一种以太坊虚拟机字节码的正式验证工具

美国伊利诺伊大学厄巴纳-香槟分校
运行时验证公司，美国
dpark69@illinois.edu

美国伊利诺伊大学香槟分校
运行时验证公司，美国
yzhng173@illinois.edu

美国伊利诺斯维萨克塞纳大学厄巴纳香槟分校
运行时验证公司，美国
msaxena2@illinois.edu

康奈尔理工学院，美国IC3，美国
运行时验证公司，美国
phil@linux.com

格里戈尔罗斯，美国伊利诺伊大学香槟分校
运行时验证公司，美国
grosu@illinois.edu

摘要

本文提出了一种针对以太坊虚拟机(EVM)字节码的正式验证工具。为了精确地解释EVM字节码的所有可能行为，我们采用了KEVM，一个完整的EVM形式语义，并实例化了Kframe的可达性逻辑定理证明器，为EVM生成一个校正构造演绎验证器。我们通过引入EVM的抽象和引理进一步优化验证器，以提高其可伸缩性。我们的EVM验证器已被用于验证各种高调的智能合同，包括ERC20令牌、以太坊Casper和DappHubMakerDAO合同。

演示视频网址：https://youtu.be/4XBcAclq0Vk

中国化学会概念

• **软件及其工程设计的→软件验证；**

关键字

以太坊，智能合约，正式验证，K框架

ACM参考格式：

大军公园、张毅、萨克斯纳、菲利普大安和格里戈尔罗斯u.2018.针对以太坊虚拟机字节码的一种正式验证工具。*在第26届ACM欧洲软件工程联合会议和软件工程基础研讨会论文集(ESEC/FSE‘18)，2018年11月4-9日，佛罗里达州布埃纳维斯塔湖，美国*。ACM，纽约，纽约，美国，4页。https://doi.org/10.1145/3236024.3264591

1 介绍

智能合同故障导致了数百万美元的资金损失，[1]并且需要严格的正式方法来确保合同实施的正确性和安全性。[2]智能合同是通常是用一种高级语言写的，比如solidity[3]或vyper[4]，然后，它被编译成以太坊虚拟机(EVM)字节码[5]这实际上是在区块链上运行的。

本文提出了一种用于EVM字节码的形式化验证工具。我们选择了EVM字节码作为验证目标语言，这样我们就可以直接验证实际执行的内容，而不需要信任编译器的正确性。为了在不遗漏任何EVM字节码的情况下精确地解释EVM字节码，我们采用了KEVM[4]，EVM的完整形式语义，并实例化了k框架的可达性逻辑定理证明器[10]，为EVM生成一个构造正确的演绎程序验证器。虽然它是听起来的，但初始的开件箱EVM验证器相对缓慢，未能证明许多正确的程序。我们通过引入特定于EVM的自定义抽象和引理，进一步优化了验证器，从而加快了在底层定理证明器中的验证搜索。我们一直在使用EVM验证器来验证高调的智能合约的完整功能正确性，包括多个ERC20令牌合约[13]，以太坊的Casper[6]合同，和DappHub的MakerDAO[7]合同。我们的验证工具和工件可以在[11]上公开获得。

贡献。我们描述了我们的主要贡献：

- 我们提出了一种正式的EVM字节码验证工具，它的能力足够强大和可伸缩，可以验证各种高调的、安全关键的智能契约。此外，据我们所知，我们的验证器是第一个采用EVM的完整形式语义的工具，能够完全解释EVM字节码的所有可能的角情况行为。有关与其他工具的比较，请参见第5节。
- 我们列举了在验证EVM字节码方面面临的重要的、具体的挑战，并提出了特定于EVM的抽象和引理来减轻这些挑战。（第2节和第3节）
- 我们提出了一个完全验证高调的ERC20令牌合同的案例研究。我们列举了在这些令牌中发现的不同行为，阐明了假设在ERC20实现中行为一致的任何API客户机的潜在安全漏洞。（第四节）

1 https://blog.ethereum.org/2016/06/19/thinking-smart-contract-security/
2 https://blog.ethereum.org/2016/09/01/formal-methods-roadmap/
3 http://solidity.readthedocs.io/en/v0.4.24/
4 https://vyper.readthedocs.io/en/latest/index.html
5 http://yellowpaper.io/
6 https://eips.ethereum.org/EIPS/eip-1011
7 https://makerdao.com/

ESEC/FSE‘18,2018年11月4-9日，美国佛罗里达州布埃纳维斯塔湖
©2018年计算机机械协会。ACMISBN978-1-4503-5573-5/11…15.00美元
https://doi.org/10.1145/3236024.3264591

2 evm验证挑战

验证EVM字节码是具有挑战性的，特别是由于内部的字节操作操作需要非线性整数算术推理，而这在一般的[7]中是不可判定的。在这里，我们提供了几个关于这些挑战的例子。

*字节管理操作。*EVM提供了三种类型的存储结构：本地内存、本地堆栈和全局存储。其中，只有本地内存是可用字节寻址的（即，用一个字节数组表示），而其他内存是可用单词寻址的（即，每个内存以一个由32字节组成的单词数组表示）。因此，一个32字节（即256位）字需要分割成32个字节块以存储在本地存储中，这32个块需要合并回来以加载到本地堆栈或全局存储中。这些字节级的分割和合并操作可以使用非线性整数算术运算进行形式化，如下所示。[8]
假设x是一个256位的整数。让x是nx的两个补体表示中的字节，其中索引0表示最不重要的位(LSB)，定义如下：

$$x_n \stackrel{\text{def}}{=} \lfloor x / 256^n \rfloor \bmod 256$$

假设合并是一个函数，它以一个字节列表作为输入，并在两者的补体解释下返回相应的整数值，递归定义为：

$$merge(x_i \cdots \bar{x}_{j+1} x_j) \stackrel{\text{def}}{=} merge(x_i \cdots x_{j+1}) * 256 + x_j \text{当我>j}$$

$$merge(x_i) \stackrel{\text{def}}{=} x_i$$

其中*和+是单词上的乘法和加法[9]。如果字节操作被盲目地编码为SMT定理，那么Z3，一个最先进的SMT求解器，超时，试图证明“x=合并(x,x)”。SMT查询可以被简化，以允许Z3有效地终止，例如，通过省略
乘法和加法的模缩减与关于遗漏的可靠性的附加推理相结合。尽管有这些改进，合并操作仍然会导致严重的性能损失，因为在内存中的每个加载/存储都需要解决大公式，这是一个非常常见的操作。

*算术溢出。*因为EVM算术指令执行模块化算术(即，+，，，/mod2)，检测算术
溢出对于防止由于意外溢出而造成的潜在安全漏洞至关重要。否则，例如，增加用户的令牌余额可能会触发溢出，导致当余额打包到低于预期值时的资金损失。没有标准的EVM级溢出检查，因此溢出检测在不同的编译器和库中有所不同。例如，Vyper编译器在256位无符号整数a和b上插入以下关于+b添加的运行时检查：

```
b == 0 || a + b > a
```

其中，+表示加法模量2。似乎可以很简单地表明上述公式等价于一个+b<2（其中
+是纯的无模还原的加成），但它不再是
琐碎的是，一旦上面的内容被编译为EVM。上述条件表达式的编译的EVM字节码可以用SMT-LIB格式编码如下：

```
（不（=(chop(+(bool2int(=b0))
                （bool2int（>（chop（+ab））a））））0）
```

其中(chopx)表示（xmod2。(bool2intx)然而，Z3未能（超时）证明上述SMT公式等同于+b<2。

*哈什碰撞。*关于SHA3哈列的精确推理是关键的。由于每次在EVM字节码中调用哈希函数时都考虑哈希算法的细节是不实际的，因此需要对哈希函数进行抽象。设计一个健全但有效的抽象并不是很简单的，因为虽然SHA3哈希不是在加密上没有冲突的，但合同开发人员假设在合同的正常执行过程中不会发生冲突。捕捉假设的一种简单方法是简单地将SHA3哈希抽象为单射函数。然而，这听起来并不仅仅是因为鸽子洞原理，因此
我们在抽象哈希函数时需要小心。

3 特定于evm的抽象

K的可达性逻辑定理证明器可以看作是一个符号模型检查器，具有关于循环和递归的共归纳推理（基本理论和实现的细节请参考[10]）。在当前的形式中，证明器通常将域推理委托给SMT求解器。底层SMT求解器的性能对整体性能至关重要。EVM字节码验证中涉及的域推理在许多情况下是不容易处理的，特别是由于非线性整数算术。我们必须设计自定义的抽象和引理，以避免不可处理的域推理，并提高了可伸缩性。

*本地内存的抽象。*我们提出了一个关于EVM局部内存的抽象方法，以允许单词级的推理。如第2节所述，由于本地内存是可字节寻址的，因此加载和存储操作涉及字和字节列表之间的转换，一般来说是不可处理的。我们的抽象通过抽象掉转换中的字节操作细节来帮助使推理更容易。具体地说，我们引入了未解释的函数抽象和词级推理的引理，如下所示。
术语nthByteOf(v、i、n)表示i两个字节的n字节（0是最重要的位），当v不适合时丢弃高阶字节
n字节。准确地说，其定义如下：

当n>i+1时(v，i，n)=nt字节(v/256，i，n-1)
无字节段(v、i、n)=vmod256 当n=i+1时

然而，当参数是符号性的时，我们希望保持它不被解释（即不展开定义），以避免昂贵的非线性算术推理。
我们在未解释的函数项上引入引理。以下引理用于对MLOAD进行符号推理

[8] 也可以使用位向量理论来形式化字节操作，但使用数学整数理论的形式化具有函数规范简洁的优点。实际上，KEVM语义采用整数形式化是因为其优势。

[9] https://keccak.team/index.html

[10] 这一假设并非不合理，因为几乎所有的区块链都严重依赖于哈希函数的抗碰撞性。

和MSTORE指令。它们捕获了这两个指令所使用的基本机制：将一个单词分解成一个字节列表，并将其合并回这个单词。首先，根据定义，我们有nthByteOf(v，i，n)的界：0nthByteOf(v，i，n)<256。合并操作的引理：

merge(nthByteOf(v，0，n)・・・nthByteOf(v，n-1，n))=v

if0≤v<2^8^n≤32内存抽象

的其他引理参考[11]。

哈什的抽象。我们不仅仅将哈希函数建模为一个内射函数，因为由于鸽子洞原理，它是不正确的。相反，我们将它抽象为一个未解释的函数，哈希，它以一个字节列表作为输入，并返回一个（无符号的）整数：

散希：{0、・・・・，255}↣ N

请注意，这个抽象操作允许发生哈希冲突的可能性。

但是，通过假设每个执行轨迹中出现的所有散列值都是无碰撞的，我们可以避免对潜在碰撞的推理。这可以通过仅对符号执行中出现的术语实例化注入性属性来实现，其方式类似于通用量词实例化。

算术简化规则。我们引入了特定于EVM的简化规则，它捕获算术属性，它将一个给定的项简化为一个更小的项。这些规则有助于提高基本定理证明者的符号推理的性能。例如，我们有以下简化规则：

(x×y)/y=x if y C0

其中/是整数除法。我们还有一个关于掩蔽的规则操作，0xff・・・f&n，如下：

m&n=n if m+1=2^1+log m^和0≤n≤m

其中，&是位向和运算符，m表示位掩码0xff・・・f.有关其他简化规则，请参阅[11]。

4 案例研究：erc20验证

我们提供了一个完全验证ERC20令牌合约[13]的高调、实际部署的实现的案例研究，这是最流行的以太坊智能合约之一，提供了维护和交换令牌的基本功能。

4.1 正式规范

ERC20标准[13]非正式地指定了ERC20令牌合同必须满足的正确性属性。然而，不幸的是，它留下了几个未指定的角落案例，这使得在令牌合同的正式验证中使用它不太理想。

我们在K框架中指定了ERC20-K[9]，这是ERC20标准的高级业务逻辑的完整形式化。
ERC20-K明确了哪些数据（如余额和津贴）由各种ERC20功能和这些功能在这些数据上的精确意义来处理。ERC20-K还阐明了ERC20标准省略了要讨论的所有角落情况的意义，例如传输到它本身或导致算术溢出的传输，

[11] 请注意，在撰写本文时，Z3未能证明这个看似琐碎的公式。事实上，这个问题已经在开发分支中得到了解决，曾经由本文的作者报道过：https://github.com/Z3Prover/z3/issues/1683

```
[转移成功]
调用数据：#abiCall数据（“传输”，#地址(TO)，#uint256（值））状态代码：
_=>EVMC_SUCCESS
输出：_=>#字节字节宽度（1,32）
日志：....(。=>#abiEventLog（来自、“传输”、#索引（#地址(FROM)）、#索
    引（#地址(TO)）、#uint256（值））)
存储：
  #有位置（{余额}，来自）|->(BAL_FROM=>BAL_FROM-Int值)
  #有位置({平衡}，TO)|->(BAL_TO=>BAL_TO+Int值)
  ...
要求：
  从=/=到库值<=IntBAL_FROM
  和BoolBAL_TO+Int值<Int(2^Int256)

[转移失败]
调用数据：#abiCall数据（“传输”，#地址(TO)，#uint256（值））状态代码：
_=>EVMC_REVERT
输出：_=>_日
志：……存储：
  #有位置（{余额}，来自）|->BAL_FROM#有位置
   ({余额}，到）                    |->bal_to
  ...
要求：
  从=/=到和库(值>IntBAL_FROM
   orBool    BAL_TO+Int值>=Int(2^Int256))
```

图1：传递函数的正式说明

遵循旨在尽量减少天然气消耗的最自然的实现。完整的规格可在[9]上找到。例如，图1显示了部分（简化的）传输规范。它指定了两种可能的行为：成功和失败。对于每种情况，它都指定了函数参数(callData)，返回值（输出），是否为异常-固化（状态代码）、生成的日志（日志）、存储更新（存储）和路径条件（需要）。具体来说，成功案例（用[传输成功]表示）指定函数成功地将值令牌从FROM帐户转移到TO帐户，并生成相应的日志消息，如果没有溢出（即，FROM帐户有足够的余额，TO帐户有足够的空间返回1（即接收令牌）。故障情况（[传输失败]）指定，如果发生溢出，该函数将在不修改帐户余额的情况下抛出异常。

4.2 正式验证

在这个案例研究中，我们考虑了三种ERC20令牌实现：VyperERC20令牌[14]，黑客金(HKG)ERC20代牌[15]，以及Open齐柏林飞艇的ERC20令牌[12]。其中，VyperERC20令牌是用Vyper编写的，其他的都是用可靠性编写的。

我们使用每种语言编译器将源代码编译为EVM字节码，并执行我们的验证器来验证编译后的EVM字节码是否满足上述规范。

[12] 转移允许四种可能的行为：定期转移的成功和失败（即从C到），以及自我转移的成功和失败（即从=到）。这里我们忽略了由于空间限制而引起的自转移行为。完整的规格，请参见[9]。[13] https://github.com/ethereum/vyper/blob/master/examples/tokens/ERC20_solidity_compatible/ERC20.vy
[14] https://github.com/ether-camp/virtual-accelerator/blob/master/contracts/StandardToken.sol
[15] https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/StandardToken.sol

表1：ERC20代合同验证时间（秒）

	VyperHKGZeppln。			VyperHKGZeppln。		
总供应36.4N/A34.3批准				33.9	48.4	35.4
balanceOf 219.7	33.3	37.3	37.1 转移	148.5	198.5	
津贴	36.442.339.6transferFrom174.4257.6 179.2					

在这个验证过程中，我们发现这些合同之间的不同行为不符合ERC20标准。由于偏离了标准，我们无法根据最初的ERC20-K规范来验证这些合同。为了证明它们是“正确的”，w.r.t.对原始规范调制偏差，我们修改了规范以捕获偏差，并根据修改后的规范进行了成功验证。表1提供了验证器的性能。下面我们将描述这些结果。

VyperERC20令牌。VyperERC20令牌根据原始规范进行了成功的验证，这意味着其完全符合ERC20标准。

HackerGold(HKG)ERC20托肯。除了众所周知的香港政府令牌的安全漏洞外，我们发现HKG令牌的实现偏离了我们的规范，如下：

- *无总供应功能：香港政府令牌中不提供总供应功能，这不符合ERC20标准。*
- 在失败中返回false：它返回false，而不是在转移和转移的失败情况中抛出异常。它不违反标准，因为建议抛出例外，但不是根据ERC20强制标准。
- 拒绝0值的传输：它不允许传输0值，立即返回false，而不记录任何事件。它不符合标准。对于任何假设符合erc20行为的API客户端，这都是一个潜在的安全漏洞。
- 无溢出保护：它不检查算术溢出，导致在溢出的情况下，接收器的平衡包围了256位无符号整数的最大值。它没有违反该标准，因为该标准没有具体规定有关它的任何要求。然而，它可能是脆弱的，因为由于接收方的余额转到低于预期，如果资金溢出，它将导致资金损失。

OpenZeppelinERC20Token。OpenZeppelinERC20令牌是由安全审计部门开发的一个高调的ERC20令牌库

咨询公司齐柏林飞艇。我们发现OpenZeppelin令牌偏离了ERC20-K规范如下：

- 拒绝发送到地址0的传输：它不允许传输到地址0，从而立即抛出异常。它没有违反该标准，因为该标准没有具体规定有关它的任何要求。然而，这是可疑的，自虽然还有许多其他无效地址不应该进行转让，但尚不清楚拒绝单个无效地址有多有用，但以每次转让交易额外的天然气消耗为代价。

[16] https://www.ethnews.com/ethercamps-hkg-token-has-a-bug-and-needs-to-be-reissued请注意，令牌合同已由齐柏林飞艇手动审计，但它们失败了要找到漏洞，这意味着需要进行严格的正式验证。

[17] https://zeppelin.solutions/security-audits

5 相关工作

虽然有一些静态分析工具[5,6,8,12]来检查某些预定义属性，但由于空间限制，这里我们考虑，只考虑由成熟的定理证明器支持的验证工具，该证明器允许推理任意（完全函数正确性）属性。具体来说，Bhargavan等。[2]和格里琴科等人。[3]提出了一种基于F*验证助手的验证工具，Amani等人。[1]提出了一个基于伊莎贝尔/HOL的工具。然而，这些工具只采用了EVM的部分、不完整的语义，因此可能会错过EVM字节码的某些关键的角情况行为，这可能会破坏验证器的可靠性。另一方面，我们的EVM验证器，是一个验证工具，从一个完整的和彻底测试的正式语义的EVM[4]，这是首次据我们所知。

确认

本文提出的工作得到了NSFCCF-1421575、NSFCNS-1619275和IOHK赠与的部分支持。

参考文献

[1] 西德尼·阿马尼，米里亚姆·贝格尔，马克西姆·博丁和马克·史台普斯。 2018.关于在以莎贝尔/HOL中验证以太坊的智能合同字节码。在第七次ACM认证项目和证明国际会议的论文集上(CPP2018)。

[2] 卡提基扬·巴加万、安托万·德拉尼亚特-拉瓦德、塞德里克·福内特、阿尼莎·戈拉穆蒂、乔治·冈蒂尔、纳迪姆·科贝西、纳塔莉亚·库拉托瓦、似乎拉斯托吉、托马斯·西博特-皮诺特、尼希尔·斯瓦米和圣地亚哥·萨内拉-贝格林。2016.智能合同的正式验证：简称。*在2016年ACM编程语言与安全分析研讨会论文集上(PLAS2016年)*。

[3] 伊利亚格里琴科，马特奥马菲，和克拉拉施奈德温。2018.一种关于以太坊智能合同安全分析的语义框架。*第七届安全与信任原则国际会议论文集（2018年后）*。

[4] 埃弗雷特·希尔登布兰特、萨克森纳、朱小然、罗德里格斯、菲利普·德安、德怀特·古斯、布兰登·摩尔、张毅、大军园、安德烈·斯蒂凡内斯库和罗苏。 2018.KEVM：以太坊虚拟机的完整语义。*第31届IEEE计算机安全基金会研讨会论文集（脑脊液2018）*。

[5] 苏克里特卡拉，众议员戈尔，莫汉达万，和苏博德沙玛。2018.宙斯：智能合同的安全性分析。*在第25届年度网络和分布式系统安全研讨会论文集(NDSS2018)*。

[6] 卢、朱公爵、奥利克尔、萨克塞纳和阿奎那。2016.制作智能合同。*在2016年ACMSIGSAC计算机和通信安全会议论文集（中国化学会2016）*。

[7] 尤里五世。玛蒂亚塞维奇。1993.*希尔伯特的第十个问题。麻省理工学院出版社*。

[8] 伊维卡尼科利克，阿什什科卢里，伊利亚谢尔盖，普拉提克萨克塞纳，和阿奎那霍布尔。2018.找到贪婪的、浪的和自杀的合同。CoRRabs/1802.06038（2018）。

[9] 格里戈尔Rosu。 2017.ERC20-K：ERC20的正式可执行规范。https://github.com/runtimeverification/erc20-semantics.

[10] 安德烈、大车公园、石用玉又、李一龙、罗素。2016.基于语义的所有语言的程序验证符。*2016年ACM签署面向对象编程、系统、语言和应用国际会议论文集(OOPSLA2016)*。

[11] 正式的验证团队。2018.已验证的智能合同。https://github.com/runtimeverification/verified-smart-contracts/。

[12] 佩塔尔·特桑科夫、安德烈·玛丽安·丹、达纳·德拉克斯勒·科恩、亚瑟·热维斯、弗洛里安·布恩兹利和马丁·T。维乔夫。 2018.安全性：智能合同的实际安全分析。CoRRabs/1806.01143（2018）。

[13] 费边·沃格尔斯特勒和维塔利克·布特林。2015.第20条令牌标准。https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md.