印达维语

工程数学问题卷2020，文章编号6191537，15页https://doi.org/10.1155/2020/6191537

*研究文章*

**Lolisa：Solidity子集的形式语法和语义**

**数学工具Coq中的程序设计语言**

# 郑阳还有杭雷

*中国电子科技大学信息与软件工程学院，成都市建设北路2段4号，邮编：610054*

通信地址应为郑阳；zhengyang@std.uestc.edu.cn

2020年8月5日收到；2020年11月11日修订；接受日期：2020年11月17日；2020年12月1日出版

学术编辑：Mohamed Shaat

版权© 2020郑阳和杭雷。这是一篇根据《知识共享署名许可证》发行的开放获取文章，允许在任何媒体上不受限制地使用、分发和复制，前提是原著被正确引用。

区块链智能合约的安全性是研究者最感兴趣的新兴问题之一。本文提出了一种用于Coq中基于以太坊的智能合约形式化验证的中间规范语言Lolisa。Lolisa的正式语法和语义包含了为以太坊区块链平台开发的Solidity编程语言的一大部分。为了增强类型安全性，Lolisa的形式语法采用了比Solidity更强的静态类型系统。此外，Lolisa还包含了大量稳定语法组件以及通用编程语言特性。因此，Solidity程序可以通过逐行对应直接翻译成Lolisa。Lolisa本质上是可推广的，可以扩展到其他编程语言。最后，将Lolisa的语法和语义封装为数学工具Coq的解释器。因此，用Lolisa编写的智能合约可以象征性地

在Coq中执行和验证。

# 1.  介绍

区块链平台[1]是新兴技术之一，旨在解决各种不同的问题，例如与加密货币[2]和分布式存储[3]相关的问题。目前，这项技术已引起金融界的兴趣[4]。以太坊是应用最广泛的区块链系统之一。以太坊最重要的特性之一是它实现了一种非常灵活的通用图灵完整编程语言，称为Solidity[5]。这允许开发任意应用程序和脚本，这些应用程序和脚本可以在以太坊虚拟机（EVM）表示的虚拟运行时环境中执行，以自动执行区块链事务。这些应用程序和脚本（即程序）统称为智能合约，已广泛应用于许多关键领域，如医疗[6]和金融领域。智能合约的使用越来越多，导致对其安全性的审查也越来越严格。智能合约可以包含特定的属性（即bug），使它们容易受到蓄意攻击，从而导致直接经济损失。一些针对智能合约的最大攻击是众所周知的，例如针对分散自治组织（DAO）和平价钱包[7]合约的攻击。事实上，智能合约中存在许多种类的细微缺陷，从事务排序依赖到错误处理的异常[8]。

本文利用了我们过去的工作，为Solidity版本0.4的一大部分定义了形式语法和操作语义。此子集在本文中表示为Lolisa，并具有以下特征。

*一致性*根据Solidity文档，Lolisa将大多数Solidity类型、操作符和机制形式化。因此，用Solidity编写的程序可以被翻译成Lolisa，反之亦然，一行一行地对应，而无需重建或抽象，这些操作会对一致性产生负面影响。

*静态式系统*Lolisa中的形式语法是使用广义代数数据类型（GADTs）[9]定义的，它将静态类型注释赋予Lolisa的所有值和表达式。这样，Lolisa在检查程序结构方面有一个比Solidity更强的静态类型系统。

*可执行和可证明*与为高级编程语言构建形式语法和语义的类似工作不同，Lolisa的形式语义是基于GERM框架和EVI定义的。因此，在理论上，使用Lolisa编写的基于以太坊的智能合约在与基于GERM框架开发的正式解释器结合使用时，可以直接在高阶逻辑定理证明助手中象征性地执行并同时自动验证其属性。

*机械化和验证*Lolisa的语法和语义是使用Coq证明助手机械化的[10]。我们还开发了一个正式的Coq验证解释器来验证Lolisa是否满足上述可执行和可证明的特征以及语义的元属性。关于我们的正式口译员的实现细节已经在另一篇论文中给出了[11]。

本文的其余部分结构如下。第二节介绍了编程语言形式化的相关工作。第3节介绍了规范语言框架的总体结构，并提供了Lolisa语法和语义的预定义。第四节阐述了Lolisa的形式抽象句法，并将其与Solidity的形式抽象句法进行了比较。第5节介绍了Lolisa的形式化动态语义，包括程序执行语义和EVM内置数据结构和函数的形式化标准库。第6节描述了Lolisa编程语言及其语义在形式验证解释器FEther中的集成。第7节讨论了我们目前工作的贡献和局限性。最后，第8节给出了我们工作的结论。

# 2.  相关工作

采用Manticore[12]和Mythril[13]等静态和动态分析工具的软件工程技术尚未被证明能有效地提高智能合约的可靠性。

KEVM[14]是使用K框架编写的EVM的形式化语义，类似于Lem[15]中进行的形式化。KEVM是可执行的，因此可以运行由EthUM基金会提供的验证测试套件。为KEVM程序进行的符号推理涉及到在可达性逻辑中指定属性并用单独的分析工具验证它们。虽然它们代表了当前可用的操作语义、公理语义的机械化形式化，以及EVM和Solidity字节码的低级编程验证工具[16]，但它们并不适合高级编程语言，如Solidity。作为回应，以太坊社区公开呼吁正式验证方案[17]，作为制定正式验证策略的共同努力的一部分[18]。模糊测试是一种高效的测试技术。目前，许多项目在智能合约中开发模糊技术来分析漏洞，如ReGuard[19]。Securify[20]是一种基于以太坊的基于静态分析的智能合约安全分析器。它在以太坊虚拟机字节码级别根据给定的安全属性验证目标智能合约的行为。Securify提供了一种特定于领域的语言，可以根据攻击报告和基本实践编写安全属性。MadMax[21]是一个静态程序分析框架，它以以太坊字节码为分析源代码，自动分析整数和内存溢出漏洞等常见漏洞。此外，它是第一个允许由动态属性定义循环规范的工具。通过这种方式，该工具可以避免验证过程中的循环爆炸。与OYENTE类似，Ehtir[22]也是一种基于规则的以太坊智能合约字节码静态分析器。该工具可以生成控制流图，并包含所有可能的执行地址。VeriSolid[23]是一个正式的验证框架，可以通过web直接访问。它的基本概念是FSolidM[24]。简而言之，VeriSolid提供了一个形式化验证框架，它提供了一种半自动开发智能合约的正确形式化规范的方法。Abdellatif和Brousmiche[25]提出了一种新的方法，可以基于形式化的模型检查语言对目标智能合约的执行行为进行建模。该技术可以通过模型检验的方法来验证目标智能合约的执行行为和权威性。

在计算机科学的其他领域，许多有趣的研究集中于为不同的高级编程语言开发操作语义的机械化形式化。Park项目[26]在JavaScript语言中给出了完全形式化的表示语义和相应的语法。CompCert项目[27]是另一个对C和GCC有影响的验证工作，它为表示为Clight的C的子集开发了一个形式化语义。这项工作构成了VST[28]和CompCertX[29]的基础。此外，在CompCert项目的基础上，对操作系统进行了一些有趣的形式验证研究。此外，还研究了JavaScipt的操作语义[30]，这对于本研究特别重要，因为Solidity是一种类似JavaScipt的编程语言。然而，这些相关著作中所定义的框架很少能直接用高阶逻辑定理证明助手来象征性地执行或分析。

# 3.  基本概念

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 图1:Lolisa架构概述。  表1：助手函数。   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 符号 | 定义 | 符号 | | 定义 | | 地图地址 | 搜索映射类型的索引地址 | 地图 | | 获取映射类型项中的值 | | EvalBoop公司 | 计算二进制运算表达式 | 蒸发 | | 计算一元运算表达式 | | memsfind公司 | 搜索所需的结构成员 | 环境检查 | | 验证当前环境 | | 环境变量 | 改变当前环境 | 继承 | | 验证继承信息 | | 初始化变量 | 初始化变量地址 | 初始化 | | 初始化函数返回地址 | |  | 表2：国家娱乐 | 行动。 | |  | | 符号 | 定义 | | 符号 | 定义 | | *M* | 内存空间 | | *ε* | 环境信息 | | Λ | 内存地址集 | | Λ乐趣 | 函数返回地址 | | Σ | 结构信息 | | Θ | 结构指针集 | | Γ | 上下文结构信息 | | Ω | 本机值集 | | Φ | 功能信息 | | F | 整体形式系统 | |  |  |  |  |  | |

Lolisa的总体架构如图1所示。表1总结了动态语义定义中使用的helper函数。表2列出了用于从程序的当前状态计算常用值的状态函数。所有这些都表明在下面的讨论中将遇到函数。具体国家的组成部分将用有关国家所订的适当希腊字母表示。如表2所示，形式内存空间的上下文表示为M，其中σ 用于表示特定的存储器状态；执行环境的上下文表示为ε; 我们分配Λ 表示一组内存地址，其中元变量α 用于表示任意地址。类似地，我们定义函数returnaddressΛ很有趣。此外，struct是Lolisa中一种重要的数据结构。因此，我们采用Σ 表示Lolisa struct信息上下文，以及Θ 用于表示结构类型的指针集。另外，以下类型的赋值可能包含变量，因此我们的类型将包含对变量类型上下文的引用，我们将其表示为Γ, Γ1等。这样的上下文是从变量名到类型的有限映射。因为程序可能还包含对Solidity程序的声明函数的引用，所以需要从函数标识符到类型的另一个映射。这个映射将简洁地表示为Φ, Φ1等。

此外，我们分配Ω 作为基本逻辑系统的本机值集。为简洁起见，在下面的讨论中，我们将指定F来表示Σ, Γ, Θ, Ω, Φ, 以及Λ. 由于篇幅的限制，我们的在线报告中提供了洛丽萨的形式化细节(https://arxiv.org/abs/1803.09885).

# 4.  Lolisa的形式语法

|  |
| --- |
| 图2:Lolisa类型的抽象语法。 |

*4.1.类型。*Lolisa类型的形式抽象语法如图2所示。支持的类型包括算术类型（各种大小和有符号的整数）、字节类型、数组类型、映射类型以及函数类型和结构类型。尽管Solidity是一种类似JavaScript的语言，但它支持指针引用。因此，Lolisa还包括基于标签地址规范的指针类型（包括指向函数的指针）。此外，这些类型的注释和相关组件可以通过Coq或其他高阶逻辑定理证明助手中的归纳枚举来容易地形式化。Lolisa不支持任何类型限定符，例如const、volatile和restrict，这些限定符在解析过程中被简单地删除。

这些类型在Lolisa中扮演两个角色。首先，它们作为语句中标识符的类型声明，其次，它们作为签名来指定用于传输类型信息的GADTs样式的值和表达式构造函数，这将在下面的部分中进行解释。在Coq形式化中，术语τ 根据规则1声明为类型，如下所示：

*τ*：类型(1)

注意，许多类型在图2中被递归地定义为参数化类型。这样，一个特定的类型依赖于指定的参数，可以抽象和表示许多不同的实体类型。

可靠度最重要的数据类型之一是映射类型。在Solidity文档[4]中，映射类型被声明为mapping（KeyType）⇒值类型）。这里，除了映射、动态大小的数组、协定和结构之外，KeyType几乎可以是任何类型。如图2所示，\_KeyType被定义为Tmap(τ地图，τ), 哪里τmap表示\u键类型和τ 表示\u ValueType。保持Lolisa中的术语类型良好并确保类型安全的最佳方法是保持类型隔离，而不是添加推论条件。因此，我们为映射中使用的\u KeyType定义了一个坐标类型typemap。特别是，Lolisa中的地址类型被视为一种特殊的结构类型，因此允许\u KeyType成为Lolisa中的结构类型。在Coq形式化中，typemap与typemap共享同一个构造器（Tmap除外），typemap类型的术语被记录为τ根据规则2绘制地图follows:.

*τ*地图：类型地图*.* (2)

在Solidity中，数组类型，根据数组索引idarray定义为Tarray（idarray，τ) 在Coq中，可以分为固定大小数组和动态大小数组。对于固定大小的数组，允许使用不同的数据结构（包括常量、变量、结构、映射和字段访问值）声明大小和索引号。它们在图2中分别被形式化为数组索引。由于Solidity中数组类型的大小可以是动态的，所以Lolisa中的动态大小数组类型被视为τ地图（Iint Signed I64）。

如图3所示，（n）维映射类型以及数组类型在智能合约中有广泛的定义。由于递归归纳定义，Lolisa可以表示

  
n维数组类型和n维映射类型比较容易，下面分别用规则3和规则4说明：

                                                                焦油id􏼂 0数组id􏼂 1T阵列􏼂...􏼂泰瑞idnτ最终的􏼃􏼃􏼃􏼃, (3)

                                               映射􏽨τ地图⇒映射􏽨τ地图⇒映射􏽨... 映射􏽨τ地图⇒τ最终的􏽩􏽩􏽩􏽩. (4)

0 0牛

我们分类τ 以及τ映射到正规形式类型和非正规形式类型。正规形式类型是指其类型规则不允许递归定义的类型，而非正规形式类型允许递归定义。例如，Tarray的范式（idarray，Tbool）应该是Tbool。在图2中，normal类型被单独定义为normal类型。

*4.2. 表达。*在正式指定了Solidity程序中可能声明和操作的所有可能的值形式之后，我们现在讨论程序中用于封装值的表达式。如第4.1节所述，所有表达式及其子表达式都用GADT定义，根据规则5，GADT由两种类型的签名进行注释，如下所示：

                                 表达式：τ0⟶ τ1⟶ 类型(5)

在这里，τ0引用当前表达式类型，并且τ1表示评估后的范式类型。例如，我们将整数变量表达式e的类型定义为exprTvid。这样，表达式的形式语法变得更清晰、更抽象，并允许严格维护Lolisa表达式的类型安全性。此外，结合使用这两种类型的注释有助于基于等价构造函数定义大量不同的表达式。当然，使用τ0和τ1根据情况可能会受到不同的限制。oα

常量表达式用于表示基本形式系统的本机值，这些值由相应的Lolisa值转换而来。因此，τ0和τ1应满足以下规则6：

*τ*0 τ1Λτ0,

|  |
| --- |
|  |
|  |  |

(6)

*τ*1 ∈ τ*纳法。*

为了满足类型形式的限制，数组类型和映射类型应该根据图2给出的类型定义进行分析和简化



*τ*最终∈ τnf，可表述为ΣΘ←τ ⟶τ′ ⟶ · · · ⟶ τnΛτn∈ τ纳法。我们把这个过程称为⇓τ.

另外，如上所述，值级别的类型信息被成功地传输到常量表达式中。例如，值v的类型为valτ1，并且常量表达式Econst的类型为

∀(τ : 类型），valτ ⟶ 出口⇓τ⇓τ. 因此，τ 在经济中（v）是由τ1.例如，Econst（Vbool（b））具有expr Tbool Tbool类型，其中τ 由Vbool（b）的Tbool指定。表达式级别的类型信息也可以以相同的方式传输到语句级别，这将在下一节中具体描述。

对于运算符表达式，Lolisa支持几乎所有的二进制和一元运算符，我们采用opclass（operator）来简化形式抽象语法。在Coq形式化中，二元和一元算子被抽象为一个归纳型op，该op也由GADTs定义，特定的算子作为它们的构造函数。通过这种方式，运算符表达式变得更加清晰和简洁，并且比使用较弱的静态类型系统时更容易扩展。二进制和一元运算符由两个类型签名进行注释，分别在规则7中给出，如下所示：

                                   操作：τ0⟶ τ1⟶ 类型(7)

|  |
| --- |
| 图4:Lolisa语句的抽象语法。 |

*4.3. 声明。*图4定义了Lolisa语句的语法。这里，几乎所有稳定的结构化控制语句（即，条件语句、循环、结构声明、修饰符定义、契约、返回、多值返回和函数调用）都受支持，但Lolisa不支持goto之类的非结构化语句和臭名昭著的“Duff&apos;s device”之类的非结构化开关。此外，匿名函数在Lolisa中是被禁止的，因为所有函数都必须有一个绑定标识符，以确保它们的格式良好。如前所述，右值（r值）e2到左值（l值）e1的赋值e1 e2、修饰符声明以及函数调用和结构声明被视为语句。此外，语句还根据正规形式和非正规形式类别进行分类，其中正规形式语句表示在求值后停止的语句，称为sttnf。实际上，虽然Solidity是一种图灵完备语言，但是用Solidity编写的智能合约程序不存在中断问题，因为程序执行受到gas的限制，gas是我们在中定义的ε 为了洛丽萨。

如图4所示，我们仍然归纳地将语句定义分为规范形式的sttnf（其类型分配必须在没有递归定义的情况下进行）和非规范形式的语句。Lolisa的范式语句被定义为sttnf。其余语句是非正规形式的语句。

*4.4. 形式抽象语法的宏定义。*Lolisa形式语法太复杂，一般用户无法采用。Lolisa语法包含与Solidity相同的成分；但是，它有更严格的正式打字规则。因此，Lolisa语法必须包含一些在Solidity中不受支持的附加组件，例如类型注释和monad类型选项。此外，Lolisa语法在Coq形式化中被正式定义为归纳谓词。因此，Lolisa代码看起来比相应的实体代码复杂得多，尽管这两种代码都显示出逐行对应。图5和图6所示的代码段说明了这种困难的一个例子。图6中质权函数中条件语句的形式Lolisa版本比图5中的原始Solidity版本复杂得多。

复杂程度给普通用户手工编写Lolisa代码和开发Lolisa与Solidity或其他语言之间的翻译带来了挑战。这在几乎所有类似的高级语言形式化研究中都是一个常见的问题。

幸运的是，Coq和其他高阶理论提供的助手提供了一种特殊的宏机制。在Coq中，这种机制被称为表示机制。这里，符号是一个符号缩写，表示由Coq自动解析的术语或术语模式。例如，可以封装Lolisa中的符号，如图7所示。

这个示例的新的正式版本产生了图8中的表示法，这表明该表示法几乎等同于原始的Solidity语法。

通过这种机制，我们可以隐藏用于验证的固定形式语法组件，从而为用户提供更简单的语法。此外，这种机制使得真实世界语言和Lolisa之间的等价性更加直观和用户友好。此外，该机制还提高了验证的自动化程度。与转换图5-8类似，我们开发了一个由词法分析器和解析器构成的转换器，用于自动将Solidity程序转换为Lolisa抽象语法树的宏定义。翻译过程如图9所示。Ethereum智能合约的文本脚本将由翻译器的词法分析器进行分析，从而生成Solidity token流。根据Lolisa的语法糖，词汇分析器将生成相应的Lolisa标记流。接下来，解析器将Solidity令牌流作为参数，生成智能合约的解析树。最后，用Lolisa令牌流替换解析树中的令牌，然后解析器重建Lolisa解析树并输出由Lolisa重写的相应形式智能契约。这样，翻译过程就可以保证机械地完成。

# 5.  形式语义学

*5.1. 表达式的求值。*表达式求值的语义是将Lolisa表达式求值为GERM框架的内存地址值的规则，这个过程包括两个部分：l值位置求值和r值位置求值。相反，修饰语表达式是一种特殊情况，不能根据这些表达式求值语义进行求值，而是根据规则进行求值

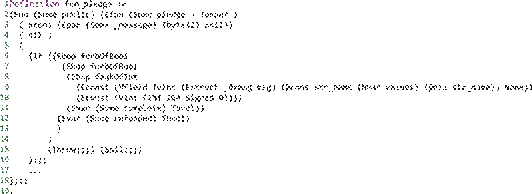
8:

|−⇓〈σ, env、fenv、Emodifier（∗ )〉⇒〈σ, env，fenv，错误〉.

(8)

在这里，⇓e表示在l值位置和r值位置计算修饰表达式的过程。图10总结了示例语义。

图5：稳定的条件语句。



|  |
| --- |
|  |
|  |  |

图6:Lolisa中图5所示的条件语句的正式版本。

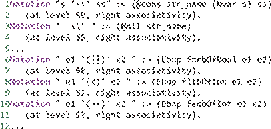


图7:Lolisa形式抽象语法树的宏定义。

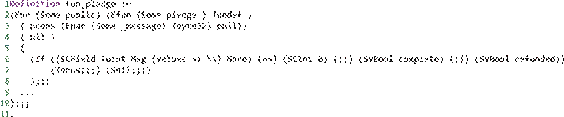
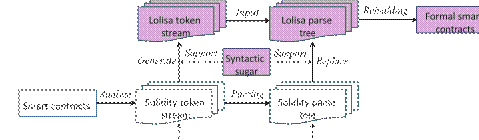


图8：使用语法缩写的图5的简化正式版本。



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | | 词法分析器 | | | |  |  | | --- | --- | |  |  | | 分析器 | | |

图9：从智能合同到其正式版本的翻译过程。

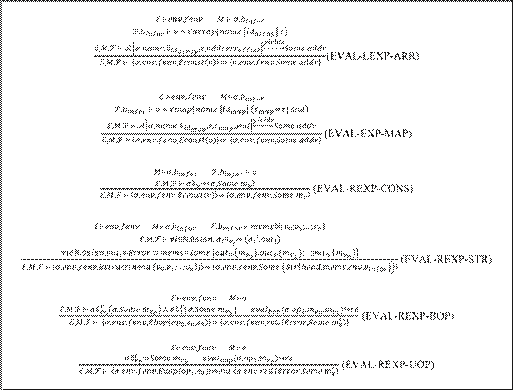


图10:Lolisa左右表达式的正式操作语义，包括数组、映射、常量、结构和二进制和

一元运算符。

*5.1.1. 在L值位置计算表达式。*在下面，我们分配⇓le表示在l值位置对表达式进行求值以产生相应的内存地址。首先，Econst构造的大多数表达式显然不能用作l值，因为其中大多数表达式直接在表达式级别表示Lolisa常量值。为简洁起见，我们指定了一个]σ, ∗ ] 表示用于搜索索引地址的数组和映射的递归过程。请注意，struct和field禁止在l值位置指定表达式，以确保Lolisa的格式和行为良好。在Lolisa中，改变结构字段的唯一方法是使用Estruct来改变所有字段或声明一个新字段。尽管这个限制对程序员或验证者来说可能不友好，但它避免了潜在的风险。

在上一节中，我们定义了数组值的语义。因此，我们可以将基于数组语义的地址搜索过程定义为规则9，它取名称，⇓身份证件，τ, 和addressoffset作为参数。类似地，我们可以在下面为映射值定义规则10：数组

|  |  |
| --- | --- |
| [答]σ, 姓名，⇓数组地址偏移量􏼕,*身份证件, τ,* | (9) |
| [答]σ, 姓名，⇓身份证件，τ, τ地图，snd􏼕. | (10) |

地图

*5.1.2. 在R值位置计算表达式。*在下面，我们分配⇓re表示在r值位置对表达式的求值，以产生相应的内存地址。

如图10所示，EVAL-REXP-CONS规则定义了常量表达式的求值。这里，我们注意到，由于常量表达式直接存储Lolisa值，因此可以通过应用⇓val直接。在表达式级别，用结构类型指定r值位置。这也是初始化或更改结构类型术语值的唯一方法。EVAL-REXP-STR规则定义了这个过程。在这里，如果对Estruct的求值失败，则求值成员值的过程将生成错误消息。否则，将获取成员的值集，并返回相应的struct memory值。最后，根据EVAL-REXP-BOP和EVAL-REXP-UOP规则定义了二进制和一元操作的语义。

由于基于GADTs的形式抽象语法定义中的静态类型限制，表达式、子表达式和操作都保证格式良好，并且不需要像Clight等其他形式语义所要求的那样使用非正式的辅助函数来检查类型依赖关系。函数evalbop和evaluop将表达式求值和所需操作的结果作为参数，并将它们组合在一起以生成新的内存值。

*5.2.状态评估。*在下面，我们分配⇓stt表示语句的求值过程，图11中总结了部分必要的操作语义

|  |
| --- |
| 图11:Lolisa的部分形式语句语义，包括环境和气体检查、契约、结构、修饰符和函数 |

通话记录。

envcheck和setgas。helper函数envcheck将当前环境env和超级环境fenv作为参数，并检查gas限制和执行级别的一致性等条件。合同声明是最重要的可靠性声明之一。在Lolisa中，合同声明涉及两个操作。首先，使用helper函数inheritcheck检查继承信息的一致性，该函数将模块上下文C中的继承关系和源代码作为参数。第二，将包括所有成员标识符的初始契约信息写入指定的存储器块中。如图11所示，合同声明的形式语义定义为EVAL-STT-CON。

作为rule EVAL-STT-STRUCT，地址是新的结构类型标识符，结构类型信息直接写入相应的内存块。

在Lolisa中，函数call语句用于应用由call语句索引的函数体。应用索引函数的过程由下面的EVALSTT-FUN-CALL规则定义。

修饰符声明是一种特殊的函数声明，它需要三个步骤，并且只包含一个限制。参数值由setpar谓词设置。如图11中EVAL-STT-MODI规则所定义的，第一步（表示为①) 初始化并设置参数。第二步（表示为②) 将修改器体存储到相应的内存块中。第三步（表示为③) 尝试初始化返回地址

Λ很有趣。由于有多个返回值，initre将返回类型列表作为参数。特别地，修饰体只能产生初始记忆状态，因此不能改变记忆状态。修饰语语义和函数语义的区别在于，函数语义包括检查限制函数的修饰语限制。具体来说，以EVAL-STT-FUN为例，在调用函数之前，将执行限制函数的修饰符。如果修饰符求值的结果是σinit，这意味着修饰符的限制检查失败，函数调用将被抛出。否则，将执行该函数。

*5.3. 标准库的开发和程序的评估。*如前所述，我们在Lolisa中开发了一个小型标准库，其中包含EVM的内置数据结构和函数，以便于使用高阶逻辑定理证明助手执行和验证在Lolisa中重写的可靠程序。这里，我们将详细讨论标准库。然后，基于语法、语义和标准库形式化，我们定义了控制Lolisa编写的程序的评估（即执行）的语义。

|  |
| --- |
| 图12:Solidity中的地址类型声明及其在Lolisa语法中作为特殊结构类型的等价物。 |

*5.3.1. 标准库的开发和程序的评估。*注意，我们假设EVM的内置数据结构和函数是正确的。这是合理的，因为首先，目前的重点是验证高级智能合约应用程序，而不是验证EVM的正确性。第二，Lolisa足够强大，可以实现EVM使用的任何数据结构或函数。因此，我们只需要在Solidity文档[4]的基础上使用Lolisa实现这些内置EVM特性的逻辑，以确保这些特性的格式良好。例如，address是一种特殊的复合类型，具有balance、send和call成员。但是，我们可以将地址视为Lolisa中的特殊结构类型，并使用Lolisa语法定义它，如图12所示。通常，requires是一个特殊的标准函数，不需要特殊的地址，根据Solidity文档，在Lolisa中定义为rule11:requires defλ􏼐∀ τ0:类型，e:扩展rτt刀具􏼑.(如果e（Snil）（Throw））。0

(11)

接下来，我们将这些数据结构和函数打包为Lolisa中的一个标准库，该库在执行用户程序之前执行。因此，EVM的所有内建函数和数据结构都可以在Lolisa中形式化，从而可以有效地模拟EVM的底层行为，而不是建立一个形式化的EVM。目前，这个标准库是一个很小的子集，只包含msg、address、block、send、call和requires。

|  |
| --- |
| *ε*ᅡ环境，芬夫ᅡFᅡ猫眼石ε, M、 F级ᅡ(（stt）ε, M、 F级ᅡ库*Mσ, b*inf或*P*                                                              环境设置气体初始环境（P（stt））􏼁fenv初始环境（P（stt））（12）  *σ* 初始内存（P（stt），lib）  *ε, 嗯，*Fᅡ〈σ, env、fenv、opars、，⇓P（标准温度）〉⇒〈σ′环境，芬夫〉,*,*  *ε*ᅡ环境，芬夫ᅡFᅡ猫眼石ε, M、 F级ᅡ(（stt）ε, M、 F级ᅡ库*Mσ, b*inf或*P*  环境设置气体初始环境（P（stt））􏼁fenv初始环境（P（stt））（13）*σ* 初始内存（P（stt），lib）〈σ恩芬夫〉.′*,* ′*,*  *ε, 嗯，*Fᅡ〈σ, env、fenv、opars、，⇓P（标准温度）〉⇒〈σ′恩夫夫〉Λ环境（气体）≤ 芬夫（气体极限）⇒*, , .* |

*5.3.2. 项目评估。*控制Lolisa程序（表示为P（stt））执行的语义由规则12和13定义，其中∞ 表示无限执行，T表示有限执行的终止条件集。

这些规则表示P（stt）执行的两个条件。在规则12规定的第一个条件下，P（stt）由于返回的stop、exit或error而在有限个步骤之后终止。在第13条规则规定的第二个条件下，P（stt）不能通过其内部逻辑终止，将经历无限多个步骤。因此，通过气体限制检查机制故意停止P（stt）。这里，opars表示可选参数的列表。此外，如第5.1节所述，初始环境env和超级环境fenv是等效的，只是它们的gas值由辅助函数initenv初始化，env的初始gas值由setgas设置。最后，初始内存状态由initmem设置，将P（stt）和标准库lib作为参数。

# 6.  基于FEther的智能合约形式化验证

如第1节所介绍的，我们为Lolisa实现了一个正式的Coq验证解释器，表示为FEther[11]，它包含了大约7000行Coq代码（不包括证明和注释）。该解释器严格遵循Lolisa的形式语法和语义，基于GERM框架。具体地说，FEther是通过计算函数（被认为是机械化的计算语义）来实现的，这相当于本文给出的Lolisa的自然语义。该实现是按照我们之前的研究[11]中的细节进行的，使用的是Coq提供的函数式编程语言Gallina。因此，费瑟可以解析Lolisa的语法来象征性地执行用Lolisa编写的正式程序。虽然正在努力证明费瑟和洛丽萨的语义之间的一致性，但费瑟可以用来证明现实世界程序的属性。这个过程不仅能有效地暴露出测试套件中的错误，这些测试套件体现了预期的行为，而且在正常的智能合约中也是如此。通过一个简单的实例说明了基于Lolisa和FEther的符号执行和验证过程。其源代码见附录A，用Lolisa编写的正式版本见附录B。在这里，很明显，如果索引映射列表中的消息发送者和当前时间小于privilegeepen或大于privilegeClose，程序将被抛出。这很容易用前面定义的归纳谓词语义手动证明。

|  |
| --- |
| 图13：使用Coq中的正式解释器FEther执行和验证附录B中的Lolisa程序。 |

同时，我们可以通过在Coq中借助FEther直接象征性地执行程序来验证这个属性，如图13所示。在使用FEther执行和验证这个Lolisa程序期间获得的形式中间存储器状态如图14所示。然后，我们可以比较机械化验证结果和手工获得的结果来验证Lolisa的语义。此外，基于Lolisa和GERM框架的FEther的应用也证明了我们提出的EVI理论是可行的。

# 7.  讨论

*7.         1.捐款。*首先，Lolisa将大多数类型、运算符和稳定机制形式化，它包含了大多数稳定语法。此外，基于Lolisa构建了一个标准库来表示EVM的内置数据结构和函数，如msg、block和send。因此，用Solidity编写的程序可以被翻译成Lolisa，反之亦然，一行一行地对应，而无需重建或抽象，这些操作会对一致性产生负面影响。

其次，Lolisa中的形式语法是使用广义代数数据类型定义的，它将静态类型注释赋予Lolisa的所有值和表达式。这样，Lolisa在检查程序结构方面有一个比Solidity更强的静态类型系统。因此，不可能在Lolisa中构造病态术语，这也有助于在稳定的源代码中发现病态术语。此外，形式语法确保Lolisa中的所有表达式和值都是确定性的。

最后，使用Coq证明助手对Lolisa的语法和语义进行了机械化。此外，在Coq中开发了一个形式化的验证解释器FEther来验证Lolisa是否满足上述可执行可证明特征和语义的元属性。与为高级编程语言构建形式化语法和语义的类似工作不同，Lolisa的形式化语义是基于FSPVM-E框架定义的。因此，用Lolisa编写的程序可以象征性地执行，并且在Coq-proof-assistant中，当与FEther一起执行时，可以直接在真实世界中作为程序执行，同时自动验证它们的属性。

*7.2. 限制。*尽管当前版本的Lolisa规范语言中的新特性提供了许多优点，但仍然存在一些局限性。

首先，因为Lolisa是Solidity的一个大子集，所以在Lolisa中省略了一些Solidity特性，比如内联组装。因此，当前版本的Lolisa current不支持某些复杂的以太坊智能合约。更新版本的Lolisa将支持这些特性。

第二，Lolisa在源代码级别上是形式化的。虽然它会在编译前分析漏洞，但当编译器不可信时，它不能保证相应字节码的正确性。一个可能的解决方案是开发Lolisa的低级版本，它执行编译器生成的字节码，然后证明执行结果的可靠性和字节码的相应执行结果之间的等价性。

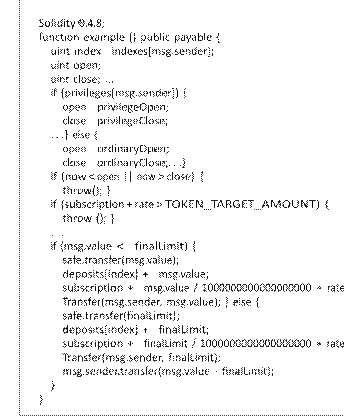
最后，虽然当前版本的Lolisa可以象征性地验证FEther，但这个过程还没有完全自动化。在某些情况下，程序员必须分析当前的证明目标并选择合适的验证策略。幸运的是，这一目标可以通过优化战术评估策略的设计来实现。

|  |
| --- |
| 图14:Coq中使用FEther执行和验证附录B中的Lolisa程序期间的形式内存状态。 |

# 8.  结论和今后的工作

在本文中，我们定义了一个大的坚固性子集的形式语法和语义，我们称之为Lolisa。根据GADTs，Lolisa的形式语法是强类型的。Lolisa的语法包含了几乎所有的语法，因此这两种语言是等价的。因此，Solidity程序可以逐行翻译成Lolisa，而无需重建或抽象，这是一种过于复杂的操作，一般程序员无法执行，并且可能会引入不一致性。此外，我们在Coq中实现了Lolisa的完全机械化，并在数学工具Coq中开发了一个基于Lolisa的形式化解释器FEther，用于验证Coq的语义

洛丽萨。通过将Lolisa的形式语义建立在我们的FSPVM-E框架[31]上，用Lolisa编写的程序可以在Coq中象征性地自动执行，从而同时验证相应的程序的可靠性。作为当前工作的结果，我们现在可以使用Lolisa直接验证以Solidity形式编写的智能合约。



算法1：案例研究合同的部分源代码。

Coq^8.8；

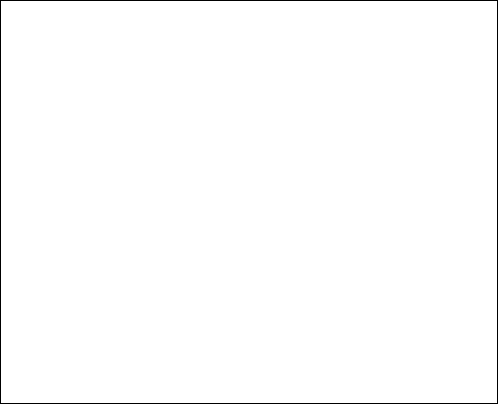
定义示例：

（有趣的公共支付（Efun（一些例子）Tundef）pnil nil）；；

（Var（部分公共）Evar（部分指数）Tuint））；；

（赋值）

（Econst（@Vmap Iaddress Tuint indexes（Mstr\u id Iaddress msg（sender））∼>>\\\)) 无）；；

（Var（一些公共的）（Evar（一些开放的）Tuint））(Var（Some public）（Evar（Some close）Tuint））(Var（Some public）（Evar（Some quota）Tuint））；；

...

（如果（经济）（@Vmap Iaddress Tbool priviledges

（Mstr\u id）消息（发送者）∼>>\\\)) 无）

（（Assignv（Evar（Some open）Tuint）（Evar（Some privilegeOpen）Tuint））；；

（Assignv（Evar（Some close）Tuint）（Evar（Some privilegeClose）Tuint））；；

...;; 无）

（（Assignv（Evar（Some open）Tuint）（Evar（Some ordinaryOpen）Tuint））；；

（Assignv（Evar（Some close）Tuint）（Evar（Some ordinaryClose）Tuint））；；

...;; 无）

（If（（Evar（Some now）Tuint）（<）（Evar（Some open）Tuint）（| |）

（Evar（有些现在）Tuint）（>）（Evar（有些接近）Tuint））

（投掷）；；无）（Snil；；无）；；

（If（（Evar（Some subscription）Tuint）（+）（Evar（Some rate）Tuint）（>）

代币（目标金额）

（投掷）；；无）（Snil；；无）；；

...

（如果（（经济）（Vfield Tuint（Fstruct\u 0xmsg）（值∼> \\) 无）

（<）（Evar（Some finalLimit）Tuint））算法2：续。

（（Fun\u call（Econst（Vfield（Tfid（Some safe））（Fstruct\u 0xaddress safe）（发送）∼> \\) 无）

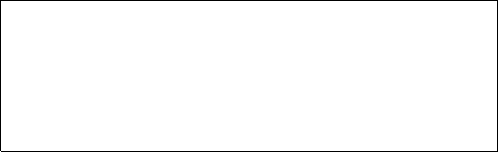
（pccons（经济）（Vfield Tuint（Fstruct\u 0xmsg）（值∼> \\) 无（pcnil））；；

（Assignv（Econst（@Vmap Iuint Tuint deposits（Mvar\u id Iuint index）None））

（（经济）（Vfield Tuint（Fstruct\u 0xmsg）（值∼> \\) 无）（+）

（经济成本（@Vmap Iuint Tuint depositions（Mar\_id Iuint index）None）））；；

（Assignv（Evar（Some subscription）Tuint）（（经济结构）

（价值观）∼> \\) 无）（+）（Evar（Some finalLimit）Tuint）（/）（Econst（Vint（INT I64 Unsigned 1000 000）））（x）（Evar（Some rate）Tuint））；；无）。。。；；无）；；零。

算法2：用Lolisa编写的算法A的正式版本。

包含Lolisa抽象语法树形式化的源文件可以在https://gitee.com/ 电子科技大学EOS\U FV/LOLISAST/tree/master/SPEC

目前，我们正致力于验证FEther的正确性，并开发可计算语义和归纳语义等价性的证明。随后，我们将在Coq符号机制的基础上实现我们提出的初步方案，沿着两条重要的途径扩展Lolisa。

我们正在进行的项目是FSPVM-E的扩展，以支持EOS区块链平台[32]，然后我们将在Coq中验证我们的新框架。此外，我们还将开发一个通用的形式化验证工具链，利用HOL-proof技术实现区块链智能合约的自动验证。

# 附录A。案例研究的源代码

如算法1所示，我们给出了案例研究合同的部分源代码。

# B.案例研究的正式版本

如算法2所示，我们给出了用Lolisa编写的算法1的正式版本。

# 数据可用性

包含用于支持本研究结果的Lolisa抽象语法树数据的形式化的源文件已存放在Gitee存储库（https://Gitee.com/UESTC\u EOS\u FV/LolisaAST/tree/master/SPEC）中。

**利益冲突**

作者声明没有利益冲突。

# 工具书类

[1] S。B.“一个点对点的电子现金系统”

2020, https://bitcoin.org/bitcoin.pdf.

[2] 答。Narayanan，J。邦诺，E。费尔滕，A。米勒和S。Goldfede，《比特币和加密货币技术：综合介绍》，普林斯顿大学出版社，美国新泽西州普林斯顿，第1版，2016年。

[3] 五。E。Buterin，“下一代智能合约和分散应用平台”，2020年，https://github. com/ethereum/wiki/wiki/白皮书。

[4] S。德米尔坎，I。Demirkan和A。McKee，“商业网络安全和会计未来的区块链技术”，《管理分析杂志》，第7卷，第2期，第189-208页，2020年。

[5] 2020以太坊实体文档。https://Solidity. readthedocs.io/en/develop/。

[6] J。麦基，J。郑，N。熊，L。詹和Y。Zhang，“使用智能合约和SGX的突发公共卫生事件大数据分布式隐私保护方法”，计算机材料与连续性，第65卷，第1期，第723-7412020页。

[7] 2020年D。A.O。攻击：代码问题导致6000万美元乙醚盗窃。https://www.coindesk. com/dao-attacked-codesissue-leads-6000万-ether-theft/。

[8] L。卢，D。H。朱，H。奥利克尔，P。萨克塞纳和A。Hobor，“使智能合约更智能”，载于ACM计算机和通信安全会议论文集，第24-28页，奥地利维也纳，2016年10月。

[9] H。十一，C。陈和G。Chen，“保护递归数据类型构造函数”，《SIGPLAN-SIGACT编程语言原理研讨会论文集》，第224-235页，新奥尔良，洛杉矶，美国，2003年1月。

[10] 2020 Coq证明助手参考手册。https://coq. inria.fr/distrib/current/refman/。

[11] Z。杨和H。Lei，“FEther:Coq中智能合约验证的可扩展定义解释器”，IEEE Access，第7卷，第37770–37791199页。

[12] T。M。位，https://github.com/trailofbits/manticore, 2020.

[13] B。M。穆勒，https://github.com/b-mueller/mythril/, 2020.

[14] E。希尔登布兰特，M。萨克塞纳，X。朱，N。罗德里格斯和P。Daian，“KEVM:以太坊虚拟机的完整语义”，《IEEE计算机安全基础研讨会论文集》，第204-217页，英国牛津，2018年7月。

[15] 是的。Hirai，“为交互式定理证明者定义以太坊虚拟机”，《金融加密和数据安全学报》，第35-47页，马耳他斯莱马，2017年4月。

[16] S。阿曼尼，M。贝格尔和M。伯丁，“去验证´ 以太坊智能合约字节码（Isabelle/HOL），《Acm Sigplan认证计划国际会议论文集》，第66-77页，美国加利福尼亚州洛杉矶，2018年1月。

[17] C。Reitwiessner，“开发更新：正式方法”，2020年，https://ethereum.org/2016/09/01/formal methods roadmap/。

[18] 第。Rizzo，“以太坊寻求智能合约确定性”，2020年，http://www.coindesk.com/ethereum-formal-verificationsmart-contracts/.

[19] C。刘，H。刘，Z。曹，Z。陈，B。陈和B。Roscoe，“ReGard:在智能合约中发现可重入性缺陷”，《IEEE/ACM软件工程国际会议论文集：Companion》，第65-68页，瑞典哥德堡，2018年5月。

[20] 第。特桑科夫，A。M。丹，D。Drachsler Cohen，A。F.热尔韦。日分¨nzli和M。T。Vechev，“Securify:智能合约的实用安全分析”，《ACM SIGSAC计算机与通信安全会议论文集》，第67-82页，加拿大安大略省多伦多市，2018年10月。

[21]不适用。格雷奇，M。香港，A。法学，L。布伦特，B。Scholz和Y。Smaragdakis，“Madmax:以太坊智能合约中的无气生存条件”，《编程语言ACM会议录》，第1-27页，美国宾夕法尼亚州费城，2018年11月。

[22]东。艾伯特，P。戈迪洛，B。利夫什，A。鲁比奥和我。E。Sergey，“以太坊字节码高级分析框架”，《验证与分析自动化技术学报》，第513-520页，美国加利福尼亚州洛杉矶，2018年10月。

[公元23年]。马弗里杜，A。拉兹卡，E。Stachtiari和A。Dubey，“Verisolid:correct by design smart contracts for ethereum”，《金融加密和数据安全学报》，第446-465页，圣基茨和尼维斯护卫舰湾，2019年2月。

[24]答。Mavridou和A。Laszka，“工具演示：用于设计安全以太坊智能合约的fsolidm”，《计算机科学讲义》，第10804卷，第270-277页，瑞士查姆施普林格，2018年。

[25]吨。阿卜杜拉蒂夫和K。Brousmiche，“基于用户和区块链行为模型的智能合约正式验证”，《新技术、移动性和安全国际会议论文集》，第1-5页，法国巴黎，2018年2月。

[26]天。A.帕克。斯泰法˘nescu和G。反渗透¸su，“KJS:a complete formal semantics of JavaScript”，Acm Sigplan Notices，第50卷，第6期，第346-356页，2015年。

[27]第X页。Leroy，S。布拉齐，D。卡斯特纳，B。肖默和M。皮斯特，¨ “Compcert-a正式验证的优化编译器，《欧洲嵌入式实时软件和系统大会论文集》，第35-62页，法国图卢兹，2016年1月。

[公元28年]。W。Appel，验证软件工具链，Springer出版社，柏林，德国，2011年第1版。

[29]右。H。顾，Z。邵，H。Chen等人，“CertiKOS:构建认证并发操作系统内核的可扩展体系结构”，《USENIX操作系统设计与实现研讨会论文集》，第653-669页，美国佐治亚州萨凡纳，2016年11月。

[30]秒。马菲斯，J。C.米切尔和A。Taly，“JavaScript的操作语义”，《亚洲编程语言和系统研讨会论文集》，第307-325页，印度班加罗尔，2008年12月。

[31]字。杨，H。雷和W。Qian，“用于确保基于以太坊的服务智能合约的可靠性和安全性的Coq混合形式验证系统”，IEEE Access，第8卷，第21411–214362020页。

[32]2020 EOS区块链文档。https://eos.io/.