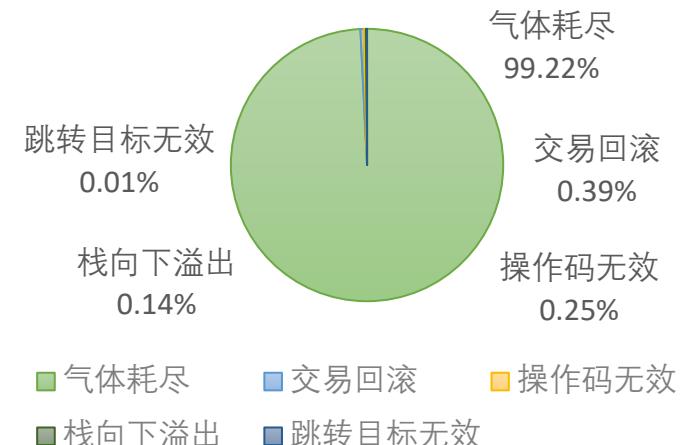


虚拟环境信息的有效性实验

- 为了验证虚拟环境信息的有效性，对比了禁用和启用时的重放成功率
- 在256个合约的数据集上重放，实验所需时间约为21天
- 无虚拟环境信息时，仅可成功重放2.35%的合约
- 有虚拟环境信息时，成功重放100%的合约

异常类型	发生次数
气体耗尽	18217388
交易回滚	71352
操作码无效	45302
栈向下溢出	25014
跳转目标无效	1030

异常发生次数占比图





错误定位有效性实验

- 发生异常时，开发者需要关注的平均函数调用结点数从14.33个减少至1个

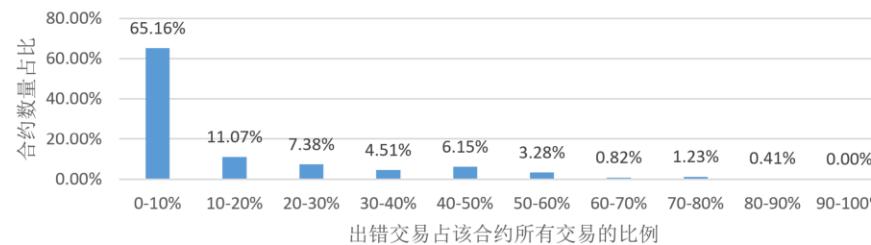


图 4-5 按出错交易占该合约所有交易的比例划分的合约数量分布直方图

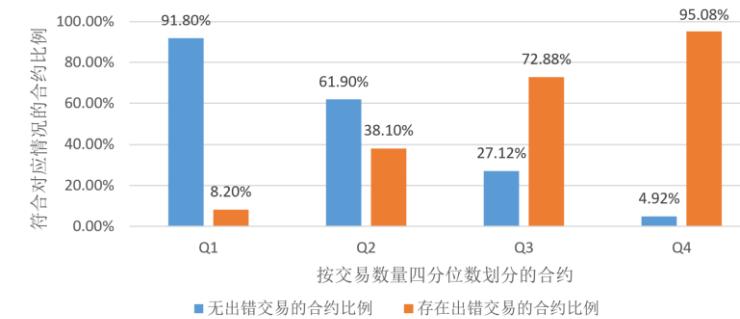


图 4-6 按交易数量四分位数划分的合约交易出错情况分布图

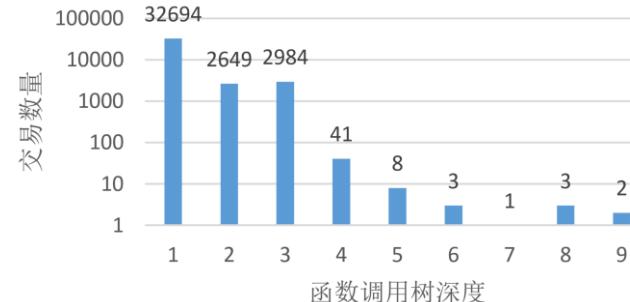


图 4-7 函数调用树的深度分布图

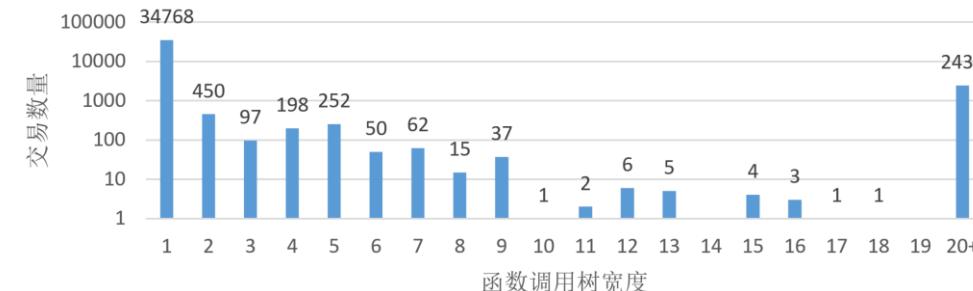


图 4-8 函数调用树的宽度分布图



支持气体消耗量优化场景

- 包括本文作者在内的Kong等人^[14]尝试优化>16万个合约，减少气体消耗量
- 描述了6种低效率的气体消耗模式，以及优化方法
- 对优化前和优化后的开源智能合约进行交易重放，验证方法的有效性

低效率的气体消耗模式示例表

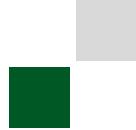
编号	模式	低效率的气体消耗模式例子	高效率的气体消耗模式例子
1	稀疏存储	<code>uint8 public decimals;</code> <code>uint256 public totalSupply;</code> <code>address public owner;</code>	<code>uint8 public decimals;</code> <code>address public owner;</code> <code>uint256 public totalSupply;</code>

气体消耗量优化结果数据表

模式编号	合约数量	交易数量	节约的气体消耗量	
			部署合约的交易	调用合约的交易
P1	11657	286696	80494110	12174283
P2	8810	229995	79077573	749724
P3	2149	59146	12262199	8860627
P4	1898	69621	-22030761	23219821
P5	7442	189270	348897	4850887
P6	3216	109215	29664663	23636855



目录



-
- 1 / 研究背景
 - 2 / Solidity语言特性实证研究
 - 3 / 智能合约交易重放测试方法
 - 4 / 总结与展望
-



本文贡献

- 开展针对Solidity语言特性的实证研究
 - 总结41种语言特性
 - 设计可扩展的特性静态分析工具SolEngine，分析172645个开源合约
 - 针对关键特性进行了案例研究，总结使用模式和它们造成的缺陷，为语言设计者、静态分析工具开发者和App开发者提供语言特性建议
- 设计并行交易重放测试方法GethReplayer
 - 复用区块链上已有的交易数据进行测试
 - 动态替换智能合约的字节码、监测环境信息来检查合约的正确性
 - 提高测试的成功率和开发者排查错误的效率

未来展望

- 提高本文方法的性能
- 支持对闭源合约进行测试



研究期间已发表的成果

➤ 论文

- An Empirical Study of Solidity Language Features, The 21st IEEE International Conference on Software Quality, Reliability, and Security, 2021.  **QRS 2021**
(EI, 第一作者)

- Characterizing and Detecting Gas-Inefficient Patterns in Smart Contracts,

Journal of Computer Science and Technology, 2022.



(CCF-B, 第二作者, 第一作者为博士生)

➤ 专利

- 一种基于源代码的智能合约优化方法及装置, 2021.

(第二发明人, 第一发明人为老师)



论文修改情况

	盲审评阅专家意见	修改情况
1	论文摘要写的略显繁琐，建议精简一下，使逻辑更清晰简单些。	已修改，在摘要中开门见山，清晰地描述研究问题，方法描述部分精简细节，突出重点，删减不必要的形容词和副词，使摘要更加简明扼要。
2	1.3节的描述过于简略，希望能够配图，说明各个章节之间的相互支撑关系。	已修改，在1.3“本文的论文结构与章节安排”一节中补充介绍了每一章的内容，并且配上论文结构示意图。加入了第三章和第四章的关系描述，让读者了解第三章对第四章起到辅助提示的作用。
3	第二章描述研究现状时，段落篇幅过长，需要对每一部分进行归类和总结，并相应分段，同时节选一些有代表性的工作，对其优缺点进行分析，而不是一些简单的文字累积。	已修改，优化了段落分配，缩短了段落的平均长度，提高读者的阅读体验。对语言设计和实证研究、智能合约测试的部分代表性工作进行了总结和分析，与本文进行对比，并补充清晰的表格。对研究现状相关工作的分类进行了优化，重新分类智能合约测试和交易重放的工作，将智能合约的编程模式经验总结归类到2.1“语言设计和实证研究”一节。
4	注意图表的标题格式，图标题在图下方，表标题在表上方，表4-1格式不符，应仔细检查所有图表。	已修改，将表4-1的标题从下方移至上方。经仔细检查，表4-1是学位论文中目前发现的唯一的格式不正确的图表。



参考文献

- [1] Coblenz M, Aldrich J, Myers B A, et al. Can advanced type systems be usable? an empirical study of ownership, assets, and typestate in obsidian[J/OL]. Proc. ACM Program. Lang., 2020, 4(OOPSLA). <https://doi.org/10.1145/3428200>.
- [2] Mastrangelo L, Hauswirth M, Nystrom N. Casting about in the dark: An empirical study of cast operations in java programs[J/OL]. Proc. ACM Program. Lang., 2019, 3(OOPSLA). <https://doi.org/10.1145/3360584>.
- [3] Uesbeck P M, Stefik A, Hanenberg S, et al. An empirical study on the impact of c++ lambdas and programmer experience[C/OL]//ICSE '16: Proceedings of the 38th International Conference on Software Engineering. New York, NY, USA: Association for Computing Machinery, 2016:760–771. <https://doi.org/10.1145/2884781.2884849>.
- [4] Peng Y, Zhang Y, Hu M. An empirical study for common language features used in python projects[Z]. 2021.
- [5] Rebouças M, Pinto G, Ebert F, et al. An empirical study on the usage of the swift programming language[C/OL]//2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER): volume 1. 2016:634–638. DOI: 10.1109/SANER.2016.66.
- [6] Mazinanian D, Tsantalis N. An empirical study on the use of css preprocessors[C/OL]//2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER): volume 1. 2016:168–178. DOI: 10.1109/SANER.2016.18.
- [7] Hartel P, van Staalduinen M. Truffle tests for free – replaying ethereum smart contracts for transparency[M/OL]. arXiv, 2019. <https://arxiv.org/abs/1907.09208>. DOI: 10.48550/ARXIV.1907.09208.
- [8] Groce A, Grieco G. Echidna-parade: A tool for diverse multicore smart contract fuzzing[M/OL]. New York, NY, USA: Association for Computing Machinery, 2021:658–661. <https://doi.org/10.1145/3460319.3469076>.
- [9] Grieco G, Song W, Cygan A, et al. Echidna: Effective, usable, and fast fuzzing for smart contracts[C/OL]//ISSTA 2020: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, NY, USA: Association for Computing Machinery, 2020:557–560. <https://doi.org/10.1145/3395363.3404366>
- [10] Chan W, Jiang B. Fuse: An architecture for smart contract fuzz testing service[C/OL]//2018 25th Asia-Pacific Software Engineering Conference (APSEC). 2018:707–708. DOI: 10.1109/APSEC.2018.00099.
- [11] Li Z, Wu H, Xu J, et al. Musc: A tool for mutation testing of ethereum smart contract[C/OL]//2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2019:1198–1201. DOI: 10.1109/ASE.2019.00136.
- [12] Barboni M, Morichetta A, Polini A. Sumo: A mutation testing strategy for solidity smart contracts[C/OL]//2021 IEEE/ACM International Conference on Automation of Software Test (AST). 2021:50–59. DOI: 10.1109/AST52587.2021.00014.
- [13] Akca S, Rajan A, Peng C. Solanalyser: A framework for analysing and testing smart contracts[C/OL]//2019 26th Asia-Pacific Software Engineering Conference (APSEC). 2019:482–489. DOI: 10.1109/APSEC48747.2019.00071.
- [14] Kong Q P, Wang Z Y, Huang Y, et al. Characterizing and detecting gas-inefficient patterns in smart contracts[J/OL]. Journal of Computer Science and Technology, 2022, 37(1):67–82. <https://doi.org/10.1007/s11390-021-1674-4>.



THANKS





给Solidity社区的建议

- 对App开发人员
 - 在开发基于Solidity的应用程序时谨慎行事
 - 始终使用最新版本的Solidity编译器
- 对静态分析工具开发人员
 - 优先考虑在其工具中支持最流行的特性，暂时忽略使用率为零的特性
- 对Solidity语言设计者
 - 在语言中引入安全且易于使用的特性
 - 弃用危险的特性，引入新的安全特性取而代之
 - 实现更多内置的原生库函数

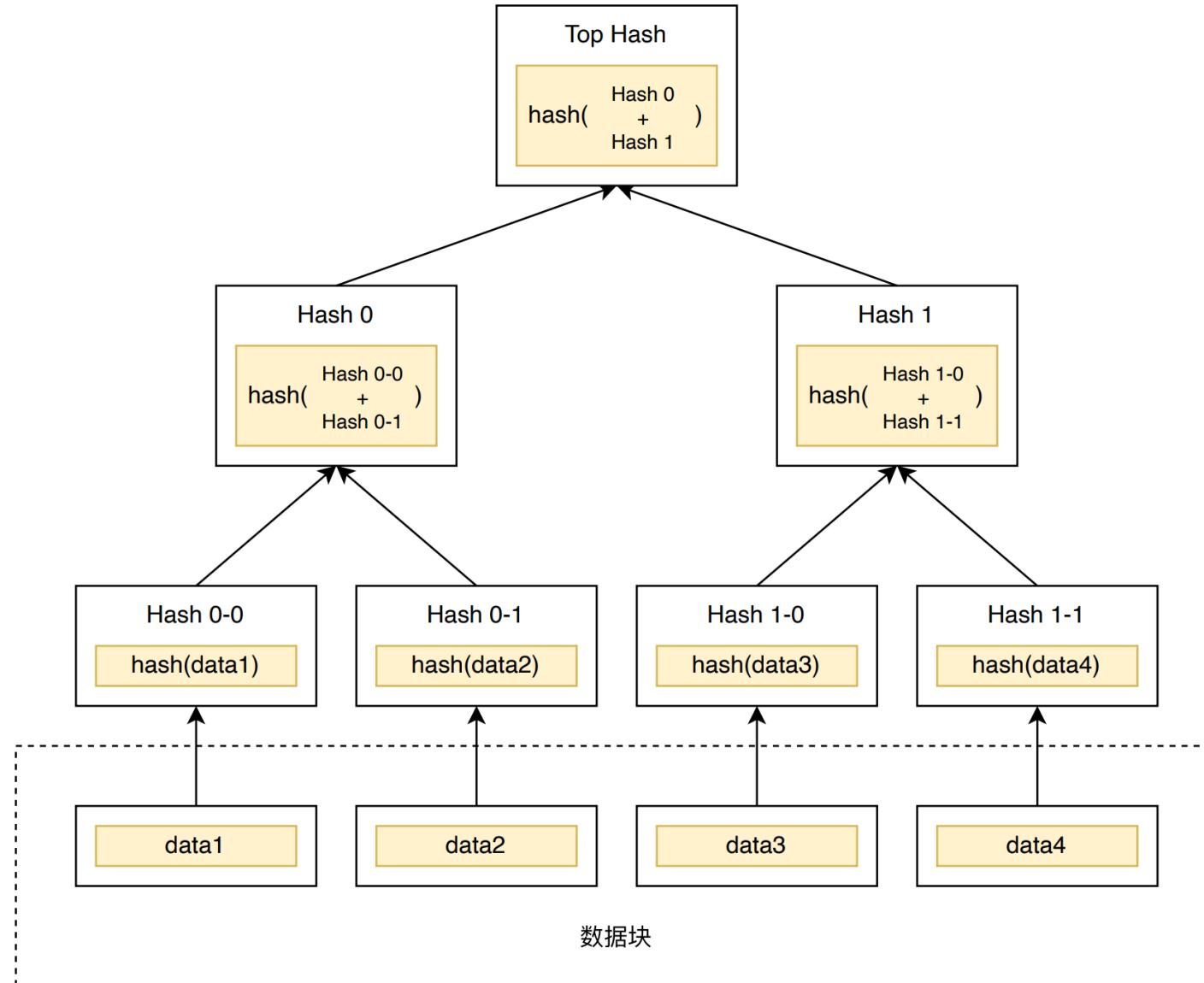


图 1-1 默克尔树

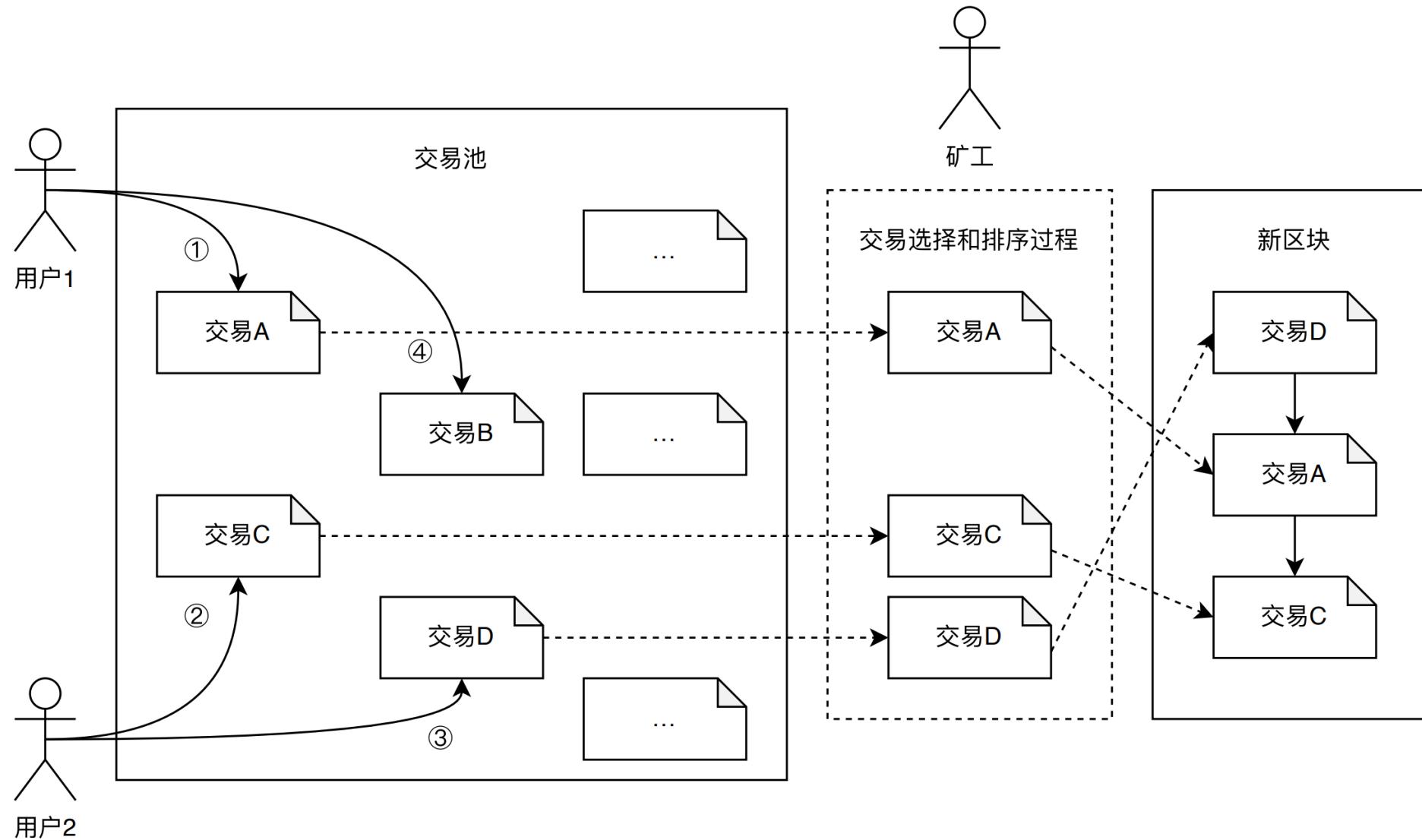


图 4-1 矿工将交易打包进区块的过程示意图



表 4-2 执行涉及到受监控地址的交易时，并行交易执行环境记录的主要信息

字段名称	含义
transaction_hash	交易哈希值
block_number	区块号
transaction_index	此交易在区块内的排列序号
hit_monitored_addresses	这笔交易中受监控的地址
hit_points	涉及受监控的地址的原因
from	交易发起地址
to	交易接收地址
value	转账金额
gas_used	气体消耗量
input	函数调用的输入参数
return	函数调用的返回值
error	错误信息
post_journal	区块链世界状态变化日志



表 4-3 交易涉及受监控的地址的五种方式

字段名称	含义
from	交易发起地址是受监控的地址
to	交易接收地址是受监控的地址
dirty	交易执行过程中，合约 A 修改了合约 B 的状态变量，合约 B 的地址是受监控的地址
callee	交易执行过程中，合约 A 调用了合约 B 的函数，合约 B 的地址是受监控的地址
readee	交易执行过程中，合约 A 读取了合约 B 的代码，合约 B 的地址是受监控的地址



表 4-4 气体查询点信息

字段名称	含义
block_number	区块号
transation_index	此交易在区块内的排列序号
call_chain_index	函数调用链索引
transient_gas	在此次查询剩余气体时，剩余可消耗的气体量
call_type	函数调用类型
to	交易接收地址，可以是外部交易或内部交易的接收地址
value	转账金额



表 4-5 异常点信息

字段名称	含义
block_number	区块号
transation_index	此交易在区块内的排列序号
call_chain_index	函数调用链索引
evm_error	虚拟机内部异常信息
call_type	函数调用类型
to	交易接收地址，可以是外部交易或内部交易的接收地址
value	转账金额



表 4-6 以太坊虚拟机可能发生的异常列表

异常名称	含义
ErrInvalidSubroutineEntry	非法的子函数入口
ErrOutOfGas	气体耗尽
ErrCodeStoreOutOfGas	创建合约存储区气体耗尽
ErrDepth	超过最大调用深度
ErrInsufficientBalance	转账所需的余额不足
ErrContractAddressCollision	合约地址冲突
ErrExecutionReverted	交易回滚
ErrMaxCodeSizeExceeded	字节码超过最大长度限制
ErrInvalidJump	非法的跳转目标
ErrWriteProtection	试图在写入保护生效时修改世界状态
ErrReturnDataOutOfBounds	返回值超出范围
ErrGasUintOverflow	气体消耗量超出uint 64 整数范围



表 4-8 低效率的气体消耗模式表

编号	模式	低效率的气体消耗模式例子	高效率的气体消耗模式例子
1	稀疏存储	<pre>uint8 public decimals; uint256 public totalSupply; address public owner;</pre>	<pre>uint8 public decimals; address public owner; uint256 public totalSupply;</pre>
2	小变量	<pre>uint32 public contractVersion = 20191203; string public contractClass = "xpetoTimestampLogger"; string public xpectoMandator = "xpecto";</pre>	<pre>uint256 public contractVersion = 20191203; string public contractClass = "xpetoTimestampLogger"; string public xpectoMandator = "xpecto";</pre>
3	重复赋值	<pre>contract ProofOfWeakFOMO{ address owner = msg.sender; constructor () { owner = msg.sender; }}</pre>	<pre>contract ProofOfWeakFOMO{ address owner; constructor () { owner = msg.sender; }}</pre>
4	频繁使用状态变量	<pre>contract ShareTokenSale{ uint256 public endTime; function startSale() { for(...){ endTime=endTime.add(x); }}</pre>	<pre>contract ShareTokenSale{ uint256 public endTime; function startSale() { uint256 temp0 = endTime; for(...){ temp0=temp0.add(x); } endTime = temp0; }}</pre>
5	未考虑短路规则	<pre>if((msg.value>=this.balance) && (frozen == false)) { msg.sender .transfer(this.balance); }</pre>	<pre>if((frozen == false) && (msg.value>=this.balance)) { msg.sender .transfer(this.balance); }</pre>
6	不准确的函数可见性	<pre>contract ImmAirDropA{ function signupUserWhitelist (address[] _userlist) public{ ... }}</pre>	<pre>contract ImmAirDropA{ function signupUserWhitelist (address[] _userlist) external{ ... }}</pre>

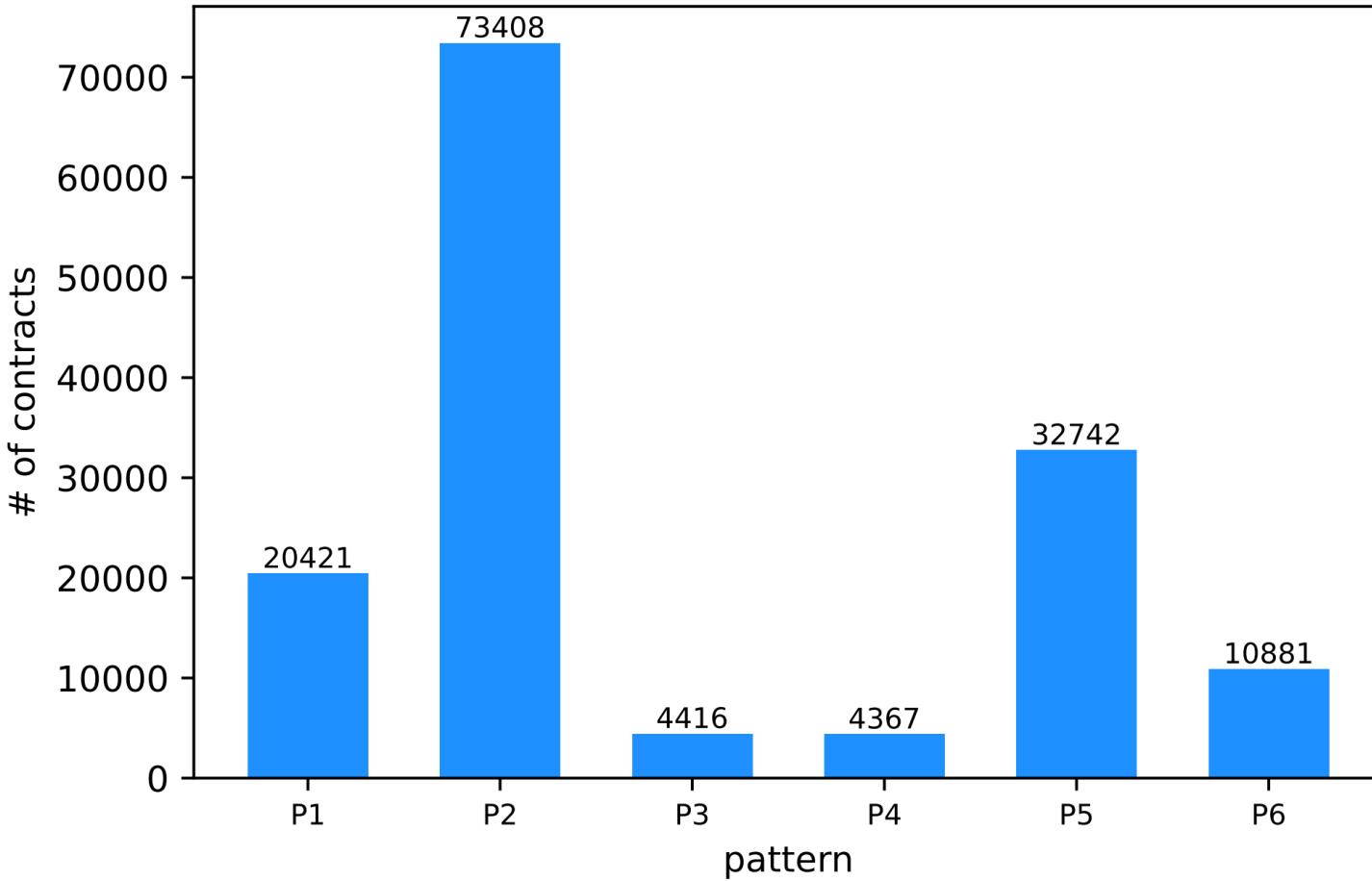


图 4-8 6 种低效率的气体消耗模式在 160301 个智能合约中的流行率



表 4-9 包含 1~6 种模式的合约数量表

合约代码包含几种低效率的气体消耗模式	对应的合约数量
1 种模式	38263
2 种模式	34454
3 种模式	8964
4 种模式	2356
5 种模式	426
6 种模式	103