收到日期：2019年2月11日，接受日期：2019年3月12日，出版日期：2019年3月15日，当前版本日期：2019年4月5日。

*数字对象标识符10.1109/ACCESS.2019.2905428*

# FEther：一个可扩展的Coq智能合约验证定义解释器



郑阳（IEEE成员）和杭雷

中国电子科技大学信息与软件工程学院，成都610054

通讯作者：郑扬（zyang。uestc@gmail.com)

**摘要**近年来，区块链技术在金融领域得到了广泛的应用。因此，区块链智能合约的安全性是当前最热门的研究课题之一。为了改进这一领域的定理证明技术，我们正在开发一个可扩展的混合验证证明引擎FEther，用于以太坊智能合约验证。FEther以Coq中solidity的一个大子集Lolisa为基础，保证了智能合约与其形式模型的一致性。FEther将符号执行与高阶逻辑定理证明相结合，提出了一套在Coq中执行和验证智能契约的自动化策略。此外，在FEther中，经过验证的代码段还可以被重用，以辅助其他属性的验证。在Coq中验证了FEther的功能正确性。FEther的执行效率已经远远超过了Coq中按照标准教程开发的口译员。据我们所知，费瑟是柯克语中第一个定义性语言的解释者。

**索引项**符号执行，形式验证，智能合约，Coq，etheruem，定义解释器。

EVI：执行验证同构

CHI：克里霍华德同构

## 一。导言

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

区块链技术[1]使用加密链接将记录添加到列表中，是当代最流行的技术之一。以太坊是一个被广泛采用的区块链系统，它实现了一种通用的图灵完整编程语言Solidity[2]。以太坊支持开发任意智能合约，可以在虚拟运行时环境（即以太坊虚拟机（EVM））中自动化区块链交易。在这里，智能合约指的是执行区块链的应用程序和脚本（即程序）。智能合约的使用越来越多，这就要求对其安全性进行更严格的审查。智能合约可以包括特定的财产，使他们暴露在蓄意攻击造成直接经济损失。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[3]和平价钱包合约[4]的攻击。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[5]。因此，必须尽可能严格地验证智能合约程序的安全性和可靠性。的性质

杨柳副主编负责协调本稿件的审稿并批准出版。

通过证明高阶逻辑定理，程序可以得到严格的验证。在标准方法中，目标软件系统的形式化模型是使用高阶定理证明助手手工抽象的。这种形式化验证技术为基于高阶逻辑理论的形式化模型设计提供了足够的自由度和灵活性，能够抽象和表达非常复杂的系统。然而，当应用于智能合约验证时，定理证明技术的自动化、可重用性、一致性和效率问题抑制了其优势。

|  |
| --- |
| 2169-3536 2019年IEEE。翻译和内容挖掘只允许用于学术研究。  37770                                                                                    也允许个人使用，但重新发布/再分配需要IEEE许可。2019年第7卷  看到了吗http://www.ieee.org/publications\_standards/publications/rights/index.html 更多信息。 |

上述问题可以通过一个形式化的符号过程虚拟机（FSPVM）[6]来解决，该虚拟机使用高阶定理证明助手直接和符号化地执行现实世界的智能合约程序。程序的属性随后由执行结果自动验证。为此，我们正在为部署在以太坊平台上的智能合约开发一个名为FSPVM-E[7]的FSPVM。FSPVM-E是在Coq（一个正式的证明管理系统）中编程的，它的灵感来自KLEE，一个用于复杂系统程序的高覆盖率测试生成器[8]。与[9]类似，FSPVM-E的符号执行在FEther中得到验证，FEther是一个支持多种符号执行类型的混合证明引擎。然而，FEther是为高阶定理证明而设计的，其验证过程建立在Hoare[10]和可达性[11]逻辑的基础上。因此，FSPVM的成功实施必须克服几个挑战[6]。

我们最近的研究已经解决了其中一些挑战。在[6]中，我们注意到在一个高阶定理证明系统中缺乏一个通用的形式内存模型来构造逻辑操作环境。因此，我们使用Coq[12]开发了一个基于高阶逻辑的通用、可扩展和可重用形式内存（GERM）框架。在我们之前的工作中，我们提出了一个Curry-Howard同构（CHI）[13]的应用扩展，将定理证明和符号执行技术结合起来。本文将其表示为执行验证同构（EVI），并采用了FSPVM的基本理论。

最后，我们开发了Solidity编程语言的一个可扩展的大子集，表示为Lolisa[14]，它等价地将现实世界的编程语言形式化为一种可扩展的中间编程语言。

本文通过克服最后一个挑战：开发验证引擎来完成FSPVM-E。我们的贡献如下。首先，我们用Coq的规范语言（Gallina）开发了一个定义解释器。这个解释器象征性地执行在GERM框架上用Lolisa编写的以太坊智能合约。执行结果用一个细菌逻辑存储状态来表示，可以在Coq中进行验证。接下来，我们实现了一套基于Ltac[12]机制的自动评估策略，FEther通过该策略完成了执行和验证过程。然后在Coq中证明FEther的正确性。本文提出的口译员评价方法是一种优化的口译员评价方法，其评价效率高于Coq标准教程中的口译员。据我们所知，费瑟是第一个混合证明引擎规范，自动和象征性地执行和验证以太坊智能合同在Coq。

本文的其余部分结构如下。第二节介绍了《拜祭》与其他相关作品的区别。第三部分介绍了本文工作的基础，包括原型系统、Lolisa的基本环境和GERM框架的初步修改。第四节介绍了FEther的理论设计和实现，以及它的自正确性证明。第5节验证了费瑟在现实世界中的案例研究，并分析了它的好处。第六节讨论费瑟的优点和局限性。本研究以第7节结束。

## 二。相关工作

自2015年以来，人们对智能合约的安全性进行了认真的研究，通过形式化的方法可以严格保证智能合约和类似轻量级程序的安全性。我们的符号执行器有几个新颖的特性，使它区别于其他方法。本节介绍了在这一领域已经报道的有趣的成就。

EVM的执行在黄皮书[15]中有正式的描述。本官方文件还提供了建立共识所需的数据、算法和参数CompatibleVMClientsandethumiImplements。然而，黄皮书并不总是阐明EVM的操作行为。在这种情况下，通常更容易查阅可执行实现以获得指导。

目前的研究主要集中在EVM安全性方面。C++实现CPP EUTUM在EVMs中扮演着安全和叛逃语义的双重角色。Lem semantics[16]是EVM的Lem[17]实现，提供了EVM的可执行语义，正式验证了智能契约。然而，Lem语义并不能精确地捕捉到契约间的执行。KEVM[18]是EVMs的一种形式语义，类似于Lem，但是是在K框架中编写的。由于KEVM是可执行的，它可以运行由EthUM基金会提供的验证测试套件。根据Hildenbrandt等人[18]，KEVM引用解释器通过了完整的40683测试EVM合规套件。然而，自正确性不能在KEVM中被完全证明，甚至不能在KEVM中被证明。此外，上述方法均不满足德布鲁恩准则[19]。

Mythril[20]是一个用于以太坊智能合约的安全分析工具。Mythril通过concolic分析发现了各种问题，但该工具是否有效地提高了智能合约的可靠性还没有得到证实。

以上研究采用的是固态字节码。编译后的字节码与实体码的一致性无法保证。然而，对于智能合约软件的编程和调试来说，高级形式化规范和相关的形式化验证工具却很少受到关注。

最后，其中一些工作集中在一个特定的领域。他们复杂的架构是不灵活的，不容易扩展到新的相关问题。

**三、 基本概念和定义**本文以我们最近的工作为基础。因此，在定义FEther的形式化规范之前，我们首先定义基本环境。

*A.预定义*

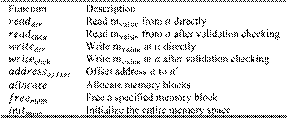
首先，EVI是CHI的一个应用扩展。简言之，池认为逻辑世界和计算世界之间存在着深刻的对应关系。为了避免在下面的EVI讨论中出现歧义，我们使用program来表示用Gallina或类似的基于CHI的规范语言编写的程序，并使用programfrw来表示用Lolisa编写的formalversonofreal world程序。这种对应关系可以根据以下三个一般原则来表达：

*类型对应于命题*; ;*证明与程序相对应*

*证明与程序评估相对应.*

在CHI中应用的深层对应使得它对于统一形式证明和程序计算非常有用。在支持CHI的证明助手（如Coq）中实现的程序是其逻辑的一级公民，这一点可以直接在支持CHI的高阶定理证明系统中得到评价，这一过程对应于相应定理中程序的命题证明。然而，大多数主流编程语言都不是基于lambda演算设计的，不能在高阶逻辑环境中进行分析。用这些语言编写的程序很难或不可能直接用CHI自动验证。执行引擎，如执行programfrw的FEther，是在基于形式操作语义的支持CHI的形式化系统中开发的，是一种特殊的程序；这形成了EVI的基础，它扩展了CHI的形式关系，包括三个推论：程序的评估对应于程序的执行frw，属性对应于类型，验证对应于证明。基于这些推论，CHI的对应关系可以扩展到第四个一般原则：验证对应于程序frw的执行，表示为EVI。更多细节见[6]。

**表1。**正式内存模型中使用的基本内存管理API。



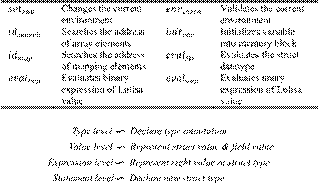
***胚芽***GERM是一个通用的、可扩展的、可重用的形式内存框架。它同时支持不同的形式验证规范，特别是在代码级别。该框架模拟物理内存硬件结构，包括一个低级形式内存空间，并提供一组简单的、非侵入式的应用程序编程接口（api）。所提出的细菌框架是独立的和可定制的，并且完全在Coq中进行了验证。表1总结了FEther的顶级接口，其中mvaluemvalue和a分别表示value类型的内存值和Laddress类型的内存地址。在特定的形式规范中，形式内存状态具有类型内存。最后，binfor表示用于环境检查的块信息。

***洛丽萨***FL是Lolisa，Solidity编程语言的一个很大的子集。在广义代数数据类型（GADTs）[21]的帮助下，Lolisa的形式语法采用了比Solidity更强的静态类型系统，从而增强了类型安全性。Lolisa包括契约声明（contract）、修饰符声明（modifier）、变量声明（Var）、结构声明（Struct）、赋值（Assign），

**表2。**动态语义定义中的状态函数。



**表3。**助手函数。



**公约1。**正式结构数据类型。

返回（Return）、多值返回（returns）、抛出（Throw）、跳过（Snil）、函数定义（Fun）、while循环

（Loopwhile）、for循环（Loopfor）、函数调用（Funcall）、条件语句（If）和序列语句。

表2总结了动态语义定义中使用的helper状态，表3列出了用于计算当前程序状态中常用值的helper函数。所有这些状态函数都将在下面的讨论中遇到。具体国家的组成部分将由有关国家签署的适当希腊字母表示。在表2中，M和E分别表示形式内存空间、特定内存状态和执行环境的上下文。证明评估是在证明上下文中执行的，表示为，…为简洁起见，下面我们用F表示整个形式系统，用env表示当前执行环境，用fenv表示env类型的超级环境*σ, 00*1*.*

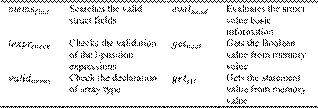
在接下来的章节中，我们将介绍提高高阶定理证明助手FI计算效率的相关分析和解决方案。

为了简化验证过程和Coq中相应的正式验证Lolisa解释器的开发，我们将Lolisa程序作为结构化程序进行维护。为此，Lolisa的语义被迫遵循以下程序反公理。结构数据类型的约定在约定1中定义。程序计数器公理是一个费瑟设计原则，它将Lolisa保持为一种结构语言。

*Axiom（程序计数器）：*如果语句s是下一个执行语句，则它必须是下一次迭代中语句序列的头。

为了避免程序中出现无限循环，FSPVM还引入了有界模型检查（BMC）[22]。幸运的是，EVM不支持无限的执行过程，因为每个执行步骤都会消耗智能合约所有者的gas。如果气体平衡不能满足限制条件，

**表4。**在优化的FEther中封装函数。



执行终止。这种设计很适合BMC的概念。因此，我们的实现使用gas来限制Solidity程序的执行。

在以下内容中，我们用通配符“”表示其他参数∗&apos;&apos; 还有那个符号|∗ 7−→ ∗|}. E是-表达式E的构造函数模式匹配的语法[23]。为了避免费瑟在下面的讨论中出现歧义，函数表示用Gallina编写的程序和函数，RWprogram表示用通用编程语言编写的实际程序。*λ*

*B.优化修改*

如前所述，在分析当前问题时，基于FSPVM的定义解释器的计算效率可能极低。这三个基本问题是按名称终止（CBNT）、信息冗余爆炸（IRE）和并发缩减（CR）。为了优化评估问题的低级计算，我们将[24]中的相应解决方案合并到FEther的实现细节中。

首先，将sequence语句s隐式地替换为等价的列表，而不是显式地定义，这避免了CBNT问题。其次，将模式匹配和可重用函数封装为优化辅助函数。表4总结了其中的一些函数。为了避免CR问题，我们最后对表达式层和值层施加限制K（与气体约束无关）。

## 四、 费瑟实施

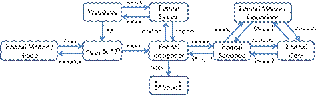
FEther是连接GERM框架、Lolisa编程语言和Coq可信核心（TCOC）的桥梁。正如我们在前面的工作中所展示的和在下面的小节中详细阐述的，FEther完全可以在Coq中构建。

*A.建筑*

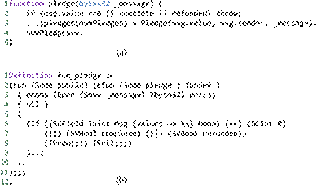
图1显示了费瑟框架的总体结构。整个以太网构建在TrustedDomainofCoQ中，逻辑上包括三个主要组件：解析器、基于Lolisa语义的ISA和验证检查机制（分别参见图1中的左、中、右部分）。解析器分析用Lolisa编写的frwprogram的语法。根据EVI理论，FEther本质上是一个用Gallina写成的巨大函数。从这个意义上说，它不同于像Smalltalk、Java和.Net这样的高级编程语言的现实世界虚拟机，后者支持字节码作为其ISA，并通过将常用代码路径的字节码转换为本机代码来实现。相反，FEther的ISA包含Lolisa语义，它指定了控制相应行为的语法标记的语义。验证检查机制包括两部分：检查结果验证（包括内存状态和值）和检查执行条件。首先，因为所有函数都容易受到未定义情况的影响，所以它们是在effect编程的帮助下开发的。更具体地说，所有函数都由可选类型标记。以Some t的形式返回一个有效结果；无效结果将作为未定义的值None返回。符号[[t]]表示术语t由可选类型标记。第二部分分别用辅助函数envcheck和pumpcheck验证了gas和K的局限性。

|  |
| --- |
| **图1。**费瑟的建筑。 |

费瑟继承了洛丽萨的低耦合特性。在同一层中，可执行语义是完全独立的，并且被封装为由一组接口连接的模块。在不同的层次上，高层语义只能通过接口访问底层语义，底层语义的实现细节对高层语义是透明的（图1中用虚线表示）。而且，高级语义的实现并不依赖于低级语义。



**图2。**费瑟的工作流程。



**图3。**从稳固到洛丽萨的翻译过程。

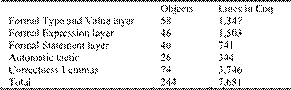
FEther的工作流程如图2所示。用户首先通过初始化GERM的形式存储空间并应用翻译器来设置初始存储状态和目标FRW程序。请注意，转换器是[14]中介绍的Lolisa的一个辅助组件。如图3所示，它通过搜索Lolisa的抽象语法树，用唯一的内存地址绑定变量标识符，并声明ML模块，将（a）的Solidity程序转换为（b）的用Lolisa编写的frwprogram。接下来，FEther解析器根据Lolisa抽象语法树分析智能契约的形式化模型，并调用相应的可执行语义。TCOC处理评估需求，并通过验证机制验证结果。尽管验证模块在逻辑上独立于其他部分（如上所述），但在实际案例中，它是在正式解释器和正式语义模块中分别实现的。因此，验证模块在图3中没有明确定义。最终的形式内存状态将在属性定理中假设。

Lolisa是由GADTs定义的，它保证了语法规范的良好构造。因此，语法正确性的附带条件不需要由FEther中定义的帮助函数进行检查。通过Coq的型式检验机构，可以检验其型式安全性。表5列出了用100个内存块构建费瑟框架的全部工作量。

*B.费瑟的指令集结构*

FEther的ISA是证明引擎的核心，遵循Lolisa的大步操作语义。如图1所示，胎儿ISA分为三层。FEther的实施如附录A所述。

**表5。**构建具有100个内存块的FEther框架的工作负载统计。



***价值层***本计画的目的是将一个机械化的语法与语义学形式化，以供高阶逻辑定理证明辅助工具直接执行与验证。因此，必须像正规体系中的本土价值观一样，对稳固性价值观进行评价。理想情况下，在正式系统中会显式地使用Solidity值或一些主流高级编程语言。然而，由于可信核心的严格类型系统和采用不同的范例，Gallina（Coq）不直接支持数组、映射和其他复杂值。因此，我们必须在现实世界语言的价值观和形式系统的本土价值观之间定义一个中间层。这个中间层通过等价的语法直接表示真实世界的值，并使用形式语义将它们转换为本机值。

通过形式化可执行语义对Lolisa值进行评估后，在基本形式化系统中计算或导出本机值信息，并确定相应的细菌内存值。在下面的部分中，ESV表示调用值语义的入口点，它由下面的规则1抽象定义。

ESVZ公司→ (∀τ : *类型,瓦尔τ)*

→ → → → (1)*记忆Blc公司环境期权价值*

这里，元变量val包含Lolisa值val和映射值valmap。除了缺少与映射类型相关的定义外，valmap与val具有相同的静态类型规则，因此可以将这两个值组合起来。每个Lolisa值与其唯一内存值之间的映射关系表示为≈.

首先，我们定义了常量值的计算语义。在Lolisa中，常量值是一组标准形式值和一组元变量vconst。vconst评估过程生成相应的内存值，用于直接记录正式系统的本机值信息。例如，考虑常量变量Vbool：∈ . Vbool的计算语义由规则2定义，然后定义ESV*（b）) t刀具vconst公司（b）) 常数博恩,环境,b信息**V刀具（不适用）)*.

ESV公司*康斯特布尔*

≡ λ(: : : *n布尔).λ(环境环境).λ 宾福**.V刀具（不适用）)*

⇒ (2） 计算语义ESVare总结在A的表11中。定理1（常数映射定理）证明了ESVis的正确性。*一些布隆,环境,宾福**常数*常数

*定理1（常数映射）：*对于所有Lolisa值vconst、环境值env和块信息b，映射*（不适用）)信息常数n,环境,b信息**v常数（不适用）)* 持有。

然后定义访问形式化内存空间和匹配索引所需的引用值（数组、映射、结构和字段访问值）的语义。相应的值定义如下，语义sesvofthearrayvalues定义在A的表12中。特别是idsearch是一个辅助的指标评价功能。具体来说，Lolisa语言支持n维数组（根据规则4），因此idsearch是一个特殊的辅助函数，用于搜索由当前n维数组索引索引的内存块。idsearch也用于initvar函数。下面我们介绍idsearch的具体实现。idsearch实现的抽象函数由规则3给出。*数组*

*身份证件搜索*:: → → → *类型L地址埃默里环境*

                               → (3)*选项地址*

*柏油**柏油**柏油**塔拉伊登*

映射值作为单链表结构存储在细菌内存模型中，并以Map的形式存储：

*女装*→ → → → → → → 在里面*选项prodvaluemapvalue**类型映射类型选项地址环境Blc公司价值*

洛丽萨。在上述表达式中，第一个参数存储初始地址，第二个参数存储成对的键值和索引值，第三个和第四个参数分别记录键值和索引值类型，第五个参数表示下一个地址。这个映射可以简单地抽象为图4。在这个设计中，这个结构支持n维映射数据类型。

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

**图4。**映射型内存块的抽象结构。

在值级别，Struct内存值由Struct数据类型表示。因此，它类似于normalform值，可以通过readcheck直接提取（见表14）。

字段访问的语义非常灵活，包括契约成员访问和结构字段访问。如果契约成员访问是从继承关系或特殊标识符派生的，例如，可以基于ML模块系统直接访问契约成员。对于字段访问语义的第二部分，Lolisa支持所有类型的struct字段访问。然而，随着公约1的引入，中间成员不能是函数。为了模拟用户节点通信函数的行为，例如发送和调用，这些通信函数必须修改相应的通信逻辑地址。例如，如下所示，字段成员地址是send函数的隐式参数。

*认捐*[i] （保证[i]）*.地址.发送.数量*

≡ (（我）*发送.地址,v,mss公司.*

因此，在表15中，如果memsfind成功地计算了类型为atype的变量ainit，它将返回一对

*（爸爸）,毫伏)*，其中Dad指成员memn的地址−1.

***表达式层***表达式的可执行语义是获取值层结果的规则，是对表达式的内存值进行计算的规则。求值需要左值（l值）和右值位置，分别表示内存地址和特定内存值。在下面的内容中，表达式层ESE的输入指针由规则5和6定义。

|  |  |
| --- | --- |
| ESE：：Z→ (∀τ: → *我*0*τ*1 *类型,出口τ*0*τ*1*) 记忆* |  |
| → → → *Blc公司环境可选Laddress*  ESE：：Z→ (∀τ: → *r*0*τ*1 *类型,出口τ*0*τ*1*) 记忆* | (5) |
| → → → *Blc公司环境期权价值* | (6) |

在正式的Lolisa语义中，修饰符表达式是一种特殊的表达式，无法在expressionlayer中进行评估（后面将在语句语义中解释）。计算语义定义为ESE≡ 埃塞≡ .*伊莫迪rmodi公司错误*

***l值位置的表达式：***以下规则定义了在l值位置（即，相应的内存地址）计算表达式的语义。位于l值位置的表达式（可以由Econst构造函数构造）在表达式级别表示Lolisa值。具体来说，左值可以指定为Varray和Vmap指定的Econst。如前所述，Varray和Vmap是指向存储在特定内存块中的值的地址指针。例如，如规则7所示，在赋值语句中用作左值的数组通常用于大多数通用编程语言。

*A*[我]= *一.* (7)

因此，Varray和Vmap不仅可以表示内存值，还可以表示内存地址。注意，剩余的值（Vstruct和Vfield）也是通过指定Econst构造函数的地址指针，但不能表示位于l值位置的表达式。因为Evar可以表示任何类型的变量和压力，包括结构和字段访问值。为了避免Vstruct和Vfield指定的Evar和Econst之间的混淆，我们设置了Vstruct和Vfield仅在值级别表示内存值的约定。

|  |
| --- |
| **图5。**存储在胚芽中的二维映射值的结构。 |

在Solidity和Lolisa中，Vfield承认了许多特殊的结构，比如msg和block，它们的成员不能随意改变。在极少数情况下，Solidity允许字段访问表达式位于左侧位置。因此，为了确保Lolisa保持良好的格式和行为，Vfield不能计算表达式中的left值。结构的字段可以通过调用Estruct来更改所有字段，或者通过声明一个新字段来更改，如下所述。尽管Vstruct和Vfield的局限性给程序员和验证者带来了不便，但它们避免了任何潜在的风险。

此外，如果构造函数是Varray，则语义选择为ESE（定义见表16）；如果构造函数是Vmap，则选择它们作为ESE（表17）。其他构造函数直接返回None。表达式级别的左常量值ese的语义在A的表18中给出。通过引入gaspspecialhelparfunctionlexprcheck来避免IRE问题，该方法封装匹配树以获取记录在有效构造函数中的值信息。*莱克斯普拉雷词汇地图lexprconst公司,*

引用表达式Evar、Efun、Econ和Epar只需要直接返回它们的地址。在A的表19中，这些引用表达式总结为Eaddr[[name]]：[[name]][[name]]。Estruct、Ebop和Euop表达式只能指定为正确的表达式值，因此它们的语义由elexprcheck函数标记（返回undefined result None）。*() expreaddr公司()eaddr公司()*

***r值位置的表达式：***以下函数描述在r值位置（即，相应的存储器值）中计算表达式的语义。常数表达式的计算见A的表20。

在ESV的帮助下，ESE直接提供各自的内存值。*价值雷克斯普康斯特*

根据约定1，结构构造函数Estruct在正确的位置表示表达式值，这是初始化或修改结构类型术语的唯一方法。right struct值的语义在A的表21中定义。助手函数evalstr包含类型匹配和值求值。类型匹配部件检查每个值的类型是否满足相应的字段。

第二部分递归地调用esvt来计算各个内存值中的值。如果求值过程产生一个None消息，则Estruct求值失败。否则，将检索成员的值集并返回相应的struct memory值。*价值*

最后，二进制和一元操作的语义分别在A的表22和23中定义。基于GADTs的形式化抽象语法定义由于静态类型的限制，所有的表达式、子表达式和操作都是格式良好的，语义不需要检查类型依赖关系。因此，不需要辅助功能。函数sevalbop和evaluop将表达式求值的结果和所需的操作作为参数，并将它们组合起来生成新的内存值。在FEther的当前版本中，上述定义禁止混合算术运算，例如“int+float”，因为Solidity不完全支持float数据类型，并且在智能合约程序中很少使用float值。因此，在实现形式化解释器时，混合运算会增加不必要的复杂性和计算负担。

***语句层***定义了语义之后，我们现在可以定义语句层了。语句语义分析Lolisa编写的FRWprograms，并评估新的内存状态。序列语句的语义没有明确定义，相关的语句定义被修改以提高求解CBNT问题的极低计算效率。我们将语句的求值过程表示为asESS，并将其抽象定义为rule8。

ESS：：Z→ → *记忆期权（列表值）)*

→ → → → *环境环境声明选项存储器*

(8)

大多数语句求值都使用帮助函数envcheck，它将当前环境env和超级环境fenv作为参数，并检查条件（gas限制和执行级别有效性）。例如，如果env和fenv中的域相等，但执行级别不同，那么程序将被终止，env将被fenv重置。如果envcheck返回true结果，则执行当前语句；否则，程序终止并恢复初始内存状态。

合同声明是最重要的可靠性声明之一。Lolisa中的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，该函数确定当前继承关系inherits是否存储在当前模块上下文C中。

在规则9中，函数inheritcheck被定义为sum类型。

*继承检查*≡ ∀(: *继承遗产列表地址),*

{= } + {6= }*继承固有的继承固有的*

(9)

其次，辅助函数writedir将初始契约信息（包括所有成员标识符）写入指定的存储器块中。合同声明的形式语义在A的表24中定义。

变量声明是Lolisa的一项基本任务。函数initvar是writedir的一个特殊类，其类型由规则10给出。

*初始化变量*:: → → → *记忆环境Blc公司选项访问*

→ → → *类型地址选项存储器*

(10)

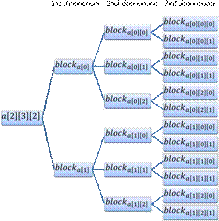
此函数将当前内存状态、变量类型、索引地址和环境信息作为参数，并初始化相应的内存块。基于细菌记忆模型，该项在数组数据类型中的初始化和定位过程与标准的数组类型形式化研究不同。函数initvar调用initarray函数来初始化相应的项。

在[14]中，普通类型是类型规则不允许递归定义的数据类型。普通类型被指定为。类型是多维数组的递归基。换句话说，如果当前元素的值为，则表示当前多维数组的递归定义的最终维度。*τ最终的τ最终τ τ最终的*

特别是，由于GERM内存模型中的每个内存块直接存储类型值为[6]的所有逻辑信息，因此无论数组元素的大小，我们只需要计算数组中逻辑元素的数量。为了说明计算过程，我们以以下类型为[iAconstid2[iAconstid3[iAconstid2]]的三维数组a[2][3][2]为简单示例。此数组的全树结构如图6所示，数学求值过程如下：*()柏油()柏油()τ最终的*

*（尺寸*1 + *大小*1 ∗ (*大小*2 + *大小*2 ∗ *大小*3*))*

= (2 + 2 ∗ (3 + 3 ∗ 2= 20*)) .*



**图6。**三维数组的示例。

注意，这个数组需要20个块的内存分配。

数组大小的计算可以归纳为规则12。利用这个公式，我们可以实现辅助函数idarray来计算并返回分配给每个维度数组的块数。此赋值的摘要在规则13中定义。请注意，规则11和13中的size1表示当前维度的大小，而不是第一个维度的大小。例如，要计算[2][3][2]的第二维度的arraysize，我们应该将规则12和13中的size=2替换为size=3。01 02

*排列*≡ + ∗ (+ ∗(...(−+ −∗ *大小*1 *大小*1 *大小*2 *大小*2 *四旬*1 *四旬*1 *四旬)))*

*尺寸J*(11)

*组大小*≡ 1*数组大小/大小*

                                                                      *尺寸J/大小*1 (12)

*伊达里*:: → → → Z（13）*索引记忆环境期权*

图7显示了一个[2][3][2]的初始化过程，它遵循它的树结构。在步骤（1）中*,* )，FEther根据规则11搜索总大小为20个块的连续内存空间。然后，算法树初始化将[2][3][2]分类为两个初始树，由[0]和[1]索引。两个组中的元素都由initarray按顺序递归初始化。例如，在recursionofa[0]中，idarray将[0]作为大小计算组索引的sesthesizeOfGroupIndexedBy a[0]*/大小*

|  |
| --- |
| **图7。**数组值初始化过程（详见正文），以及在内存空间中的[2][3][2]数组的最终结构。 |

从块0到块9的块。步骤（2）分配内存块。因为[0]也是整个数组的起始地址，所以它被分配给Block0（图7的步骤（2）中的深蓝色块）。为了分配第二维度的内存块，initarray必须继续到下一个级别，递归地初始化由[0][0]、[0][1]和[0][2]索引的子组。在步骤（3）中，由[0][0]索引的组的信息存储在Block1中，Block1需要size+*,* 02 *大小**大小**/大小*从区块1到区块3的3个区块（图7步骤（3）中的绿色区块）。为了分配第三维的内存块，initarrray在步骤（4）和（5）中继续深度递归，这两个步骤初始化了[0][0][X]∈ {01}. [0][0][X]中的元素属于类型，因此这是组大小为的叶节点级别*,十,τ最终的*03*/大小*1.换句话说，a[0][0][0]和a[0][0][1]是单元素组。两个组都连续地存储在Block2和Block3中（图7的步骤（4）中的橙色块）。步骤（6）–（8）将内存状态mstate0恢复到第二维度的递归级别，并对[0][1][X]重复步骤（3）–（5）。对其余组重复此过程，直到整个阵列信息已存储到相应的内存块中。内存空间中[2][3][2]的最后一个结构，从图7中的Block0到Block19。这里，左栏（9）是真实结构，右栏（10）是组分类。

idsearch可以用类似的算法来实现。但是，由于idsearch直接定位索引组而不是搜索每个组，因此其核心过程是addressoffset+，。偏移量公式如规则14所示。例如，在定位[0][1][1]的块时，偏移量计算为0∗ 10 + 1 ∗ 3 + 1 ∗ 1+2=6，初始地址为Block0。因此，[0][1][1]的信息存储在块6中。*(抵消,初始化地址)*

十*n*

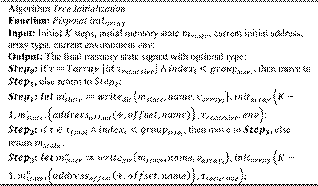
*抵消*≡ *i索引XI*∗ *组大小我*+ (*n*− 1*)* (14)

=1

一般来说，如果瓦利达雷∧ = , 数组空间可以由表6初始化，称为树初始化。*(τ) 分配（M状态,排列) 一些初始地址*

在运行该算法之后，FEther中的符号执行比其他使用list数据类型的形式化更准确地模拟了n维数组在实际硬件中的初始化和分配行为。数组可以由许多有趣的算法抽象，例如

**表6。**初始化函数的算法。*数组*



树结构映射[25]或图形映射[26]，但这些算法的优点部分被缺点所抵消。例如，虽然它们可以表示一个有限的内存空间，但它们的规范和形式结构非常复杂，很难扩展。此外，要修改数组元素，操作必须逐个搜索每个节点，如果没有依赖类型，溢出问题很难检查。在一种基于细菌记忆模型的算法中，数组存储在固定大小的连续内存空间中，而不需要依赖类型的帮助[27]。验证器可以形式化地模拟地址偏移过程，通过检查存储在内存块中的头标志来检查数组溢出问题，并通过索引相应的内存地址来直接修改数组块。因此，验证过程变得更容易和更准确。

假设基于GERM的当前逻辑上下文有足够的逻辑内存空间，并且每个标识符都有一个有效的空闲地址，initvar表示第一次设置索引内存块，writedir总是成功的。本场景中的变量声明语义在A的表25中定义。

结构数据类型声明的语义在A的表26中定义。按照约定1，语句级的结构声明使用地址标识符str声明新的结构类型。stris mems的字段成员列表。例如，图8定义了由Lolisa重写的Solidity的内置地址数据类型。0xu地址是str，其余字段是mems。ess将结构类型信息直接记录到地址为str的内存块中。*ττ τττstr公司τ*



**图8。**Solidity中的地址类型声明，以及Lolisa语法中的等效特殊结构类型。

在Lolisa中，函数调用语句展开存储在相应内存地址中的函数体。函数调用的语义如表27所示。在第一步中，函数调用尝试读取存储在相应内存地址中的函数声明语句。如果读取成功，第二步将当前执行环境级别设置为0，并且（在setenv的帮助下）将域设置为被调用的函数标识符。在最后一步中，使用新的env&apos;执行函数体。*呼叫*

修饰符声明是特殊的函数声明，需要三个步骤，包括一个限制。参数值由setpar谓词设置。如A中表28所定义，第一步初始化并设置参数。第二步将修饰符体存储到相应的内存块中，第三步尝试初始化返回地址。在一致性规则下，修饰符主体可以返回检查标志，但不能更改内存状态。因此，在FEther中，我们在GERM框架中添加了一个特殊的布尔型内存块，由0xmodifier索引。如果修改器检查成功，则将块设置为true并指定为；否则，设置为false并指定为，这意味着不能修改其他块。*3乐趣σ真的吗σmfalse公司*

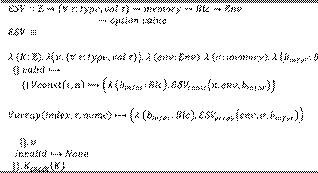
为了保证类型安全，Lolisa分别定义了单返回值函数和多返回值函数。然而，如A的表29所示，我们将它们组合起来，使得返回类型和修饰符限制都定义为列表。评估由repeat函数完成。与修饰语语义不同，函数语义检查限制函数的修饰语限制。具体来说，所有限制函数的修饰符都是在调用函数之前执行的。如果修饰符求值的modifchck结果为真，则执行该函数；否则终止。特别是，如果modifchck发现修改过的内存状态，则丢弃执行。

赋值语句语义是基于表达式求值语义的。如果对r值表达式求值的结果是由字段访问生成的函数指针，则返回值由函数调用语义求值。赋值语句的语义在A的表30中定义。

*C.费瑟分析器*

为了分析FRWprograms的语法单元，必须将语义集成到一个在ISA上易于实现的解析器中。如图2所示，解析器有三个层来解析这三个语法层。这些层的功能是验证环境、解构输入语法单元、将语法单元映射到相应的语义ES以及将存储在Sto中的信息传输到ES。例如，考虑表7中的值层。首先，ESV检查K限制。然后通过模式匹配将输入值v分解为特定的构造函数。最后，逻辑数据被传输到各自的语义中。*我我我我*

**表7。**值层解析器的简单示例。



因此，解析器可以概括为类型判断15和16，其中valid表示验证过程。

E``σ,= S≈ 锿*环境,芬文binfor有效（千）,环境,芬夫) 是的我我*

(15)

环境足迹⇒ 锿*,M,我（args）) 我σ,环境,芬夫,宾福,参数*

E``σ,F`=S≈ 锿*环境,芬文宾福K有效（千）,环境,芬夫) 假我我*

(16)

环境足迹⇒ *,M,我（args）) 没有*

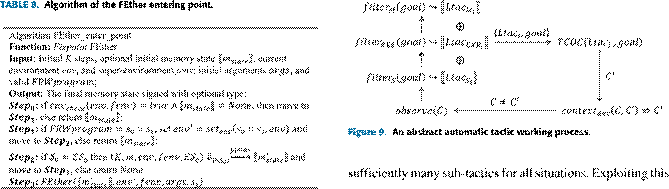
Sneeds中的信息在传输到ES之前需要进行部分预处理。首先，我们必须检查ESis Efun的构造函数是否正确。如果是真的，我们必须传递各自的名称而不是名称。此操作记录为*我我rexpreaddr公司3有趣的*

|∗) 7−→ *依繁钢管舞（oaddr公司）,τ,readchck公司σ,环境,宾福,3乐趣*. 如上所述，在评估ESSand ESS之后，我们必须将当前环境更改为停止函数执行的超级环境。此外，由于ESS、ESE和esr等语义可递归调用ESE，因此必须将特定的ESE和ESS定义为递归函数。最后，解析器语句级集成了两个较低的级别，还定义了FEther的入口点（参见表8中给出的FEther\u enter\u point算法）。*重新物件呼叫rexpr公司防喷器rexpr公司计量单位rexpr公司rexpr公司*

管理费瑟中Lolisa程序执行的规则由规则（EXE-F）和（EXE-IF）定义，如本页底部所示，其中符号∞ 表示无限执行，T是有限执行的终止条件集。

*D.自动化策略*

自动定理证明是形式化验证研究的核心课题。许多高阶定理证明助手提供了简化程序计算过程和自动构造证明的策略或类似机制。利用人工建模技术，可以在不同的程序中构造具有显著不同结构和验证过程的形式化模型。因此，设计一套在不同程序中自动验证模型的策略几乎是不可能的。

  
费瑟回避了上述问题。根据EVI理论，FEther符号执行对应于函数评估和程序验证（见规则17）。也就是说，通过简化FEther中的程序求值过程，统一了高阶定理证明助手中不同程序的验证过程。由于FEther执行的情况构成了一个固定的有限集合{}，因此我们可以利用Ltac机制来设计FEther的原始自动策略。该策略模型由内存操作、K代价和语义简化三部分组成。*s*0*,s*1*,...,山猫*

*嗯，M,*F`*ins P公司执行*≡ *P评估*≡ *P验证*(17)

工作流如图9所示。当Coq的证明域打开时，observe函数扫描当前上下文C以获得当前目标。各部分依次抓住当前目标的操作特点，选择匹配策略。将所选策略组合成解决方案策略Ltaci，以解决TCOC中的目标。在contextdec中，新的上下文Cis与C进行了比较。如果C与C相同，则当前策略不能自动求解目标，策略模型终止。否则，策略模型不断试图简化C。0 0 0

表达式18表示存储操作部分的副主题unfold-u-modify策略。此子策略捕获writedir函数的部分操作特性，并使用基本的内置策略评估扫描的writedir。

***Ltac公司****展开并修改*：=将目标与

|[| − 上下文[？Y（？X:内存）（？Z:值）]]

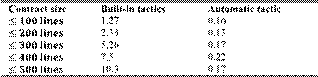
⇒ 展开Y-in；cbn输入∗∗

        结束（18）*.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| E``σ,F`EF`EF`==*环境,芬文宾福猫眼石,M,P（标准测试）),M,库环境凝析气（初始环境）（第（标准测试）)))芬夫初始环境（第（标准测试）)) σ 初始化内存（第（标准测试）)，库)*  （执行-F）  引用​​`H⇒,小时σ0，我*,M,恶臭,环境*0*,芬夫,参数,佩克塞特T环境*0*,芬夫*   |  |  |  | | --- | --- | --- | | E``σ,F`EF`EF`==*环境,芬文宾福猫眼石,M,P（标准测试）),M,库环境凝析气（初始环境）（第（标准测试）)))芬夫初始环境（第（标准测试）)) σ 初始化内存（第（标准测试）)，库)* |  |  | |  | *.* | （EXE-IF） |   环境足迹*,M,恶臭**,环境*0*,芬夫,参数,P**执行*H⇒,∞ 小时σ0，我∨ *环境*0*,芬夫环境*0*（天然气）)*  *执行,T*0            0                                                 → (¬H⇒ 小时σ ,我*芬夫（气体极限）)) 环境,芬夫* |

合同规模与证明规模的平均比率见表9。500行以外的智能合同被排除在本分析之外，因为大型合同的规模

**表9。**定理证明策略中的证明规模与契约规模之比。



受天然气成本的限制。表9的第二列和第三列分别列出了使用Coq的内置策略和我们的自动策略的比率。显然，自动策略减少了大量的证明工作量。此外，根据我们的实验结果，比率在一定范围内的浮动受目标契约复杂度的影响。具体来说，通过内置战术获得的比率范围约为−05到+100，而自动战术的范围从大约−因此，自动战术比直接应用的内建战术具有更好的通用性。*....*

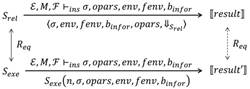
*E。自我正确性认证*

费瑟解释器完全是用Coq构造的，这使得它比其他程序验证和分析工具具有天然的优势。Coq的核心是可信计算库（TCB）[28]，它满足de Bruijn准则。在几乎所有的程序分析工具中，TCB自我验证都是有争议和矛盾的，因此程序验证（分析）工具的TCB是否满足de-Bruijn准则是验证可信度的一个重要指标。

FEther的正确性是通过关系定义和计算定义的一致性、本质属性的正确性和语义的元属性来证明的。

首先，我们必须证明Lolisa（归纳关系形式）的操作语义[29]等价于操作语义（可执行函数形式）。正如CompCert项目[25]所期望的那样，我们检查关系语义中的每个求值是否对应于可执行语义中的符号执行。为此，我们构建了一个仿真图。在相同的条件下，关系语义和可执行语义必须具有相同的可观察效果（评估过程的相同轨迹）。这一要求体现在以下仿真图理论中。

**定理**（simulationdiagram）LetEF`作为初始评估环境，让Req表示两个术语之间的等价关系。那么，任何关系语义Srel和可执行语义Sexe必须满足以下模拟图：*,M,ins公司σ,猫眼石, 环境,芬夫,宾福*



其次，我们必须证明可执行语义的基础行为的正确性。作为一个简单的例子，我们构造了引理test\u引理if\u false，它证明了以下执行的正确性：对于所有语句s和s，如果if语句条件为false，则FEther必须执行false分支的语句。通过类似的过程，我们证明了几乎所有可执行的语义都表现出标准的行为。00

*引理（Test\u Lemma\u If\u False）：*

∀ 0 = → = 0→ 0→*如果假if州n环境通行证, 如果为假经济（Vbool错误）) 如果州如果如果假s sn>*

*FEther n initm pass env env if假*（FEther n init \u m pass设置）*（环境)环境*0*.*

最后，我们证明了这些语义的元属性。每层中最基本的属性是进度和保存属性，它们维护规范的静态类型安全。例如，在引理表达式类型安全中定义了表达式层的进程和保存。由于Lolisa是用GADTs定义的强类型语言，通过简化语义函数可以很容易地证明表达式的进步性和保持性。其他层的进度和保存性能也得到了类似的认证。除了元属性之外，我们还证明了Coq中所有语义的执行决定论。引理执行决定论是相关证明的一个例子。

*引理（表达式类型安全）：*

1.              如果e:和e7−→ , 然后e:0。*出口τ*0*τ*1 *e*00 *出口τ*0*τ*1

2.              如果e:，那么要么e7存在，要么e7存在−→ .*出口τ*0*τ*1*（五）)* 0 *e*0

*引理（执行决定论）：**最终**涅夫帕斯*,

*费瑟n m pass env env s**最终*→

hh 0Ⅱ=→*费瑟n m pass env env s最终*

*米最终*= 0*米最终的*

目前，核心功能已经完全验证。正确性证明包括74个定理和引理，以及大约4000行Coq证明代码。

**五。智能系统的形式化验证**

## 费瑟契约

为了展示费瑟在现实世界实践中的力量，本节以从契约演示[2]中提取的智能契约为例，说明使用费瑟的验证过程和特征。接下来，我们将费瑟与其他类似作品进行比较。实验环境是五台相同的个人计算机，硬件相当于8gb内存和3.20ghz CPU。所有计算机都运行在windows10和coqide8.8上。

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

**图10。**钱包功能的正式版本。

*A.案例研究：混合验证*

首先，智能合约的.sol文件在翻译人员的协助下，自动逐行从Solidity翻译成Lolisa。作为一个简单的例子，我们考虑附录B中编码的wallet函数。该函数执行首次硬币发行，是从[2]中提取的实体契约的一部分，其形式化模型转化为图10，作为图10与附录B的比较，严格保证了智能契约的一致性。

钱包最重要的要求之一（不是及时的）是应用程序时间验证。显然，当前时间低于privilegeepen或高于privilegeClose是无效的。因此，如果钱包功能中的当前时间超出打开到关闭的范围，则必须放弃智能合约。

根据EVI理论，提出的FSPVM的验证是基于同时Hoare逻辑和可达性逻辑的。同时，FEther中的验证结合了高阶定理证明和符号执行。借助费瑟，程序员可以机械地定义公式摘要（19）后面的hoarestyle属性，其中通配符∗&apos;&apos; 表示其他特定参数。

P{minit}fetherminitfrwp程序∗)Q{M最终}（19）*(,,*

根据可达性逻辑，Hoare逻辑的推导等价于可信操作语义的执行。因此，对费瑟的执行可以看作是一种基于霍尔逻辑的推导。推理过程由表达式（20）给出。特定的初始存储状态minit是程序验证的前提。

在每个语句cici的语义指导下，FEther逻辑地修改当前的内存状态mi−一个新的后条件Qi{}（即cici的前提条件）。这些定理只需在执行final语句后判断最终输出的内存状态mn是否与正确的内存状态mfinal匹配。最重要的是，该验证过程在建议的FSPVM中是自动化的。1 *惯性矩*

P{minit}c0费瑟−米→初始化C0∗) 问题0{m0}c1*(,,*

FEtherm0c1型∗)*(,,*

                                     −→ 问题1{m1}c2→→ cnQn{mn}

?

                                    ←→ Q{M最终}（20）

在这个过程中，验证者可以通过以不同的方式定义前提条件来改变验证模式（包括静态的、共同的和选择性的符号执行）。例如，程序员可以通过以下三种方法改变钱包功能。

### 1） 静态符号执行

基本的验证模式是静态符号执行。当初始参数用量词归纳定义时，例如∀ 以及∃, 传统的符号执行将遍历所有情况。例如，图11中定义的引理no\u in\u time（用红框标记）定义了INT I64 Unsigned？xa和INT I64 Unsigned？yb，定义了？X和？Y作为表示时间的所有可能情况的归纳值∀(x:IntINT I64无符号x和∀(y:IntINT I64无符号y。并将其写入图中定义的初始存储状态m3中。接下来，钱包功能的时间要求应在一阶逻辑下定义为∧ ∨ (∧ .*() () )() )()（t）<倍y>t型) t型>十y小于)*

|  |
| --- |
| **图11。**使用抽象符号参数执行和验证钱包。 |

**图12。**手动验证钱包功能。

显然，预期的最终内存状态应该是智能合约终止。

最后，如图11所示，使用为费瑟设计的自动策略，钱包功能和相应的引理可以在5.772秒内仅在1行验证码下自动执行和验证。如图12所示，如果用户直接调用Coq的内置策略，即使证明代码只是简单的优化，也可以用21行证明代码验证引理。与使用Coq提供的内置策略进行手工验证相比，FEther具有更高的自动化程度。

### 2） 共同处决

第二，费瑟支持获得真实输入的共形符号执行。为了精确地模拟现实世界硬件上的执行过程，FEther被构建在一个虚拟的执行环境中。因此，祭物的执行可以看作是一种特殊的动态分析。如图13所示，entering points test和code wallet未被修改，privilegeOpen、providegclose和now被替换为特定值0、3和4，用蓝色框标记。其他约束仍然是归纳定义为抽象符号。然后用无输入时间引理证明了具有特定输入的共执行函数的正确性。因为

|  |
| --- |
| **图13。**钱包功能的共同验证。    **图14。**钱包功能的选择性符号执行。 |

输入被指定，可能的执行路径的数目被限制，并且执行时间减少到5.534s。

这种模式为测试留下了可扩展的空间。程序员可以开发辅助工具来生成修改输入值的自动测试脚本。

### 3） 选择性符号执行

第三，钱包的功能可以通过利用有选择的象征性执行费瑟。如图14所示，程序员可以提取核心代码段if（现在是| |）{（）；}从钱包函数，并表示它的一个新的定义，如msp&apos;在红盒，这可以单独验证的msp\ u正确引理。将已验证的msp&apos;组合到wallet函数中后，可以通过调用msp\ u correct引理来完成no\ u in\ time引理的验证。显然，msp\u correct还可以帮助使用msp&apos;代码段的证明。*<打开现在>关闭扔*

|  |
| --- |
| **图15。**Coq中钱包功能的调试。  **表10。**FEther语义与现有软件质量工具的特征比较。 |

这样，新属性的验证就可以反复调用已验证的代码段，提高了定理证明技术的可重用性。此外，FEther还可以简化循环证明。在高阶定理证明的标准方法中，程序循环是通过人工识别不变量来证明的。然而，搜索简单循环的循环不变量是一个繁琐的过程。通过结合符号执行和高阶定理证明，我们同时方便了BMC的使用和循环不变量的搜索。利用BMC，我们首先将FEther限制为K或更少的FRWprogram执行。一般来说，如果≤ 吉隆坡≤ K） 如果一个FRWprogram中的一个能生成相应的最终内存状态，则FRWprogram中存在的循环可以在有限时间内直接展开为一组相同的正常序列语句，如规则21所述。如果FRWprogram在K次执行后不能生成相应的最终内存状态，我们可以将loop语句设置为断点（通过选择性符号执行），并将FRWprogram分为两部分，分别表示为头部和尾部。接下来，我们必须定位循环不变量，并将它们封装到一个不变量内存状态I{mi}，它作为头部的最终内存状态和尾部的初始内存状态。这一程序载于下文第21条。

P{minit}c0型→→ ciI{mi}（头和*)*

                            ciI{mi}→→ cnQ{mfinal}（尾部（21）*)*

在霍尔逻辑的组合规则下，我们得到了P{minit}c0→→ cnQ{mfinal}。这样，可以自动证明简单循环，减少搜索循环不变量的工作量。此外，对于无法通过模型检验和符号执行技术验证的复杂循环，可以通过高阶定理证明技术进行证明。

### 4） 调试机制

最后，FEther为用户提供了一种调试机制。由于FEther是在细菌记忆模型中开发的，因此它提供了诸如step之类的调试策略，从而可以对智能合约进行逐步调试。在使用FEther的Lolisa程序的执行和验证过程中获得的正式中间存储器状态显示在证明上下文中（图15的右面板）。通过这种方式，程序员可以跟踪中间内存状态来定位bug。

显然，FEther的用户可以灵活地选择最合适的方法来验证他们的程序。

*B.功能比较概述*

费瑟是第一个混合符号执行引擎以太坊智能合约。为了说明FEtheroverthesolversofothertools的优势，我们需要一个完整的基准测试，例如测试套件或分析时间。然而，由于FEther是基于Coq构建的，并且直接执行和验证智能合约的源代码，而不是在字节码级别编译源代码，因此很难找到这样的基准。为了公平比较，我们反而比较了FEther和其他工具中各种特性的存在和不存在。下面列出并定义了比较的特征：

•规范：适合作为EVM语言的正式规范

•Exec.：具体测试的可执行文件

•认证：可认证的自我正确性

•验证：EVM程序的可验证属性

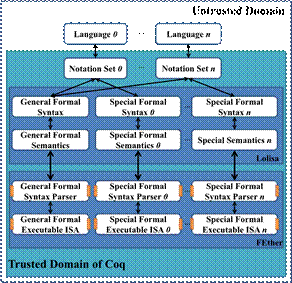
•调试：提供交互式EVM调试器

•气体：分析EVM项目气体复杂性的工具

•级别：代码的分析或验证级别

•逻辑：支持的基本逻辑类型

•混合验证：支持混合验证方法



**图16。**将Lolisa扩展到其他通用编程语言的详细体系结构。

表10概述了特征比较的结果。显然，只有FEther、KEVM的核心和Mythril支持Spec、Exec、Verif、Debug和Gas特性。FEther的Certif特性是“验证”而不是“测试”，这提高了FEther的可靠性（至少在理论上）超过了KEVM和Mythril等测试方法。此外，FEther的执行和验证级别是“稳定”而不是“字节码”，这避免了编译过程中的错误风险。费瑟还支持高阶逻辑，提高了表达能力。此外，费瑟的基本验证理论是归纳构造演算，而不是可满足性模理论或布尔可满足性问题。因此，无法评估和验证的情况并不存在。最后，FEther是唯一支持混合形式验证的工具。

根据我们之前的实验结果[24]，优化后的当前版本的FEther的符号执行时间约为0.03s/语句，当

**表11。**常量Lolisa值的语义。

当初始参数由量词归纳定义时，初始参数被指定为大约0.07s。FEther的执行效率远远超过了Coq中按照Coq标准教程开发的口译员。当前版本还支持验证遵循以太坊ERC20标准的智能合约模型。

#### 六、 讨论

*A.贡献*

本文克服了我们以前工作中提到的最后一个挑战：完成FSPVM-E的证明引擎。我们现在强调目前工作的重大贡献。首先，我们确认FEther保持了源代码的一致性和相应的形式规范。据我们所知，费瑟是以太坊第一个支持Coq混合验证技术的验证引擎。其次，它提供了一种调试机制，程序员可以通过该机制直接调试Coq中的目标智能契约。第三，费瑟的正确性在Coq中得到了充分的证明，这意味着费瑟是一个可靠的证明引擎。第四，我们为费瑟提供了一套专有的自动策略，这将帮助程序员以高度自动化的方式完成他们的属性验证。最后，优化了FEther的高级评价效率。我们确认了我们以前的工作在Coq中构建认证可执行证明引擎的效用。

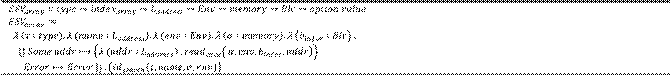
*B.可扩展性和通用性*

显然，中间层的定义解释程序必须忠实地捕获用真实编程语言编写的程序的预期行为。从灵活性的角度来看，同一个解释器也应该适用于多种编程语言。因此，费瑟设计从开发之初就考虑了可扩展性和通用性。

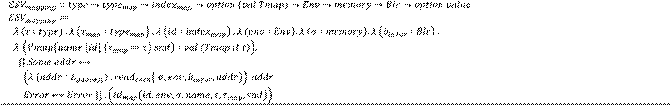
正如在[14]中提到的，我们故意在Lolisa中加入可扩展空间。这个空间足以扩展诸如指针形式化和实现独立的操作符定义等特性。通过添加新的代码，它可以很容易地结合主流编程语言的特性

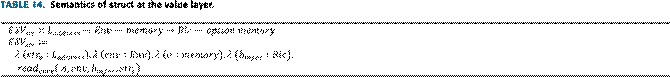


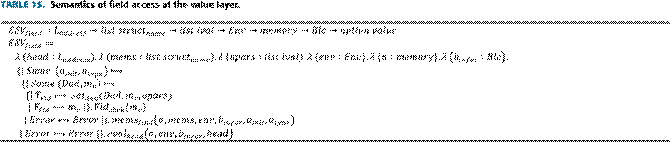
**表12。**值层数组类型的语义。



**表13。**映射值的语义。





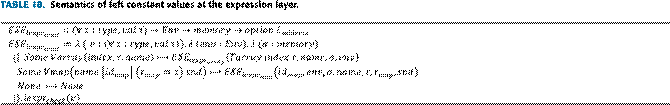


**表16。**表达式层左数组值的语义。



**表17。**值层字段访问的语义。





**表19。**表达式层的引用表达式Evar、Efun、Econ和Epar的语义。



在形式抽象语法和相应的形式语义中键入规则构造函数。此外，通过将Lolisa封装在语法糖符号N中，简化了Lolisa的形式语法。如规则22和23所示，Lolisa被视为核心形式语言，对现实世界的用户是透明的。Lolisa的形式语法和语义在逻辑上分为一般成分G和n特殊成分S（见下面的规则22）。通用的*我*

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

**表20。**表达式层上右常量值的语义。

  
**表26。**结构声明的语义。

编程语言lca可以被形式化为Lolisa子集G∪S、 其中Lis用语法符号N象征性地表示。这里，语法符号与L的原始语法符号几乎相同。此方法为每个lw分配一个满足N的相应符号集N⊆ . 这种关系由下面的规则23定义，也提高了Lolisa的可扩展性。*我我我我我我我我洛丽萨*

*洛丽萨*(22)

                                 ∀∈ 北爱尔兰↔ N≡ G∪ S（23）*我我我我*

|  |
| --- |
| **表27。**函数调用语句的语义。 |

作为Lolisa各自的定义解释者，FEther继承了Lolisa的可扩展性优势，并支持其所有的语法和语义。此外，在同一级别上，任何可执行语义i独立于任何其他语义，并且所有同一级别的语义被封装到一个独立的模块M中（参见下面的规则24和25）。高层语义可以在不同的Ms中访问底层语义的api，但实现细节是透明的。因此，如图16所示，FEther也很容易扩展到Lolisa中的新可执行语义，而不影响旧语义。

                       h i公司\

       M∀∈ N、 6==∅ (24)*奥杜列克我,j我j. 我是j*

*百米*:=

*C.局限性*

尽管《费瑟》当前版本中的小说特色赋予了它许多优点，但也存在一些局限性。

首先，FEther在源代码级别上运行。虽然它不会在编译过程中导入漏洞，但当编译器不可信时，它不能保证字节码的正确性。一个可能的解决方案是开发FEther的低级版本，执行编译生成的字节码。然后必须证明稳固性执行结果和字节码的相应执行结果之间的等价性。

其次，与其他符号执行工具类似，FEther遍历所有可能的执行路径，这有可能导致路径爆炸问题。然而，考虑到以太坊智能合约是轻量级甚至轻量级程序，几乎可以排除路径爆炸问题。此外，在确实遇到路径爆炸问题的情况下，可以通过定理证明技术将执行合并为不变量，并进行人工证明。这个解决方案将利用费瑟的选择性符号执行模式。

最后，虽然当前版本的FEther通过一些简单的自动策略实现了属性验证，但它还不是完全自动化的。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。

#### 七。结论和今后的工作

本文解决了FSPVM蓝图的最后一个挑战：在Coq中开发一个定义解释器。这个名为FEther的解释器支持以太坊智能合约形式验证的混合符号执行。基于细菌记忆模型，FEther准确地模拟了Coq中Solidity的执行行为，保证了源代码与相应形式化模型的一致性。为了评估费瑟执行过程中的完整情况，我们还设计了一套基于Coq的Ltac机制的策略，并将它们组合成一个庞大的自动策略。通过这种策略，FEther可以在一个具有高度自动化和可重用性的符号虚拟机中半自动地执行和验证不同的智能契约。为了证明费瑟在现实世界中的力量，通过费瑟中的传统符号执行（同时共现和选择性符号执行）验证了一个样本智能契约。我们还比较了费瑟的基本特征和相关工具的核心。费瑟的自我正确性已经通过证明Coq中的主要功能得到了证实。当前版本的FEther支持按照ERC20标准验证智能合约。最后，讨论了FEther的可扩展性和通用性，提出了系统化简化和扩展FEther的初步方案，从而支持多种通用编程语言的形式化。

我们希望FSPVM-E将变得足够强大和用户友好，便于普通程序员进行程序验证。目前，我们正在正式制定EOS区块链平台的高级智能合约开发语言[30]。我们还打算扩展和优化目前版本的费瑟。未来的版本将支持稳定的汇编语言和相应的字节码，如[31]。接下来，我们将扩展FSPVM-E以同时支持以太坊和EOS。将在GERM平台上开发这些语言的正式验证解释器。然后，我们将构建一个基于EVI的区块链智能合约通用形式化验证工具链。最后，我们将构建一个通用的形式化算法1部分钱包智能合约源代码。

Function wallet（）public payable{uint index=索引[msg.sender]；uint打开；uint关闭；配额；单价；uint部分限制；完全限制；最终限制；

if（privileges[msg.sender]）{open=privilegeOpen；关闭=特权；配额=privilegeQuota；

速率=速率else{open=普通打开；关闭=普通关闭；配额=普通配额；

rate=普通费率；

}

if（now<open II now>close）{revert（）；

}if（subscription>=TOKEN\_TARGET\_AMOUNT）{revert（）；

}if（index=0）{revert（）；

}if（deposits[index]>=quota）{revert（）；

}if（msg.val ue=O）{revert（）；

}如果（msg.val ue%10000000000000000！=0）{revert（）；

}

partiLimit=配额-存款[指数]；总限额=（（TOKEN\u TARGET\u AMOUNT-subscription）-（TOKEN\u TARGET\u AMOUNT-subscription）%费率）/费率\*

1000000000000000000;

如果（部分限制<=总限制）{

final Limit=partiLimit；}else{final Limit=总限额；

}

if（msg.value<=final limit）{安全传输（msg.value）；存款[指数]+=消息值；

订阅+=msg.val ueI I 000000000000000000000000\*费率；

传输（msg.sender，msg.value）；

}其他{

安全转移；存款[指数]+=最终限额；

认购+=最终限额I 00000000000000000000\*费率；传输（msg.sender，finalimit）；msg.sender.transfer（msg.vaI ue-fi naI Limit）；

}

}



基于EVI的区块链智能合约验证工具链，最终目标是智能合约自动验证。

**附录A**

可执行的语义是11到30。

**附录B**

见算法1。

## 致谢

作者希望感谢Marisa在实验验证过程中给予的友好帮助，以及在本手稿编写过程中给予的口头语言帮助。

## 参考文献

[1] S。中本。比特币：点对点电子现金系统。访问时间：2008年[在线]。可用：https://bitcoin.org/bitcoin.pdf

[2] . 访问日期：2018年7月2日[在线]。可用：https://solidity.readthedocs.io/en/develop/*以太坊实体文档*

[3] (2016). 被攻击的DAO：代码问题导致6000万美元

*盗窃*. 访问日期：2017年6月17日[在线]。可用：https://www.coindesk. com/dao-attacked-code-issue-leads-6000万-ether-theft/

[4] *以太坊奇偶校验黑客可能影响50万以太币或146美元*

*百万*. 访问日期：2017年12月2日[在线]。可用：https://www. crowdfundinsider.com/2017/11/124200-以太坊-parity-hack-mayampact-eth-500000-1.46亿/

[5] 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”，在Proc。ACM SIGSAC配置计算机。公社。Securit，美国纽约州纽约市，2016年10月，第254-269页。

[6] Z。杨和H。“雷。”Coq中的一个通用形式内存框架，用于验证基于高阶逻辑定理证明的程序的属性，增加了“.”访问日期：2018年3月27日。[在线]。可用：https://arxiv.org/abs/1803.00403

[7] Z。杨和H。Lei，“智能合约验证的正式流程虚拟机”，国际期刊。可执行性工程，第14卷，第8期，第1726-1734页，2018年8月。

[8] C。卡达尔，D。邓巴和D。Engler，“KLEE:复杂系统程序高覆盖率测试的独立自动生成”，在Proc。OSDI，2008年，第209-224页。

[9] B。Ekici et al.，“SMTCoq:将SMT解算器集成到Coq中的插件”，在Proc。内部配置计算机。辅助验证，2017年，第126-133页。

[10] 五。R。普拉特，“弗洛约霍尔逻辑的语义考虑”，在Proc。证监会，1976年10月，第109-121页。

[11] G。反渗透ş美国和美国。Ş聚四氟乙烯ănescu，“从霍尔逻辑到匹配逻辑可达性”，在Proc。内景交响乐团。正式方法，第74362018卷，第387-402页。

[12] . 查阅日期：2018年7月23日。*Coq证明助手参考手册*

[在线]。可用：https://coq.inria.fr/distrib/current/refman/

[13] 第。Wadler，“作为类型的命题”，Common。ACM，第58卷，第12期，第75-84页，2015年12月。

[14] Z。杨和H。“雷。”Lolisa:solidity编程语言的一个子集的形式语法和语义。“”访问日期：2018年4月1日。[在线]。可用：https://arxiv.org/abs/1803.09885

[15] G。木头。以太坊：一个安全的分散式通用交易账本。访问日期：2018年4月[在线]。可用：http://yellowpaper.io/

[16] 是的。Hirai，“为交互式定理证明者定义以太坊虚拟机”，在Proc。国际金融密码会议。《数据安全》，第10323卷，第2017卷，第520-535页。

[17] D。第。穆利根，S。欧文斯，K。E。格雷，L。里奇和P。Sewell，“Lem：现实世界语义的可重用工程”，在Proc。ICFP，2014年9月，第49卷，第9期，第175-188页。

[18] E。Hildenbrandt et al.，“KEVM:以太坊虚拟机的完整形式语义”，在Proc。IEEE 31计算机。塞库。找到了。症状(CSF），2018年7月，第204-217页。

[19] H。巴伦德雷格特和E。巴伦森，“形式证明中的自给自足计算”，J。自动。《推理》，第28卷，第3期，第321-336页，2002年4月。

[20] . 查阅日期：2018年4月23日。*Mythril文档和用户手册*

[在线]。可用：https://github.com/b-mueller/mythril/

[21]小时。十一，C。陈和G。Chen，“保护递归数据类型构造函数”，ACM SIGPLAN，第38卷，第1期，第224-235页，2003年1月。

[22]答。比尔，A。西马特，E。克拉克和Y。朱，“没有BDDs的符号模型检查”，在程序中。塔卡斯，荷兰阿姆斯特丹，1999年，第193-207页。

[23]页。计算机科学逻辑方法中的“带构造函数的类型化lambda演算的语义”。德国布伦瑞克：技术大学ä布伦瑞克，2011年，第561-576页。[在线]。Available:https网址：lmcs.epiciences.org/page/lmcs-ev

[24]字。杨和H。Lei，“高阶逻辑定理证明系统中可执行形式解释程序的优化”，IEEE Access，第5卷，第70331-70348页，2018年。[在线]。可用：https://ieeexplore.ieee. 组织/文件/8531607

[25] . 访问日期：2018年4月23日[在线]。可用：http://compcert. inria.fr/man/manual.pdf*CompCert C验证编译器：文档和用户手册*

[26]右。O&apos;Connor，“简单性：区块链的新语言”，在Proc。研讨会计划。朗。肛门。美国纽约州纽约市，2017年10月，第107-120页。

[公元27年]。以上和P。Dybjer，“工作中的依赖类型”。LNCS，2009年，第57-99页。

[28]乙。兰普森，M。阿巴迪，M。Burrows和E。Wobber，“分布式系统中的认证：理论与实践”，ACM Trans。计算机。《系统》，第10卷，第4期，第265-310页，1992年11月。

[29]天。米勒，“逻辑中操作语义规范的形式化”，电子版。理论笔记。计算机。科学，第246卷，第147-165页，2009年8月。

[30] . 访问日期：2018年4月23日[在线]。可用：https://eos.io/*EOS区块链平台*

[31]秒。阿曼尼，M。Bé凝胶，M。博丁和M。Staples，Proc中的“在Isabelle/HOL中验证以太坊智能合约字节码”。第七届ACM SIGPLAN国际会议认证课程证明，2018年1月，第66-77页。

[32]我。谢尔盖，A。库马尔和A。霍博尔(2018). &apos;&apos;Scilla：一种智能合约中级语言。可用：https://arxiv.org/abs/ 1801.00687

郑洋于2017年获得中国电子科技大学信息与软件工程学院学士学位，目前正在攻读博士学位。学位。他的研究兴趣包括编程语言理论、形式化方法和程序验证。

杭雷获得博士学位。1997年获中国电子科技大学计算机科学学位，毕业后在计算机系担任教授，从事实时嵌入式操作系统、操作系统安全和程序验证等领域的研究，现任中国电子科技大学信息技术学院教授（博士生导师）-

信息与软件工程。他的研究兴趣包括大数据分析、机器学习和程序验证。

