**智能合约的形式化验证**

短文

Karthikeyan Bhargavan，安托万Delignat Lavaud，Cé德里克·福奈特，122

Anitha Gollamudi，Georges Gonthier，Nadim Kobeissi，Natalia Kulatova，3211

Aseem Rastogi，Thomas Sibut Pinote，Nikhil Swamy，212

还有圣地亚哥·萨内拉-Bé圭林12

因里亚

{karthikeyan.bhargavan，nadim.kobeissi，natalia.kulatova，thomas.sibut pinote}@inria.fr 2 3

                                               哈佛大学微软研究院

{antdl，fournet，gonthier，aseemr，nswamy，santiago}@microsoft.comagollamudi@g.harvard.edu

# 摘要

以太坊是一个加密货币框架，它使用区块链技术提供一个开放的全球计算平台，称为以太坊虚拟机（EVM）。EVM在一个简单的堆栈机器上执行字节码。程序员通常不写EVM代码；相反，他们可以用类似JavaScript的语言编程，称为Solidity，可以编译成字节码。由于EVM的主要目的是执行管理和传输数字资产（称为Ether）的智能合约，因此安全性至关重要。然而，编写安全的智能合约可能非常困难：由于以太坊的开放性，程序和假名用户都可以调用其他程序的公共方法，从而导致可信和不可信代码的潜在危险组合。最近，dao合同遭到攻击，利用EVM语义的微妙细节，将价值约5000万美元的乙醚转移到攻击者的控制之下，这就说明了这种风险。在本文中，我们提出了一个分析和验证以太坊契约的运行时安全性和功能正确性的框架，并将其翻译成一种面向程序验证的函数式编程语言F。*?*

# 1.        导言

|  |
| --- |
| ACM承认此贡献是由雇员、承包商或国家政府的附属机构编写或共同编写的。因此，政府保留非排他性、免版税的权利，仅为政府目的出版或复制本文，或允许其他人这样做。*2016年10月24日，奥地利维也纳*  内政部：http://dx.doi.org/10.1145/2993600.2993611 |

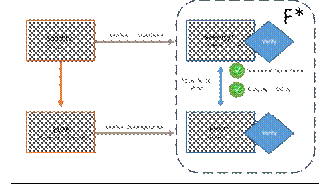
由比特币开创的区块链技术[7]提供了一个全球一致的只附加分类账，不依赖于中央可信机构。在比特币中，这个分类账记录虚拟货币的交易，虚拟货币是通过一个叫做挖掘的过程创建的。在工作证明挖掘方案中，网络的每个节点都可以通过找到一个SHA256哈希值与某个难度阈值匹配的格式化值（其中包括要出现在该块中的所有事务），来获得将下一个事务块附加到分类帐的权利。该系统旨在确保以恒定的速度开采区块：当提交的区块太多太快时，难度增加，从而提高开采的计算成本。

以太坊同样基于工作证明在区块链上创建了一种称为以太的虚拟货币。以太坊的账本明显比比特币更为通用：它以以太坊虚拟机（EVM）字节码的形式存储图灵完整程序，并以对该代码的函数调用的形式启用事务，并以参数的形式提供附加数据。可编程合约还可以访问非易失性存储器和日志事件，两者都记录在账本中。图1显示了以太坊网络中工作流的高级视图。

交易的发起人为交易的执行支付费用，以气体为单位。设法追加一个区块（包括交易）的矿工可以以指定的天然气价格申请转换成乙醚的费用。有些操作比其他操作更昂贵：例如，写入分类帐或启动事务比对堆栈值进行算术运算要贵四个数量级。因此，以太坊可以被认为是一个分布式计算平台，任何人都可以通过支付相关的天然气费用来运行代码（他们的和其他的）。

系统的完整性依赖于大多数矿工的诚实：矿工可能会试图通过不运行程序或错误地运行程序来作弊，但诚实的矿工会拒绝阻止和分叉链。由于最长的链条被认为是有效的（特别是支付费用和区块奖励），每个矿工都受到激励，不作弊，并检查彼此的工作。尽管如此，完整性只确保契约代码按照EVM语义的规定正确执行，而不保证契约的行为符合其程序员的预期，这是本文的重点。

以太坊目前运行的智能合约管理数百万美元，使其安全性高度敏感。例如，最近在TheDAO[2]中利用了一种有充分证据的重入攻击的变体，这是一个实施分散自治风险投资基金的合同，导致价值5000多万美元的以太被盗。回收资金需要硬叉的区块链，违背法律是制度的前提。这样的攻击引发了这样一个问题：在将合同上传到以太坊之前，是否可以通过静态分析[6]来防止类似的bug。



|  |
| --- |
| **图1：以太坊网络中的工作流概述** |

**图2：我们的验证架构概要**

在本文中，我们概述了一个框架来分析和正式验证以太坊智能合约使用F[9]，一个函数式编程语言，旨在程序验证。这种契约通常是用Solidity[3]编写的，一种类似JavaScript的语言，并编译成EVM字节码。但是，分析较低级别的EVM代码也很有用，因为Solidity编译器可能不完全可信，恶意代码可能是用字节码而不是Solidity编写的。对于一个给定的智能合约，获得稳定的源代码和EVM字节码显然是有利的，但是我们在设计我们的架构时假设验证器可能只有字节码。在撰写本文时，112802份合同中只有396份的源代码可以在http://etherscan.io.*?*

我们的智能合约验证体系结构使用了一种双管齐下的方法，如图2所示。我们开发了一种基于语言的方法来验证智能合约，并提供了两个基于F中浅嵌入的原型工具（即，合约被转换为FProgram，该程序调用用于所有以太坊操作的Fruntime库）：*? ? ?*

实体公司实体合同F（第2节）。它允许我们在源代码级别验证功能正确性规范（比如契约不变量）和运行时错误的安全性。*? ?*

EVMdecompiles将EVM字节码编译成更简洁的Fcode，隐藏堆栈机器的细节（第-*? ?*

第3节）。它允许我们分析低级属性，例如完成调用或事务所需的气体量的界限。

Flanguage附带了一个丰富的类型系统，其中包括依赖类型和一元效应，我们将其应用于为SMT解算器生成自动查询，然后可以静态验证契约的此类属性。*?*

本文给出了这两种形式的验证的初步结果，但是在将来，我们计划使用我们的工具来逐个测试Solidity编译器的正确性。给定一个可靠的程序和据称功能等价的EVM字节码，我们将把它们翻译成FAN，并用关系推理验证它们的等价性[1]。*?*

# 2.        从固态到F\*

根据之前关于JavaScript程序基于类型分析的工作[8]的精神，我们提倡一种方法，程序员可以通过将契约嵌入F中来验证契约的高级目标。在本节中，我们将介绍一个将Solidity转换为fan的工具和一个基于Ftypechecking的嵌入式Fcontracts自动分析示例。*?? ? ?*

坚固性计划由许多合同声明组成。一旦编译到EVM，契约将使用一种特殊的帐户创建事务来安装，该事务为契约分配一个地址。与比特币不同的是，在比特币中，地址是帐户公钥的散列，以太坊地址可以毫无区别地引用合同或用户公钥。类似地，事务和方法调用之间没有区别：向契约发送Ether将隐式调用fallback函数（Solidity契约的未命名方法）。

事实上，区块链中已编译的契约由单个入口点组成，该入口点根据传入的事务（被视为接收到的消息消息）决定调用哪个方法代码。Solidity合约的方法可以访问包含合约信息的环境全局变量（例如，this.balance中合约的当前余额）、用于调用合约方法的事务（例如msg.sender中的源地址和msg.value中接收的乙醚量），或者在其中挖掘调用事务的块（例如block.timestamp中的miner的timestamp）。

|  |
| --- |
| hsoliditi:：=（hcontracti）\*hstatementi:：=ε                                                                                                                        |htypei@identifier（&apos;=&apos;hexpressioni）(\*十二月\*）  hContract:：=&apos;合同&apos;@标识符&apos;{&apos;（hsti）\*&apos;}&apos;|&apos;if（&apos;hexpressioni&apos;）&apos;hstatementi（&apos;else&apos;hstatementi）？                   hsti:：=htypedef i | hstatedef i | hmethodi |{&apos;（hstatementi&apos;；&apos;）\*&apos;}&apos;                                                                                                                        |“返回”（hexpressioni）？                      htypedef i:：=&apos;struct&apos;@identifier&apos;{&apos;（htypei@identifier）| &apos;&apos;                          &apos;;&apos;)\* &apos;}&apos; | 己糖加压素                     htypei:：=&apos;uint&apos;|&apos;address&apos;|&apos;bool&apos;六角印刷体i:：=hliterali                   |&apos;映射（&apos;htypei&apos;=>&apos;htypei&apos;）&apos;| hlhs expressioni&apos;（&apos;（hexpressioni&apos;，&apos;）\*&apos;）&apos;                   |@标识符| hexpressioni hbinopi hexpressioni                                                                                                                        |hunopi hexpressioni公司                  hstatedef i:：=htypei@identifier | hlhs expressioni&apos;=&apos;hexpressioni                                                                                                                        |hlhs表达式  hmethodi:：=&apos;函数&apos;（@identifier）？&apos;（）&apos;（hqualifieri）\*                        “”hlhs表达式i:：={                             （&apos;var&apos;（@identifier（&apos;=&apos;hexpressioni）？&apos;，&apos;）+）？|@标识符                           （hstatementi“”）\*“”| hlhs expressioni&apos;[“hlhs expressioni&apos;]”;}                                                                                                                        |hlhs表达式“.”@标识符  hqualifieri:：=&apos;私有&apos;|&apos;公共&apos;|&apos;内部&apos;                   |&apos;返回（&apos;htypei（@identifier）？&apos;）&apos;hliterali:：=h函数i                                                                                                                        |&apos;{&apos;（@identifier&apos;：&apos;hexpressioni&apos;，&apos;）\*&apos;}&apos;                     hbinopi:：=&apos;+&apos;|&apos;-&apos;|&apos;\*&apos;|&apos;/&apos;|&apos;%&apos;|&apos;[&apos;（hexpressioni&apos;，&apos;）\*&apos;]                   | &apos;&&&apos; | &apos;||&apos; | &apos;==&apos; | &apos;!=&apos; | &apos;>&apos; | &apos;<&apos; | &apos;>=&apos; | &apos;<=&apos; | @数字|@地址|@布尔值  胡诺皮：：=&apos;+&apos;|&apos;-&apos;|&apos;！&apos;    **图3:**坚固性*?* |

在这项探索性工作中，我们考虑了一个图3中指定的Solidity的受限子集。值得注意的是，我们考虑的片段不包括循环，但支持递归。契约中的三种主要声明类型是类型声明、属性声明和方法。类型声明由类似C的结构和枚举以及映射（作为哈希表实现的关联数组）组成。尽管属性和方法让人想起了面向对象编程，但这是一个令人困惑的类比：合同是由账户创建事务“实例化”的；这将在全局存储中分配协定的属性并调用构造函数（与协定同名的方法）。尽管有C++和java类似的访问修饰符，契约的所有属性都存储在Enthunm分类帐中，因此，所有契约的内部状态都是完全公开的。方法在EVM中被编译成一个函数，当事务被发送到契约的地址时，该函数将运行。此事务处理程序将请求的方法签名与非内部方法列表匹配，并调用相关方法。当找不到匹配项时，将调用回退处理程序。

## 2.1        翻译成F\*

我们将坚固性翻译为F过程如下：*?*

1.    合同被翻译成模块；*?*

2.    类型声明被转换为类型声明：枚举成为空数据构造函数的和，结构成为记录，映射成为fmap；*?*

3.    所有合同属性都打包在一个state记录中，其中每个属性都是一个freeference；*?*

4.    合同方法转化为函数；*?*

5.    如果重写了具有continuation的语句，这取决于一个分支是以return结束还是以throw结束（移动另一个分支中的continuation）还是不结束（然后在每个分支中复制continuation）。

6.    赋值转换如下：我们保持一个局部、状态和环境全局变量名的环境：将局部变量声明和赋值转换为let绑定；全局变量被库调用替换；状态属性替换为状态类型的更新；

7.    内置方法调用（例如address.send（））被库调用替换。

图4显示了一个实例Solidity契约及其f翻译。翻译增加的唯一类型注释是对合同方法的影响，如第2.2节所述。翻译后的契约使用Solidity库，它定义映射类型（对映射的引用）和相关函数update\u map和lookup。此库还为无符号256位整数和算术运算符定义基数字类型uint。*?*

## 2.2        检测易受攻击模式的效果

图4中的示例说明了Solidity编程的两个主要缺陷。首先，许多合同编写者没有意识到send及其变体不能保证成功（send返回bool）。对于Solidity程序员来说，这种语言特性是令人惊讶的，因为所有其他运行时错误（如气体耗尽或调用堆栈溢出）都会触发异常。此类异常（包括由throw触发的异常）安全地恢复所有事务和对合同属性的所有更改。这不是send的情况：当返回false时，程序员需要手动撤消任何副作用，例如通过写入（！地址发送（x））。**伊夫罗**

MyBank中说明的另一个陷阱是重入性。因为事务也是方法调用，所以将资金转移到-

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **合同**我的银行{  **映射地址uint**(⇒ ) 余额；  **函数msgmsg**存款（）{余额[.sender]+=.value；  }  **函数uint ifmsgmsgmsg**取款（金额）{（余额[.sender]≥ amount）{.sender.call.value（amount）（）；余额[发送方]−= 金额；}  } | 模块MyBank  开放性  类型state={余额：映射地址uint；}val存储：状态={balances=ref empty map}  let deposit（）：Eth unit=更新map store.balances msg.sender  （添加（lookup store.balances msg.sender）msg.value）  让我们退出(amount:uint)：Eth unit=if（ge（lookup store.balances msg.sender）amount）则调用fallback msg.sender amount；更新map store.balances msg.sender |   **函数常量returnsIntreturn msg**Balance（）（{（sub（lookup store.balances msg.sender）amount）余额[.sender]；                   }让平衡（）：Eth uint=               }查找store.balances msg.sender    **图4：一个简单的银行合同，在坚固的翻译成**F*?* |

其他契约（使用send或直接调用fallback方法）是程序控制的转移。事实上，我的银行遭受了一种类似的行为，允许排水道。考虑以下恶意合同：

**合同**恶意的{

**地址uint**业主；数量；我的银行；

**功能**恶意（MyBank）{

bank=银行；所有者=.sender；**味精**

}

**功能**排水管（）{

金额=.value；银行存款价值（金额）（；//转发所有气体组。提取。值（0）（金额）；**味精**

}Claim（）{owner.send（.balance）；}**函数this**

**函数ifmsg**（）{//回退方法（.gas>50000）

银行取款金额（0）（金额）；

}

}

通过在MyBank合同转移的某个点递归调用MyBank合同，该合同可以耗尽MyBank合同的余额。攻击是通过调用Drain方法触发的，该方法在MyBank中存款并立即取款。当draw方法将资金转移回时，它还通过调用其fallback方法将控制权转移回恶意的，而恶意的fallback方法又发出一个可重入调用来提取。第二次取款调用中的if条件仍然满足，因为余额仅在原始调用恶意收益率控制后更新，因此相同数量的乙醚可以多次转移。进一步的重新进入调用仅受提供给初始调用以排放的气体的限制。还要注意，从后续调用返回到恶意的fallback方法将导致余额映射中的条目多次递减，从而导致整数下溢。攻击的一种变体可能会以这种方式破坏余额，然后再从银行自由提取任何剩余资金。请注意，在MyBank中使用send（amount）而不是msg.sender.call.value（amount）可以减轻攻击，因为与方法调用不同，send只为调用分配2300 gas。

使用F的effect系统，我们现在展示了如何检测翻译契约中的一些易受攻击的模式，例如不检查外部调用的结果。基本构造是一个组合的异常和状态monad（有关详细信息，请参见[9]），具有以下签名：*?*

EST（a:Type）=h0:heap//输入堆

→发送failed:bool // 发送失败标志

→Tot（选项a∗heap）//结果和堆，或异常∗ bool）//new failure flag return（a:Type）（x:a）：EST a=fun h0 b0→一些（x，h0），b0

绑定（a:类型）（b:类型）（f:测试a）（g:测试a）→EST b）：EST b=

乐趣h0 b0→

将f h0 b0与

|无，b1→None，b1//f中的异常：无输出

|一些（x，h1），b1→g x h1 b1//运行g，进位标志

此Fcode捕获返回类型a的值的fcomputions的类型。monad带有send\u failed标志，用于记录send（）或外部调用到目前为止是否失败。在这个单子的基础上，可以执行几种不同的样式；例如，您可能希望强制约定在发送失败时总是抛出，方法是在类型系统中编写一个小的引用监视器。例如，我们根据EST定义了以下影响：*? ?*

效应Eth（a:类型）=EST a

        （有趣b0级→not b0）//在非故障状态下启动

（乐趣h0 b0 r b1）→

//发送失败时怎么办=⇒

（将r与

|没有→True//异常

|一些（，h1）→no mods h0 h1））//无写入

标准库将throw的post条件定义为（fun h0 b0 r b1）→b0=b1∧ 是None r）和send as的post条件（fun h0 b0 r b1）→r==一些（b1，h0））。

通过在Eth effect中对提取的方法进行类型检查，我们可以防止危险的模式，例如外部调用之后的无条件状态更新。这些模式在编译时触发Ftypechecking错误。请注意，Eth施加的安全条件不足以防止重入攻击，因为无法保证外部调用前后的状态修改保持契约的功能不变。因此，这种分析有助于检测危险模式和加强故障处理方式，但它不能取代合同的手动规范和证明。*? ?*

## 2.3        评价

尽管我们的工具有局限性（特别是，它不支持许多坚固的句法特征），我们还是能够翻译和打字检查从我们收集的396份合同中的46份https://etherscan.io. 我们已经可以找到几个在Eth效应中无效的。虽然这些结果仍然是初步的，但它们表明，对公布的合同进行更大规模的分析将具有重大价值。

# 3.        将字节码反编译为F\*

在本节中，我们将介绍EVM，一个EVM字节码的反编译器，我们使用它来分析实体源不可用的合同（以太坊区块链中的大多数实时合同就是这样），并检查合同的低级属性。反编译器的第三个用例，我们在本文中没有进一步探讨，就是将evmt与Solidity一起使用，检查Solidity程序和Solidity编译器输出的字节码之间的等价性，这样不仅可以确保编译器不会引入bug，而且在源代码级别验证的任何属性都会被保留。例如，可以使用具有关系精化类型的fw的rF[1]版本来执行这种等价性证明。*?? ? ? ?*

EVM将存储在区块链中的合约字节码作为输入，并将其转换为F表示。反编译器基于符号求值执行堆栈分析，以识别程序中的跳转目标，并检测堆栈不足和溢出。结果是一个等价的Fprogram，直观地说，它在一台拥有无限数量的单个赋值寄存器的机器上运行，我们将这些寄存器转换为函数let绑定。*? ??*

EVM是一种基于堆栈的机器，字大小为256位[10]。字节码程序可以访问以字数组为模型的字寻址非易失性存储器、以字节数组为模型的字寻址易失性存储器和仅附加的不可读事件日志。指令集包括常用的算术和逻辑操作（如加法、异或）、堆栈和内存操作（如PUSH、POP、MSTORE、MLOAD、SSTORE、SLOAD）、控制流操作（如JUMP、CALL、RETURN）、检查环境和区块链的指令（如BALANCE、TIMESTAMP），以及EVM特有的专门说明（如SHA3、创建、自杀）。作为一个特性，指令JUMPDEST用于在协定的代码部分中标记有效的跳转目标，但在运行时表现为NOP。这便于在反编译期间识别潜在的跳转目的地，因为跳转到无效地址会停止执行。

由EVMmarks堆栈单元格所做的静态分析，分为3种类型：1.初始化单元格为Void，2.操作结果为Local，3.推送操作的立即参数为Constant。分析识别可跳转的地址和块，从一个可跳转地址开始并以停止或控制流指令结束的连续代码段（我们将条件语句的分支视为独立的块）。块摘要由其入口点的地址、最终指令以及初始堆栈和最终堆栈的表示形式组成，这些表示形式总结了块对堆栈的影响。入口点可以是0地址、用JUMPDEST标记的地址或条件的直通地址。*?*

在静态分析的基础上，EVMemits使用let绑定中绑定的变量而不是堆栈单元来实现Fcode。许多指令可以通过这种方式消除，因此分析可以准确地计算剩余代码中指令的偏移量。因为被删除的指令可能会产生汽油费，我们通过在代码中插入对burn的调用来精确地跟踪它们的耗油量，burn是一个库函数，其唯一作用是累积汽油费，它配备了一个Ftype，可以完全捕捉这种效果。*? ? ?*

设x 29=pow[0x02uy][0xA0uy]in设x 30=sub x 29[0x01uy]in设x 31=get caller（）in设x 32=land x 31 x 30 in burn 17；//操作码SUB，CALLER，AND，PUSH1 00，SWAP1，DUP2 mstore[0x00uy]x 32；

燃烧9；//操作码PUSH1 20，DUP2，DUP2

mstore[0x20uy][0x00uy]；燃烧9；//操作码PUSH1 40，SWAP1，SWAP2 let x 33=sha3[0x00uy][0x40uy]in let x 34=sload x 33 in burn 9；//操作码PUSH160、SWAP1、DUP2

mstore[0x60uy]x 34；loadLocal[0x60uy][0x20uy]//返回值



**图5：合同中方法的反编译代码，使用气体消耗量进行检测。**余额我的银行

图5显示了从图4中MyBank合同的Balance方法反编译的Fcode；相应的EVM字节码是396字节长。*?*

我们为字节码操作编写了一个参考成本模型，可以用来证明契约方法的气体消耗的界限。作为示例，图6示出了反编译为ft的MyBank契约的入口点的类型注释，该注释说明对Balance函数的方法调用将消耗至多390个单位的气体。然后可以通过Ftypechecking自动释放此注释。*? ?*

val银行：单位→ST字

（需要（乐趣）→选择h mem=0∧ 选择h气体=0∧ 非零（eqw

（div（get calldataload[0x00uy]）（pow[0x02uy][0xE0uy]））

[0xF8uy；0xF8uy；0xA9uy；0x12uy]））//平衡散列（确保（fun h0h1型→选择h1气体≤ 390))

让我的银行（）=

燃烧6；//操作码PUSH1 60，PUSH1 40

mstore[0x40uy][0x60uy]；

*...*

设x28=eqw[0xF8uy；0xF8uy；0xA9uy；0x12uy]x 3英寸烧伤10；//操作码JUMPI如果非零x 28则开始（\*偏移：165\*）

*...* //Balance方法end的反编译代码



**图6:对调用的方法消耗的气体有界的证明。**余额我的银行

# 4.        结论

我们使用Fto验证智能合约的初步实验表明，Fis的类型和效果系统足够灵活，可以捕获和证明合约程序员感兴趣的属性。我们的方法基于现有验证框架中的浅层嵌入和类型检查，便于探索以Solidity和EVM字节码编码的契约的形式化验证。另一方面，专用于这些语言的静态工具可能更易于使用(例如，我们缺少用简单的源代码术语解释Ftypechecking错误的工具。）*? ? ?*

与这项工作并行的是，Luu等人[6]使用符号执行来检测EVM字节码程序中的缺陷，而一个实验性的Why3[5]正式验证后端现在可以从Solidity web IDE中获得[4]。

我们考虑的示例非常简单，因此我们不必在F中编写EVM的完整实现。我们计划完成一个经过验证的引用实现，并使用它来验证Solidity编译器的输出在功能上是否等同于源代码契约。我们在OCaml中实现了EVMand和solidity。使用Finstead实现和验证这些工具的一部分是很有趣的。例如，我们可以尝试证明在EVMis中完成的堆栈和控制流分析是关于堆栈机器语义的。*?? ? ? ?*

# 5.        致谢

我们感谢来自卡利亚里大学的尼古拉·阿特齐、马西莫·巴托莱蒂和蒂齐亚娜·西莫利指出了早期草案第2节攻击代码中的一个缺陷。

# 工具书类

[1] G。巴特，C。福奈特，B。Gr公司´伊戈尔，P.-Y。甲状腺肿。斯瓦米和S。扎内拉-B´埃古林。密码系统的概率关系验证

实现。在第41届ACM SIGPLAN-SIGACT编程语言原理研讨会上，POPL&apos;14，第193-205页。ACM，2014。

[2] 五。巴特林。关键更新：Dao漏洞。https://blog.ethereum.org/2016/06/17/ 关于dao漏洞的关键更新，2016年。

[3] 以太坊。Solidity文档–0.2.0版。http://solidity.readthedocs.io/, 2016.

[4] 以太坊。实体浏览器。https://ethereum.github.io/browser-solidity, 2016.

[5] J.-C。菲利亚ˆ特雷和A。帕斯凯维奇。为什么3-课程与校准者见面的地方。在第22届欧洲编程研讨会上，ESOP&apos;13，第7792卷《计算机科学讲稿》，第125-128页。Springer，2013。

[6] L。卢，D.-H。朱，H。奥利克尔，P。萨克塞纳和A。霍博尔。使智能合约更智能。

密码学ePrint档案，报告2016/63320016。http://eprint.iacr.org/2016/633.

[7] S。中本。比特币：一种点对点的电子现金系统。http://bitcoin.org/bitcoin.pdf.

[8] N。斯瓦米，C。福奈特，A。拉斯托吉，K。Bhargavan，J。陈，P。Strub和G。M。比尔曼。渐进式键入安全地嵌入javascript。2014年出版，第425-438页。ACM，2014年。

[9] N。斯瓦米，C。人权倡议¸tcu，C。凯勒，A。拉斯托吉，

A.Delignat Lavaud，S。森林，K。巴尔加万，

C.福奈特，P.-Y。Strub，M。科尔维斯，J.-K。津多侯´e、 和S。扎内拉-B´埃古林。F\*中的依赖型与多单体效应。在第43届ACM SIGPLAN-SIGACT编程语言原理研讨会上，POPL&apos;16，第256-270页。ACM，2016。

[10] G。木头。以太坊：一个安全的分散式通用交易账本。http://gavwood.com/paper.pdf.