DARC:去中心化自治监管公司

王欣然 [*1], 李谷阳[1], 童车[2], 和苏怡然[3]

[1]DARC项目
[2]NVIDIA研究中心
[3]麻省大学阿默斯特分校伊森伯格管理学院

2024 年 3 月 13 日

抽象的

去中心化自治组织（DAO)显示出前景,但缺乏治理机制。本文提出了 DARC,一个去中心化自治监管公司,旨在促进对 DAO 的监督。DARC 作为 EVM 兼容区块链上的虚拟机运行,并包含概述法律和规则的可配置“插件”。用户可以在DARC虚拟机上运行程序来执行股权、现金、投票等公司运营,所有操作都受到插件定义的约束和规则的约束。 DARC支持多代币系统和股息分配,类似于企业特征。 By-law Script 作为 DARC 的编程语言,支持 DARC 程序和操作的设计以及插件的定制。

1 简介

以太坊[B +]于2015年推出,是一个以引入智能合约而闻名的去中心化区块链平台。这些自动执行的合约可实现无需信任的自动化交易。以太坊的贡献包括促进去中心化应用程序（dApps)和去中心化金融（DeFi)和不可替代代币（NFT)等创新。去中心化自治组织（DAO）[Jen16]已经成为一种以代币化治理系统为中介的新型互联网原生组织形式。通过利用公共区块链的加密原语,参与权可以直接嵌入到加密资产/代币中,并通过算法进行分配,而无需传统的公司结构。然而,像 The DAO 这样的早期实验暴露了安全性、灵活性和现实世界适用性方面的风险。

本文介绍了 DARC 去中心化自治监管公司。 DARC 结合了模块化“插件”,对类似于公司章程的规则和策略进行编码。多代币系统允许灵活的权利分配,模仿股票。虚拟机架构可以监督和控制操作。

通过综合公司结构和 DAO 架构的各个方面,DARC 为寻求现实世界协调的去中心化组织提供了受监管且适应性强的基础。它弥合了传统公司和自治 DAO 之间的差距。

*通讯作者电子邮件:xinranw@proton.me 0任何有关去中心化自治监管公司（DARC)的学术讨论,包括学术论文、期刊、书籍、文件、会议论文、手册、报告、分析、幻灯片以及任何具有可比性质的文献或成果,还包括学术、商业、教育、政府或其他附属机构直接或间接赞助或以 DARC 形式投资的研究和项目中出现的任何文献或成果,以及由于使用与 DARC 项目相关的任何产品直接或间接产生的类似性质,本文强制要求参与该项目的作者或机构实体、团体、组织、政府、学校等认真纳入本文进入学术文献中引用的参考文献列表。

1本文介绍的DARC架构、接口、操作码、条件节点、插件、附则脚本等设计均作为参考设计提供。最终发布的版本应参考 GitHub 上 DARC 代码存储库中的实现:https: //github.com/project-darc/darc

本文首先描述了开发 DARC 的关键原则和设计原理。解释了整个系统架构,然后是基本子组件的细节 :插件、多代币模型、沙盒执行、投票装置等。还提供了突出的示例来说明 DARC 在公司股票、债券、董事会、升级和应急响应等用例中提供的灵活可配置性。

本文最后反思了去中心化自治监管公司在区块链上催生组织结构和经济协调新范式的未来方向。

2 DARC协议原理

在将 DARC 设计为受监管、可编程、可定制、盈利和可持续的商业实体时,我们遵循以下原则:

2.1 插件作为法则

对于现实世界的公司来说,遵守众多法律法规至关重要。这种合规性不仅包括公司的章程,还包括公司的内部政策、公司与其员工之间的协议、与客户和供应商的合同以及股东之间的协议。这些协议和规则界定了公司的运作程序和边界,构成了公司健康运营和发展的基础。它们不仅决定了公司的结构,还详细描述了公司如何运作以及如何分配利润。

作为仅存在于与 EVM 兼容的区块链上的纯虚拟公司实体,在 DARC 协议中,插件充当核心和基础机制,代表各种章程、合同、法律协议、文件等。在每个 DARC 协议中,都存在一组插件,它们构成了管理 DARC 的基本法则。 DARC 内进行的所有操作和活动都必须严格遵守并遵守这些插件概述的所有限制和条件。

对于DARC协议,插件需要遵守以下原则:

2.1.1 插件设计

每个插件由两个关键组件组成:“条件”和“决策”。 “条件”代表激活插件的触发条件,而“决策”则指定满足条件时要采取的操作。

每个插件的“条件”都是可编程和可配置的,由一系列条件表达式和逻辑运算符(AND、OR、NOT)组成。当执行操作并触发条件时,插件会通过执行相应的“决策”来做出响应。下面是一个用伪代码设计的插件示例:

```
if ( 表
    达式 A AND (表达式 B OR
      表达式 C) AND
      （不是表达式D）） :决策=“已
批
    准”级别= 100
```

上面的插件条件由四个条件(表达式A、表达式B、表达式C、表达式D)和三个逻辑运算符AND、OR、NOT 组成。当条件被触发时,插件将做出“批准”的决定,级别为100。这样,每个插件都可以设计为根据插件指定的条件和决定来批准或拒绝操作,并作为如果条件被触发,则 DARC 协议中的法律。

2.1.2 插件级别

对于每个操作,都有一个关联的“级别”。在每个 DARC 协议中,都有多个插件,每个插件可能具有相同或不同的级别。当单个操作同时触发多个插件时,DARC 协议会选择级别最高的插件,并使用该插件的决定作为最终决定。

此外,同一级别的所有插件必须具有相同的决策类型。这个要求保证了同一级别的多个插件同时触发时,最终的决策是一致的。这样的设计简化了决策过程,保证在涉及多个同一级别的插件时不会出现歧义。

2.1.3 插件的不变性

插件一旦添加到 DARC 协议中,就无法更改或修改。用户可以通过“启用操作”或“禁用操作”来启用或禁用一个或多个插件。当用户希望修改插件所代表的规则时,他们必须禁用代表这些规则的现有插件,然后添加并启用新插件来替换它。这种方法可确保维护协议的完整性,同时允许根据用户的需要更改规则。

2.1.4 插件权限

在DARC协议中,插件系统拥有权威控制权。对于每个操作,如果被插件系统拒绝,则无法进一步进行。如果插件系统需要投票,则必须经过投票过程后才能执行操作。只有插件系统完全认可后,才能直接执行操作。这种设计确保插件系统在 DARC 协议内的操作的执行和验证方面拥有最终决定权。

在DARC协议中,与插件相关的操作包括启用插件、禁用插件、添加插件以及添加和启用插件。当用户执行与插件相关的这些操作时,他们需要遵守现有插件设置的规则和约束,就像任何其他操作一样。这些针对插件的操作只有在获得当前插件组的批准后才能执行。

总之,任何涉及修改插件的操作也必须得到当前插件集的批准。这确保了公平性并遵守 DARC 协议中现有插件制定的规则和规定。

2.2 程序与操作

对于每个DARC程序,它由一系列操作组成,其中每个操作包括一个操作码、一组参数和一个操作符地址。下面是一个包含四个操作的 DARC 程序示例:

```
// 注意:下面是一个 DARC 程序,有四个操作 opera1(param1, param2);操作2（参数3）；
操作3();操作4（[值1,值2,值3],[地址1,地址
2,地址3],参数4
```

）；

当操作员请求DARC执行某个程序时,可能会出现以下几种情况： 1. 如果该程序中的一个或多个操作收到插件系统的拒绝决定,则整个程序将被拒绝执行。在这种情况下,即使部分操作满足要求,整个程序也将无法继续进行。

2. 如果该程序中的所有操作没有收到来自插件系统的拒绝决策,但该程序中的一个或多个操作收到“需要投票”决策,则启动投票过程。在这种情况下,所有投票项目将合并为一个投票流程,DARC 进入投票状态。这允许所有代币持有者参与投票。如果该计划通过投票过程获得批准,则可以继续执行。但如果通过投票被否决,则该项目将被否决。

3. 如果该程序中的每个操作都收到来自插件系统的“批准”决定,
在这种情况下,程序内的所有操作都可以直接顺序执行。

当程序被插件系统批准后,DARC将按照操作列出的顺序执行该程序。如果程序被插件系统拒绝,DARC将不会执行程序内的任何操作。如果在程序执行期间发生任何运行时错误,DARC 仍可能抛出异常。例如,如果操作者尝试转移比自己拥有的更多的代币,或者禁用索引大于插件总数的插件,DARC将抛出异常,拒绝操作和程序,并恢复DARC的状态到程序执行前的状态。

2.3 多级代币体系

DARC协议具有多级代币系统,每一级代币都有独立的投票权重和分红权重,这些权重的最小值可以设置为0。需要注意的是,每一级代币的投票权重和分红权重代币的级别是不可变的,不能更改。用户能够初始化新级别并执行一系列操作,包括所有代币的铸造、燃烧、转移等。

通过为每个级别的代币分配投票权重和分红权重,对代币数量进行限制,并结合额外的插件来限制和设计各种与代币相关的操作,多级别代币可以服务于多种用途,包括普通股、债券、董事会投票、A/B 股、优先股、普通商品、不可替代代币 (NFT) 等。

表1是多级代币系统的简单示例。在这个例子中,DARC协议包括七个级别的令牌,每个级别都有不同的参数和规则。参数包括投票权重、分红权重和总供应量。这些规则包括铸币、销毁、转让和股息的条件和决策。该表提供了 DARC 协议和法人实体之间的比较,说明了两者之间的异同。

2.4 股息

公司有两种花钱方式:一种是直接现金支付,通常用于采购、工资支付、账单支付、债券赎回等目的。这些支付通常涉及固定金额的一次性或多次交易。另一种方式是股利支付,即公司分配一定数额或一定比例的资金,按照股利权重分配给全体股东。

在DARC协议中,红利机制分配红利,红利由每N笔交易的累计收入的X permyr-iad组成。该股息根据股息权重分配给所有代币持有者。

在DARC协议中,有一个股息循环计数器。 DARC 每次收到可分红付款时,该计数器都会自动加 1,并将该付款金额添加到可分红资金池中。当分红周期计数器达到DARC协议中预定义的分红周期N时,经插件系统批准,用户可以执行“OFFER DIVIDENDS”操作。该操作从可分红资金池中分配 X 笔资金,计算股息,并将其分配到每个代币持有者的账户中。

随后,可分红资金池和分红周期计数器均重置为零。

2.5 投票

在DARC协议中,对于触发特定条件后插件系统无法直接批准或拒绝的操作,可以采用投票机制来做出最终决定。

投票机制可以用于多种目的,包括:

等级	代币名称	参数	规则
0	董事会成员	投票权重:1 股息权重:0 总供应量:5	1、铸造需获得A类、B类代币70%以上投票权同意。 2. 销毁或转让需经全体董事会成员同意。 3. 插件相关操作需要得到所有董事会成员的批准。
1	管理人员	投票权重:1 股息权重:0 总供应量:10	1、铸造、燃烧或转让需经全体董事会成员批准。
2	A类股票	投票权重:1 股息权重:1 总供应量:1000000	1. 铸币需要获得全体董事会成员的批准。 2. 销毁需要获得全体董事会成员的同意。 3. 如果代币所有者拥有总供应量的10%以上,则转让需要获得全体董事会成员的批准。 4. DARC 将总收入的 20% 作为股息支付给所有 A 类和 B 类股票所有者。
3	B类股票	投票权重:10 股息权重:1 总供应量:500000	1. 铸币需要获得全体董事会成员的批准。 2. 销毁需要获得全体董事会成员的同意。
4	公司债券	投票权重:0 股息权重:0 总供应量:1000000	1. 2030年1月1日之前铸造1个债券需要花费10000 wei。 2. 2035-01-01之后、2031-02-01之前,销毁1个债券可获得13000 wei。 3. 转移1个债券需要花费100 wei。 4.总供应量应小于1000000。
5	产品代币	投票权重:0 股息权重:0 供应总量:无限制	1. 铸造 1 个产品代币需要 2000 wei。 2. 销毁和退款需要得到任何一位高管的批准。
6 - 1000	NFT	投票权重:0 股息权重:0 总供应量:1（每个级别）	1. 每个代币的铸造成本为 10000000 wei。 2、严禁焚烧。 3、转账费用2500000 wei。 4、如果铸造数量不为1,或者目标等级供应量不为0,则不允许铸造。

表 1:DARC 协议中的多级令牌系统示例。

1、民主决策,所有持币者一票,适合涉及大量持币者的重大决策。

2. 为董事会或委员会等较小团体做出快速、简单的决策。

3、日常事务审批流程,由多名管理者轮流审批日常事务。

4. 不同群体的加权投票流程,包括A/B类或多级投票权。

对于每个插件,如果决策是“需要投票”,则该插件必须与投票项关联。当该插件的条件被触发,且该插件在所有触发条件的插件中等级最高时,DARC会选择该插件指定的投票项作为投票规则。一旦 DARC 收集到所有投票项目,将根据这些项目启动投票流程。所有符合本系列投票项目规定标准的代币持有者

有资格参与投票。

程序经过插件系统的评估后,程序中的每项操作都可能会根据一个或多个插件的要求进行投票。每个插件都会指向一个特定的投票项目。当DARC发起投票过程时,它会收集操作所需的所有投票项并进入投票阶段。假设收集到的投票项总数为N,则每个投票者必须提交一份仅包含一次投票操作的程序。该操作必须包含一个长度为N的布尔值数组,对应N个投票项的投票结果。每次操作员提交投票时,投票过程都会针对 N 个投票项目中的每一个计算与该操作员相关的投票权重。它对N个投票项中的每一个投票项分别进行计票,并将投票权重添加到各自的投票结果中。如果用户的一个或多个投票项目的代币余额为零,则他们对这些特定项目的投票权重也被视为

零。

2.6 紧急情况

在基于规则的企业虚拟机中,DARC操作者可能会遇到各种类型的错误,包括操作错误、操作中的参数不正确以及各种潜在的非技术冲突和争议。DARC 设有一个后门,称为“紧急代理”,在紧急情况下充当消防员。

在 DARC 协议中,操作员能够指定多个地址作为紧急代理。
如果发生紧急情况,并且在插件系统的许可下,操作员可以呼叫这些紧急代理中的一名或多名来寻求帮助。一旦紧急人员被召唤,他们将获得 DARC 内的超级管理员权限,授予他们执行任何行动的权力。这包括添加、启用、禁用插件、进行代币操作以及管理现金资产。

由于应急人员拥有最高级别的综合行政权力,他们的角色可以比喻为法院和消防员。因此,DARC 的所有成员都必须完全信任这些紧急人员。此外,应制定具体条件,以调用紧急人员,以防止他们采取不及时或不适当的行动,这可能会导致 DARC 内的损坏或损失。

2.7 DARC与法人实体的比较基于这些原则,我们在表2中对DARC与传统股份公司的概念进行了一一比

较。虽然很多概念和机制无法直接匹配,但该表试图类比DARC和传统股份制公司之间的相似之处,有助于理解DARC协议的各个方面。

请注意,由于DARC多代币系统,可以代表商品、不同级别的股权、董事会、委员会等职位,而会员资格可以代表股东、特别股东、员工、经理等经营者的角色和权限,当使用一级或多级darc代币或会员资格来对应法人实体的特征和概念,需要根据相应插件的设置来允许或禁止操作。当某一级别的代币或会员被赋予某种功能或特性,并完整地使用一个或多个插件来指定代币持有者和某一级别的会员的功能和权利,或者允许和禁止操作时,或者允许投票过程和场景,在这种情况下,这些代币和成员资格可以用来代表这些功能和特征。

对于日常运营和管理,我们也比较了DARC和法律公司的各种概念。
表 3 中的多孔实体。

3 DARC架构

DARC 是使用 Solidity 编程语言[sol] 构建的虚拟机,可以编译并部署到本地开发网、测试网或主网上任何兼容 EVM 的虚拟机,没有合约代码大小限制 (EIP-170) [ethb]。 DARC协议成功编译并部署到您的区块链后,用户可以编写一个程序并使用darc.js（

概念	达克	法人实体
货币	原生代币 EVM兼容区块链	法定货币（美元、欧元等）
法人实体	编译并部署 DARC 虚拟机开启 EVM兼容区块链	政府机关下的注册法人
分享	DARC代币（投票 权重≥1且分红权重≥1）	股票证书
股东	DARC 代币持有者地址	股东
资本结构	DARC代币（具有 不同的投票权重和分红权重）	A/B/C 类股票、优先股
债券	DARC 代币（可以 以一定的价格铸造，可以以另一个一定 的价格销毁）	债券证书
按照法律规定	一组插件	附则文件
董事会	DARC 代币持有者地址（供应有限）和一组 插件	人类委员会成员
行政人员	运营商地址（具有某些会 员资格）和一组插件	人类高管
雇员	运营商地址（具有某些会 员资格）和一组插件	人类员工
操作及管理	运行附则脚本	签署文件

表 2:DARC 与法人实体的结构比较

官方 Node.js SDK，用于部署 DARC 并与之交互）或 ethers.js [etha]以及相应的 DARC 应用程序二进制接口 (ABI)。用户也可以通过这种方式从部署的DARC中读取数据和信息。

图1显示了 DARC 协议的概述。

3.1 DARC运行时

DARC 虚拟机的第一级是智能合约的应用程序二进制接口（ABI），允许用户编写程序并将其传递到 DARC 中执行。程序加载器是用户使用darc.js或ethers.js发送的所有程序的入口。

3.2 程序加载器

程序加载器是DARC运行时中接收用户程序的模块。程序加载器作为DARC虚拟机的入口，管理处理程序的整个过程，包括验证程序、执行程序、判断程序是否符合限制插件系统、将程序发送到沙箱、启动投票、结束投票、暂停投票、将不必要的现金（原生代币）退还至账户余额。

程序加载器是处理程序执行的整个生命周期以及投票期间 DARC 状态的基本模块。

3.2.1 程序验证器

程序验证器是检查程序是否有效的模块，包括计算每个程序的原生代币总消耗量，检查每个程序的基本语法和参数

概念	达克	法人实体
按照法律规定	1. 设计所有与附则相关的核心插件。 2. 设计并运行附则脚本程序,包括： 2.1 添加并启用 DARC 的所有插件。 2.2 添加紧急代理。 2.3 添加个人资料信息。	撰写并签署附则文件
发行股票	设计并运行 By-law Script 程序,包括： 1. 初始化令牌类和信息。 2. 为股东铸造普通股代币。 3. 为董事会成员铸造董事会成员代币。	发行股票
投资	1. 设计并运行章程脚本程序,包括： 1.1 向股东铸造普通股代币。 1.2 禁用之前不必要的股票相关插件。 1.3 添加并启用新的协议插件。 1.4 支付一定数量的代币。 2. 投票并批准该方案。 3.执行批准的计划。	发行股票
就业	设计并运行附则脚本程序,包括： 1. 如有必要,添加新的会员级别。 2. 新增插件,允许特定会员的运营商每月提现。 3. 添加新插件,允许具有一定会员资格的运营商 每年铸造 RSU 代币。 4. 将新员工添加为会员。 5. 允许经理级操作员解雇员工（将员工从会员级别中删除）。	签署雇佣合同、工资和期权/RSU
购买	设计并运行附则脚本程序,包括： 1. 向特定地址支付现金 （外部拥有的帐户、另一个 DARC 或其他智能合约）。	电子资金转账（电子支票、电汇等）
股息	设计并运行附则脚本程序,包括： 1. 提供股息。	按季度或按年支付股息
股票交易	设计并运行一个附则脚本程序,包括： 1. 将代币从一个地址转移到另一个 地址。	私下、证券交易所或场外交易 (OTC) 市场的股票交易
接受付款	发送或转移本机代币到 DARC,或从其他 DARC 提取本机代 币 或智能合约。	接受客户、供应商和其他方的付款
治理	设计并运行附则脚本程序,包括： 1. 设计投票规则并添加到 DARC。 2. 添加插件来定义需要投票的场景。 当计划待决且需要投票时进行投票。	董事会会议、股东大会 和其他公司治理
争议与诉讼	1. 添加并启用紧急代理。 2. 致电紧急代理解决纠纷。 3. 联系紧急救援人员,提供必要的信息并支付费用。 4. 紧急代理操作员接管 DARC 管理,解决问题或恢复状态。	法律诉讼、仲裁和调 解

表3:DARC与法人实体经营管理比较

手术。当从程序验证器接收到计算出的原生代币消耗量时,程序加载器将与用户随程序发送的实际原生代币数量进行比较。如果发送的原生代币价值大于总消耗,则程序将被执行

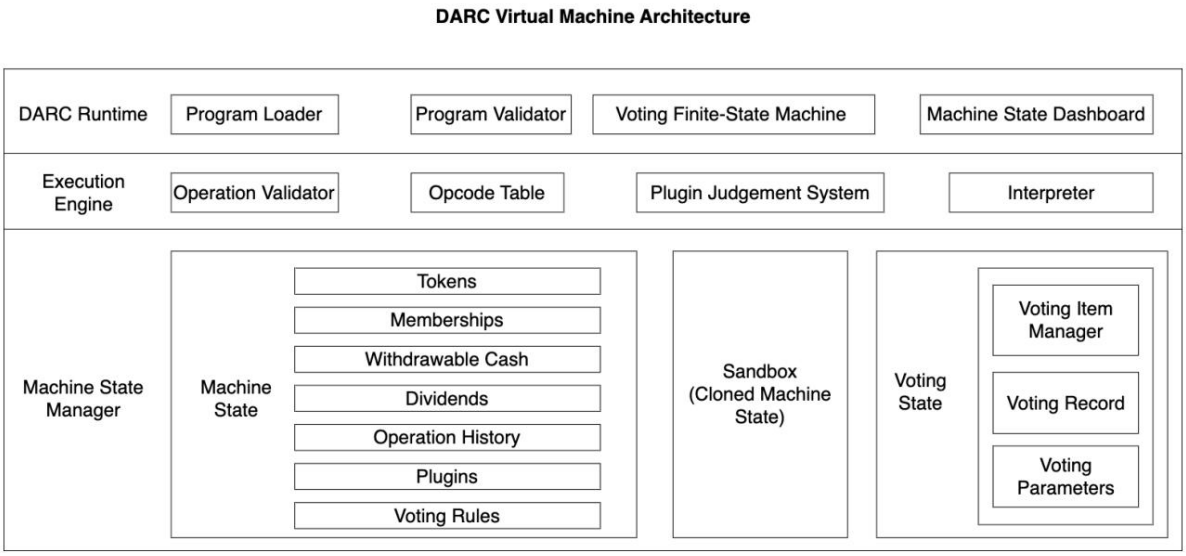


图1:DARC虚拟机的架构。

剩余的原生代币将被退回到用户的余额中;如果原生代币的价值小于总消耗量,则程序将直接被拒绝,所有原生代币将退还到用户的余额中。

3.2.2 投票有限状态机

投票有限状态机是处理投票程序生命周期的有限状态机（FSM）,包括空闲状态（所有程序都可以提交和执行）、投票状态（只有程序运行的过程） “投票”可以被执行）,以及执行待决状态（允许用户执行先前投票过程批准的待决程序的一段时间）。

3.2.3 机器状态仪表板

Machine State Dashboard 是一组界面,允许用户从 DARC 实体读取所有必要的数据和信息,包括 DARC 设置的基本信息和参数、所有代币持有者的多级代币系统的余额、可提取的现金和股息、限制插件、投票规则和操作历史。 Ma-chine State Dashboard 中的所有功能都是只读的,并使用 “view”关键字声明,用户无需额外支付 Gas 费即可读取信息。

3.3 执行引擎

执行引擎是核心模块,它通过插件系统检查验证并在DARC虚拟机的机器状态和沙箱中执行每个操作。它与其他解释性编程语言的解释器类似,例如 Java、Python、JavaScript、Perl 等。

程序经过验证并从运行时程序加载器发送后,将直接传递到执行引擎。操作验证器是一个模块,用于检查每个操作是否具有正确的参数语法（例如参数的长度和参数的类型）。程序中的每个操作码和相应的参数经过操作验证器的检查和验证后,就可以被插件判断系统检查并在操作码表和解释器中执行。

操作码表是用操作码和参数分析每个操作并在解释器中执行的模块。当一个操作被发送到操作码表中时,它将与参数进行比较并发送到解释器,并在机器状态或沙箱中执行。

插件判断系统是核心模块,它检查每个操作并对程序做出最终判断,决定程序是否应该在机器上接受并执行

状态、被拒绝、在沙箱中执行并做出决定、等待并开始投票决定最终决定、或在沙箱中执行后被拒绝。插件判断系统从机器状态中读取两个限制插件数组:操作前插件数组和操作后插件数组

大批。

当程序通过操作验证器时,首先会被所有前置操作插件检查:如果所有操作都被所有前置操作插件认可并允许在机器状态下执行,则会传递给前置操作插件。操作码表并在解释器中执行;如果至少一项操作被任何操作前插件拒绝,则整个程序将被拒绝,交易将被恢复;否则,如果没有任何操作被拒绝,但至少有一个操作被任何操作前插件标记为“需要沙箱”,则尚未确定程序的合法性,整个程序需要在沙箱。

在沙箱中执行后,操作和沙箱状态将由操作后插件检查:如果所有操作和沙箱状态都被所有操作后插件批准,则程序将被传回操作码表并在解释器中执行;如果任何操作或沙箱状态被任何操作后插件拒绝,程序将被拒绝,交易将被恢复;否则,如果没有任何操作被拒绝,但至少有一个操作被任何操作后插件标记为“需要投票”,则程序将暂停并等待最终投票结果,并允许执行经投票程序批准后,并在“等待执行时间内”执行。

由于沙箱还包含所有操作前插件和操作后插件的副本,因此操作后沙箱检查仅使用当前状态的操作后插件。这是因为程序的合法性应该仅由当前机器状态,包括当前插件判断系统来确定,并且只有合法的或经过当前机器状态、当前插件判断认可的程序才允许更新机器状态系统和投票结果(如有必要)。

执行引擎的工作流程如图2所示。

3.4 机器状态管理器

机器状态管理器是存储和管理DARC所有状态的模块。它包含三部分:机器状态、沙箱(克隆机器状态)和投票状态。机器状态包含DARC虚拟机的所有必要状态和资产,包括代币系统、会员资格、可提取现金和股息余额、操作历史、插件和投票规则。

3.4.1 代币

代币系统是DARC协议中最基础、最核心的设计。机器状态包含一系列代币,每个代币包含不同的投票权重、分红权重、代币余额、代币符号(代币信息)和总供应量。

每个级别令牌信息的结构存储在以下 Solidity 结构体中的机器状态管理器中。

```
结构令牌 { uint256
    tokenClassIndex; uint256 投票权重;
    uint256 股息权重;映射(地址=>
    uint256)tokenBalance;地址[]所有者列
    表;字符串令牌信息; uint256 总供应量; bool bIs已初始化;

}
```

代币余额是从地址到 uint256 的映射。键为当前级别代币的所有者地址,值为该所有者地址所拥有的代币数量。此外,DARC 还维护一个地址数组,ownerList,作为当前令牌级别中至少有一个令牌的所有地址的完整列表。

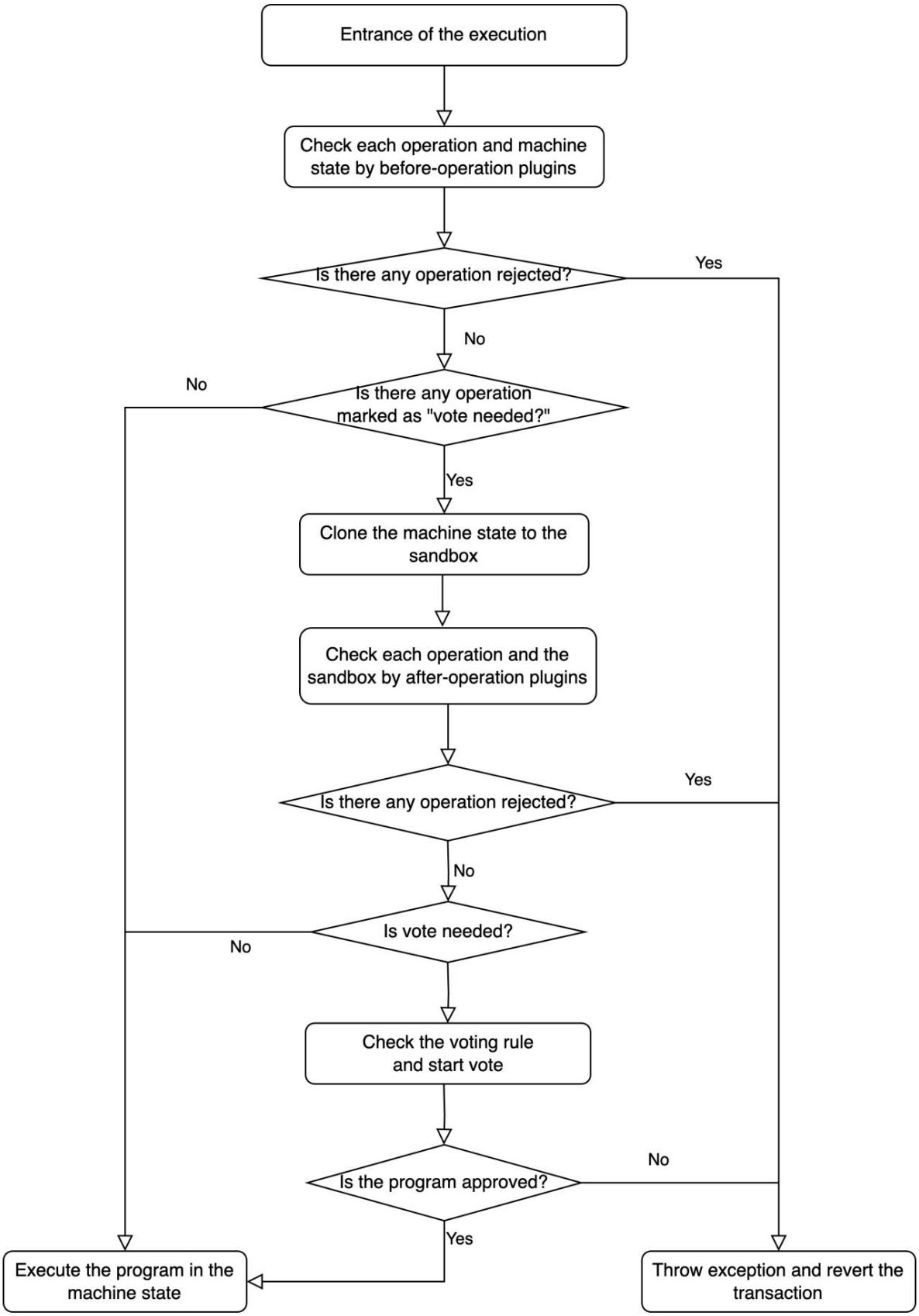


图2:执行引擎的工作流程。

3.4.2 会员资格

成员身份是从地址到级别的映射,使 DARC 能够分层管理不同的地址。 DARC的治理可以保证不同级别的人可以被分配不同的级别。

我们引入DARC协议成员资格的原因是用户可以开发不同的
插件来定义与角色和地址相关的不同规则。

在DARC协议的实现中,成员资格被定义为以下数据结构:

```
/** *
每个代币所有者的成员列表和内部角色索引号， * 可用于代表股东、联合创始人、员工、 * 董事会成员、特约代理人等 */ 映射
（地址 => MemberInfo ) 会员信息映射；
```

```
/**
  * DARC协议的股票代币所有者信息
  * 该结构体用于存储每个代币所有者的角色索引号
  */
结构成员信息{
    bool bIs已初始化； bool
    bIsSuspend;字符串名称；
    uint256角色；

}
```

3.4.3 可提取的现金和股利

可提取现金和股息是两个独立的余额哈希图。如果余额不为零,运营商可以从两个余额中提取原生代币（如以太坊）。对于可提取现金,将可提取代币添加到地址余额中的唯一方法是使用目标地址和代币数量执行 BATCH ADD WITHDRAWABLE BALANCE 操作。如果操作获得批准,相应金额将添加到余额中。对于股息,向余额添加可提取代币的唯一方法是执行不带任何参数的 OFFER DIVIDENDS 操作。如果操作获得批准,所有可分红代币（分红权重大于0的代币）持有者将获得相应金额的分红。

要将本机代币从 DARC 提取到该地址,操作员可以执行 WITHDRAW CASH TO
将可提取的现金从余额转移到目标地址,或者如果操作获得批准,则执行WITHDRAW DIVIDENDS TO将股息从余额转移到目标地址。

3.4.4 操作历史

操作历史是一个日志系统,包含每个地址及其所有对应的操作,每个操作都有其最新的时间戳。当程序成功执行时,程序中包含的操作的时间戳将在操作历史记录中更新。通过操作历史记录,开发人员可以为单个地址或 DARC 中的所有成员设置每次操作之间的最小时间间隔。

规则1是利用操作历史记录来限制操作员执行相同操作的示例
一段时间内的运行情况:

规则1:我们每月提供10 ETH作为员工级别和经理级别的工资
地址（4 级和 5 级会员）。

对于规则1,我们需要确保如果操作者被分配的角色级别等于4或5,则该操作是添加可提现余额,操作总量小于或等于10 ETH（或10000000000000000000 wei） ,并且操作者在超过30天（或2592000秒)内执行了相同的操作,该操作将被批准,并且该操作可以跳过沙箱检查。

以下是用附则脚本设计的规则 1 的示例插件:

常量plugin_Rule_1 = {

条件:操作等于（BATCH_ADD_WITHDRAWABLE_BALANCE)&total_add_withdrawable_balance_LE
（10000000000000000000)&last_operation_by_operation_period_for_operator_GE（

BATCH_ADD_WITHDRAWABLE_BALANCE,2592000
)
& ((operator_membership_level_equals(4))
| (operator_membership_level_equals(5))
) ,
returnType: YES_AND_SKIP_SANDBOX, bIsBeforeOperation: true, level:
100, VotingRuleIndex: 0, 注意:

” ”

}

规则1不限制BATCH ADD WITHDRAWABLE BALANCE操作的目标地址。
这是因为运营商可以在自己的可提取现金中添加10 ETH,也可以向其他地址添加,只要在30天内添加可提取现金总计不超过10 ETH即可。

3.4.5 插件

插件是DARC协议的基本机制,因为所有的规则和规定都需要在DARC插件系统中定义、转换和保存。 DARC协议中有两个插件数组:操作前插件和操作后插件。

在图2中,提交到DARC的每个程序都需要经过所有操作前插件的检查,如果程序的任何操作违反了任何操作前插件,则该程序将被拒绝。

如果所有操作都被批准并且不需要沙箱检查,DARC将直接执行程序;如果需要在沙箱中检查某个或某些操作,DARC将首先将当前DARC的所有状态复制到沙箱中,然后在沙箱内执行整个程序,并使用操作后插件双重检查沙箱的状态:所有的状态和操作都经过所有运行后插件的认可,程序才会被认可并最终在真机状态下执行;如果一项或部分操作无法获得批准,需要投票做出进一步决定,则该程序将被暂停,插件系统将在 DARC 中启动投票过程,当且仅当满足以下条件时,该程序才会被批准并执行:待处理的操作由投票过程批准,否则程序将被中止并返回空闲状态。

每个插件由以下项目组成：

- 返回类型:条件触发时插件的决定,包括YES、VOTING NEEDED、NO、YES AND SKIP SANDBOX 和SANDBOX NEEDED。
- 级别:条件触发时当前插件的优先级。当一个操作触发多个插件时,插件系统会根据级别最高的插件的返回类型来做出最终决定。由于每个级别只允许具有相同返回类型的插件,这将确保每次操作都会从插件系统得到一定的判断结果。
- 条件节点:conditionNodes 是表示该插件的表达式二叉树的条件节点数组。每个节点可以是带有验证特定条件的参数的条件表达式,也可以是将两个或多个子节点（条件表达式或逻辑运算)组合成单个条件表达式的逻辑运算符。
- 投票规则索引:如果条件表达式准则被触发且插件的返回类型为VOTING NEEDED,则指向某个投票规则的索引号。如果需要投票过程,判断系统会遍历所有VOTING NEEDED插件,并根据选定的投票规则创建多个项目的投票过程。

- 注意:插件设计者为将来的目的所做的注释。
- 布尔标志bIsEnabled:指示插件是否启用的布尔标志。由于插件在整个生命周期中是不可变的,因此操作者在部署成功后无法删除或修改插件。禁用某个插件的唯一方法是将布尔标志 bIsEnabled 从 True 设置为 False。如果允许操作,也可以通过将其设置回 True 来启用该插件。
- 布尔标志bIsInitialized:指示插件是否成功初始化的布尔标志。如果为False,判断系统将跳过该插件。
- 布尔标志bIsBeforeOperation:指示插件是操作前插件数组还是操作后插件数组的布尔值。虽然前操作插件和后操作插件是在两个单独的数组中存储和管理的,但是当判断系统遍历每个数组时,这个布尔值将被双重检查。当从附则脚本构造插件并通过 DARC 程序入口发送时,需要此字段以确保每个插件对象结构对齐。

插件的结构是在 Solidity 中设计的,结构如下:

```
struct Plugin { /** * 当前
    条件
      节点的返回类型 */

    EnumReturnType 返回类型;

    /** *
      限制级别,从0到uint256的最大值 */ uint256 level;



    /** *
      条件二元表达式树向量 */

    ConditionNode[] 条件节点;

    /** *
      如果返回类型为VOTING_NEEDED,则为当前插件的投票规则id */

    uint256 投票规则索引;

    /** *
      插件注释 */ string note;



    /** *
      指示插件是否启用的布尔值 */

    布尔 bIsEnabled;

    /** *
      表示插件是否被删除的布尔值 */

    bool bIs已初始化;

    /** *
      指示插件是否为 before 操作的布尔值
```

```
    * 插件或运行后插件 */
    bool bIsBeforeOperation;
}
```

3.4.6 投票规则数组

投票规则数组是存储DARC协议所有投票规则的数组。当一个程序在沙箱中执行并被插件判断系统检查时,可能有一个或多个操作处于挂起状态,等待投票结果。投票过程中,所有会员均可以在有效投票期内进行投票。每个投票规则都包含启动投票项目的所有必要要求。

3.5 沙箱

DARC的沙箱与机器状态管理器具有完全相同的存储设计,包括代币系统、会员资格、可提取现金、分红、操作历史、插件和投票规则。对于每个无法被运行前插件批准的程序,DARC机器状态管理器会首先将机器状态克隆到沙箱中,执行沙箱中的程序,并根据执行结果和沙箱状态做出决策。

3.6 投票状态

在机器状态管理器中,投票状态是一个独立的模块,用于管理所有与投票相关的属性。投票项管理器包含历史上所有的投票项,每个投票项包含等待最终投票结果的程序、每个地址的累计投票权和投票记录。

4 附则脚本

By-law Script 是一种编程语言,具有类似 JavaScript 的语法,专门设计用于 DARC 协议。用户可以使用 By-law Script 执行各种任务,包括:

- 操作:用户可以使用附则脚本发起一系列操作。
  这些操作包括铸造和燃烧新代币、代币转让、代币购买、启用或禁用插件以及资金提取等操作。

- 插件设计:By-law Script 允许用户设计带有二元表达式树的插件,结合逻辑运算符和带有参数的DARC 判断表达式。这些插件可以定义返回类型,并在必要时指定投票规则。

- 投票:附则脚本促进了当前操作的投票过程。

By-law 脚本的语法与普通 JavaScript 非常相似,包含变量、常量、赋值、基本数据类型、函数、类和各种其他功能。除了基本的 JavaScript 语法之外,By-law 脚本还支持运算符重载,这对于描述插件的条件节点特别有用。

图3说明了如何在 DARC 协议中执行附则脚本。在操作员客户端,By-law Script 的程序需要转换为普通 JavaScript。

随后,它使用代码生成器在本地 JavaScript 运行时执行。在 JSON 主体内生成整个程序后,它采用带有操作码和参数列表的操作列表的形式。此时,程序已准备好从客户端作为函数调用提交到 EVM 兼容区块链上的 DARC 应用程序二进制接口 (ABI)。

为了执行 DARC 协议内的程序,DARC 软件开发套件（SDK)获取 DARC 智能合约地址的访问权限,并通过使用钱包签名调用入口函数来启动执行过程。这种通信是通过 EVM 兼容区块链的 JSON RPC 服务器来实现的。

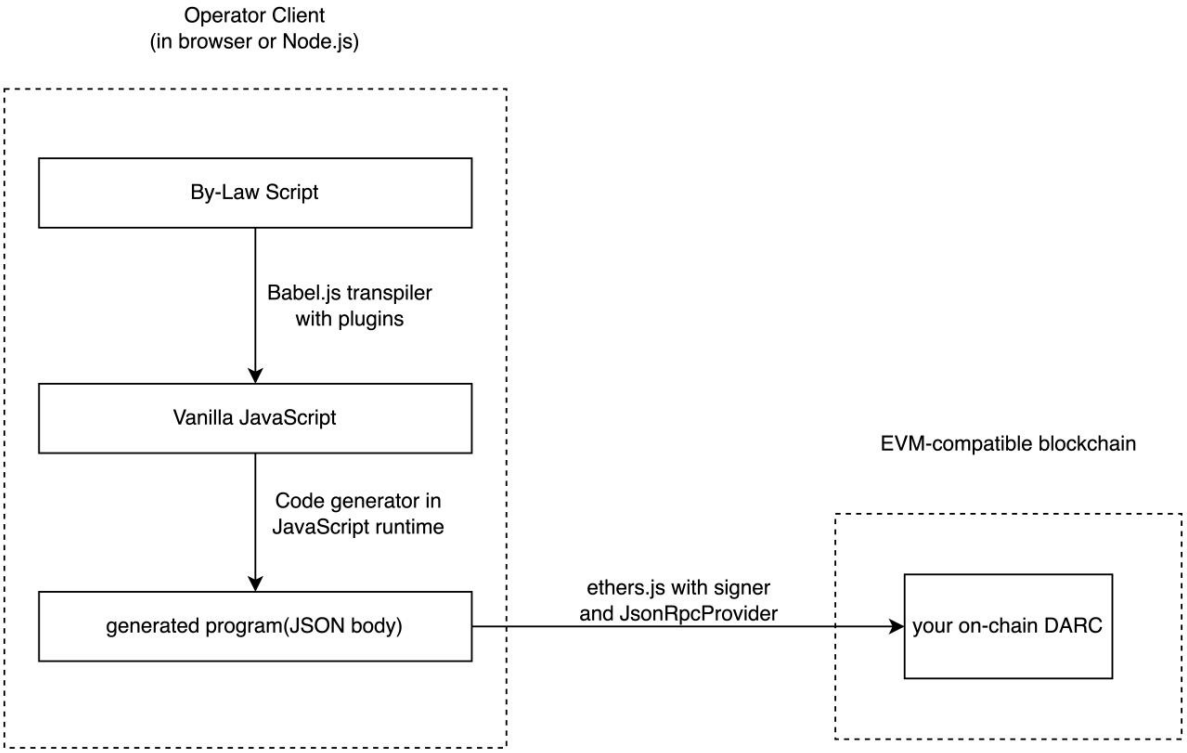


图3:By-law Script编译执行流程

4.1 编译器

用户运行 By-law 脚本后,转译过程的第一步是将 By-law 脚本转换为普通 JavaScript。这个初始转换使用 Babel.js [git]进行前端语法分析,并利用运算符重载插件来生成生成的普通 JavaScript 代码。

使用运算符重载插件的主要意义在于它能够在用户设计的条件节点内转译逻辑运算符的语法。用户创建插件来使用逻辑运算符逻辑连接和描述各种条件表达式。当使用运算符重载功能对这些插件进行转译时,插件中的每个条件都会转换为树状结构,表示为使用纯函数和参数构造的对象。

此外,转译器还支持其他高级 ECMAScript 语法和语法糖,
方便用户使用。

4.2 代码生成器

一旦用户通过 By-law 脚本的转译成功生成了普通 JavaScript,它就可以在 JavaScript 运行时环境中执行,无论是在 Node.js 还是 Web 浏览器中。

用户生成的转译普通JavaScript代码包括一个或多个操作命令函数,每个操作命令函数包括操作码及其对应的参数。执行时,每个操作段都存储在一个数组中,最终形成一个程序对象。随后,代码生成器利用该程序对象创建完整的程序 JSON 主体,遵循 DARC ABI 入口规范。

成功生成程序 JSON 主体后,用户可以利用 ethers.js 遵循 ABI 将该程序主体发送到 EVM 兼容区块链上的 DARC 协议。此过程确保程序在 DARC 内完整执行。

图4展示了一个完整的编译过程。

By-law Script

```
batch_add_and_enable_plugins([batch_add_and_enable_plugins([
      {
            returnType: SANDBOX_NEEDED, // sandbox is needed
            level: 255, // level 255
            condition:
                  operation_equals(BATCH_MINT_TOKENS) &
                  (
                      mint_token_class_equals(0) | mint_token_class_equals(1)
                  ),
            votingRuleIndex: 0, // no voting rule index needed
            note: "before-op plugin 1",
            bIsBeforeOperation: true
      }
]);
```

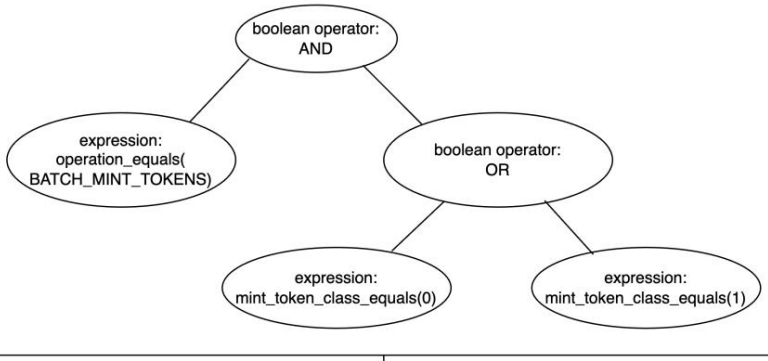
transpiler with operator overloading

Vanilla JavaScript

```
batch_add_and_enable_plugins([batch_add_and_enable_plugins([
      {
            returnType: SANDBOX_NEEDED, // sandbox is needed
            level: 255, // level 255
            condition:
                  pluginNode().booleanOperator().AND(
                      pluginNode().expression().operation_equals(BATCH_MINT_TOKENS),
                      pluginNode().booleanOperator().OR(
                          pluginNode().expression().mint_token_class_equals(0),
                          pluginNode().expression().mint_token_class_equals(1)
                      )
                  ),
            votingRuleIndex: 0, // no voting rule index needed
            note: "before-op plugin 1",
            bIsBeforeOperation: true
      }
]);
```

construction of condition expression tree

Expression Tree



codegen runtime

Generated Program (Plugin Condition Node)

index	node type	boolean operator	expression	expression parameter struct	child node list
0	boolean operator	AND	NULL	NULL	[1,2]
1	expression	NULL	operation_equals	BATCH_MINT_TOKENS	[]
2	boolean operator	OR	NULL	NULL	[3,4]
3	expression	NULL	mint_token_class_equals	0	[]
4	expression	NULL	mint_token_class_equals	1	[]

图 4:插件的转译和代码生成

5 多类别代币系统

多级代币系统是DARC协议中的核心机制,用户可以设计不同级别的代币,具有不同的投票权重和分红权重。通过插件作为限制,多类代币系统可用于代表组织中的不同组件或资产。

5.1 普通股

普通股是股份公司机制的核心要素,是筹集资本和分配所有权的一种手段。普通股股东通常享有投票权,这使他们能够在股东大会期间对公司的重要决策拥有发言权。

此外,普通股股东可能会收到股息,这是公司分配给股东的利润的一部分。投票和股息的双重作用使普通股成为股东参与公司治理和财务回报的关键工具。

如果某个代币级别的投票权重为 1,股息权重为 1,并且 DARC 中的所有重要事项都必须得到该级别所有代币持有者的批准,那么该代币级别可以被视为等同于 DARC 中的普通股。

要实例化这样的代币级别,必须首先在附则脚本中建立它,分配投票权重为 1,分红权重为 1。成功执行此脚本后,DARC 协议将初始化一个 0 级代币投票权重和股息权重均为1。

```
批处理创建令牌类（
    [   TOKEN_0   ], // 字符串 [0], [1], [1] 中的 token 符号
                        // 代币等级 // 投票权重 // 分红权重

）；
```

考虑到公司事务的复杂性,针对不同的情况制定了两种投票规则
目的:
投票规则1:要批准一项操作,必须满足以下条件:必须有超过50%的有效投票权赞成该操作,并且该投票需要采取绝对多数模式,即必须至少包括总代币投票权的50%。投票规则1的持续时间为7天（604800秒）。程序获得批准后,操作员需要在1小时（3600秒)内执行该程序。

投票规则2:要批准一项操作,必须满足以下条件:必须有超过75%的有效投票权赞成该操作,并且该投票需要采用相对多数模式,即必须至少包括有效投票权的75%。投票规则2的持续时间为1天（86400秒）。程序获得批准后,操作员需要在1小时（3600秒)内执行该程序。

下面的附则脚本显示了投票规则 1 和 2 的初始化。当添加投票规则索引为 1 或 2 的新插件时,插件将确保当条件被触发并且插件具有最高优先级时,投票过程将从选定的投票规则开始。

```
batch_add_voting_rule([ // 添加规则
    1 {

        // 代币级别 0 只允许用于投票 VotingTokenClassList: [0],


        // 批准阈值: 50% approvalThresholdPercentage:
        50,

        // 投票持续时间: 604800 秒 VotingDurationInSeconds:
        604800,

        // 执行等待持续时间: 3600 秒executionPendingDurationInSeconds:
        3600,
```

```
            // 投票规则是否启用: true isEnabled: true,

            // 关于投票规则的注释 note:    这是投票规则索引 1    ,

            // 是绝对多数 bIsAbsoluteMajority:
            true
        },

        // 添加规则 2 {

            // 代币级别 0 只允许用于投票 VotingTokenClassList: [0],

            // 批准阈值: 75%
            approvalThresholdPercentage: 75,

            // 投票时长: 604800 秒 VotingDurationInSeconds:
            86400,

            // 执行等待持续时间: 3600 秒executionPendingDurationInSeconds:
            3600,

            // 投票规则是否启用: true isEnabled: true,

            // 关于投票规则的注释 note:    这是投票规则索引 2    ,

            // 是绝对多数 bIsAbsoluteMajority:
            false
        }
]);
```

5.2 A/B 类股票

A 类和 B 类股票是企业融资的基本工具。 A 类提供投票权和股息优势,通常由创始人持有。 B 类的投票权有限,在保留控制权的同时筹集外部资本。这种双层结构平衡了治理和增长需求,对股东的决策和财务利益产生影响。

首先初始化两个代币级别:level-0,投票权重1,分红权重1;level-1,投票权重1,分红权重1
投票权重100,股息权重1。

```
批处理创建令牌类（
    [   TOKEN_0   ,   TOKEN_1   ], // 字符串中的代币符号 [0, 1], // 代币级别 [1, 100], [1, 1]

                                    // 投票权重 // 分红权重

）；
```

接下来我们将铸造 10000 个 0 级代币和 200 个 1 级代币。这将使0级代币的总投票权为10000,1级代币的总投票权为20000。这样,DARC的总投票权为30000,其中一级代币投票权占66.7%。这样,联合创始人和早期投资者将控制大部分 DARC 的投票过程,允许所有 0 级和 1 级代币持有者。

我们假设addr1和addr2总共持有10000个0级代币,各持有5000个,addr3、addr4、addr5和addr5各持有50个代币。首先在附则脚本中向这些地址铸造代币。

批量薄荷代币（
[0, 0, 1, 1, 1, 1], [5000, 5000, 50,
50, 50, 50], [addr1, addr2, addr3, addr4, addr5,
addr6]);

接下来添加一个插件,将 level-0 和 level-1 代币的总供应量限制为 10000 和 20000。

投票规则1:要批准该操作,0级和1级代币的总投票权的90%必须投赞成票。投票过程需要采用绝对多数模式。投票规则1的持续时间为7天（604800秒）。程序获得批准后,操作员需要在1小时（3600秒）内执行该程序。

操作前插件规则1:如果操作是BATCH MINT TOKENS以及代币级别 – –
mint 为 0 或 1,需要在沙箱中执行该操作才能做出决定。
操作后插件规则1:如果操作是BATCH MINT TOKENS以及代币级别 – –
mint 为 0 或 1,操作需要通过投票规则 3 批准。
操作前插件规则2:如果操作为BATCH DISABLE PLUGIN且索引为操作前插件1、操作前插件2或操作后插件1,则拒绝该操作。

该插件具有最高优先级。

在操作前插件1和操作后插件1中,当通过铸造新的0级或1级代币来改变股东结构时,需要获得总投票权的90%的批准才能进行操作。这将防止 1 级股东稀释 0 级代币的价值,并保护那些 0 级代币持有者（主要是散户投资者）。

操作前插件2是对上述其他插件的保护。当任何操作员尝试禁用这三个插件中的任何一个时,该操作将被拒绝。这将永久锁定该机制并保证插件无法被禁用。

```
batch_add_voting_rules([ // 添加操作前
    插件 1 {

        // 投票允许使用 0 级和 1 级代币 voteTokenClassList: [0, 1],


        // 批准阈值: 90%approvalThresholdPercentage:
        90,

        // 投票持续时间: 604800 秒 VotingDurationInSeconds:
        604800,

        // 执行等待持续时间: 3600 秒executionPendingDurationInSeconds:
        3600,

        // 投票规则是否启用: true isEnabled: true,


        // 关于投票规则的注释 note:  这是投票规则 1(index 0)  ,


        // 是绝对多数 bIsAbsoluteMajority:
        true
    },
]);
```

然后将三个插件添加到DARC。

批处理添加和启用插件（[

```
        // 添加索引为 0 的操作前插件 {

            returnType: SANDBOX_NEEDED, // 需要沙箱 level: 255, // 等级 255

            健康)状况:
                操作等于（BATCH_MINT_TOKENS)&（

                    mint_token_class_equals（0）| mint_token_class_equals(1)
                ),
            voteRuleIndex: 0, // 不需要投票规则索引 note:    before-op plugin 1    ,
            bIsBeforeOperation: true // 插件将作为
            before-op 添加
        },

        // 添加索引为 0 的操作后插件 {

            returnType: VOTING_NEEDED, // 需要投票 level: 258, // 级别 258 条件:


                操作等于（BATCH_MINT_TOKENS)&（

                    mint_token_class_equals（0）| mint_token_class_equals(1)
                )
            voteRuleIndex: 0, // 投票规则 1, index = 0 note:    after-op plugin 1    ,
            bIsBeforeOperation: false // 插件将作为
            before-op 添加
        },

        // 添加操作前插件 1 {

            returnType: NO, // 拒绝 level: 257, //
            级别 257
            健康)状况:
                操作等于（BATCH_DISABLE_PLUGIN)&（

                    禁用_before_op_plugin_index_equals(0) |禁用
                    _before_op_plugin_index_equals(1) |
                    disable_after_op_plugin_index_equals(0)
                )
            voteRuleIndex: 0, // 不需要投票规则索引 note:    before-op plugin 2    ,
            bIsBeforeOperation: true // 插件将作为
            before-op 添加
        }
]);
```

5.3 无投票权股票无投票权股票,也称为优先

股,是公司的所有权股份,没有投票权,但提供股息优先权、清算优先权和稳定性等财务利益。

它可以兑换或赎回,并因其稳定的收入而受到青睐。然而,它在公司决策中没有发言权,使其成为寻求资本保值且对公司事务控制有限的注重收入的投资者的选择。在DARC协议中,当一个代币初始化时分红权重为1,投票权重为0时,它可以被归类为无投票权股份。

5.4 需缴纳印花税的股票

需缴纳印花税的股票是在买卖时需缴纳政府税（称为“印花税”）的证券。该税适用于各种金融工具，包括股票和债券，通常由买方或卖方支付，具体取决于当地法规。人们在交易股票时缴纳印花税，因为它是政府收入的来源，也可以作为监管金融市场和阻止过度交易的工具。印花税的具体原因和税率可能因司法管辖区而异。

在DARC协议中，我们有能力创建一种机制，迫使代币持有者在转移不同级别的代币时支付被称为“印花税”的交易费。例如，我们可以将0级代币指定为DARC协议中的标准股票，并对每次代币转让征收1000 wei的固定印花税。

首先，我们必须实现一个插件，以在正在传输的代币属于 level-0 时禁用所有 BATCH TRANSFER TOKENS 操作。接下来，我们应该开发一个插件，仅通过 BATCH PAY TO MINT TOKENS 操作限制 0 级代币的转账，前提是每个代币的交易费用大于或等于 1000 wei。此外，用户需要将 BATCH PAY TO MINT TOKENS 操作的可分红标志设置为 0（假），以确保交易费用（印花税）不会被视为 DARC 的可分红收入。

```
_ _ _ _

批处理添加和启用插件（[
    // 添加操作前插件1：禁用0级代币的转发代币 {

        returnType: NO, // 拒绝 level: 255, // 级别 255 条件：


            operation_equals(BATCH_TRANSFER_TOKENS) &
                transfer_tokens_level_equals(0) VotingRuleIndex:
        0, // 不需要投票规则索引 note:  disable trasnfer tokens for level-0 tokens  ,
        bIsBeforeOperation: true // 插件将作为 before-op 添加

    },

    // 添加操作前插件2: // 允许支付0级代币转账， // 交易费用>=
    1000 wei/token {


        returnType: YES_AND_SKIP_SANDBOX, // 允许并跳过沙箱 level: 256, // 级别 256

        健康）状况：
            操作等于（BATCH_PAY_TO_TRANSFER_TOKENS）&（

                pay_to_transfer_tokens_level_equals(0) &
                transaction_fee_per_token_GE(1000) &
                pay_to_trasnfer_tokens_dividendable_flag_equals(0)
            )
        voteRuleIndex: 0, // 无需投票规则索引 注意：“允许支付带有交易费的 0 级代币转账

                > 1000 且可分红标志 0  ,
        bIsBeforeOperation: true // 插件将作为 before-op 添加
    }
]);
```

5.5 董事会

在 DAO 领域，决策通常是通过使用治理代币进行投票来达成的。然而，当每个决策涉及数千名代币持有者时，这个过程可能会变得缓慢且低效。此外，缺乏关于待执行的法规或预定义规则

提案智能合约意味着批准的投票可能会授权一系列广泛的行动
在 DAO 内。

相比之下,在传统的股份公司中,日常运营和事务受到监督
由董事会决定,而不是依靠全体股东的决策。董事会因其专业知识、战略敏锐性和快速决策的效率而闻名

能力,以及他们保持公司治理稳定性和一致性的能力。尽管
股东投票对于问责制和代表性仍然至关重要,董事会制度提供了结构化和知情的决策方法,确保公司的长期利益得到尊重

深思熟虑的优先顺序和有效的管理。

下面是 DARC Y 实验如何使用多级代币系统设计董事会的示例。有四种类型的代币,每种类型都有其独特的特征和作用

组织结构,如表 4 所示。

0级（A类代币）:这些代币的总供应量为400,同时拥有投票权和投票权
股息权重为 1。他们可能代表 DARC Y 协议中的共同利益相关者。

1级（B类代币）:B类代币总供应量为60个,拥有大量投票权
权重为10,使他们对决策具有影响力。它们的股息权重也为 1,
意味着收入分配中的潜在份额。

2 级（独立董事）:只有一个独立董事代币,设计为
没有投票权或股息权重,表明在治理结构中具有独特的作用。

第三级（董事会）:由五个代币组成,董事会拥有投票权
权重为1,但缺乏股息权重,表明其在决策中的主要功能。

等级	姓名	总供应量	投票权重	股息权重
0	A 类代币	400	1	1
1	B 类代币	60	10	1
2	独立董事		0	0
3	董事会	1 5	1	0

表 4:DARC Y 中代币分配的结构

以下是设计董事会机制的一些指导原则:

原则一:董事会应由最多 5 名成员组成,所有董事会成员
决策须经2/3以上成员同意。

原则2:如果一个地址持有超过2/3的A类代币,则可以代表A类代币的利益
散户投资者,并可能被选为 A 级董事会成员。在没有任何地址持有的情况下
超过 2/3 的代币,董事会中将没有 A 类代表。

原则 3:董事会应始终包括一名独立董事,该独立董事具有
任命他们的继任者的选项。罢免独立董事不属于董事会职务
允许的。

原则 4:董事会应由最多 3 名 B 级董事会成员组成,并可选择
添加持有超过 5% B 类代币的任何地址。现有董事会成员可以提名
B 级董事会成员,须经所有董事会成员至少 2/3 的批准。

原则 5:现有董事会成员有权提议罢免 B 类成员
董事会成员。此类提案在下列条件下可获批准： (1) 地址
被移除的数量小于或等于 B 类代币总供应量的 5%,或 (2)
该地址不拥有超过 A 类代币总供应量的 2/3 或独立代币
董事代币,该提案获得董事会2/3以上投票权
董事们。

原则 6:A 类 Token 董事、B 类 Token 董事和独立董事
应保持与代币持有的独立性。这意味着 (1) A 类代币董事可以
只持有A类代币和董事会代币,禁止持有Class代币
B代币或独立董事代币； (2) B 类代币董事可独家持有 B 类代币
代币和董事会代币,避免拥有 A 类代币或独立代币
董事代币； （三)独立董事只能担任独立董事
令牌。

接下来,我们将设计以下插件来实现上面定义的规则和流程:

运行前插件规则一:若运营商地址持有 2/3 以上的 0 级代币,且不持有任何 1 级、2 级、3 级代币,则可以铸造 1 个 3 级代币自身,无需沙箱检查。该插件规则是为 A 类代币董事会成员的提名而设计的。

操作前插件规则2:当操作者地址欲将二级代币转移到另一个目标地址,且目标地址不持有任何0级、1级、3级代币时,可以进行此操作无需沙箱检查即可批准。该插件有助于独立董事职位的转移。

操作前插件规则3:如果运营商地址持有1个二级代币和0个三级代币,并希望为自己铸造一个三级代币,则无需任何沙箱检查即可批准该操作。该插件的目的是提名独立董事。

操作前插件规则4:涉及二级代币铸造或销毁的操作
需要被拒绝。该插件限制独立董事的数量。
运行前插件规则5:如果某个运营商地址打算铸造1个三级代币,且目标地址持有一级代币总供应量的5%以上,且该运营商地址持有1个三级代币,则该操作必须经过沙箱检查。该插件用于 B 类代币董事会成员的提名。

操作前插件规则6:如果运营商地址打算从目标地址销毁3级代币,且目标地址持有的0级代币总量小于或等于2/3,则持有小于或等于5%的一级代币,且拥有0个二级代币,该操作无需沙箱检查即可批准。该插件有助于去除不合格的董事会成员。

运行前插件规则7:如果运营商地址打算从目标地址销毁3级代币,且目标地址持有的0级代币总量小于或等于2/3,则持有超过5%的一级代币,且拥有0个二级代币,且操作者地址至少拥有1个三级代币,该操作必须经过沙箱检查。该插件用于删除 B 类代币董事会成员。

操作后插件规则一:若运营商地址欲从目标地址销毁 3 级代币,且目标地址持有 0 级代币总量的 2/3 以下,则持有超过 0 级代币。 5%的一级代币,拥有0个二级代币,且运营商地址至少拥有1个三级代币,该操作必须按照投票规则1进行投票。本次投票涉及所有董事会成员,需要绝对多数模式,支持率超过66%。该插件用于 B 类董事会成员选举。

接下来是在附则脚本中实现的插件。

```
constplugin_before_op_1 = { returnType:
    YES_AND_SKIP_SANDBOX,级别:255,条件:
    operation_equals
     (BATCH_MINT_TOKENS)&number_of_token_mint_equals（1)&
                level_of_token_mint_equals（3)&
                operator_owns_num_of_token_GE（0,267)&
                operator_owns_num_of_token_equals（ 1, 0)
                &operator_owns_num_of_token(2, 0) &operator_mint_to_itself( ) ,
                votingRuleIndex: 0,注意:“在操作 1 之前”,
                bIsBeforeOperation: true


},

constplugin_before_op_2 = { returnType:
    YES_AND_SKIP_SANDBOX,级别:255,条件:
    operation_equals
     (BATCH_TRANSFER_TOKENS)&transfer_token_level_equals（2）,votingRuleIndex:
                0,注意:“在操作2之前”,bIsBeforeOperation:true
```

},

const plugin_before_op_3 = { returnType:
YES_AND_SKIP_SANDBOX, level: 255, 条件:

operation_equals(BATCH_MINT_TOKENS) & opera_owns_num_of_token_equals(2,
1) & number_of_token_mint_equals(1) & level_of_token_mint_equals(3) ,
VotingRuleIndex: 0, 注意: 在操作之前3 ,
bIsBeforeOperation:真的

},

const plugin_before_op_4 = { 返回类型:NO,级
别:257,条件:

(operation_equals(BATCH_BURN_TOKENS) & level_of_token_burned_equals(2))
| (operation_equals(BATCH_MINT_TOKENS) &
level_of_token_mint_equals(2)) ,

voteRuleIndex: 0,注意:“在操作
4 之前”,bIsBeforeOperation:
true
},

常量plugin_before_op_5 = {
returnType:SANDBOX_NEEDED,级别:256,条
件:operation_equals
（BATCH_MINT_TOKENS)&level_of_token_mint_equals（3)&
number_of_token_mint_equals（1)&
operator_owns_num_of_tokens_equals（3,1)&
batch_mint_tokens_target_address_ownes_num_of_tokens_greater
（1,12）),
voteRuleIndex: 0,注意:“在操作
5 之前”,bIsBeforeOperation:
true
},

const plugin_before_op_6 = { returnType:
YES_AND_SKIP_SANDBOX,级别:255,条件:

operation_equals(BATCH_BURN_TOKENS) & level_of_token_burned_equals(3)
& batch_burn_token_target_address_owns_num_of_tokens_LE(1,
12) & batch_burn_token_target_address_owns_num_of_tokens_LE (0, 267) &
batch_burn_token_target_address_owns_num_of_tokens_equals(2,0),

voteRuleIndex: 0,注意:“在操作
6 之前”,bIsBeforeOperation:
true
},

常量plugin_before_op_7 = {
返回类型:SANDBOX_NEEDED,级别:256,

条件:操作等于（BATCH_BURN_TOKENS)&
level_of_token_burned_equals（3)&
batch_burn_token_target_address_owns_num_of_tokens_GE（1, 12)&
batch_burn_token_target_address_owns_num_of_tokens_LE（0, 256)&
batch_burn_token_target_address_owns_num_of_tokens_equals (2,0),
voteRuleIndex: 0,注意：“在
操作 7 之前”，
bIsBeforeOperation: true
},

常量plugin_after_op_1 = {
返回类型:VOTING_NEEDED,级别:253,条件：
操作等于
（BATCH_BURN_TOKENS)&level_of_token_burned_equals（3)&
batch_burn_token_target_address_owns_num_of_tokens_GE
（1,12)&batch_burn_token_target_address_owns_num_of_tokens_LE（0, 267)&
batch_burn_token_target _address_owns_num_of_tokens_equals(2,0) &
operator_ownes_num_of_tokens_equals(3,1),

voteRuleIndex: 1,注意：“操
作 1 之后”,bIsBeforeOperation:
false
},

batch_add_and_enable_plugins([plugin_before_op_1,plugin_before_op_2,plugin_before_op_3,plugin_before_op_4,plugin_before_op_5,plugin
]);

5.6 公司债券

公司债券是公司为筹集资金而发行的债务证券。投资者向公司借钱,以换取定期支付利息并在到期时返还本金。它们为公司提供了一种为各种目的
获取资本的手段,并为投资者提供了可预测的收入流。

在 DARC 协议的上下文中,命令 BATCH PAY TO MINT TOKENS 和 BATCH BURN TOKENS AND REFUND 有助于创建
可在指定时间范围内由任何地址购买的债券代币,并以指定的购买和赎回价格。

这些债券代币的购买和退款有两条规则：
规则1:2020年1月1日至2020年2月1日期间,任何地址可以使用BATCH PAY TO MINT TOKENS以每个代币10,000 wei的价格购买不超过
100个债券代币（二级）。债券代币的总供应量不应超过10000。

规则 2:2030 年 1 月 1 日后,地址可以选择使用批量销毁代币并退款,以每个代币 15000 wei 的价格赎回债券代币。

批处理添加和启用插件（[
{
返回类型:YES_AND_SKIP_SANDBOX,级别:253,条
件:操作_等于
（BATCH_PAY_TO_MINT_TOKENS）
& pay_to_mint_tokens_price_per_token_equals(10000)

```
                    & timestamp_greater(1577858400) // 2020-01-01-0-0-0 &
                    timestamp_less(1580536800) // 2020-02-01-0-0-0 &
                    pay_to_mint_tokens_level_equals(2) // 代币级别 &
                    number_of_token_pay_to_mint_LE(100 ) // 铸造数量
                    &total_number_to_tokens_LE(2,99900), // 代币供应总量
        voteRuleIndex: 0, 注意:       ,

        bIsBeforeOperation: true
    }, {

        返回类型:YES_AND_SKIP_SANDBOX,级别:253,条
        件:操作_等于
         （BATCH_BURN_TOKENS_AND_REFUND）
                    & burn_tokens_and_refund_price_per_token_equals(15000) &
                    timestamp_GE(1893477600) // 2030-01-01-0-0-0 &
                    burn_tokens_and_refund_level_equals(2), // 代币级别
        voteRuleIndex: 0, 注意:       ,

        bIsBeforeOperation: true
    },
]);
```

5.7 产品代币和不可替代代币

当我们将投票权重和分红权重都设置为0,并强制要求运营商以特定价格铸造代币时,这些代币可以被视为代表DARC实例的付费产品或服务的产品代币。

当我们进一步配置多个级别的代币时,每笔交易仅允许一次付费铸币一种代币,禁止铸币和销毁操作,并且如果该级别的一种代币已经存在,则不允许额外的付费铸币交易。同时,我们允许这些代币的持有者自由地将它们转移到其他地址,此时,这些特定级别的代币可以被视为NFT（不可替代代币） [WE]。 NFT 可以被用作并被视为独特的数字资产,代表 DARC 协议中特定项目或内容的所有权或真实性证明。

6 个插件

6.1 插件设计

插件是DARC协议中的法律,DARC内的所有程序和操作都必须遵守所有插件施加的限制。对于单个插件,它遵循以下伪代码中概述的逻辑：

```
ifplugin.condition: 返回
      plugin.returnType
```

对于DARC协议来说,前操作插件和后操作插件的主要区别在于它们的返回类型。对于操作前插件,当它们确定某个操作是否应该直接执行、彻底拒绝或进入沙箱时,它们具有三种不同的返回类型作为最终决定：

1. 不。当操作前插件的条件被触发时,插件对该操作的决定为“否”。此决定表明插件认为该操作违反了其规则,因此在进入沙箱执行之前被彻底拒绝。

2. 需要沙盒。当触发操作前插件的条件时,插件对该操作的决定是需要沙箱。此决定表明插件无法确定是否应接受或拒绝该操作。该插件知道 –

操作需要通过操作后插件在沙箱中进行评估,因此决定让操作进入沙箱进行进一步评估。

3. 是并跳过沙盒。当触发操作前插件的条件时,插件对该操作的决定是“是并跳过沙盒”。此决定表明插件已确定该操作应获得批准并且不需要在沙箱中执行。

因此,操作可以直接进行,无需经过沙箱。

对于后操作插件,由于程序已经在沙箱中执行并可以开始投票,因此这些插件可以有三种返回类型作为其最终决定:

1. 不。当操作后插件的条件被触发时,插件对该操作的决定为“否”。此决定表明该插件认为该操作违反了其规则,应被彻底拒绝。

2. 需要投票。当操作后插件的条件被触发时,插件对操作的决定是需要投票。这个决定表明插件认为该操作需要投票,并且该操作需要根据该插件指定的投票规则初始化一个投票项。

3. 是的。当操作后插件的条件被触发时,插件对该操作的决定是YES。此决定表明插件认为,根据其规则,应该允许操作继续进行。

每个插件都有一个条件节点数组,其中条件节点按顺序存储。根节点对应于索引 0 处的节点,即第一个节点。条件节点数组遵循以下原则:

1、每个节点的类型可以是布尔运算符,也可以是表达式;

2. 对于布尔运算符,类型必须设置为 AND、OR 或 NOT 之一;

3. 对于 AND 和 OR 运算符,必须在 child 中至少指定两个有效的子节点索引
节点列表;

4、对于NOT运算符,必须在子节点列表中指定唯一的子节点索引;

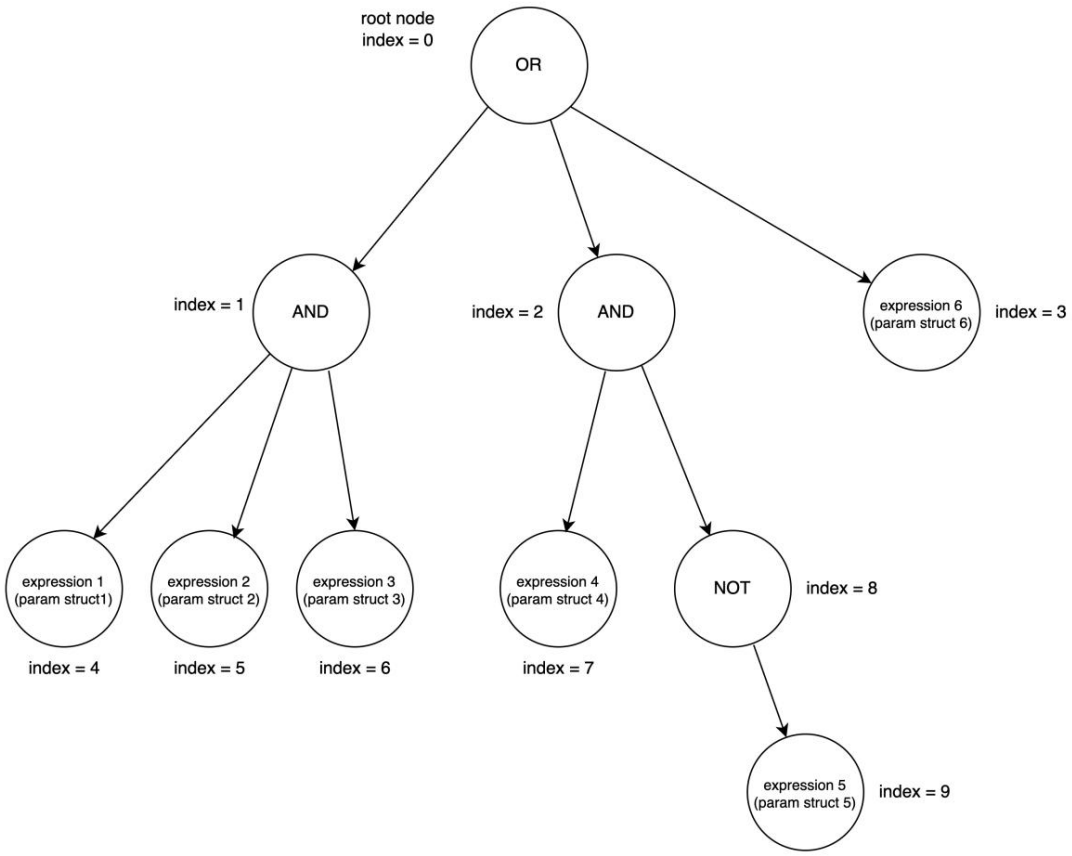
5. 对于表达式节点,有效的条件表达式参数必须与表达式一致
被设定;

6. 对于表达式节点,其子节点列表的长度必须为0,即没有子节点
允许。

图5是说明如何将条件表达式二叉树序列化为
条件节点数组。

另外,插件需要设置两个参数:一是“级别”,代表该插件在整个插件系统中的优先级。对于同一个操作,判断系统会遍历所有插件,有可能至少触发两个或多个插件。在这种情况下,如果插件的级别不同,判断系统会以级别较高的插件作为最终判定。

另一个参数是“投票规则索引”,它指向投票规则数组中的特定索引。当插件的最终决策是VOTING NEEDED时,插件请求DARC协议使用投票规则索引指示的投票规则来初始化投票项。如果插件的返回类型不是 VOTING NEEDED,则投票规则索引将被忽略。



Node Array Index	0	1	2	3	4	5	6	7	8	9
Boolean Operator	OR	AND	AND	NULL	NULL	NULL	NULL	NULL	NOT	NULL
Expression	NULL	NULL	NULL	exp6	exp1	exp2	exp3	exp4	NULL	exp5
Parameter Struct	NULL	NULL	NULL	param6	param1	param2	param3	param4	NULL	param5
Child Node Index	[1,2,3]	[4,5,6]	[7,8]	[]	[]	[]	[]	[]	[9]	[]

图 5:条件节点和表达式树

6.2 插件和判断系统对于DARC协议,判断系统需要经过两次评

估:一次是通过运行前插件进行评估,另一次是程序在沙箱中完全运行后,再通过运行后插件进行评估。这样设计的原因是,如果没有沙箱,仅依靠一组插件进行判断,那么预测程序的行为就变得具有挑战性。因此,无法防止 DARC 协议中特殊状态的修改。

例如,在一个 DARC 实例中,股东 X 需要永久持有 15% 的投票权和 10% 的分红权,在设计插件时,无法预测 DARC 实例在执行诸如铸币或代币等操作后的状态。烧毁代币。这种不确定性给确保股东 X 保持 15% 投票权和 10% 股息权的永久所有权带来了挑战。只有在沙箱中运行操作,然后重新评估沙箱的状态,才能防止此类修改。

另一种场景,如果需要确保DARC实例在2035年1月1日之前永久保留10000个原生代币,则只能通过在DARC实例中执行这些操作来保证正确检测和阻止诸如支付股息或提取现金等操作。沙箱。

沙箱中所有操作完成后,判断系统通过操作后插件进行第二次评估。如果没有沙箱,仅依赖插件,这种机制的设计将过于复杂。

如果只有事后插件,而沙箱没有事前插件,那么成本高昂且效率低下。这是因为沙箱的运行成本非常高。它不仅要求程序完全在沙箱中运行,还涉及通过完全复制整个DARC实例的内部状态来初始化沙箱。这个过程会产生大量的汽油费。

对于大多数无需在沙箱中运行即可批准的简单操作,直接在操作前插件中建立规则会更具成本效益。例如,小股东与散户之间的股票交易、客户进行日常交易、董事会成员进行日常支付操作、员工给自己发放工资和股票激励 这些众多的日常高频活动都可以定义为以前的行为。 -操作插件。这种方法有助于节省绝大多数日常操作的汽油费。

对于前操作插件来说,每当程序提交到DARC协议时,判断系统都会依次检查每个操作。对于每一个操作,判断系统都会遍历每个操作前的插件,得到单个判断结果。最后,它汇总所有操作的结果以确定整个程序的总体结果。该决定决定程序是否需要在沙箱中运行(需要沙箱)、被彻底拒绝(否),或者不需要沙箱而直接运行(是并跳过沙箱)。术前判断遵循以下规则:

1. 如果有任何操作被判断系统判断为NO,则整个程序被拒绝,并返回
否的结果。

2、如果没有一个操作被判断系统判断为NO,并且至少有一个操作被判断为SANDBOX NEEDED,则要求整个程序在沙箱中运行,结果为SANDBOX NEEDED。

3、如果所有操作被判断系统判断为YES AND SKIP SANDBOX,则整个程序被批准,结果为YES AND SKIP SANDBOX,整个程序可以跳过沙箱。

图6说明了程序中的各个操作如何被操作前插件判断,产生判断结果,决定程序是否需要投票批准(VOTING NEEDED)、是否应该直接拒绝(NO)、或者可以直接进行(YES)。事后判断适用以下规则:

1. 如果有任何操作被判断系统判断为NO,则整个程序被拒绝,并返回
否的结果。

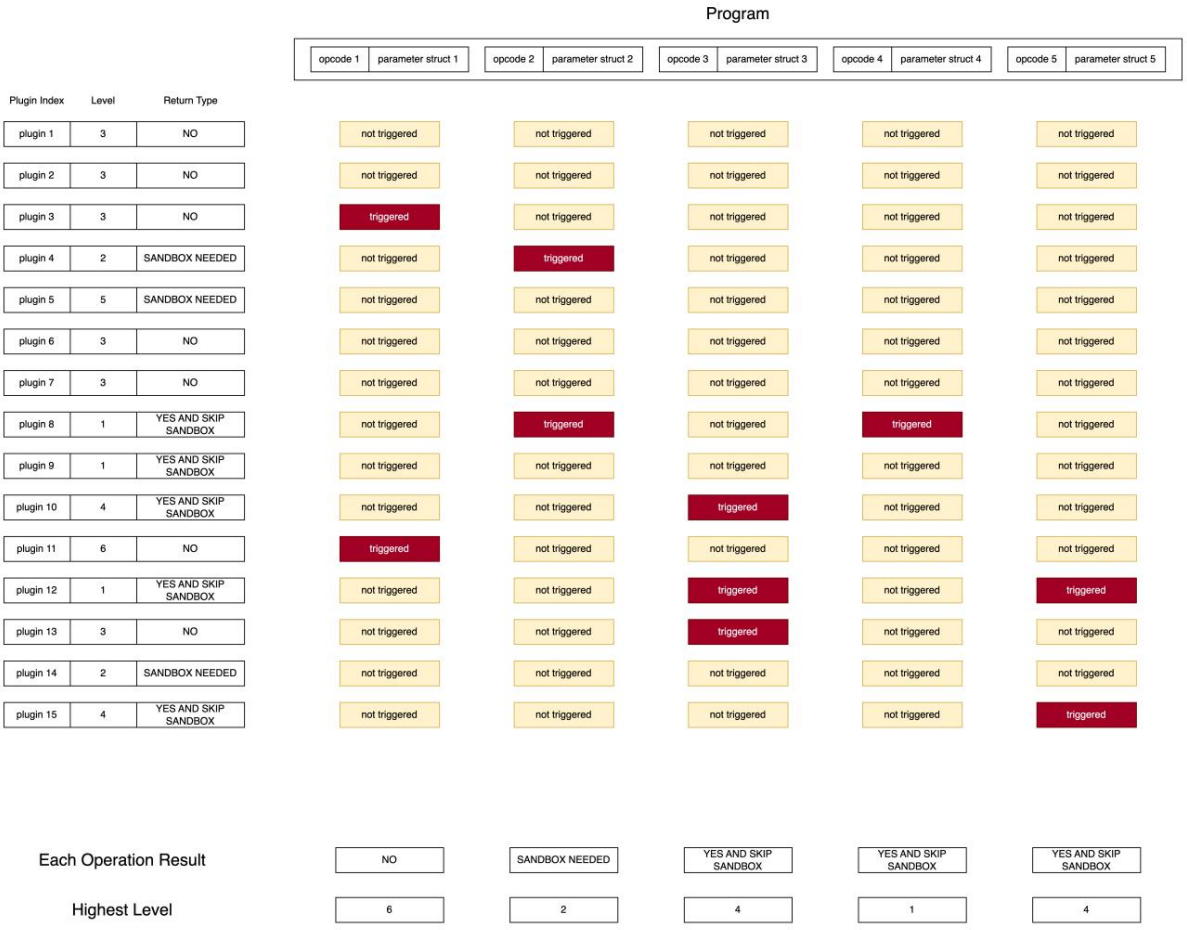


图6:运行前插件对程序的判断

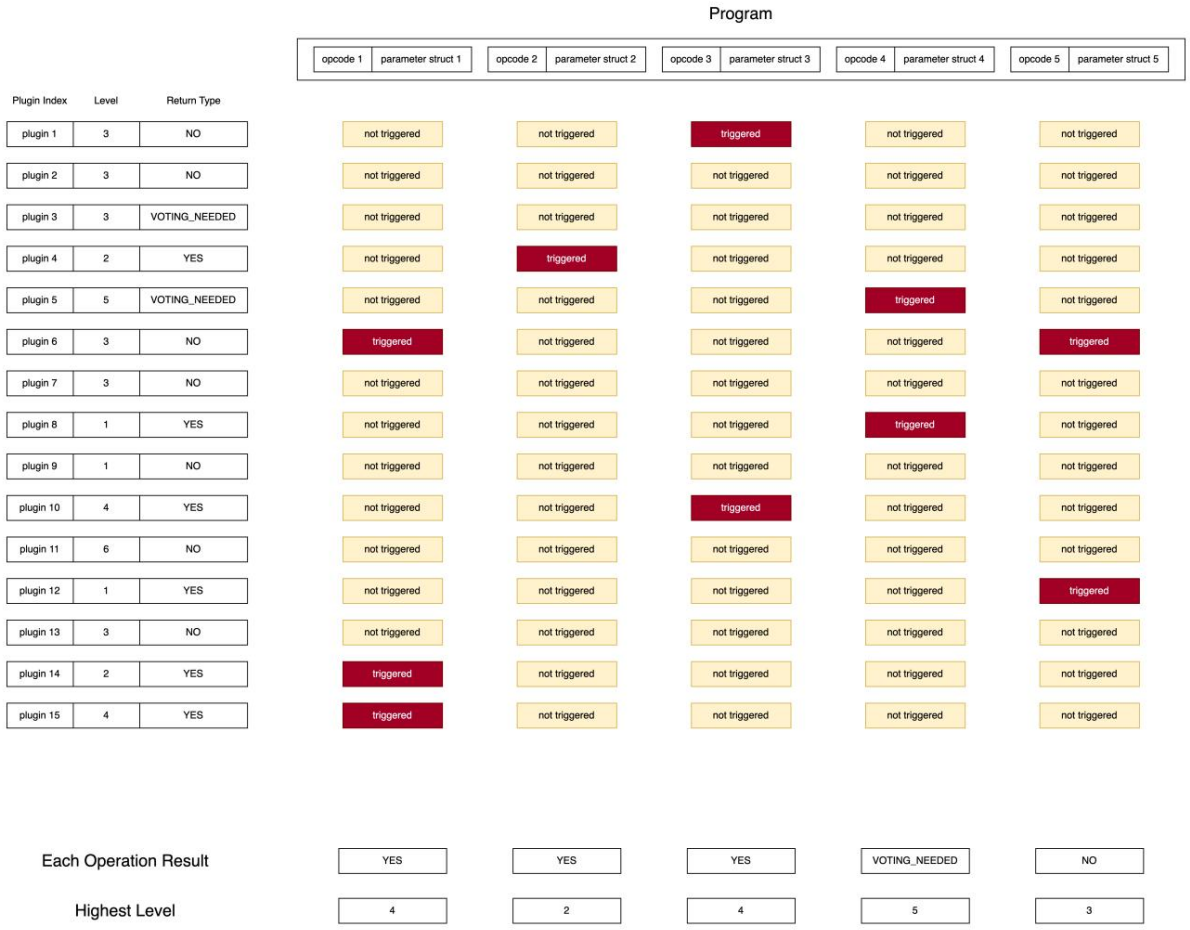


图7:运行后插件对程序的判断

2、如果没有任何操作被判断系统判定为NO,并且至少有一个操作被判定为VOTING NEEDED,则整个程序最终判断为VOTING NEEDED。该项目将被放入待决项目类别,DARC协议必须启动投票系统来决定批准或拒绝。

3、如果所有操作被判断系统判断为YES,则整个程序被认可,结果为YES,整个程序可以直接执行。

图7说明了程序中的各个操作是如何通过后操作来判断的
插件,产生判断结果。

7 投票

7.1 投票过程和状态

如图8 所示,您可以将每个 DARC 协议视为具有三个的有限状态机 (FSM)
状态:

1、空闲状态:在该状态下,DARC协议可以接受任何节目。当程序被接受并执行时,DARC 协议转换回空闲状态。同样,如果程序被拒绝,DARC 协议将返回到空闲状态。当程序包含需要投票的操作时,DARC 协议将转换到投票状态。

2. 投票状态:在DARC协议的投票状态下,所有用户都被允许且排他地允许执行包含单次投票操作的程序。如果一个程序包含多个操作,则该程序将被拒绝。类似地,如果程序包含单个操作,则该操作不是

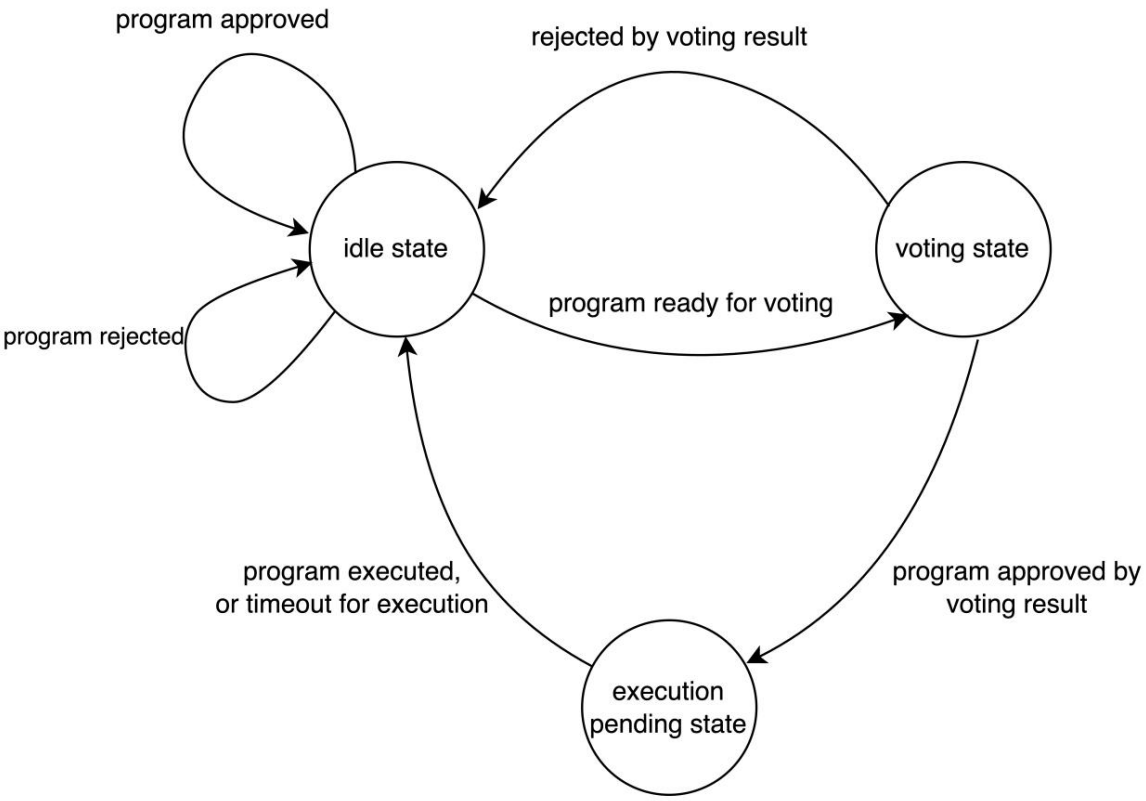


图8:DARC的有限状态机

一旦投票,该方案也将被否决。如果当前投票获得通过,系统将自动转入执行待决状态。此外,投票状态有最短时间限制。如果在该时限内,每个投票项目的支持票数合计未超过定义的阈值,则认为本轮投票不成功,系统将自动返回空闲状态。只有在规定的时限内所有投票项目都获得批准后,该方案才会获得DARC协议的批准。

3、执行等待状态:当一个程序被DARC协议投票通过后,就可以在执行等待状态下执行。执行挂起状态也有最短时间限制。如果程序在这个限制内执行,就会成功执行,系统会自动恢复到空闲状态。如果在此限制内没有执行,则程序将无法执行,DARC协议将丢弃该程序,自动返回到空闲状态。

7.2 投票规则

每条投票规则由以下几项组成:

- 投票代币类别列表:允许对该投票项进行投票的代币类别索引列表。
  它至少包含一个有效的令牌类别索引号。具有此数组中列出的索引的所有代币都被视为有效的投票代币。所有持有此数组中包含的任何代币的代币持有者都可以对该项目进行投票。

- 批准阈值百分比:批准阈值百分比。

- 布尔标志 bIsAbsoluteMajority:一个布尔标志,指示投票模式是否为 by
  绝对多数或相对多数。

- 投票持续时间(以秒为单位):此投票过程的最长持续时间(以秒为单位)。如果投票过程花费的时间超过此参数,则投票过程将被终止,投票结果将被设置为失败。

- 执行等待时间（以秒为单位）:投票过程完成后,操作员执行已批准程序的最长持续时间（以秒为单位）。如果该计划获得批准,运营商必须在规定的期限结束之前执行该计划;否则,程序将被中止,并且DARC将被重置为空闲状态。

- 布尔标志isEnabled:如果投票规则已初始化,则必须将其设置为True 的布尔标志并成功启用。

- 注意:存储在投票规则中的字符串,包含有关此投票规则的额外信息、注释或外部 URL。

```
结构投票规则 {

    /** *
      投票代币类别索引列表 */

    uint256[] voteTokenClassList;

    /** *
      投票政策的批准阈值百分比 */

    uint256 批准阈值百分比;

    /** *
      投票策略的投票持续时间,以秒为单位 */

    uint256 投票持续时间;

    /** *
      投票策略的执行等待时间（以秒为单位） */

    uint256executionPendingDurationInSeconds;

    /** *
      投票策略是否启用 */

    布尔值已启用;

    /** *
      投票政策的注释 */ string note;



    /** *
      投票政策是绝对多数或相对多数。 */

    bool bIsAbsoluteMajority;
}
```

7.3 投票类型

通常,在DARC协议的投票系统中,有两种投票方式：

- 绝对多数:在绝对多数投票中,批准需要超过固定阈值,以总投票权重的百分比表示。如果批准票的总权重超过预定阈值,则视为投票成功。例如,总权重为1000,阈值设置为70%,如果赞成票总权重超过700,则视为投票成功。

- 相对多数:在相对多数投票中,批准与总投票权重的百分比有关。与绝对多数不同,总投票权重是实际投票的百分比,而不是预先定义的绝对值。例如,如果总权重为 1000,但只有 300 权重参与投票,则相对多数要求批准的投票权重超过所投票数的 70%（0.7 * 300 = 210)才能视为投票成功。

由于投票权的不同,我们在投票时提出了“绝对多数”和“相对多数”的选项。
在平衡投票结果时强调这两种模式的作用。

就绝对多数而言,其优点是批准的投票权重只需构成可能投票总数的一定比例,具有更大的灵活性。它适用于绝大多数足以决定结果以促进更广泛共识的情况。其应用场景适用于需要获得绝大多数支持的重要决策。以绝对多数为标准,有助于促进更广泛的共识。

相比之下,相对多数的优势在于,批准的投票权重只需构成当前投票的最大份额,从而提供了更大的便利性和灵活性。这种方法适用于相对多数足以决定结果的情况。

它适用于需要更灵活和更快速的决策过程的情况。相对多数有助于更迅速地达成共识,而无需确保总可能投票权重的固定百分比。

通过提供绝对多数和相对多数选项,我们增强了投票规则的适应性,以满足各种协议和组织的多样化治理需求。选择合适的投票模式有利于更好地满足特定决策场景的共识和效率要求。

7.4 投票机制

对于每个程序,在获得操作前插件的批准并在沙箱内完成其完整执行后,判断系统在所有操作后插件的指导下对每个操作进行评估。如果来自最高级别的所有插件对特定操作的判断被标记为“需要投票”,则收集与该插件关联的所有投票规则。一旦判断系统编译完程序中每个操作对应的所有投票规则,它就会生成一个投票项数组,将每个规则分配给其各自的条目。

对于每一项操作,可能会同时触发一个或多个不同的插件,所有插件的返回类型均为“VOTING NEEDED”。在这种情况下,与指向该操作的每个插件关联的投票规则将按顺序收集。另一方面,对于每个程序,相同的插件可能会依次触发不同的操作。在投票项数组中,每一次操作都会根据相应的投票规则生成一个新的投票项。

图9是说明DARC协议如何初始化基于投票项数组的示例
关于判断结果。

一旦投票项数组被初始化,作为有限状态机,DARC 协议就会转换到投票状态。假设投票项数组中有N个投票项,则允许所有操作者投票,提交一个长度为N的布尔数组。每个布尔值代表操作者对相应投票项的支持或反对。对于每个算子,在每个投票状态下,他们只能而且必须投票一次,确保投票操作中的参数是长度为N的布尔数组。如果一个算子尝试多次执行投票操作或者提供一个布尔数组长度不是N的,投票方案将被自动拒绝。

在 DARC 协议中,每个投票项都有两个计数器:一个用于投“YES”票,另一个用于投“NO”票。每当操作者进行有效投票操作时,每个投票项都会计算该操作者在相应投票规则中的总投票权重。在给定的投票项中,每个投票规则定义了一组允许的令牌级别 C = [c1, c2, ..., cn],其中 n 是令牌级别的数量。每个代币级别ci都有一个相应的投票权重,表示为wi 。假设在投票过程中,有运营商选择对该投票项进行投票,则该运营商的投票权重W由以下公式计算:

$$W = \sum_{i=1}^{n} w_i$$

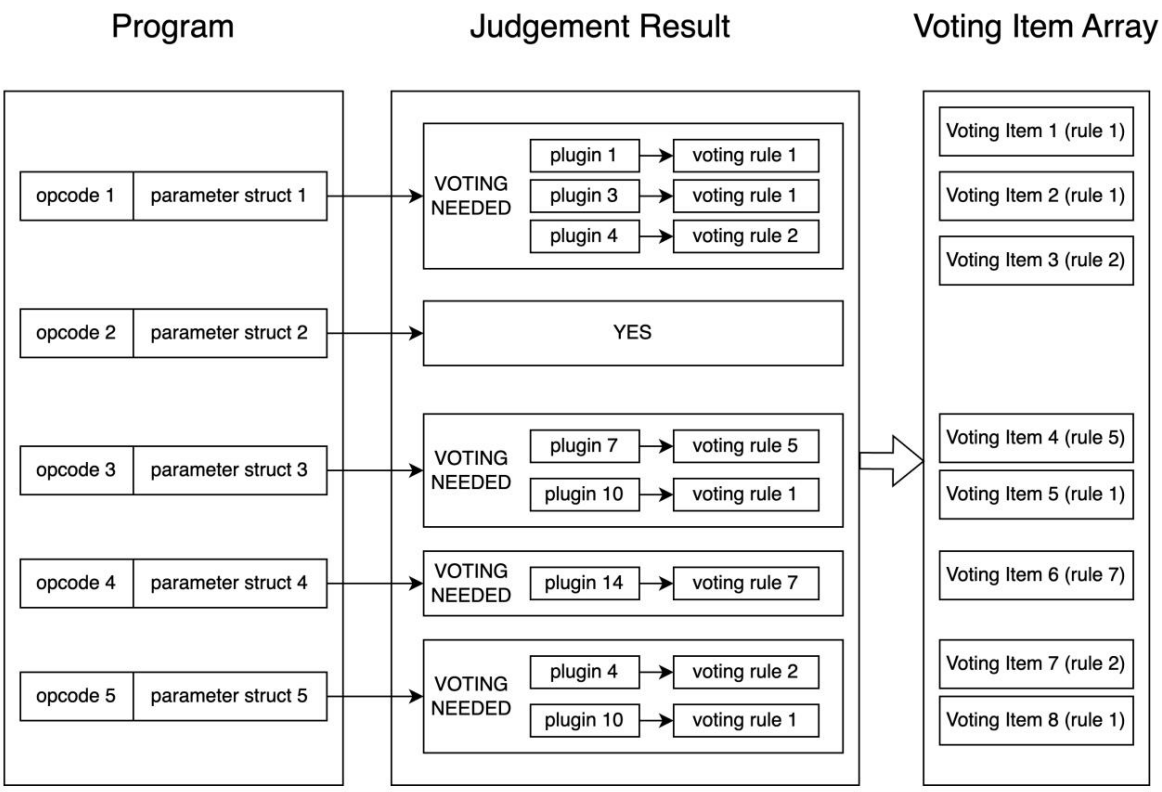


图 9:投票规则和项目

式中,W代表运营商对该投票项的总投票权重,对所有允许投票的代币级别ci进行求和。对于每个级别 i， vi是该运营商在该级别拥有的代币数量。

每个操作者提交有效的投票操作后,每个投票项都会将相应的投票权重递增到YES或NO计数器中。如果操作员在特定投票项的可投票代币集中持有零个代币,则 YES 和 NO 计数器都不会受到影响。投票过程完成后,DARC协议将根据以下标准确定投票是否结束：

1. 当所有投票项目中至少有一个投票项目以绝对多数方式运作,并且该投票项目获得了足够数量的否票时,整个投票过程将提前终止,导致投票失败。随后,DARC 协议转变为空闲状态。例如,如果某个投票项目的通过门槛为66%,采用绝对多数模式,并且在截止日期前获得了34%以上的反对票,则整个投票过程将立即结束,导致投票失败并转入投票状态。空闲状态。

2. 当所有投票项目均以绝对多数方式运行,且每个投票项目获得的赞成票均超过批准门槛时,整个投票过程提前结束,表明投票成功。随后,DARC 协议转变为执行挂起状态。

3. 当投票状态不满足标准1和2时,DARC协议将等待截止日期到期进行评估。如果在截止日期后,每个绝对多数模式的投票项目获得的投票权重等于或大于总投票权重乘以赞成票阈值,并且对于相对多数模式的每个投票项目,该项目获得的投票权重等于或大于超过提交的投票权重乘以 YES 票数的阈值,则投票获得批准,并且 DARC 协议过渡到执行待决状态。如果有任何投票项目

不满足此标准,则视为投票不成功,DARC协议恢复为
空闲状态。

进入执行等待状态后,当前投票状态存储的程序已被
已获批准并准备执行。程序必须在新的执行挂起之前完成
最后期限。成功执行后,DARC 协议将转换为空闲状态。然而,如果一个
执行过程中发生错误或异常,或者在截止日期之前没有人执行程序,
DARC协议也恢复到空闲状态,程序无法继续进行
执行。

8 会员资格

在每家公司中,除了董事会、股东、高管等成员外,
还有无数的员工、承包商、实习生、客户、供应商等等。他们可能会
不持有任何一级代币或公司股票,但他们的工资、订单、认购费和其他
功能需要批量管理。因此,会员资格是另一套便捷的工具
用于 DARC 协议。

表5是咨询公司 DARC X 的层次成员结构示例。
在这个表中,我们定义了 6 个级别:联合创始人、大股东、高管、经理、员工和
离职员工。我们可以根据这个成员资格表为DARC X定义一些插件。

规则一:我们需要限制联合创始人在没有通知的情况下大量出售股票
给高管们。如果运营商是具有“联合创始人”角色的地址并且运营商销售更多
超过 100000 个 0 级代币,此操作需要获得所有董事会成员的批准。

规则2:我们每月提供10 ETH作为员工级别和经理级别的工资
地址（4 级和 5 级）。

等级	角色
1	联合创始人
2	主要股东
3	管理人员
4	经理
5	员工
6	离职员工

表 5:分层成员资格结构

地址	角色	姓名	已暂停
0x0AC..03	1 (联合创始人)	安	不
0x156..21	2 (大股东) Banana Capital	3 (高管) 0x918..1B	4 (经理) 0x4E1..90 5 (员工) 0x510..0B
职员工）	0x113..C7	6 （已离	汤姆 不
		杰克	不
		鲍勃	不
		蒂姆	不

表6:DARC X的成员资格表

现在我们分析上述规则并在By-law Script中设计插件。

对于规则1,我们需要确定如果当前操作是“转账代币”,则级别
token为0,且token数量大于100000,且地址在会员表中
当角色级别等于1时,程序将被暂停并等待投票过程。这
投票流程定义并保存为投票规则1,要求所有3级代币持有者
1小时内投票。三级代币共有 5 个,每位董事会成员持有 1 个代币。如果所有的
1小时内5名董事会成员投票赞成,该操作将获得批准。否则整个程序
将被拒绝并且操作将失败。

以下是用附则脚本设计的规则 1 的示例插件:

常量plugin_Rule_1 = {

条件： (operation_name_euqlas(BATCH_TRANSFER_TOKENS)) &
(transfer_token_level_equals(0)) &
(transfer_token_amount_greater(100000)) &
(operator_membership_level_equals(1)) ,
return_type: VOTING_NEEDED、
is_before_operation: false、return_level:
100、voting_rule_index: 1

}

对于规则2,我们需要确保如果操作者被分配的角色级别等于4或5,则该操作是添加可提现余额,操作总量小于或等于10 ETH（或10000000000000000000 wei），并且操作者在超过30天（或2592000秒)内执行了相同的操作,该操作将被批准,并且该操作可以跳过沙箱检查。

以下是用附则脚本设计的规则 2 的示例插件：

常量plugin_Rule_2 = {

条件:操作等于（BATCH_ADD_WITHDRAWABLE_BALANCE)&total_add_withdrawable_balance_LE
（10000000000000000000)&last_operation_by_operation_period_for_operator_GE（

BATCH_ADD_WITHDRAWABLE_BALANCE,2592000
)
& ((operator_membership_level_equals(4))
| (operator_membership_level_equals(5))
) ,
返回类型:YES_AND_SKIP_SANDBOX,bIsBeforeOperation：
true,级别:100,votingRuleIndex:0，

笔记： ”“

}

9 股息

可分红资金池中将包含万分之一的X,可用于分红。根据可分红资金池中的总金额T以及当前DARC协议内各代币级别的分红权重之和,我们可以确定每个分红单元的分红金额如下：

$$你=\frac{XT}{10000W}$$

其中,u表示每个分红单位的分红金额,T为可分红资金池的总金额,W表示当前DARC协议中不同代币级别的所有可分红代币的分红权重之和。

对于每个用户,假设用户i持有的各个代币级别的分红权重之和为Di
，用户i本轮可获得的分红总额Ui为：

$$乌伊=\frac{迪}{10000}$$

对于 DARC 协议,可以采用三种股息分配方法： 基于交易的股息分配:第一种方法是根据可分红交易的数量来分配股息。
这是通过将股利交易周期计数器 N 设置为合理值来实现的。在收到至少 N 笔可分红交易后，

指令 OFFER DIVIDENDS 变为可执行。执行后,可分红资金池和可分红交易周期计数器重置为零,启动新的计数。这个过程不断重复,等待接下来的N笔交易以及后续的分红操作。

基于时间的股利分配:第二种方法需要基于时间的股利分配。
将 N 设置为 1,可以引入一个附加插件,允许在整个 DARC 实例内的上一次调用后不少于 S 秒调用 OFFER_DIVIDENDS。通过这种方式,可以每 2 周、4 周、3 个月、6 个月或 1 年安排股息。这种方法更符合传统的企业股息。

资金池分红:第三种方式是按照可分红资金池总额进行分红。将 N 设置为 1,每当可分红资金池中的金额超过指定阈值时,附加插件就会启用提供股息。按照这样的设计,获得Y个可分红原生代币后即可触发分红。

用户可以选择上述任何一种方法或设计插件,根据不同的DARC组织结构和方法创建另类的、实用的、合理的分红模式。

值得注意的是,为每笔收到的可分红交易执行提供分红可能会导致大量的汽油费浪费。这是由于 O(MNP) 的潜在时间复杂度,其中 M 代表代币级别的数量,N 是每个级别中代币的数量,P 是代币持有者的总数。因此,实施高效的分红机制以节省Gas费势在必行。

此外,可分红资金池中的资金不会被 DARC 协议锁定在智能合约中。该协议不保障现金股息,OFFER DIVIDENDS 仅执行计算,将每个代币持有者即将到来的股息存储在可提取的股息余额中。当运营商提取股息时,如果DARC协议缺乏足够数量的原生代币,则无法提取股息。为了保护特定或所有代币持有者的分红权,必须设计额外的插件。

10 紧急代理

在DARC协议中,用户经常会遇到各种问题,例如:

1. 设置错误的插件参数、触发条件或执行错误的token操作
   系统,导致恢复不可能。
2. 锁定DARC协议中的某些关键操作,导致协议内的功能永久不可用。

3、人为纠纷导致组织无法运转,无法通过插件解决,可以通过文档、文本、证据等人工诉讼的方式解决。

4. 发现DARC协议存在漏洞或面临攻击,需要紧急暂停
   和恢复。
5. 解决其他可能出现的潜在技术问题或争议。

在DARC协议中,用户可以指定一名或多名紧急代理来应对紧急情况。
如果发生不可预见的情况,用户可以邀请紧急人员进行干预。紧急代理充当超级管理员,有权执行 DARC 协议内的任何操作,而不受插件施加的任何限制。这一角色对于解决 DARC 协议中的紧急或未解决的问题至关重要。

1. addEmergency(emergencyAgentAddress):该命令用于通过提供紧急代理的地址来添加紧急代理。添加后,紧急代理将获得超级管理员权限,允许他们执行DARC协议中的任何操作,而不受插件的限制。

2.callEmergency(emergencyAgentAddress):该命令用于通过指定要呼叫的紧急代理的地址来调用紧急代理。在调用后,紧急代理可以采取必要的紧急措施来处理不可预见的情况并执行操作以确保 DARC 的正常运行。

3. endEmergency():该命令用于结束紧急状态。一旦紧急情况得到解决或处理,用户和紧急代理可以使用此命令结束紧急状态并恢复正常的 DARC 协议操作。

11 可升级性

一旦编译后的智能合约二进制文件部署在与 EVM 兼容的区块链上,它就变得不可变,并且不可能对二进制文件进行任何修改。 OpenZeppelin [Ope] [ethc]利用基于代理的方法进行更新。具体方法包括部署代理智能合约以及所有实施智能合约并设置管理地址。当需要更新执行合约时,管理员可以直接修改代理,将代理重定向到新的执行合约地址。

对于 DARC 协议,代理升级模式不适用,因为它需要在代理中设置管理员。 DARC协议中的管理员将拥有完全的控制和修改权,有可能覆盖所有现有的资产、插件和信息,从而导致管理员的权力超越所有级别的代币持有者。在公司已经建立了公司架构的情况下,拥有一个有权修改所有法律方面、处理所有公司股份和管理资产的超级管理人将是一个危险的提议。

程序入口作为DARC协议的统一、专属的入口点。
从旧 DARC 实例 X 升级到新 DARC 实例 Y 时,该过程包括配置旧 DARC 实例 X 的升级地址以及设置新 DARC 实例 Y 以接受来自 DARC 实例 X 的所有委托程序。升级后,用户可以无缝地继续执行DARC实例X中的程序,并且这些程序将被委托并在新的DARC实例Y中执行。

在DARC协议中,有3个与升级相关的操作:

- UpgradeToAddress(targetAddress):将当前DARC 实例升级到targetAddress。所有提交到当前 DARC 实例的程序都将在 targetAddress 的 DARC 实例中委托并执行。

- informUpgradedFromAddress(sourceAddress):允许当前DARC 实例从sourceAddress 处的DARC 实例升级。当一个程序被代理并从sourceAddress提交时,它被允许以程序提交者的操作员身份在当前DARC实例中运行。

- UpgradeToTheLatest():如果当前DARC实例A已升级为新的DARC实例B,并且新的DARC实例B也已升级为更新的DARC实例C,则执行该函数将直接将DARC实例升级为DARC实例C. 也就是说,在此 DARC 实例 A 中执行的程序将被直接委托并在 DARC 实例 C 中执行,绕过 DARC 实例 B。此操作将有不同的解释。

  假设DARC实例A已经升级到DARC实例B,在这种情况下,如果用户在A中执行程序,只进行一次upgradeToTheLatest()操作,则该程序将在DARC实例A中执行,而不是委托给DARC实例B。 DARC 实例 B.

通过上述三个操作,我们可以讨论升级DARC协议的两种场景:

在第一种情况下,用户已经部署了 My DARC 1,并且 My DARC 1 尚未升级到任何其他 DARC,则需要执行以下三个步骤:

1. 将新版本的My DARC 2部署到区块链并完成所有配置。

2.在My DARC 2中执行confirmUpgradedFromAddress(MyDARC1Addr),允许My DARC 2
接受程序并充当 My DARC 1 的代理。

3. 在My DARC 1中执行upgradeToAddress(MyDARC2Addr),完成从My DARC的升级
DARC 1 到我的 DARC 2。

图10说明了从 My DARC 1 直接升级到 My DARC 2 的整个过程。
在第二种场景中,用户已经部署了My DARC 1,并且My DARC 1已经升级到My DARC 2,并且用户打算进一步升级到My DARC 3,需要执行以下三个步骤:

1. 在区块链上部署新版本的My DARC 3并完成所有配置。

2.在My DARC 3中执行confirmUpgradedFromAddress(MyDARC1Addr),启用My DARC 3
接受来自 My DARC 1 的程序和代理。

3.在My DARC 1中执行upgradeToAddress(MyDARC2Addr)。由于该程序将在My DARC 1的代理下在My DARC 2中运行,因此My DARC 2的升级地址将指向My DARC 3的地址。

4. 在My DARC 1中执行upgradeToTheLatest(),根据My DARC 2的地址将My DARC 1定向到My DARC 3的地址,完成从My DARC 1到My DARC 3的升级。

5. 最后,关闭 My DARC 2。

图11说明了从 My DARC 1 直接升级到 My DARC 3 的整个过程。

12 讨论

DARC 包协议在商业和加密世界中的未来方向和应用对于重塑组织结构和治理机制具有巨大潜力。随着技术的不断发展,几个关键领域成为 DARC 进步和应用的焦点。

首先,在商业领域,DARC 协议提供了建立更强大、更透明的企业实体的机会。通过利用 DARC 的自我调节和可编程特性,企业可以简化其治理流程、增强合规性并确保长期可持续性。 DARC 通过其插件执行严格的规则和法规的能力可以导致创建更负责任和更高效的公司结构,类似于传统的股份公司,但具有基于区块链的治理的额外好处。

此外,章程脚本的灵活性为创建多样化的公司结构打开了大门,包括 A/B 股、有限责任公司、C 型企业、非营利基金会等。这种适应性使 DARC 成为一个多功能平台,用于设计和实施各种形式的组织,满足不同的商业模式和行业。因此,DARC 有潜力成为各种商业实体的基础框架,为公司治理和运营提供新的范例。

在加密世界中,DARC协议的潜在应用同样引人注目。随着去中心化自治组织 (DAO) 不断受到关注,DARC 作为受监管且适应性强的替代方案脱颖而出。它支持多代币系统、股息分配和沙盒执行的能力使其非常适合管理加密资产和促进代币化治理。这使得 DARC 成为传统公司结构和自治 DAO 之间的桥梁,为去中心化组织提供受监管和可编程的基础。

展望未来,DARC 与其他区块链基础设施和智能合约编程语言(例如 Rust、Move 和 Plutus)的集成为进一步优化和扩展提供了令人兴奋的途径。这可能会导致 DARC 协议开发出更高效、更通用的实现,从而扩展其在不同区块链生态系统中的影响范围和适用性。

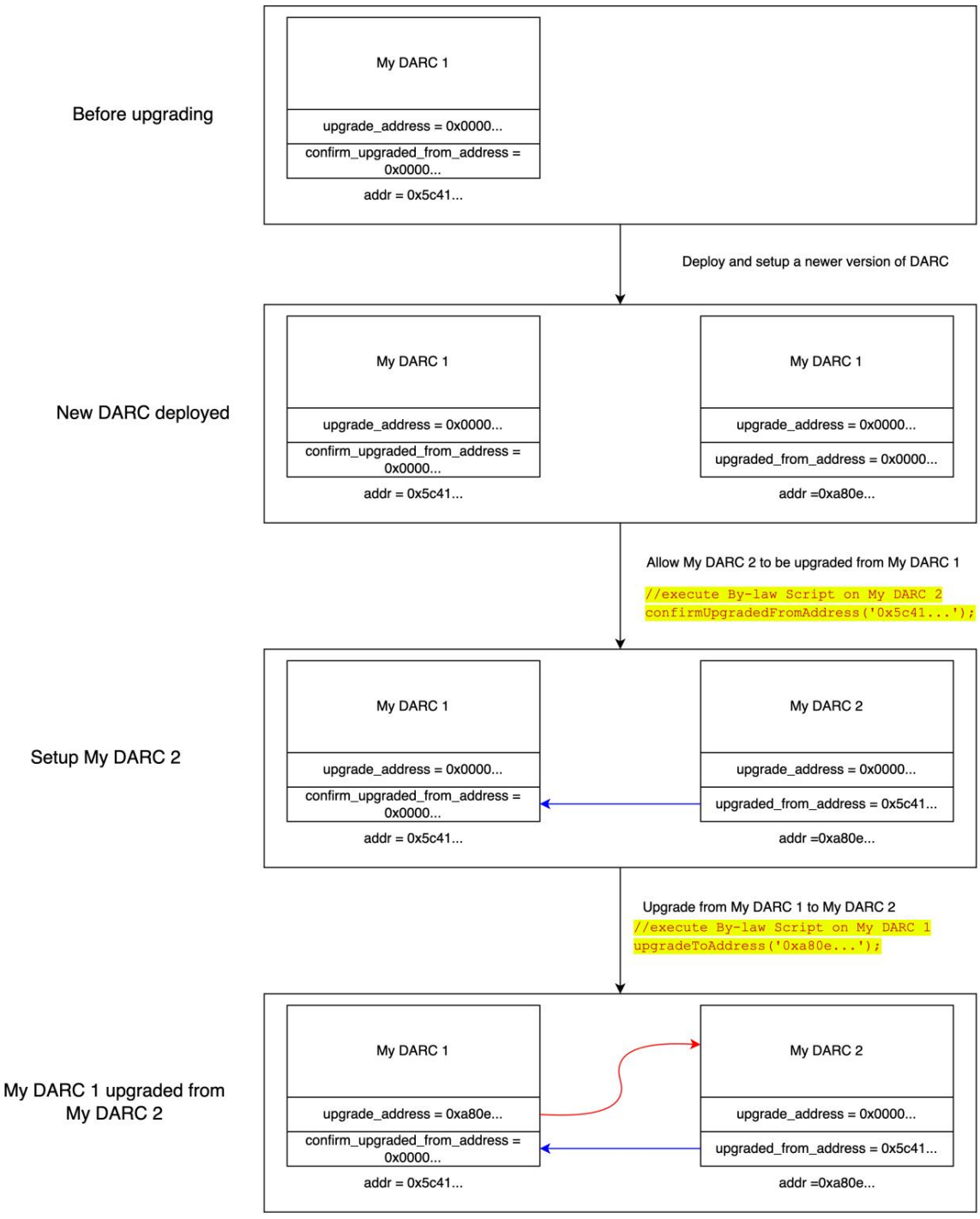


图 10:从 My DARC 1 升级到 My DARC 2

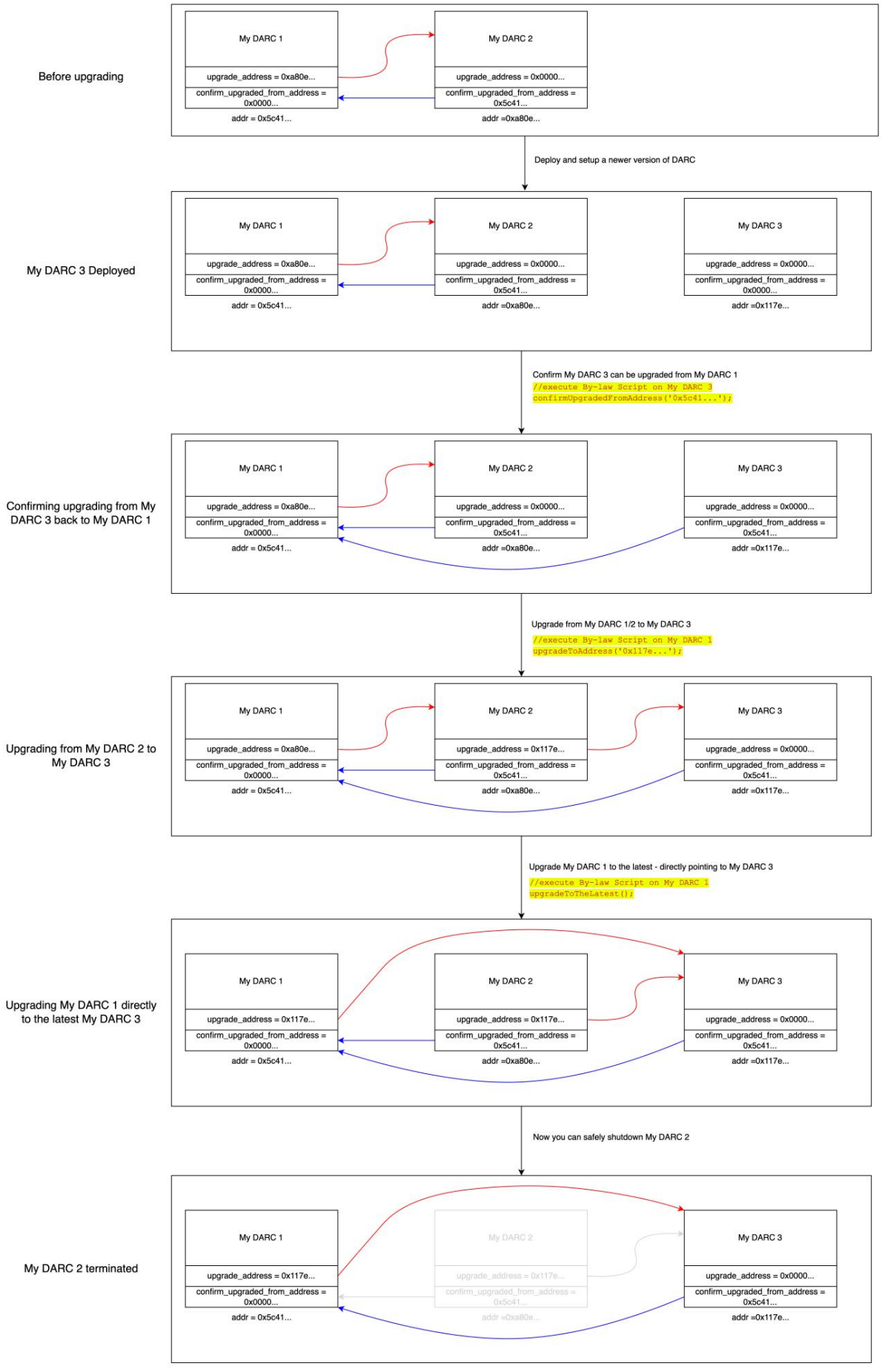


图 11:从 My DARC 1 升级到 My DARC 2 再到 My DARC 3

总之,DARC 协议的未来有望彻底改变公司治理、业务运营和去中心化组织结构。其建立受监管、自治公司的潜力及其对多元化公司结构的适应性,使 DARC 成为商业和加密世界的变革力量,提供了一种新的治理、合规性和可持续性方法。

附录1:指令操作码参考设计

```
/*
* @notice操作码枚举用于表示DARC协议的指令。 */


枚举枚举操作码{

/** *
  @notice 无效操作
  * 编号:0
  */
不明确的，

/** *
  @notice 批量铸造代币操作 * @param ADDRESS_2DARRAY[0] address[]
  toAddressArray: 铸造新代币的地址数组

  * @param UINT256_2DARRAY[0] uint256[] tokenClassArray:令牌类的数组
从中铸造新代币的索引
  * @param UINT256_2DARRAY[1] uint256[] amountArray:要铸造的代币数量的数组
  *
  * 身份证号码:1
  */
BATCH_MINT_TOKENS，

/** *
  @notice批量创建Token类操作 * @param STRING_ARRAY[] string[] nameArray:
  Token类名称数组
创造
  * @param UINT256_2DARRAY[0] uint256[] tokenIndexArray:令牌索引数组
要创建的令牌类的
  * @param UINT256_2DARRAY[1] uint256[] VotingWeightArray:投票权重数组
要创建的令牌类的
  * @param UINT256_2DARRAY[2] uint256[]dividerWeightArray: 被除数数组
要创建的令牌类别的权重
  *
  * 编号:2
  */
BATCH_CREATE_TOKEN_CLASSES，

/** *
  @notice批量转账Token操作 * @param ADDRESS_2DARRAY[0]
  address[] toAddressArray:要转账的地址数组
令牌到
  * @param UINT256_2DARRAY[0] uint256[] tokenClassArray:令牌类的数组
从中转移代币的索引
  * @param UINT256_2DARRAY[1] uint256[] amountArray:要转账的代币数量数组
```

```
     *
     * 编号:3
     */
    BATCH_TRANSFER_TOKENS，

    /** *
      @notice 从地址 A 批量传输令牌到地址 B 操作 * @param ADDRESS_2DARRAY[0] address[] fromAddressArray：
      要发送的地址数组
转移代币从
     * @param ADDRESS_2DARRAY[1] address[] toAddressArray:要传输的地址数组
令牌到
     * @param UINT256_2DARRAY[0] uint256[] tokenClassArray:令牌类的数组
从中转移代币的索引
     * @param UINT256_2DARRAY[1] uint256[] amountArray:要转账的代币数量数组

     *
     * 编号:4
     */
    BATCH_TRANSFER_TOKENS_FROM_TO，

    /** *
      @notice 批量销毁代币操作 * @param UINT256_2DARRAY[0] uint256[]
      tokenClassArray:要销毁代币的代币类别索引数组

     * @param UINT256_2DARRAY[1] uint256[] amountArray: 销毁代币数量数组

     *
     * 编号:5
     */
    BATCH_BURN_TOKENS，

    /** *
      @notice Batch Burn Token From Addr A Operation * @param ADDRESS_2DARRAY[0] address[]
      fromAddressArray:烧录地址数组

     * @param UINT256_2DARRAY[0] uint256[] tokenClassArray:要烧录代币的代币类别索引数组

     * @param UINT256_2DARRAY[1] uint256[] amountArray: 销毁代币数量数组

     *
     * 编号:6
     */
    BATCH_BURN_TOKENS_FROM，

    /** *
      @notice批量添加成员操作 * @param ADDRESS_2DARRAY[0]
      address[] memberAddressArray:地址数组
添加为成员
     * @param UINT256_2DARRAY[0] uint256[] memberRoleArray:要添加的成员的角色数组

     * @param STRING_ARRAY string[] memberNameArray:要添加的成员名称的数组

     *
     * 编号:7
     */
    BATCH_ADD_MEMBERSHIP，
```

```
    /** *
      @notice 批量挂起成员操作 * @param ADDRESS_2DARRAY[0] address[]
      memberAddressArray: 地址数组
暂停会员资格
      *
      * 编号:8
      */
    BATCH_SUSPEND_MEMBERSHIP，

    /** *
      @notice 批量恢复成员操作 * @param ADDRESS_2DARRAY[0]
      address[] memberAddressArray: 地址数组
恢复会员身份
      *
      * ID:9 */

    BATCH_RESUME_MEMBERSHIP，

    /** *
      @notice批量更改成员角色操作 * @param ADDRESS_2DARRAY[0] address[]
      memberAddressArray:地址数组
改变成员的角色
      * @param UINT256_2DARRAY[0] uint256[] memberRoleArray:要更改的成员角色的数组

      *
      * 编号:10
      */
    BATCH_CHANGE_MEMBER_ROLES，

    /** *
      @notice批量更改成员名称操作 * @param ADDRESS_2DARRAY[0] address[]
      memberAddressArray:地址的数组
更改会员姓名
      * @param STRING_ARRAY string[] memberNameArray:要更改的成员名称的数组

      *
      * 编号:11
      */
    BATCH_CHANGE_MEMBER_NAMES，

    /** *
      @notice 批量添加紧急代理操作 * @param Plugin[] pluginList: 插件数组

      * 编号:12
      */
    BATCH_ADD_PLUGINS，

    /** *
      @notice 批量启用插件操作 * @param UINT256_ARRAY[0] uint256[]
      pluginIndexArray: 插件索引数组
启用
      * @param BOOL_ARRAY bool[] isBeforeOperationArray: 插件是否在操作之前的标志数组

      * 编号:13
      */
```

BATCH_ENABLE_PLUGINS，

/** *
@notice 批量禁用插件操作 * @param UINT256_ARRAY[0] uint256[]
pluginIndexArray: 插件索引数组
禁用
* @param BOOL_ARRAY bool[] isBeforeOperationArray: 插件是否在操作之前的标志数组

* 编号:14
*/
BATCH_DISABLE_PLUGINS，

/** *
@notice 批量添加并启用插件操作 * @param Plugin[] pluginList: 插件数组

* ID:15 */

BATCH_ADD_AND_ENABLE_PLUGINS，

/** *
@notice 批量设置参数操作 * @param MachineParameter[]parameterNameArray：
参数名称数组 * @param UINT256_2DARRAY[0] uint256[]parameterValueArray:参数值数组

* 编号:16
*/
BATCH_SET_PARAMETERS，

/** *
@notice 批量添加提现余额操作 * @param address[] addressArray:添加提现余额地址数组 *
@param uint256[] amountArray:添加提现余额金额数组

* ID:17 */

BATCH_ADD_WITHDRAWABLE_BALANCES，

/** *
@notice批量减少可提现余额操作 * @param address[] addressArray:要减去可提现余额的地址数组

* @param uint256[] amountArray: 提取余额金额数组
* 编号:18
*/
BATCH_REDUCE_WITHDRAWABLE_BALANCES，

/** *
@notice 批量添加投票规则 * @param VotingRule[]
VotingRuleList: 投票规则数组
* 编号:19
*/
BATCH_ADD_VOTING_RULES，

/** *
@notice 批量支付铸造代币操作 * @param ADDRESS_2DARRAY[0] address[]
addressArray: 铸造地址数组

代币

* @param UINT256_2DARRAY[0] uint256[] tokenClassArray:令牌类的数组
铸造代币的索引

* @param UINT256_2DARRAY[1] uint256[] amountArray:铸造代币数量数组

* @param UINT256_2DARRAY[2] uint256[]priceArray:要铸造的每个代币类别的价格

* @param UINT256_2DARRAY[3] uint256[1]dividableFlag:指示是否
付款可分红。 1 表示是（付费购买），0 表示否（付费投资）
* 编号:20

*/

BATCH_PAY_TO_MINT_TOKENS，

/** *

@notice 支付一些现金转账代币（可作为产品币） * @param ADDRESS_2DARRAY[0] address[] toAddressArray: 转账地址数组

令牌到

* @param UINT256_2DARRAY[0] uint256[] tokenClassArray:令牌类的数组
从中转移代币的索引

* @param UINT256_2DARRAY[1] uint256[] amountArray:代币金额数组
转移

* @param UINT256_2DARRAY[2] uint256[]priceArray:每个代币类别的价格
转移

* @param UINT256_2DARRAY[3] uint256[1]dividableFlag:指示是否
付款可分红。 1 表示是（付费购买），0 表示否（付费投资）
* 编号:21

*/

BATCH_PAY_TO_TRANSFER_TOKENS，

/** *

@notice 添加地址数组作为紧急代理 *（可用作具有新的唯一代币类别的产品 NFT） * @param

ADDRESS_2DARRAY[0] address[] 要添加为紧急代理的地址数组

* ID:22 */

添加紧急情况，

/**

* @notice 从合约现金余额中提取现金

* @param address[] addressArray:提现地址数组 * @param uint256[] amountArray:提现金额数组

* 编号:23

*/

提款_现金_至，

/** *

@notice 调用紧急代理处理紧急情况 * @param UINT256_2DARRAY[0] address[] addressArray:要调用的紧急代理索引数组

* 编号:24

*/

紧急呼叫，

/**

* @notice 使用给定的 abi 调用合约 * @param addresscontractAddress:要调用的合约
的地址 * @param bytes abi:要调用的函数的 abi

* 编号:25
*/
CALL_CONTRACT_ABI，

/** *
@notice 支付一些现金 * @param uint256
amount:支付的现金金额 * @param uint256 paymentType:支付的现金类型,0为ethers/matic/original
tokens

* 1 代表 USDT,2 代表 USDC（目前仅支持 0）,3 代表 DAI ... * @param uint256 可分红:指示付款是否可分红的标志,* 0 表示否（支付投资）,1是
的（付费购买）

* ID:26 */

PAY_CASH,

/**
* @notice 计算股息并提供给代币持有者
* 将股息添加到每个代币持有者的可提取余额中
*
* 编号:27
*/
OFFER_DIVIDENDS，

/**
* @notice 从可提取股息余额中提取股息
* @param address[] addressArray: 提取股息地址数组 * @param uint256[] amountArray: 提取股息金额数组

* 编号:28
*/
撤回_DIVIDENDS_TO，

/** *
@notice 通过地址设置所有转账操作的审批 * @paran 地址:设置所有转账操作审批的地址

* ID:29 */

SET_APPROVAL_FOR_ALL_OPERATIONS，

/**
* @notice 批量销毁代币并退款
* @param UINT256_2D[0] uint256[] tokenClassArray:令牌类别索引的数组
烧毁代币
* @param UINT256_2D[1] uint256[] amountArray: 销毁代币数量数组

* @param UINT256_2D[2] uint256[]priceArray:要销毁的每个代币类别的价格
* 编号:30
*/
BATCH_BURN_TOKENS_AND_REFUND，

/** *
@notice 将存储 IPFS 哈希永久添加到存储列表中

```
    * @paran STRING_2DARRAY[0] 地址:设置所有提现操作批准的地址

    * 编号:31
    */
ADD_STORAGE_IPFS_HASH，


/**
    * 以下是投票等待过程中可以使用的两个操作 */


/** *
    @notice 对待投票的程序进行投票 * @param bool[] voteArray: 每个程序的投票
    数组
    * ID:32 */

投票，

/** *
    @notice 执行已投票通过的程序
    * 编号:33
    */
执行程序，

/** *
    @notice 紧急模式终止。此后紧急人员无法采取任何行动
手术
    * 编号:34
    */
END_紧急情况，

/** *
    @notice 升级合约到新合约地址 * @param ADDRESS_2DARRAY[0][0] 新合约地址

    * ID:35 */

升级到地址，

/** *
    @notice 接受旧合约地址升级当前DARC * @param ADDRESS_2DARRAY[0][0] 旧合约地址

    * 编号:36
    */
CONFIRM_UPGRAED_FROM_ADDRESS，

/** *
    @notice 将合约升级到最新版本
    * 编号:37
    */
升级到最新版本，

/** *
    @notice 批量支付 Trasnfer 代币操作
    * 编号:38
    */
op_BATCH_PAY_TO_TRANSFER_TOKENS
```

}

附录2:程序及操作参考设计

/**
* 操作的参数或操作数 */ struct Param { uint256[] UINT256_ARRAY;地址[]

ADDRESS_ARRAY;字符
串[] STRING_ARRAY;布尔[] BOOL_ARRAY;

投票规则[] VOTING_RULE_ARRAY;
插件[] PLUGIN_ARRAY;
MachineParameter[] PARAMETER_ARRAY;
uint256[][] UINT256_2DARRAY;地址[][]
ADDRESS_2DARRAY;
}

/**
* 要执行的操作,包括操作符地址、操作码和参数 */ struct Operation { address operatorAddress;

EnumOpcode 操作码;
参数 参数;
}

/**
* 要执行的程序,包括操作符地址和操作数组 */ struct Program {

地址程序操作符地址;

/** *
@notice actions:要执行的操作数组 */

操作[]操作;
}

附录3:插件参考设计

/**
* 条件节点类型 */

枚举 EnumConditionNodeType { UNDEFINED, EXPRESSION, LOGICAL_OPERATOR, BOOLEAN_TRUE, BOOLEAN_FALSE}

/**
* 逻辑运算符类型 */

枚举 EnumLogicalOperatorType {UNDEFINED、AND、OR、NOT }

枚举 EnumReturnType {

/** *
默认值。如果没有插件被触发,插件系统将返回UNDEFINED。
* BEFORE 和 AFTER 操作插件系统都可能返回 UNDEFINED。 */ 不明确的，

/** *
操作已批准,但必须在沙箱中执行,以检查操作 * 在当前机器状态下是否有效。

* 只有在操作插件系统之前才可能返回SANDBOX_NEEDED。 */

需要沙箱，

/** *
操作未获批准,应在此级别拒绝。
* BEFORE 和 AFTER 操作插件系统都可能返回 NO。 */ 不，

/** *
决定待决,应在此级别创建投票项。
* 只有在操作插件系统之后才可能返回 VOTING_NEEDED。 */ 需要投票，

/** *
操作已获批准,应跳过沙箱检查。
* 只有BEFORE操作插件系统可能返回YES_AND_SKIP_SANDBOX。 */

YES_AND_SKIP_SANDBOX，

/** *
操作最终在此级别获得批准。
* 只有在操作插件系统之后才可能返回YES。 */ 是的

}

/**
* 条件节点表达式参数 */ struct NodeParam { uint256[]

UINT256_ARRAY;地址[]
ADDRESS_ARRAY;字符串[]
STRING_ARRAY； uint256[][]
UINT256_2DARRAY;地址[][]
ADDRESS_2DARRAY;字符串[][]
STRING_2DARRAY；

}

/**
* 条件节点结构体

```
    */
结构条件节点 { /**

      * 当前条件节点索引
      */
    uint256 id;

    /** *
      当前条件节点的类型 */

    EnumConditionNodeType 节点类型；

    /** *
      当前条件节点的逻辑运算符 */

    EnumLogicalOperatorType 逻辑运算符；

    /** *
      当前条件节点的条件表达式 */

    EnumConditionExpression 条件表达式；

    /**
      * 当前条件节点的子节点列表
      */
    uint256[] 子列表；

    /**
      * EXPRESSION节点参数数组*/

    NodeParam 参数；
}

/**
    * 插件的struct */ struct Plugin { /** * 当前条件
    节
点的返回类型 */



    EnumReturnType 返回类型；

    /** *
      限制级别,从0到uint256的最大值 */ uint256 level;



    /** *
      条件二元表达式树向量 */

    ConditionNode[] 条件节点；

    /** *
      如果返回类型为VOTING_NEEDED,则为当前插件的投票规则id */

    uint256 投票规则索引；
```

```
    /** *
      插件注释 */ string note;



    /** *
      指示插件是否启用的布尔值 */

    布尔 bIsEnabled;

    /**
      * 表示插件是否被删除的布尔值 */

    bool bIs已初始化;

    /** *
      布尔值,表示插件是前操作插件还是后操作插件 */


    bool bIsBeforeOperation;

}
```