Azure区块链中智能合约的形式化规范与验证

岳鹏王书文杜K。拉希里朔陈荣潘

*德克萨斯大学奥斯丁微软研究微软研究德克萨斯大学奥斯丁*ypwang@cs.utexas.edu shuvendu@microsoft.com shuochen@microsoft.com rpan@cs.utexas.edu

伊西尔·迪利格·科迪出生于伊玛德·纳西尔

# 德克萨斯大学奥斯丁微软天青微软天青

                   isil@cs.utexas.edu coborn@microsoft.com imnaseer@microsoft.com

*摘要*-确保智能合约的正确性对于确保基于区块链的系统的信任至关重要。本文研究了微软提供的企业区块链asa服务Azure区块链工作台中智能合约的安全性。作为这项研究的一部分，我们将智能合约的语义一致性形式化为一个具有访问控制策略的状态机模型，并开发了一个

高度自动化的正式验证器，可以产生证据以及反例。我们已经应用我们的验证器VERISOL来分析Azure附带的所有合同

区块链工作台，其中包括应用程序示例以及一个用于授权证明（PoA）的治理合同。我们在这些已发布的智能合约中发现了以前未知的bug。在修复了这些bug之后，VERISOL能够成功地对所有这些契约执行完全验证。

一。导言

作为维护和保护共享账本的分散和分布式共识协议，区块链被视为对不同领域具有深远影响的颠覆性技术。因此，包括Microsoft、IBM、Amazon、SAP和Oracle在内的主要云平台公司都在提供区块链即服务（BaaS）解决方案，主要针对企业场景，如金融服务、供应链、托管和联合体治理。Gartner最近的一项研究预测，到2030年，区块链的业务增值可能超过3.1万亿美元[16]。

在区块链上运行的程序被称为智能合约。流行的以太坊区块链提供了一种基于底层堆栈的字节码语言，在以太坊虚拟机（EVM）上执行。诸如Solidity和Serpent之类的高级语言已经被开发出来，使传统的应用程序开发人员能够编写智能合约。然而，由于区块链交易是不可变的，智能合约代码中的漏洞具有毁灭性的后果，智能合约中的漏洞导致了一些引人注目的攻击，这些攻击破坏了对底层区块链技术的信任。例如，臭名昭著的TheDAO漏洞[1]导致价值近6000万美元的以太丢失，平价钱包漏洞导致价值1.69亿美元的以太被永久锁定[4]。这些事件的唯一补救办法是硬分叉区块链，并将其中一个分叉恢复到事件发生前的状态。然而，这种补救措施本身是毁灭性的，因为它挫败了区块链的核心价值，如不变性、分散信任和自治。这种情况下，除了编写正确的代码外，智能合约程序员没有其他选择。

出于智能合约代码中错误的严重后果，最近的工作研究了许多类型的安全错误，例如重入性、整数下溢/溢出以及与以太坊上的delegatecalls相关的问题。虽然这些低级错误由于公共区块链上的高可见性事件而引起了广泛关注，但我们认为，BaaS基础设施和企业场景带来了一系列有趣但研究较少的安全问题。

在本文中，我们介绍了我们在Azure区块链环境下对智能合约正确性的研究，Azure区块链是微软提供的一个BaaS解决方案[3]。具体来说，我们关注一个名为Azure Blockchain Workbench（简称Workbench）的云服务[6]、[7]。Workbench允许企业客户轻松构建和部署集成active directory、数据库、web UI、blob存储等的智能合约应用程序。客户实现智能合约应用程序（满足应用程序策略中指定的要求）并将其上载到Workbench。然后将代码部署到底层区块链分类账中，作为端到端应用程序使用。除了客户（应用程序）智能合约之外，工作台系统本身还包括定制底层分布式区块链共识协议的智能合约。Workbench为以太坊区块链的治理提供了一个这样的智能合约，该合约使用授权证明（PoA）共识协议来验证交易。Workbench依赖于PoA治理契约的正确性，在Azure上提供可信的区块链。

Workbench体系结构中的客户契约实现了复杂的业务逻辑，从JSON文件中指定的高级有限状态机（FSM）工作流策略开始。直观地说，工作流描述了（a）一组称为角色的用户类别，（b）契约的不同状态，以及（c）每个状态下仅限于每个角色的已启用操作（或功能）集。高级策略对于围绕状态机抽象设计契约以及指定操作所需的访问控制非常有用。虽然这些状态机在智能合约设计过程中提供了强大的抽象模式，但决定一个给定的智能合约是否忠实地实现了预期的FSM是非常重要的。在本文中，我们将语义一致性检查定义为判断客户契约是否正确地实现以FSM表示的底层工作流策略的问题。给定一个描述工作流和契约C的工作台策略，我们的方法首先构造一个新的契约C0，使得C在语义上符合当且仅当C0没有使任何断言失败。*π π*

为了自动检查智能契约（如C0或PoA治理）中断言的正确性，我们开发了一个新的验证器VERISOL，用于以Solidity形式编写的智能契约。VERISOL是一个通用的坚固性验证器，不依赖于Workbench。验证器将Solidity程序的语义编码到低级中间验证语言Boogie中，并利用精心设计的Boogie验证管道[14]进行验证和反例生成。特别是，VERISOL利用现有的有界模型检查工具CORRAL[24]为Boogie生成断言冲突的见证，并利用Boogie的实际验证条件生成器自动进行正确性证明。此外，VERISOL使用单项式谓词抽象[17]，[22]来自动推断所谓的契约不变量，这对于语义一致性的自动验证至关重要。

为了评估VERISOL的有效性和效率，我们对工作台附带的所有11个示例应用程序以及区块链本身的PoA治理契约进行了实验。VERISOL在这些已发布的智能合约中发现了4个以前未知的缺陷，所有这些缺陷都已被开发人员确认为真正的缺陷（其中许多在编写本文时已修复）。实验结果也证明了VERISOL的实用性，它可以在不费力的情况下对所有固定契约进行完全验证；最值得注意的是，VERISOL可以在平均1.7秒内自动验证11个样本智能合约固定版本中的10个。

贡献。本文的贡献如下：

1） 我们研究BaaS产品Workbench中智能合约的安全性。

2） 我们将Workbench应用程序策略语言形式化，并定义契约和策略之间的语义一致性检查问题。

3） 我们开发了一个新的形式验证器VERISOL的智能合约写在坚实。

4） 我们对Workbench附带的所有合同执行VERISOL评估。这包括所有应用程序示例以及高度信任的PoA治理契约。

5） 我们报告以前未知的错误，已经确认和几个已经修复。

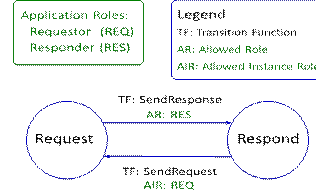


图1.HelloBlockchain应用程序的工作流策略图。

二。概述

在本节中，我们给出了一个示例契约的工作台应用程序策略的示例，并描述了在契约和策略之间进行语义一致性检查的方法。

# A.工作台应用程序策略

Workbench要求每个客户提供一个在JSON文件中表示应用程序高级工作流的策略（或模型）[1]。策略由几个属性组成，例如应用程序名称和描述、一组角色以及一组工作流。例如，图1为一个名为HelloBlockchain的简单应用程序提供了策略的非正式图示[2]. 应用程序由两个全局角色组成（请参阅“应用程序角色”），即请求者和响应者。非正式地说，每个角色代表一组用户地址，并为应用程序公开的各种操作提供访问控制或权限。我们将全局角色与实例角色区分开来，因为后者适用于工作流的特定实例。实例角色通常是与全局角色关联的用户地址的子集。

如图1所示，简单的HelloBlockchain应用程序由一个具有两种状态的工作流组成，即请求和响应。数据成员（或字段）包括在用户地址范围内的实例角色成员（请求者和响应者）。除了构造函数SendRequest和SendResponse之外，工作流还包含两个操作（或函数），这两个函数都以字符串作为参数。

工作流中的转换由开始状态、操作或函数、访问控制列表和一组后续状态组成。图1描述了两种转换，两种状态各一种。例如，如果用户属于响应者角色（AR）并调用动作SendResponse，则应用程序可以从请求转换为响应。“应用程序实例角色”（AIR）是指工作流的实例角色数据成员，该工作流存储全局角色的成员（例如请求者）。例如，图1中从Respond到Request的转换使用AIR，并且仅当用户地址与实例数据变量Requestor中存储的值匹配时才允许。

pragma固体度^0.4.20；

合同HelloBlockchain{

//状态集枚举StateType{请求，响应}

//属性列表StateType public State；

向公众请求者致辞；向公众回应者讲话；字符串公共请求消息；字符串public ResponseMessage；//构造函数HelloBlockchain（字符串消息）

构造函数\u checker（）



公众的

{

请求者=消息发送者；

RequestMessage=消息；

State=StateType.Request；

}

//调用此函数以发送请求函数SendRequest（string requestMessage）

SendRequest\u checker（）公用



{

RequestMessage=请求消息；

State=StateType.Request；

}

//调用此函数以发送响应函数SendResponse（string responseMessage）SendResponse\u checker（）public



{

Responder=msg.sender；

ResponseMessage=响应消息；

State=StateType.Respond；

}

## <修改器定义>

}

图2.HelloBlockchain应用的坚固性合同。

# B.Workbench应用程序智能合约

指定应用程序策略后，用户为相应的区块链分类账提供智能合约，以实现工作流。目前，Workbench支持针对以太坊上的应用程序的流行语言Solidity。图2描述了在HelloBlockchain应用程序中实现HelloBlockchain工作流的Solidity智能合约。在本小节中，我们首先忽略代码中带下划线的部分。契约将配置中的数据成员声明为具有适当类型的状态变量。实现工作流的每个契约都定义了一个附加的状态变量state来跟踪工作流的当前状态。契约包含构造函数和策略中定义的两个函数，以及匹配的签名。这些函数设置状态变量并适当地更新状态变量以反映状态转换。

Workbench服务允许用户上载策略、Solidity代码，并可以选择添加用户和执行配置允许的各种操作。尽管智能合约驱动应用程序，但该策略用于公开给定用户在每个状态下启用的操作集。策略和稳定代码之间的差异可能导致不符合高级策略的意外状态转换。为了确保应用程序的正确功能和安全性，必须验证Solidity程序在语义上是否符合策略配置的预期含义。

# C.语义一致性验证

给定应用程序策略和智能合约，我们定义了两者之间的语义一致性问题，以确保智能合约遵守策略（第III-B节）。此外，我们还将语义一致性验证问题简化为在一个工具化的坚固性程序上检查断言。对于HelloBlockchain应用程序，插装是通过添加图2中带下划线的修饰符调用来提供的。修饰符是一种坚固的语言构造，它允许用执行前后执行的代码包装函数调用。

函数nondet（）返回（bool）//没有定义

//Checker修饰符修饰符构造函数\u Checker（）

{

\_; 断言（nondet（）/\*全局角色请求程序）\*/

==>状态==StateType.Request）；

}

修改器SendRequest\u checker（）

{

StateType oldState=状态；address oldRequestor=请求者；

\_; 断言（（msg.sender==oldRequestor&&oldState==StateType.Respond）

==>状态==StateType.Request）；

}修改器SendResponse\u checker（）

{

StateType oldState=状态；

\_; 断言（（nondet（）/\*全局角色响应程序\*/&&oldState==StateType.Request）

==>状态==StateType.Respond）；

}

图3.插入指令的HelloBlockchain应用程序的修饰符定义。

图3显示了用于指导一致性检查的修饰符的定义。直观地说，我们用检查来包装构造函数和函数，以确保它们正确地实现FSM状态转换。例如，如果FSM在具有访问控制的用户调用函数时从一个状态转换到另一个状态，那么我们将插入的定义，以确保从满足访问控制开始的任何执行都应转换到。*s*1 *s*2 *f交流电fs*1 *交流电s*2

最后，给出了工具化的实体程序，我们使用一个新的实体形式验证器VERISOL静态地释放断言。验证器可以找到反例（以涉及调用构造函数和公共方法的一系列事务的形式）并自动构造语义一致性的证明。请注意，即使简单的HelloBlockchain示例不包含任何无限循环或递归，验证语义一致性仍然需要对涉及对两个公共函数的无限次调用的执行进行推理。我们证明了VERISOL能够为经过良好测试的Workbench应用程序发现严重违反一致性属性的情况，并为Workbench附带的大多数应用程序示例自动构造归纳证明。

三、 工作台策略的语义一致性检查

在本节中，我们将形式化在第二节中非正式介绍的Workbench应用程序策略。我们的形式化可以看作是官方工作台应用程序JSON模式文档的数学表示。

# A.工作台应用程序策略的形式化

应用程序的工作台策略允许用户描述（i）应用程序的数据成员和操作，（ii）应用程序的高级状态机视图，以及（iii）状态转换的基于角色的访问控制。基于角色的访问控制为在开放和对抗环境中部署智能合约提供了安全保障；高级状态机自然地捕获了工作流的本质，工作流基于用户的某些操作在一组状态之间进行。

更正式地说，工作台应用程序策略是一对

（R，W）其中R是一组用于访问控制的全局角色，W是一组定义一种有限状态机的工作流。具体而言，工作流由元组hS、s0、Rw、F、F0、ac0定义，γi其中：

•S是一组有限状态，以及∈ S是初始状态*s*0

•Rw是一组有限的实例角色（id:t），其中是一个标识符，是从R中提取的角色*身份证件t型*

•F（id0，…，idk）是一组动作（函数），F0表示初始动作（构造函数）

• ⊆ R是发起者角色，用于限制可以创建契约实例的用户*交流电*0

• ⊆ S× F× (Rw公司∪ （右）× 2S是一组过渡。给定一个跃迁=（s，f，ac，s），我们分别写表示和*γ τ τ.是的，τ.f、f、，τ.交流电，τ.Ss、 f、acS*

直观地说，S定义了合同可以处于的不同的“状态”，并描述了通过执行某些动作，哪个状态可以过渡到什么其他状态。转换还由角色（可以是全局角色或实例角色）进行“保护”，这些角色限定了允许哪些用户执行这些操作。如前面第二节所述，每个“角色”对应一组用户（即区块链上的地址）。在工作台策略中使用实例角色允许契约的不同实例授权不同的用户执行某些操作。*γ*

# B.语义一致性

给定一个contract C和一个Workbench应用程序策略，C和之间的语义一致性要求contract C忠实地实现由指定的策略。在本小节中，我们首先定义契约的一些语法要求，然后将契约和策略之间的语义一致性形式化。*ππ π*

*句法一致性。*给定一个客户机契约C和一个policy=（R，W），我们的语法一致性要求规定每个∈ W实现所有实例状态变量以及每个函数的定义。此外，每个契约函数都有一个名为的参数，该参数是表示调用该函数的用户或契约的区块链地址。最后，每个契约都应该包含一个状态变量，该变量的范围在Sw之上∈ W。*π w发件人软件w*

*语义一致性。*我们使用Floyd-Hoare三元组形式化了智能合约的语义一致性需求{φ} S码{ψ} 表示以满足状态开始的语句的任何执行都会导致满足状态（如果执行终止）。我们可以将契约C和策略之间的语义一致性定义为一组Hoare三元组，每对（m，s）一个，其中是契约中的一个方法，是工作台策略中的一个状态。在高层，想法很简单：我们坚持认为，当函数沿着转换执行时，产生的状态转换应该与工作台策略一致。*Sφ ψ Sπ 米s*

给定应用程序策略=（R，W）和工作流=hS，s0，Rw，F，F0，ac0，γ我∈ W、 我们可以通过使用以下Hoare三元组来形式化这个高级概念：*π w*

1） 启动。

{发件人∈ ac0}F0（v1，…，vk）{sw=s0}

Hoare三元组声明，创建具有适当访问控制的工作流实例将导致建立初始状态。*交流电*0

2） 连续性。设=（s1，f，ac，S2）为中的过渡。然后，对于每个这样的转换，语义一致性要求以下Hoare三元组有效：*τ γ*

{发件人∈ 交流电∧ sw=s1}f（v1，…，vk）{sw∈ S2}

在这里，前提条件检查两个事实：第一，必须满足访问控制，第二，启动状态必须是。后置条件断言，合同中方法的实现导致了一种根据策略是有效的状态。*发件人s*1*fπ*

# C.语义一致性检查工具

如第二节所述，我们的方法通过（a）使用断言来检测契约，以及（b）使用验证工具来检查断言都不能失败来检查坚固性契约的语义一致性。我们将在本小节中解释我们的检测策略，并让读者参阅第四节，以了解我们的验证工具链的描述。

我们的仪器方法在很大程度上依赖于稳定的修饰符结构。修饰符的语法非常类似于函数定义，具有一个名称和参数列表，以及一个可以在范围内引用参数和全局变量的主体。没有任何参数的修饰符定义的一般结构是[2]：

修饰符Foo（）{

预测试；后stmt；}

其中，前stmt和后stmt是坚固性声明。将此修改器应用于功能栏时，

函数栏（int x）Foo（）{Bar stmt；}

Solidity编译器将Bar的主体转换为在Bar stmt之前（之后）执行pre stmt（分别为post stmt）。这提供了一种方便的方法，可以在过程的多个返回站点注入代码，还可以在执行构造函数代码之前注入代码（因为构造函数可以隐式调用其他基类构造函数）。

在描述实际的检查之前，我们现在定义两个helper谓词。设（ac）是对集合中的成员进行编码的谓词：*P发件人交流电*

                                       *错误，ac*= {}

msgsender=q，ac={q∈Rw}nondet（）ac={r∈R}*.*

 *P*（ac1）∨ P（ac2）ac=ac1∪ ac2型

这里的nondet是一个没有副作用的Solidity函数，它在每次调用时返回一个不确定的布尔值。为了进行静态验证，可以在没有任何定义的情况下声明函数。这允许我们对成员资格检查进行建模∈ 在区块链上没有全球角色的情况下，ac保守。*发件人*

接下来，我们为一组状态S0中契约状态的成员定义一个谓词⊆ S使用（S0）如下：*α*

                                      0

                                          *假*S{}

*α*（S0）={s∈S}*.*

 α(S1）∨ α(S2），S=S1∪S2级

我们现在可以使用这些谓词来定义下面的源代码转换：

*建造师。*我们向构造函数添加以下修饰符：

修饰符构造函数\u checker（）{

\_;

断言（（ac0）⇒ α({（s0}）；*P*

}

这里，断言确保构造函数在由具有访问控制的用户执行时设置正确的初始状态

*交流电*0.

*其他功能。*对于函数，让={τ ∈ γ | τ = (s1，g，ac，S2）}是调用的所有转换的集合。*克γ克. 克*

呼叫（\*）；while（真）{*f*0

如果（\*）呼叫（\*）；else if（\*）调用（\*）；*f*1*f*2

...

else if（\*）调用（\*）；*fn公司*

}

图4.坚固性合同的安全带

修饰符g\u checker（）{

//复制旧状态

StateType oldState=；//复制旧实例角色变量*软件*

... \_; 断言（旧（P(τ.交流）∧ α({τ.s} ））⇒ α(τ.S） ）；五*τ*∈γ*克*

}

这里，插入指令的代码首先将变量和Rw中的所有变量复制到相应的“旧”副本中。接下来，断言检查如果函数是在转换中执行的，那么状态转换到S中的一个后续状态。表示法old（e）用*软件ττ.软件*

“old”拷贝，保存函数入口的值。图3显示了在第二节中描述的正在运行的示例HelloBlockchain的修饰符定义尽管我们显示了nondet（）以突出显示全局角色的问题，但是可以安全地将nondet（）替换为true，因为函数只在任何断言中以负的形式出现。事实上，这个观察也允许我们为运行时检查添加简化的断言。最后，由于连接分布在断言上，我们可以为实现中的每个转换使用一个断言来替换单个断言。

四、 用VERISOL进行形式化验证

在本节中，我们将介绍名为

VERISOL用于检查Solidity智能合约中断言的正确性。因为我们的验证器是建立在Boogie工具链之上的，所以它可以用于验证和反例生成。

# A.一般方法

设C={λx~0.f0，λx~1.f1。。。，λx~n.fn}是一个带有断言的智能合约，其中：

•是建造商*λ十~*0*f*0

•用于∈ [1，n]是公共职能*λx~i.fi公司我*

我们的验证方法基于找到满足以下Hoare三元组的契约不变量I：

|  |  |
| --- | --- |
| |={true}f0{I} | (1) |
| |={I}fi{I}全部∈ [1，n]*我* | (2) |

这里，第一个条件说明契约不变量是由构造函数建立的，第二个条件说明I是归纳的，也就是说，它被C中的每个公共函数所保留。注意，这样的契约不变量足以

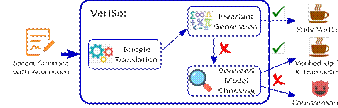


图5.VERISOL的工作流程示意图。

在契约的任何可能的函数调用序列下，建立契约中所有断言的有效性。要了解为什么会出现这种情况，请考虑一个调用C中函数的“工具”，如图4所示。观察上面列出的Hoare三元组（1）和（2）本质上表明I是这个线束中回路的一个归纳不变量；因此，契约不变量I在契约的任何函数调用序列下都过度接近契约的状态。此外，当函数包含断言时，Hoare三元组{I}fi{I}只有在I强到足以暗示断言条件时才能被证明。因此，（1）和（2）中的Hoare三元组的有效性确定了在合同的所有可能使用模式下的正确性。

# B.概述

现在，我们将描述一个名为VERISOL的工具的设计，该工具用于检查智能合约的安全性。VERISOL基于第IV-A节中概述的证明方法，其工作流程如图5所示。在高级阶段，VERISOL采用带有断言注释的坚固性合同C作为输入，并产生以下三种结果之一：

•这意味着C中的断言在任何使用场景下都不会失败。*完全验证：*

•这表明C能够找到至少一个契约函数的输入和调用序列，在该序列下，一个断言肯定会失败。*反驳：*

•当VERISOL既不能验证也不能反驳合同的正确性时，它会执行有界验证，以确定合同在交易前是安全的。这本质上相当于将图4中的“harnese”循环展开几次，然后验证在展开的版本中断言没有失败。*部分验证：kk*

VERISOL由三个模块组成，即（a）Boogie

*翻译*从Solidity程序中，（b）生成不变量以推断契约不变量、循环不变量和过程摘要，以及（c）有界模型检查以探索所有事务中直到用户指定深度的断言失败。在剩下的小节中，我们将更详细地讨论每个组件。*k*

*计算机断层扫描*∈ 合同名称et∈ SolElemTypes：：=整数|字符串|地址

|  |  |
| --- | --- |
| *圣*∈ 土壤类型 | ：：=et | ct | et⇒ 圣 |
| *东南方*∈ ::=*索莱克斯* | *c*|x | op（）|[]*东南方,...,东南方东南方东南方* |
| | | 消息发送器| x.长度 |
| *不锈钢*∈ ::=*溶剂* | x:=| x[]…[]：=y |；*东南方东南方东南方不锈钢不锈钢* |
| | | require（）| assert（）*东南方东南方* |
| | | if（）{}else{}*东南方不锈钢不锈钢* |
| | | while（）做{}| xpush（）*东南方不锈钢.东南方* |
| | | *东南方*：=f（）|：=f（）*东南方~ 东南方东南方.东南方~* |
| | | *东南方*：=新（）|：=新[]（）*C东南方~ 东南方t型东南方* |
| | | *东南方*：=新（t1⇒ t2级⇒ ... ⇒ 塔卡⇒ t） （） |

图6.一个稳定语言的子集。表示合同名称，表示基本实体类型，f表示函数名称。*Ct型*

# C.稳定的布吉翻译

在本小节中，我们正式描述了将Solidity源代码翻译成Boogie中间验证语言的过程。我们先简单描述一下Solidity和Boogie，然后讨论一下我们的翻译。

坚实的语言。图6显示了用于形式化的Solidity的核心子集。在较高的层次上，Solidity是一种类型化的面向对象编程语言，它内置了对基本验证结构的支持，例如表示前置条件的require结构。

核心语言中的类型包括整数、字符串、契约、地址和映射。我们使用符号⇒ τ2表示从类型值到类型值的映射（其中可以是嵌套映射类型）。因为数组可以看作是一种特殊的映射形式⇒ τ, 我们不引入单独的数组类型来简化表示。*τ*1 *τ*1 *τ*2 *τ*2 *整数*

作为标准，一致性表达式包括常量、局部变量、状态变量（即标准面向对象语言术语中的字段）、一元/二元运算符（表示）和数组/映射查找[e0]。给定一个数组x，表达式x.length将生成该数组的长度，msg.sender将生成发起当前函数调用的协定或用户的地址。**op公司***e*

核心Solidity语言中的语句包括赋值、条件语句、循环、顺序组合、数组插入（push）、内部和外部函数调用、契约实例创建和动态分配。构造require用于指定函数的前提条件，assert检查其输入的计算结果是否为true，否则终止执行。坚固性区分了两种类型的函数调用：内部函数调用和外部函数调用。内部调用se:=f（）调用函数f并保持msg.sender不变。外部调用se:=f（）调用se所指向的协定实例中的函数f（可能包括此函数），并将其用作被调用方的msg.sender。*东南方~ 东南方*0*.东南方~* 0

*bbt公司*∈ boogielem类型：：=int | Ref bt∈ BoogieTypes:：=bbt |[bbt]bt

|  |  |
| --- | --- |
| *e*∈ ::=*出口* | *c*|x | op（）| uf（）*e,...,ee,...,e* |
| | | x[e]…[e]|∀我：bbt:：*e* |
| *圣*∈ ::=*Stmts公司* | **跳过**|浩劫x | x:=*e* |
| | | x[e]…[e]：=e |
| | | **假设***e*|断言*e* |
| | | **呼叫***~*x:=（）|；*fe,...,e圣圣* |
| | | **如果**（）{}其他{}而（）做{}*e圣圣e圣* |

图7.布吉语的一个子集。

我们省略了语言的几个方面，这些方面被设计成我们的核心语言。这包括对修饰符、库和结构的相当全面的支持。

布吉语。因为我们的目标是将Solidity翻译成Boogie，所以我们还简要介绍了Boogie中间验证语言。如图7所示，Boogie中的类型包括integers（）、references（）和嵌套映射。表达式（expr）由常量、变量、算术和逻辑运算符（）、未解释函数（uf）、映射查找和量化表达式组成。Boogie中的标准语句（Stmts）由skip、variable和array/map赋值、顺序组合、条件语句和循环组成。x语句将适当类型的任意值赋给变量x。过程调用（x:=（））返回可以存储在局部变量中的值向量。当and语句的参数求值为true时，它们的行为就像no ops，否则终止执行。断言失败被认为是执行失败，而假设失败则会阻止执行。**intRefophavoc调用断言假定***~fe,...,e*

从坚固型到布吉型。我们定义了一个函数：SolTypes→ BoogieTypes，将实体类型转换为Boogie中的类型，如下所示：*µ*

|  |  |  |  |
| --- | --- | --- | --- |
| *µ*（圣） | *.*  = |   **内景***,*    **裁判***,*   裁判， | *圣*∈{整数，字符串}st∈{地址}∪ 合同名称  *.*  *圣*=电子技师⇒ 圣 |

具体来说，我们把实数整数和字符串转换成布尔整数；地址、合同名称和到Boogie引用的映射。请注意，我们在Boogie中将Solidity字符串表示为整数，因为Solidity只允许核心语言中的字符串之间进行相等检查。

从坚实到粗野的表情。我们使用“e”形式的判断，将我们的翻译从Solidity翻译成Boogie表达式，→ χ 在图8中，其中是一个Solidity表达式，是相应的Boogie表达式。当Solidity局部变量和表达式msg.sender分别直接映射到Boogie局部变量和参数时，Solidity中的状态变量被转换为数组查找。具体来说，对于contract的每个状态变量x，我们引入一个映射xfrom contract实例x∈ = 邮件发送者*eχ CCo局部变量v*

（变量1）（发件人）

             `十→ x​​`msg.sender邮件发件人→ v*,,*

# 类型（c）6=字符串x∈ 状态变量（C）

（常数1）*C*[此]（Var2）

                `c级，→ 抄送​​`x→ 十*,*

# 类型（c）=字符串c=哈希（c）`c，→ stroint（c）0

0) （常数2）

`十→ χ*,*

（长度）`X长度→ 长度[χ]*.,*

                             `e1号，→ χ1个`e2，→ χ2

*类型*（e2）=t1*类型*（e1[e2]）=t2

（地图）



图8.将实体表达式编码为Boogie表达式的推理规则。Type（e）是一个函数，它返回Solidity表达式的静态类型。*e*

存储在其状态变量x中的值。因此，在Boogie中，状态变量x的读取被建模为x[this]。接下来，我们使用一个名为stroint的未解释函数将稳定的字符串常量转换为Boogie整数，该函数应用于字符串的散列[3]。如前所述，字符串到整数的转换不会导致不精确，因为Solidity只允许字符串类型的变量之间进行相等检查。*C*

与我们处理状态变量类似，我们的Boogie编码还引入了一个名为Length的数组，将每个Solidity数组映射到相应的长度。因此，坚固性表达式xlength被翻译为Length[χ] x的Boogie编码在哪里。*.χ*

由于潜在的混叠问题，数组/映射查找的转换要复杂一些。首先，基本思想是对于每种类型的地图⇒ t2级*t型*1 ，我们介绍一个布吉地图*τ* 是类型的Boogie编码（即=µ(t1），是类型的Boogie编码（即=µ(t2）*t型*1 *τ τ*0 *t型*2 *τ*0 ). 直观地，将每个数组/映射对象映射到它的内容，这些内容依次表示为一个映射。因此，我们可以认为是一个二维映射，其中第一个维度是实体映射的地址，第二个维度是查找键。对于嵌套映射表达式*e*1 类型⇒ t2其中是一个嵌套的映射/数组，请注意，我们在1中查找，因为映射和数组是动态分配的。直观地说，在我们的编码中，可以动态分配的所有内容都用类型表示，以允许潜在的混叠。*t型*1 *t型*2 *e*1 *Mµ***裁判**（t）)

# 例1.假设契约C有一个状态变量x

*坚固型*映射（int=>int[]），对应于我们的核心Solidity语言中的类型int=>（int=>int）。

*表达式*x[0][1]将被翻译成Boogie表达式[e][1][xC[]][0]*M***内景***哪里e是M这使用图8中的规则。*

从坚实到不羁。图9显示了从Solidity到Boogie语句的转换，使用的是表示Solidity语句被转换为Boogie语句的形式判断。因为图9中的大多数规则都是自解释的，所以我们只解释赋值、函数调用和动态分配的转换。*ω sω*

*函数调用。*Solidity函数有两个隐式参数，即receiver对象的参数和调用者区块链地址的msg.sender。因此，在将Solidity调用转换为相应的Boogie版本时，我们在Boogie版本中显式地传递这些参数。但是，回想一下，implicit msg.sender参数的值根据调用是外部调用还是内部调用而变化。对于内部调用，msg.sender保持不变，而对于外部调用，msg.sender将成为当前的接收方对象。对于这两种类型的调用，我们的翻译引入了一个条件语句来处理动态分派。具体地说，我们的Boogie编码引入了一个映射来存储分配站点上接收器对象的动态类型，函数调用的转换根据接收器对象的内容调用方法的正确版本。*τ τ*

*动态分配。*在帮助程序New（如图10所示）的帮助下，稳定的动态内存分配被转换成Boogie代码，它总是返回一个新的引用。如图10所示，新过程是使用全局映射Alloc实现的，以指示是否分配了引用。所有三种类型的动态内存分配（contract、array和map）都使用这个helper新过程来生成Boogie代码。

在契约创建的情况下（在图9中标记为NewCont），我们生成的Boogie代码除了分配新内存之外，还更新了前面提到的映射。具体来说，给定New返回的新分配的引用，我们将[v]初始化为be，并根据Solidity语义的要求调用的构造函数。*τ vτCC*

接下来，让我们考虑图9中规则NewArr中描述的数组对象的分配。因此，除了分配新内存外，数组分配的转换还更新长度数组，并将数组中的所有元素初始化为零（或null）。

最后，规则NewMap展示了如何使用一个名为MapInit（如图10所示）的辅助Boogie过程将Solidity中的映射分配转换为Boogie，以进行映射初始化。给定一个多维映射，MapInit过程迭代地分配低维映射，并确保存储在映射中的值不会像以前分配的任何其他引用那样彼此别名。*n*

# 例2.固体代码

*十*:= *新建（int=>int=>int）（）*

*转换为以下Boogie代码：*

1    *呼叫v:=新建（）；假设长度[v]=0；*

2    *假设*∀*i：：长度[M***内景***[v] [i]]=0；*

3    *假设*∀*我：：*¬*分配[M***内景***[v] [我]]；*

4    *呼叫NewUnbounded（）；*

5    *假设*∀*分配[**[v] [我]]；*

6    *假设*∀*i、 j：：i=j*∨ MintRef公司*[v] [我]**[v] [j]；*

7    *假设*∀*i、 记者：M***内景***[M[v] [i]][j]=0；*

8    *xC公司[这个]：=v；*

# 首先，我们分配一个新的参考v，并将v的长度和每个内部映射v[i]初始化为零（第1-2行）。其次，我们为每个内部映射v[i]分配新的引用

*（第3-5行），并确保内部映射的引用v*[i] [j]6=j[i][j]*以及v不别名如果我（第6行）。最后，我们初始化每个元素v归零并指定参考v到状态变量x（第7-8行）。*

# D.不变生成

如前所述，将Solidity代码转换为Boogie允许VERISOL利用Boogie周围的现有生态系统，包括高效的验证条件生成[25]。然而，为了完全自动化验证（甚至对于无循环和无递归的契约），我们仍然需要推断一个合适的契约不变量，如第IV-B节所讨论的。具体来说，回想一下契约不变量必须满足以下两个属性：

1） |={true}f0{I}

2） |={I}fi{I}所有∈ [1，n]*我*

VERISOL使用单项式谓词抽象（[17]、[22]、[23]）来自动推断满足上述属性的契约不变量。具体地说，契约不变量推理算法将所有候选谓词的连接作为归纳不变量进行推测，并在无法证明候选谓词归纳的基础上逐步削弱它。这个算法即使在大型的例子中也收敛得相当快，但是依赖于从一个必要谓词的超集开始。在VERISOL的当前实现中，我们通过实例化谓词模板获得候选不变量，其中谓词模板要么相等，要么不相等。在这里，可以用对应于工作台策略中的角色和状态的变量以及常量来实例化表达式。我们发现这些候选谓词对于自动验证工作台契约的语义一致性是足够通用的；但是，其他类型的契约可能需要额外的谓词。*e*1 */电子*2 *./ e*1*，电子*2

# E。有界模型校验

如果VERISOL无法使用单项式谓词抽象来验证契约的正确性，它将使用assertiondirected有界验证器CORRAL[24]来查找导致断言冲突的事务序列。CORRAL通过展开循环时间来分析图4中的工具，并使用抽象细化技术的组合（包括嵌套过程的延迟内联）以可伸缩的方式寻找反例。因此，当VERISOL无法验证属性时，它要么成功地找到一个反例，要么用事务验证是否缺少任何反例。*kk*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | `e1:=e2:=χ2`require（e）`assert（e）*χ*1 **假定断言***χ χ*                           `是的，→ χ ` s1`s2`e段，→ χ ` s*ω*1 *ω*2 *ω*  （条件）（循环）                       `if（e）{s1}else{s2}(χ) {ω1} 其他{ω2} `while（e）do{s}(χ) 做{ω}**如果**  `呃，→ χ雷伊，→ χi=0，…，n新鲜：（）j=1，…，mω ≡ 如果(τ[this]=C1）{呼叫v:=fC（thismsg\u发送方）；χr:=v}否则如果。。。*五希杰<类型这*1*,χ*1*,...,χn.公司，*  **否则如果**(τ[this]=Cm）{呼叫v:=fC（thismsg\u发送方）；χr:=v}*米,χ*1*,...,χn.公司，*  （内部通话） | | | `er:=f（e1，…，en）*ω*  `呃，→ χ雷伊，→ χi=0，…，n新鲜：（e0）j=1，…，mω ≡ 如果(τ[χ0]=C1{呼叫v:=fC(χ0,χ1,...,χn、 本）；χr:=v}否则如果。。。*五希杰<类型*1  **否则如果**(τ[χ0]=Cm）{呼叫v:=fC(χ0,χ1,...,χn、 本）；χr:=v}*米*  （外部调用）  `er:=e0.f（e1，…，en）*ω*                                                               `呃，→ χ雷伊，→ χi=1，…，n新鲜*v*  *ω* ≡ 调用v:=New（）；假设τ[v] =C；呼叫f0C（v，χ1,...,χn、 本）；χr:=v  （NewCont）`er:=新（e1，…，en）*Cω* |  | | `电子  `[]：=e→ χ` 是的，→ χ 新鲜的`v[i]，→ χ十十*.*长度++*ω r,rv我*  （推）≡ 调用v:=New（）；长度[v]：=χ; 假设∀我：：χi=0；χr:=v*ω*                 `X推力（e）*.ω*                                                                                                                         `er:=新[]（e）ω*t型* | （纽瓦尔） | | `哦，呃，→ χr刷新`v[i1]…[in]，→ χ*ω*1 *v*             `s2级≡ 调用v:=New（）；调用MapInit（v，n）；假设∀i1，…，英寸：χ = 0; χr:=v*ω*2 *ω*  （续）  `s1；s2；ω2`er:=新建（t1）⇒ ... ⇒ 田纳西州⇒ t） （）*ω*1*ω* | （新地图） |                        `e1号，→ χ1个`e2，→ χ2个`e，→ χ  （Asgn公司）（Asrt）  图9.将坚固性语句编码为Boogie语句的推理规则。Type（e）是一个函数，它返回Solidity表达式的静态类型。符号表示合同中的功能表示合同的构造者。关系表示子类型关系。是一个全局Boogie映射，它将接收器对象映射到它们的动态类型。为简洁起见，省略了普遍量化的Boogie变量的类型。*efC公司fC<τ* |

变量分配：[Ref]布尔；过程New（）返回（ret:Ref）{

浩劫；假设（！分配[回收]；Alloc[ret]：=真；

}过程NewUnbounded（）{var oldAlloc:[Ref]bool；oldAlloc:=分配；大破坏分配；假设∀i:：oldAlloc[i]==>分配[i]；

}

过程MapInit（v:Ref，n:int）{

var j：整数；j:=1；长度[v]：=0；而（j<n）{

假设∀i1，…，ij：：长度[（v，i1，…，ij）]=0；假设∀i1，…，ij:：¬分配[（v，i1，…，ij）]；调用NewUnbounded（）；假设∀i1，…，ij：：分配[（v，i1，…，ij）]；假定*χχχ*

j:=j+1；}

}

图10辅助布吉程序。术语（v，i1，…，ij）表示实体表达式[i1]…[ij]的翻译结果。为简洁起见，省略了普遍量化的Boogie变量的类型。*χv*

五。评价

我们通过对Workbench附带的智能合约进行两组实验来评估VERISOL的有效性和效率：（i）Workbench样本的语义一致性检查，以及（ii）PoA治理合约的安全性和安全性检查。我们评估的目的是回答以下研究问题：

RQ1当检查工作台应用程序策略的语义一致性时，VERISOL是如何执行的？

RQ2 VERISOL对具有复杂数据结构（如PoA）的智能合约的适用性如何？

实验装置。由于资源有限，我们将每个基准测试的超时时间设置为1小时。所有的实验都是在一台装有英特尔至强（R）E5-1620v3cpu和32GB物理内存、运行ubuntu14.04操作系统的机器上进行的。

;

# A.Workbench应用程序策略的语义一致性

基准。我们在Azure区块链的Github存储库[5]上收集了Workbench附带的所有示例智能合约及其相应的应用程序策略。这些智能契约及其策略描述了在真实企业用例中具有代表性的各种工作流场景。智能合约具有多种稳定特性，如数组、嵌套合约创建、外部调用、枚举类型和相互递归。对于每个智能合约C及其应用程序策略，我们执行如下程序插装*π*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 表一  语义一致性与工作台应用程序策略的实验结果。   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 姓名 | 说明 | 原始  SLOC公司 | 仪器  SLOC公司 | 初始化状态 | 修复后的状态 | 时间（s） | | 资产转让 | 出售高价值资产 | 192 | 444 | 驳斥 | 充分核实 | 2.1 | | 基本环境 | 所有权记录 | 43 | 95 | 充分核实 | 充分核实 | 1.5 | | 集市物品清单 | 销售项目的多工作流场景 | 98 | 175 | 驳斥 | 充分核实 | 2.3 | | 缺陷计数器 | 制造商使用阵列进行产品计数 | 31 | 68 | 充分核实 | 充分核实 | 1.3 | | 数码锁 | 共享数字锁定的文件 | 129 | 260 | 驳斥 | 充分核实 | 1.7 | | FrequerFlyerWards公司 | 使用数组计算飞行常客奖励 | 47 | 90 | 充分核实 | 充分核实 | 1.3 | | 地狱锁链 | 请求和响应（图1） | 32 | 78 | 充分核实 | 充分核实 | 1.3 | | 乒乓球 | 双人游戏的多工作流 | 74 | 136 | 驳斥 | 完全验证（手动） | 2.1 | | 制冷剂运输 | 物联网监测的物源场景 | 118 | 187 | 充分核实 | 充分核实 | 2.2 | | 房间恒温器 | 恒温器的安装和使用 | 42 | 99 | 充分核实 | 充分核实 | 1.3 | | 简单营销场所 | 业主和买方交易 | 62 | 118 | 充分核实 | 充分核实 | 1.4 | | 平均 | - | 79 | 159 | - | - | 1.7 | |

在第III-C节中解释，以获得合同C0。注意，C0的断言失败并不等同于C和之间的语义一致性，因此我们在我们的基准集中包含了这种插入指令的智能契约。*π*

主要结果。表一总结了我们第一次实验评估的结果。在这里，“Name”列给出了合同的名称，“Description”列描述了合同的使用场景。下两列给出了第III-C节所述仪器前后的固体代码行数。最后三列显示了主要的验证结果：特别是，“Init Status”显示了在原始智能合约上应用VERISOL的结果，“Status after Fix”显示了我们手动修复错误（如果有）后VERISOL的结果。修复可能需要更改策略或合同，具体取决于合同作者的反馈。最后，“Time”显示VERISOL在应用于固定契约时的运行时间（以秒为单位）。

我们的实验结果表明，VERISOL对于检查工作台契约和它们应该实现的策略之间的语义一致性是有用的。特别是，VERISOL在11份经过良好测试的合同中发现了4份缺陷，并精确地指出了导致违规的痕迹。我们的结果还表明，VERISOL可以有效地自动化语义一致性证明，因为它可以在修复原始bug之后成功地验证所有契约。此外，除了乒乓球比赛外，11个合同中有10个合同的不变推理技术足以使证明完全按下按钮。契约不变量的候选模板不足以满足pingpongame契约，这主要是由于两个契约之间存在相互递归的函数。这要求我们为相互递归的过程手动提供一个函数摘要，该过程声明由msg.sender表示的发送方契约的状态变量上的不变量（例如，[msg.sender]=s1）∨ sw[msg.sender]=s2，其中是发送方合同的状态）。这说明我们可以实现底层sound-Boogie模块验证的强大功能，用适度的手工来执行非平凡的证明*软件软件硅*

函数Accept（）public

{

如果（msg.sender！=InstanceBuyer&&msg.sender！=实例所有者）{

还原（）；

} ...

如果（msg.sender==InstanceBuyer）{。。。

}

其他{

//msg.sender必须是InstanceOwner//如果（State==StateType.oncentalAcceptance）{State=StateType.SellerAccepted；

}

else if（State==StateType.BuyerAccepted）{

//不一致：JSON转换

//to StateType.SellerAccepted State=StateType.Accepted；

}

} ...

}

图11.资产转移的Buggy函数接受。

头顶。我们目前正致力于将契约不变推断的模板扩展到更丰富的模板，用于推断递归过程的后置条件。

错误分析。我们报告并分析了

VERISOL在Azure区块链工作台示例合同中找到。这些bug可以分为两类：（i）错误的状态转换，和（ii）错误的初始状态。我们简要讨论这两类错误。

*状态转换不正确。*当契约中某个函数的实现违反了策略规定的状态转换时，就会出现此类错误。VERISOL在资产转让和乒乓球比赛合同中发现了此类不符合项。例如，让我们把资产转让[4]作为一个具体的例子。在这个契约中，动作由一个角色或实例角色变量中msg.sender的成员身份来保护（参见图11）。VERISOL发现在Accept函数中从状态BuyerAccepted到状态Accepted的转换在策略中没有匹配的转换。具体来说，当调用函数Accept和msg.sender等于实例角色变量InstanceOwner时，策略允许从BuyerAccepted转换为SellerAccepted。但是，函数Accept的实现将转换为Accepted状态，而不是selleraaccepted状态。从有界验证器的角度来看，这是一个相当深的bug，因为它至少需要6个事务才能到达从初始状态接受的状态BuyerAccepted。

*初始状态不正确。*当智能合约的初始状态没有按照相应策略的指示建立时，就会出现此类错误。我们在DigitalLocker和BazaarItemListing中发现了此类不符合项。例如，DigitalLocker的策略要求请求智能合约的初始状态，但实现最终错误地将初始状态设置为DocumentReview。在BazaarItemListing基准中，开发人员无法设置合同的初始状态，尽管策略要求将其设置为ItemAvailable。

# B.PoA治理契约的安全性

在本节中，我们将讨论将VERISOL应用于PoA治理契约的经验。我们首先介绍了PoA的一些背景知识，然后讨论了实验结果。

PoA治理合同背景。除了应用程序示例之外，Workbench还提供了一个核心智能合约，它构成了Workbench系统堆栈的一个组成部分。PoA是许可区块链的流行工作证明（PoW）共识协议的替代方案，该协议由一组运行该协议的节点组成，并验证要附加到将在分类账上提交的块的交易[9]。验证器属于不同的组织，其中每个组织由一个管理员表示。用于管理员添加、删除、投票和验证程序集管理的协议作为PoA治理契约实现。它实现奇偶校验以太坊的ValidatorSet契约接口，并分布在Azure区块链github上[8]。智能合约由五个组件合约（ValidatorSet、SimpleValidatorSet、AdminValidatorSet、Admin、AdminSet）组成，总计约600行代码。PoA治理契约的正确性支撑了Workbench上的信任以及Azure区块链产品的其余部分。

智能合约使用了一些特性，使其成为可靠智能合约推理的挑战性基准。我们在此概述其中一些：

•契约使用多级继承，因为顶级契约AdminValidatorSet派生自契约SimpleValidatorSet，而契约SimpleValidatorSet又派生自ValidatorSet接口。

•它使用复杂的访问控制，使用坚固性修饰符来限制哪些用户和合同可以改变不同合同的状态。

•契约维护深度嵌套的映射和数组，以存储不同管理员的验证程序集。

•契约使用嵌套循环和过程对数组进行迭代，并使用算术运算对多数投票进行推理。

属性。我们研究了PoA合同的三个关键属性：

P1：至少有一个管理员：网络启动时只有一个管理员，但是管理员的集合永远不会变为空。如果违反此属性，整个网络将进入冻结状态，随后的任何事务都将恢复。

P2:AdminSet的正确性：AdminSet是一个契约，它公开一个set接口来执行常量时间操作，比如查找。由于Solidity不允许枚举映射中的所有键，因此该集合被实现为成员数组addressList和布尔映射inSet的组合，以将成员映射为true。属性检查这两个数据结构之间的耦合—（i）addressList没有重复的元素，（ii）inSet[a]为真，前提是存在addressList[j]==a的索引。*j*

P3：元素删除：从数组中删除元素是PoA合同的常用操作。PoA的正确性依赖于仅对作为数组成员的元素调用此过程。

发现了虫子。为了检查上述三个正确性属性（P1）、（P2）、（P3），我们首先用适当的断言对PoA治理契约进行注释，然后使用VERISOL对它们进行分析。除了发现先前已知的违反“至少一个管理员”属性的情况外，VERISOL还发现了其他一些已经被开发人员确认和修复的bug。特别是，VERISOL发现了一个导致违反属性（P3）的bug：当管理员发布一个事务以从其验证程序列表中删除验证程序时，删除后将发出对event InitiateChange的调用（使用deleteArrayElement）。要保持更改，需要另一个finalizeChange调用。但是，该实现实际上允许两个连续调用InitiateChange，而不需要调用finalizeChange。因此，此错误可能导致PoA协定无法删除启动要删除的验证程序。*十十*

除了检查上述三个属性的手动添加的断言之外，PoA治理契约还包含由原始开发人员添加的其他断言。有趣的是，VERISOL还发现了违反这些原始断言的行为。然而，这些断言失败是由于开发人员错误地使用assert而不是require。尽管require和assert失败都会恢复执行，但Solidity建议仅对违反内部不变量的情况使用assert，这些情况在运行时不会失败。VERISOL发现了五个这样的断言误用实例。

无限验证。与客户端契约的语义一致性检查问题不同，验证PoA契约的属性（P1），（P2），（P3）需要非平凡的量化不变量和关于深度嵌套数组的推理。因此，我们尝试通过手动提出契约/循环不变量和方法前置和后置条件来半自动验证PoA契约。此外，一些属性的归纳证明还需要引入原始代码中不存在的ghost变量。在PoA治理契约中对这些属性的全自动验证是一个雄心勃勃但令人兴奋的未来工作领域。

六、 相关工作

在本节中，我们将讨论在确保智能合约的安全性方面的前期工作。现有的智能合约安全技术可以大致分为不同的类别，包括寻找易受攻击模式的静态方法、形式化验证技术和运行时检查。此外，在形式化语言（如K框架）中对EVM的语义进行了形式化的研究[20]。最后，有几项工作讨论了智能合约中漏洞的调查和分类[13]、[26]、[28]。

*静态分析。*静态分析工具集基于数据流分析或符号执行的选择，以发现已知易受攻击模式的变体。这种模式包括使用可重入性、事务排序依赖性、将以太发送到可能导致以太丢失的无约束地址、使用块时间戳、错误处理异常、在无约束地址上调用自杀等。基于符号执行的工具包括Oyente[26]、MAIAN[28]、Manticore[10]，和Mythril++[11]。另一方面，也存在一些基于数据流的工具，如Securify[29]和Slither[12]。最后，MadMax工具[18]执行静态分析，以发现与气体不足异常相关的漏洞。这些工具既不检查语义一致性，也不验证断言。相反，它们大多找到已知易受攻击模式的实例，并且不提供任何可靠性或完整性保证。另一方面，VERISOL不需要考虑气体消耗，因为它分析了坚固性代码，并且它还需要将漏洞表示为正式规范。

*正式验证。*F\*[15]和Zeus[21]使用形式验证来检查智能合约的正确性。这些方法将可靠性分别转化为形式化验证语言F\*和LLVM，然后应用基于F\*的验证器和约束horn子句求解器来检验翻译程序的正确性。尽管基于F\*的方法是相当有表现力的，但是该工具只覆盖了一小部分没有循环的可靠度，并且需要大量的用户指导来证明用户指定的断言。宙斯的设计与VERISOL的相似之处在于，它将Solidity转换为一种中间语言，并使用基于SMT的求解器来解决验证问题。然而，这两部作品在能力上有一些不同之处。首先，本文的主要贡献之一是智能合约的语义一致性检查问题，而Zeus没有解决这个问题。第二，与我们对Boogie翻译的正式处理不同，Zeus只提供了对LLVM翻译的非正式描述，没有在嵌套数组和映射存在的情况下定义内存模型。不幸的是，我们无法获得宙斯的副本来尝试我们的例子，这使得我们很难在Solidity代码中进行释放断言的实验性比较。

*其他方法。*除了静态分析器和形式化验证工具之外，还有其他方法通过借鉴线性化的思想在运行时强制执行安全的可重入模式[19]。与本文相关的另一项工作是FSolidM[27]，它提供了一种使用有限状态机指定智能契约的方法，其中的操作是用Solidity编写的。尽管它们的状态机模型与我们的工作台策略有相似之处，但它们不考虑访问控制，并且操作没有嵌套的过程调用或循环。最后，FSolidM工具不提供任何静态或动态验证支持。

七。结论

在这项工作中，我们描述了在工业环境中智能合约的自动形式验证的第一个用途。我们为工作台应用程序配置提供了形式化的语义，并执行了自动程序插装来执行这些规范。我们描述了一个使用Boogie工具链的形式化验证工具VERISOL，并举例说明了它在非平凡智能合约验证和bug发现中的应用。在不久的将来，我们将致力于增加常用企业工作流中使用的Solidity语言的更多特性，并探索更复杂的推理，以推断更复杂的契约不变量。

参考文献

[1] 扎实地向初学者讲解道的功用。https://medium.com/\空间因子\@m{}MyPaoG/

explaining-the-dao-exploit-for-初学者-in-solidity-80ee84f0d470，2016年。

[2] 坚固性：功能修饰符。https://solidity.readthedocs.io/en/v0.4.24/ 合同结构.html#函数修饰符，2016。

[3] Azure区块链。https://azure.microsoft.com/en-us/solutions/ 区块链/，2017年。

[4] 平价：把1.69亿美元以太坊放在冰上的虫子？是的，它已经在待办事项清单上好几个月了。https://www.theregister.co.uk/2017/11/16/parity\_ 缺陷未修复，2017年。

[5] workbench的应用程序和智能合约示例。https://github.com/Azure-Samples/blockchain/tree/master/ 区块链工作台/应用程序和智能合约样本，2018年。

[6] Azure区块链内容和示例。https://github.com/ Azure样本/区块链，2018年。

[7] Azure区块链工作台。https://azure.microsoft.com/en-us/features/ 区块链工作台/，2018年。

[8] 蔚蓝的以太坊。https://github.com/Azure-Samples/blockchain/tree/ azure上的master/ledger/template/ethereum，2018年。

[9] azure上以太坊的权威证明。https://azure.microsoft.com/en-us/ 2018年azure/上的博客/以太坊权威证明。

[10] 曼蒂科尔。https://github.com/trailofbits/manticore, 2018.

[11] Mythril classic：用于以太坊智能合约的安全分析工具。https://mythril.ai, 2018.

[12] 固体源分析仪。https://github.com/trailofbits/slither, 2018.

[13] Nicola Atzei、Massimo Bartoletti和Tiziana Cimoli。对以太坊智能合约（sok）攻击的调查。《安全与信任原则——第六届国际会议》，波斯特，2017年4月22日至29日，瑞典乌普萨拉，会议记录，第164-186页，2017年。

[14] 迈克尔·巴奈特、博尔·尤埃文·张、罗伯特·德林、巴特·雅各布斯和K。鲁斯坦M。莱诺。Boogie：面向对象程序的模块化可重用验证器。组件和对象的正式方法，第四届国际研讨会，FMCO 2005，荷兰阿姆斯特丹，2005年11月1-4日，修订讲座，第364-387页，2005年。

[15] Karthikeyan Bhargavan，安托万Delignat Lavaud，Cé德里克·福内特、阿妮莎·戈拉穆迪、乔治·冈蒂尔、纳迪姆·科贝西、纳塔利娅·库拉托娃、阿塞姆·拉斯托吉、托马斯·西布特·皮诺特、尼基尔·斯瓦米和圣地亚哥·赞内拉

        Bé盖林。智能合约的形式化验证：短文。在

*2016年ACM编程语言与安全分析研讨会论文集，PLAS@CCS 2016年10月24日，奥地利维也纳*，第91-96页，2016年。

[16] 计算机世界。到2023年，区块链将产生超过106亿美元的收入。https://www.

computerworld.com/article/3237465/enterprise-applications/blockchain-to-generate-more-106b-in-revenue-by-2023.html。

[17] 科马克·弗拉纳根和K·鲁斯坦·M·莱诺。Houdini是esc/java的注解助手。欧洲正式方法国际研讨会，第500-517页。施普林格，2001年。

[18] 内维尔·格雷奇、迈克尔·孔、安东·尤里塞维奇、莱克西·布伦特、伯恩哈德·斯科尔斯和扬尼斯·斯马拉达基斯。Madmax：以太坊智能合约中没有汽油的情况下生存。PACMPL，2（OOPSLA）：116:1–116:27，2018年。

[19] 谢莉·格罗斯曼、伊泰·亚伯拉罕、盖伊·戈兰·盖塔、扬·米查列夫斯基、诺姆·里涅茨基、穆利·萨吉夫和约尼·佐哈尔。智能合约应用中有效无回调对象的在线检测。PACMPL，2（POPL）：48:1–48:28，2018年。

[20] 埃弗雷特·希尔登布兰特、马纳斯维·萨克塞纳、尼桑·罗德里格斯、朱小冉、，

菲利普戴安，德怀特古思，布兰登M。Moore、Daejun Park、Yi Zhang、Andrei Stefanescu和Grigore Rosu。KEVM：以太坊虚拟机的完整形式语义。在第31届IEEE计算机安全基础研讨会上，CSF 2018，英国牛津，2018年7月9日至12日，第204–217页，2018。

[21]Sukrit Kalla、Seep Goel、Mohan Dhawan和Subodh Sharma。宙斯：分析智能合约的安全性。在第25届年度网络和分布式系统安全研讨会上，NDSS 2018，美国加利福尼亚州圣地亚哥，2018年2月18-21日。

[22]Shuvendu Lahiri和Shaz Qadeer。单项式和小句谓词抽象的复杂性和算法。在国际自动扣除会议（2009年）。Springer Verlag，2009年3月。

[23]Shuvendu K。Lahiri、Shaz Qadeer、Juan P。加洛蒂，简W。冯，还有

托马斯·维斯。模块内推理。在计算机辅助验证中，

*第21届国际会议，CAV 2009，格勒诺布尔，法国，2009年6月26日至7月2日*，第493–508页，2009年。

[24]阿卡什·拉尔、沙兹·卡迪尔和舒文杜·K。拉希里。可达模理论的求解器。计算机辅助验证-第24届国际会议，CAV 2012，加州伯克利，美国，2012年7月7日至13日会议记录，第7358卷，第427-443页。斯普林格，2012。

[25]克。鲁斯坦M。莱诺。最薄弱的前提条件。信息处理。Lett.，93（6）：281–2882005年。

[26]卢伊卢乌、杜希普朱、赫里希·奥利克尔、普拉蒂克·萨克塞纳和阿奎那·霍博尔。使智能合约更智能。《2016年ACM SIGSAC计算机与通信安全会议记录》，奥地利维也纳，2016年10月24-28日，第254-269页。

[27]Anastasia Mavridou和Aron Laszka。工具演示：用于设计安全以太坊智能合约的Fsolidm。安全与信任原则-第七届国际会议，2018年后，作为欧洲软件理论与实践联合会议的一部分，ETAPS 2018，希腊塞萨洛尼基，2018年4月14-20日，会议记录，第270-277页，2018年。

[28]Ivica Nikolic、Aashish Kolluri、Ilya Sergey、Prateek Saxena和Aquinas Hobor。发现贪婪的，挥霍的，自杀性的合同规模。

第34届计算机安全应用年会论文集，ACSAC 2018，圣胡安，PR，USA，2018年12月3-7日，第653-663页，2018年。

[29]Petar Tsankov、Andrei Marian Dan、Dana Drachsler Cohen、Arthur Gervais、Florian Bünzli和Martin T。维切夫。安全化：智能合约的实用安全分析。《2018年ACM SIGSAC计算机与通信安全会议记录》，CCS 2018，加拿大多伦多，2018年10月15-19日，第67-82页。

[[1]](" \l "_ftnref1" \o ")https://docs.microsoft.com/en-us/azure/blockchain/workbench/ 配置

[[2]](" \l "_ftnref2" \o ")示例相关详细信息可在相关网页上找到：https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench/应用程序和智能合约示例/hello blockchain

[[3]](" \l "_ftnref3" \o ")我们假设散列函数是无冲突的。在我们的实现中，我们通过保持从每个字符串到计数器的映射来实现这一点。

[[4]](" \l "_ftnref4" \o ")https://github.com/Azure-Samples/blockchain/tree/master/ 区块链工作台/应用程序和智能合约示例/资产转移