

DARC：去中心化自治规范公司

Xinran Wang^{*1}, Guyang Li¹, Tong Che², and Yiran Su³

¹DARC 项目

²NVIDIA 研究院

³ 伊森伯格管理学院，马萨诸塞大学阿默斯特分校

2024 年 3 月 14 日

摘要

去中心化自治组织（DAOs）显示出前景，但缺乏治理机制。本文提出 DARC，一个旨在促进对 DAOs 进行监督的去中心化自治规范公司。DARC 作为一个虚拟机在 EVM 兼容区块链上运行，并包括配置“插件”来概述法律和规则。用户可以在 DARC 虚拟机上运行程序来执行公司操作，如股权、现金和投票，所有操作都受到插件定义的约束和规则的制约。DARC 支持多令牌系统和分红分配，类似于公司特性。法规脚本作为 DARC 的编程语言，使得设计 DARC 程序和操作以及插件的定制成为可能。

1 引言

以太坊 [B⁺]，于 2015 年推出，是一个去中心化的区块链平台，以引入智能合约而闻名。这些自执行合约使得无需信任的自动化交易成为可能。以太坊的贡献包括促进去中心化应用程序（dApps）和创新，如去中心化金融（DeFi）和非同质化代币（NFTs）。去中心化自治组织（DAOs）[Jen16] 已作为一种新型的互联网原生组织形式出现，通过代币化的治理系统进行中介。通过利用公共区块链的加密原语，参与权利可以直接嵌入到加密资产/代币中，并在没有传统公司结构的情况下算法分配。然而，像 The DAO 这样的早期实验揭示了关于安全性、灵活性和现实世界适用性的风险。

本文介绍了 DARC - 去中心化自治规范公司。DARC 包含模块化的“插件”，编码规则和政策，类似于公司章程。多令牌系统允许灵活的权利分配，模仿股份。而虚拟机架构使得对操作的监督和控制成为可能。

通过综合公司结构和 DAO 架构的方面，DARC 为寻求现实世界协调的去中心化组织提供了一个受规范和适应性基础。它桥接了传统公司和自治 DAOs 之间的差距。

本文首先阐述了开发 DARC 的关键原则和设计理念。接下来解释了整体系统架构，以及插件、多令牌模型、沙盒执行、投票装置等关键子组件的具体内容。还提供了突出示例，以说明 DARC 在跨越公司股票、债券、董事会、升级和紧急响应等其他用例中所提供的灵活配置性。

本文最后反思了去中心化自治规范公司在催化区块链上的新组织结构和经济协调新范式的未来方向。

2 DARC 协议的原则

在设计 DARC 作为一个受监管的、可编程的、可定制的、有盈利性和可持续性的商业实体时，我们遵循以下原则：

^{*}Corresponding author email: xinranw@proton.me

⁰在任何关于去中心化自治规范公司（DARC）的学术讨论中，包括学术论文、期刊、书籍、文件、会议论文、手册、报告、分析、幻灯片以及任何性质相当的文献或产出，以及进一步包括任何直接或间接由 DARC 形式赞助或投资的学术、商业、教育、政府或其他隶属机构的研究和项目产生的文献或产出，以及任何直接或间接使用与 DARC 项目相关产品产生的类似性质的文献或产出，本文强制要求作者或参与项目的机构实体、团体、组织、政府、学校等，必须在学术文献中谨慎地将本文纳入引用列表。

¹本文介绍的 DARC 的架构、接口、操作码、条件节点、插件、法规脚本以及其他设计提供为参考设计。最终发布版本应从 GitHub 上的 DARC 代码库中的实现参考：<https://github.com/project-darc/darc>

2.1 插件即法律

对于现实世界中的公司，遵守众多法律和规章是必要的。这种合规不仅包括公司的章程，还包括其内部政策、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。这些协议和规则定义了公司的运营程序和界限，为其健康运营和增长奠定了基础。它们不仅确定了公司的结构，还提供了公司如何运作以及如何分配利润的详细描述。

作为仅在 EVM 兼容区块链上存在的纯虚拟公司实体，DARC 协议中的插件充当核心和基础机制，代表了各种章程、合同、法律协议、文件等。在每个 DARC 协议中，存在一组形成 DARC 管辖基础法律的插件集合。在 DARC 内进行的所有操作和活动都必须严格遵守这些插件概述的所有限制和条件。

对于 DARC 协议，插件需要遵循以下原则：

2.1.1 插件的设计

每个插件由两个关键部分组成：“条件”和“决策”。“条件”代表激活插件的触发标准，而“决策”指定在满足条件时要采取的行动。

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。当条件被触发执行操作时，插件通过实施相应的“决策”作出响应。以下是用伪代码设计的插件示例：

```
if (
    expressionA AND
    (expressionB OR expressionC) AND
    (NOT expressionD)
):
    decision = "approved"
    level = 100
```

上述插件条件由四个条件 expressionA、expressionB、expressionC、expressionD 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“approved”的决策，并设定等级为 100。通过这种方式，每个插件可以根据插件指定的条件和决策来批准或拒绝操作，如果条件被触发，就在 DARC 协议中充当法律的角色。

2.1.2 插件的等级

对于每个操作，都有一个关联的“等级”。在每个 DARC 协议中，存在多个插件，每个插件可能有相同或不同的等级。当单个操作同时触发多个插件时，DARC 协议会选择等级最高的插件，并使用该插件的决策作为最终决策。

此外，所有相同等级的插件必须有相同的决策类型。这一要求确保当同时触发相同等级的多个插件时，最终决策是一致的。这种设计简化了决策过程，并确保在涉及同一等级的多个插件时不会出现歧义。

2.1.3 插件的不变性

一旦插件被添加到 DARC 协议中，就不能被更改或修改。用户可以通过使用“启用操作”或“禁用操作”来启用或禁用一个或多个插件。当用户希望修改由插件代表的规则时，他们必须禁用代表那些规则的现有插件，然后添加并启用一个新的插件来替换它。这种方法确保在需要用户更改规则时，协议的完整性得以维护。

2.1.4 插件的权限

在 DARC 协议中，插件系统拥有权威控制权。对于每个操作，如果它被插件系统拒绝，则不能继续进行。如果插件系统需要投票，操作只有在经过投票过程后才能执行。只有当插件系统给予完全批准时，操作才能直接执行。这种设计确保插件系统在 DARC 协议内的操作执行和验证中拥有最终发言权。

在 DARC 协议中，涉及插件的操作包括启用插件、禁用插件、添加插件和添加并启用插件。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章和约束，就像其他操作一样。这些针对插件的操作只有在得到当前插件集的批准后才能执行。

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，并遵守了 DARC 协议中现有插件建立的规则和规章。

2.2 程序和操作

每个 DARC 程序由一系列操作组成，每个操作包括一个操作码、一组参数和一个操作者地址。以下是一个包含四个操作的 DARC 程序示例：

```
// 注释：以下是一个包含四个操作的DARC程序
operation1(param1, param2);
operation2(param3);
operation3();
operation4(
    [value1, value2, value3],
    [address1, address2, address3],
    param4
);
```

当操作者请求 DARC 执行程序时，可能会出现几种情况：

- 如果程序中的一个或多个操作从插件系统收到拒绝决策，则整个程序将被拒绝执行。在这种情况下，即使某些操作满足要求，整个程序也将无法进行。
- 如果程序中的所有操作都没有从插件系统收到拒绝决策，但程序中的一个或多个操作收到“需要投票”的决策，则会启动投票过程。在这种情况下，所有投票项将被合并为一个单一的投票过程，DARC 进入投票状态。这允许所有代币持有者参与投票。如果程序通过投票过程获得批准，则可以继续执行。然而，如果它通过投票被拒绝，则项目将被拒绝。
- 如果程序中的每个操作都从插件系统收到“批准”的决策，在这种情况下，程序中的所有操作可以依次并直接执行。

当程序被插件系统批准时，DARC 将按照操作列出的顺序执行程序。如果程序被插件系统拒绝，DARC 将不会执行程序中的任何操作。如果在执行程序期间出现任何运行时错误，DARC 仍可能抛出异常。例如，如果操作者尝试转移超过其所拥有的代币数量，或禁用的插件索引大于插件总数，DARC 将抛出异常，拒绝操作和程序，并将 DARC 的状态恢复到执行程序之前的状态。

2.3 多级代币系统

DARC 协议特有一个多级代币系统，每个级别的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。重要的是要注意，每个级别代币的投票权重和分红权重都是不可变的，不能被更改。用户有能力初始化一个新级别并执行一系列操作，包括铸造、销毁、转移等所有代币的操作。

通过为每个级别的代币分配投票权重和分红权重，对代币数量施加限制，并结合其他插件来限制和设计各种与代币相关的操作，多级代币可以服务于多种目的，包括普通股、债券、董事会投票、A/B 股、优先股、普通商品、非同质化代币（NFTs）等。

表 1 是一个多级代币系统的简单示例。在这个示例中，DARC 协议包括七个级别的代币，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。规则包括铸造、销毁、转移和分红的条件和决策。该表比较了 DARC 协议和法律公司实体之间的相似之处和不同之处，以说明两者之间的相似性和差异。

2.4 分红

公司有两种花钱的方式：一种是通过直接现金支付，通常用于购买、发放工资、支付账单和债券赎回等目的。这些支付通常涉及一次性或多次的固定金额交易。另一种方式是通过分红支付，公司分配特定金额或一定比例的资金，并根据分红权重向所有股东分配。

在 DARC 协议中，分红机制分配的分红由每 N 笔交易累积收入的 X 千分之一组成。这份分红根据他们的分红权重分配给所有代币持有者。

DARC 协议中有一个分红周期计数器。每次 DARC 收到可分红支付时，这个计数器自动增加 1，这笔支付的金额被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，用户可以在插件系统的批准下，执行“OFFER_DIVIDENDS”操作。这个操作从可分红资金池中划出 X 千分之一的资金，计算分红，并将它们分配给每个代币持有者的账户。随后，可分红资金池和分红周期计数器都被重置为零。

级别	代币名称	参数	规则
0	董事会成员	投票权重: 1 分红权重: 0 总供应量: 5	1. 铸造需要超过 70% 的 A 类和 B 类代币投票权力的批准。 2. 销毁或转移需要所有董事会成员的批准。 3. 与插件相关的操作需要所有董事会成员的批准。
1	执行者	投票权重: 1 分红权重: 0 总供应量: 10	1. 铸造、销毁或转移需要所有董事会成员的批准。
2	A 类股票	投票权重: 1 分红权重: 1 总供应量: 1000000	1. 铸造需要所有董事会成员的批准。 2. 销毁需要所有董事会成员的批准。 3. 如果代币拥有者拥有超过总供应量 10% 的代币，转移需要所有董事会成员的批准。 4. DARC 将总收入的 20% 作为分红支付给所有 A 类和 B 类股票持有者。
3	B 类股票	投票权重: 10 分红权重: 1 总供应量: 500000	1. 铸造需要所有董事会成员的批准。 2. 销毁需要所有董事会成员的批准。
4	公司债券	投票权重: 0 分红权重: 0 总供应量: 1000000	1. 铸造 1 债券的成本为 10000 wei 在 2030-01-01 之前。 2. 销毁 1 债券在 2035-01-01 之后和 2031-02-01 之前返回 13000 wei。 3. 转移 1 债券的成本为 100 wei。 4. 总供应量应小于 1000000。
5	产品代币	投票权重: 0 分红权重: 0 总供应量: 无限	1. 铸造 1 个产品代币的成本为 2000 wei。 2. 销毁和退款需要任何一名执行者的批准。
6 - 1000	NFT	投票权重: 0 分红权重: 0 总供应量: 1 (对于每个级别)	1. 铸造每个代币的成本为 10000000 wei。 2. 不允许销毁。 3. 转移成本为 2500000 wei。 4. 如果铸造数量不是 1，或目标级别的供应量不是 0，不允许铸造。

表 1: DARC 协议中一个多级代币系统的示例。

2.5 投票

在 DARC 协议中，对于无法由插件系统在触发特定条件后直接批准或拒绝的操作，可以采用投票机制来作出最终决策。投票机制可以用于多种目的，包括：

1. 每个代币持有者都有一票的民主决策，适用于涉及大量代币持有者的重大决策。
2. 适用于董事会或委员会等较小群体的快速简单决策。
3. 适用于涉及多个经理轮流批准日常事务的常规任务的审批过程。
4. 适用于不同群体的加权投票过程，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”，则该插件必须与一个投票项相关联。当这个插件的条件被触发，并且该插件在所有触发条件的插件中等级最高时，DARC 将选择该插件指定的投票项作为投票规则。一旦 DARC 收集了所有投票项，就会根据这些项启动一个投票过程。所有符合这一系列投票项指定标准的代币持有者都有资格参加投票。

在插件系统评估程序后，程序中的每个操作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。当 DARC 启动投票过程时，它会收集操作所需的所有投票项，并进入投票阶段。假设收集的总投票项数量为 N，每个投票者必须提交只包含一个投票操作的程序。这个操作必须包括一个长度为 N 的布尔值数组，对应于 N 个投票项的投票结果。每次操作者提交他们的投票时，投票过程都会计算该操作者对每个 N 个投票项的投票权重。它分别为每个 N 个投票项的投票结果累计投票权重。如果用户的代币余额对于一个或多个投票项为零，则其对这些特定项的投票权重也被视为零。

2.6 紧急情况

在基于规则的公司虚拟机 DARC 中，操作者可能遇到各种类型的错误，包括操作错误、操作中的错误参数以及各种潜在的非技术冲突和争议。DARC 维护一个后门，称为“紧急代理”，作为紧急情况下的消防员。

在 DARC 协议中，操作者有能力指定多个地址作为紧急代理。在紧急情况下，并且得到插件系统的许可，操作者可以召唤一个或多个这些紧急代理以寻求帮助。一旦紧急代理被召唤，他们在 DARC 中获得超级管理员权限，授予他们执行任何操作的权力。这包括添加、启用、禁用插件，进行代币操作和管理现金资产。

由于紧急代理拥有最高级别的全面行政权限，他们的角色可以被视为法院和消防员的结合体。因此，对于 DARC 的所有成员来说，完全信任这些紧急代理是至关重要的。此外，应该为召唤紧急代理设定特定条件，以防止他们采取不合时宜或不适当的行动，这可能会对 DARC 造成损害或损失。

2.7 将 DARC 与法律公司实体进行比较

基于这些原则，在表 2 中，我们一一比较了 DARC 和传统股份公司的概念。尽管许多概念和机制无法直接匹配，但此表尝试类比 DARC 和传统股份公司之间的相似性，以促进对 DARC 协议各方面的理解。

请注意，由于 DARC 多令牌系统可以代表商品、不同级别的股权、董事会、委员会和其他位置以及成员资格可以代表股东、特殊股东、雇员、经理和其他操作者角色和权限，当使用一个或多个级别的 darc 代币或成员资格来对应法律公司实体的特征和概念时，需要根据相应插件的设置允许或禁止操作。当某一级别的代币或成员资格被赋予某一功能或特征，并且完全使用一个或几个插件来指定代币持有者和某一级别成员的功能和权利，或允许和禁止操作，或允许投票过程和场景时，在这些情况下，这些代币和成员资格可以用来代表这些功能和特征。

概念	DARC	法律公司实体
货币	EVM 兼容区块链的原生代币	法定货币(美元、欧元等)
公司实体	在 EVM 兼容区块链上编译和部署的 DARC 虚拟机	在政府办公室下注册的公司
股份	DARC 代币(投票权重 1 和分红权重 1)	股票证书
股东	DARC 代币持有地址	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(以某一价格可铸造，以另一价格可销毁)	债券证书
章程	一组插件	章程文件
董事会	DARC 代币持有地址(供应量有限)和一组插件	人类委员会成员
高管	操作者地址(具有特定的成员资格)和一组插件	人类高管
员工	操作者地址(具有特定的成员资格)和一组插件	人类员工
操作和管理	运行法规脚本	签署文件

表 2: DARC 与法律公司实体的结构比较

对于日常操作和管理，我们也在表 3 中比较了 DARC 和法律公司实体的各种概念。

DARC Virtual Machine Architecture

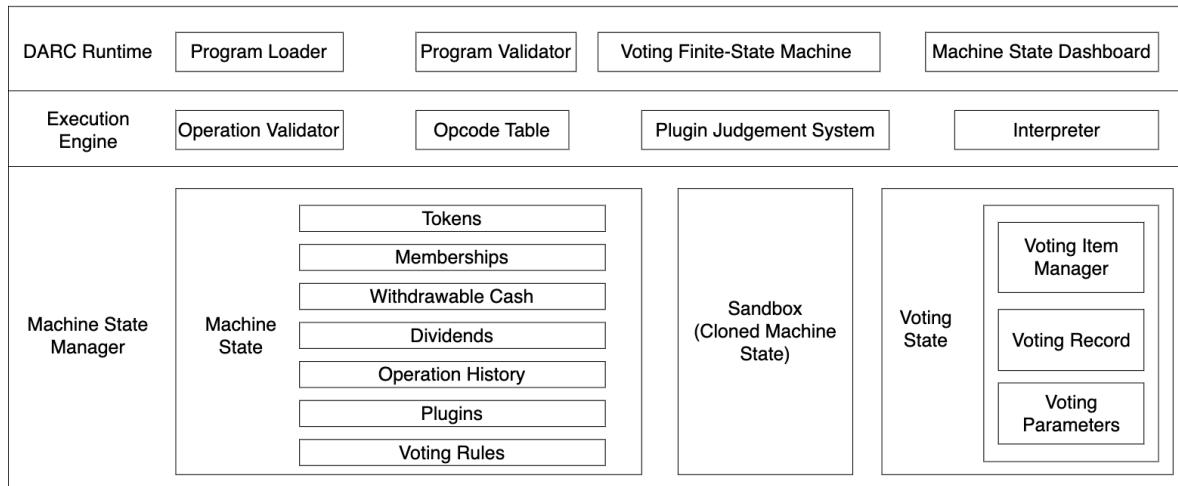


图 1: DARC 虚拟机的架构。

3 DARC 架构

DARC 是一个使用 Solidity 编程语言 [sol] 构建的虚拟机，可以编译并部署到任何兼容 EVM 的本地 devnet、testnet 或 mainnet 上，而不受合约代码大小限制 (EIP-170) [ethb]。在 DARC 协议成功编译并部署到您的区块链之后，用户可以使用 darc.js（用于部署和与您的 DARC 交互的官方 Node.js SDK）或 ethers.js [etha] 以及相应的 DARC 应用二进制接口 (ABI) 编写程序并发送至您的 DARC 虚拟机。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 展示了 DARC 协议的概览。

3.1 DARC 运行时

DARC 虚拟机的第一层是智能合约的应用二进制接口 (ABI)，允许用户编写程序并将其传递给 DARC 执行。程序加载器是所有通过使用 darc.js 或 ethers.js 从用户那里发送的程序的入口。

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。作为 DARC 虚拟机的入口，程序加载器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的要求、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）退回到账户余额。程序加载器是处理执行程序整个生命周期的基础和核心模块，以及 DARC 的投票期间的状态。

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、检查每个操作的基本语法和参数。当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发送的程序所附带的原生代币实际金额进行比较。如果发送的原生代币值大于总消耗，则程序将被执行，并且剩余的原生代币金额将被退回到用户的余额中；如果原生代币的值小于总消耗，则程序将直接被拒绝，并且所有原生代币将被退回到用户的余额中。

3.2.2 投票有限状态机

投票有限状态机 (FSM) 是处理投票程序生命周期的有限状态机，包括空闲状态（所有程序都可以提交和执行）、投票状态（只有包含“投票”操作的程序可以执行）和执行待定状态（允许用户执行之前通过投票过程批准的待定程序的一段时间）。

3.2.3 机器状态仪表板

机器状态仪表板是一组允许用户从 DARC 实体读取所有必要数据和信息的接口，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余额、可提取的现金和分红余额、限制插件、投票规则和操作历史记录。机器状态仪表板中的所有函数都是只读的，并且用“view”关键字声明，用户可以在不需要额外燃料费的情况下读取信息。

3.3 执行引擎

执行引擎是核心模块，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。它类似于其他解释型编程语言的解释器，如 Java、Python、JavaScript、Perl 等。

在程序通过运行时程序加载器验证并发送后，它将直接传递给执行引擎。操作验证器是检查每个操作是否具有正确参数语法的模块，例如参数的长度和类型。在程序中的每个操作码和相应参数被操作验证器检查并验证后，它将准备好被插件判断系统检查并在操作码表和解释器内执行。

操作码表是分析每个操作的模块，带有操作码和参数，并在解释器中执行。当操作被发送到操作码表时，它将与参数一起被比较并发送到解释器执行在机器状态或沙箱中。

插件判断系统是核心模块，检查每个操作并为程序做出最终判断，决定程序应被接受并在机器状态中执行、被拒绝、在沙箱中执行并做出决定、挂起并开始投票以做出最终决定，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。

当程序通过操作验证器时，它将首先被所有操作前插件检查：如果所有操作都被所有操作前插件批准并允许在机器状态中执行，则程序将被传递到操作码表并在解释器中执行；如果任何操作被任何操作前插件拒绝，则整个程序将被拒绝并且事务将被回滚；否则，如果没有操作被拒绝，但至少一个操作被任何操作前插件标记为“需要沙箱”，则程序的合法性尚未确定，整个程序需要在沙箱中执行。

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，则程序将被拒绝并且事务将被回滚；否则，如果没有操作被拒绝但至少一个操作被任何操作后插件标记为“需要投票”，则程序将被挂起并等待最终投票结果，并且只有在通过投票过程批准并在“执行待定时间”内执行后才被允许执行。

由于沙箱还包含所有操作前插件和操作后插件的副本，操作后沙箱检查仅使用当前状态中的操作后插件。这是因为程序的合法性应仅由当前机器状态决定，包括当前插件判断系统，而机器状态只允许由当前机器状态、当前插件判断系统和必要时的投票结果批准的程序更新。

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（克隆的机器状态）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、成员资格、可提取的现金和分红余额、操作历史、插件和投票规则。

3.4.1 代币

代币系统是 DARC 协议中最基本和核心的设计。机器状态包含一个代币数组，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。

每个级别代币的信息结构存储在机器状态管理器中，如下 Solidity 结构所示。

```
struct Token {
    uint256 tokenClassIndex;
    uint256 votingWeight;
    uint256 dividendWeight;
    mapping (address => uint256) tokenBalance;
    address[] ownerList;
    string tokenInfo;
    uint256 totalSupply;
    bool bIsInitialized;
}
```

代币余额是从地址到 uint256 的映射。键是当前级别代币的拥有者地址，值是该拥有者地址所拥有的代币数量。DARC 还维护一个地址数组 ownerList，作为拥有至少一个当前级别代币的所有地址的完整列表。

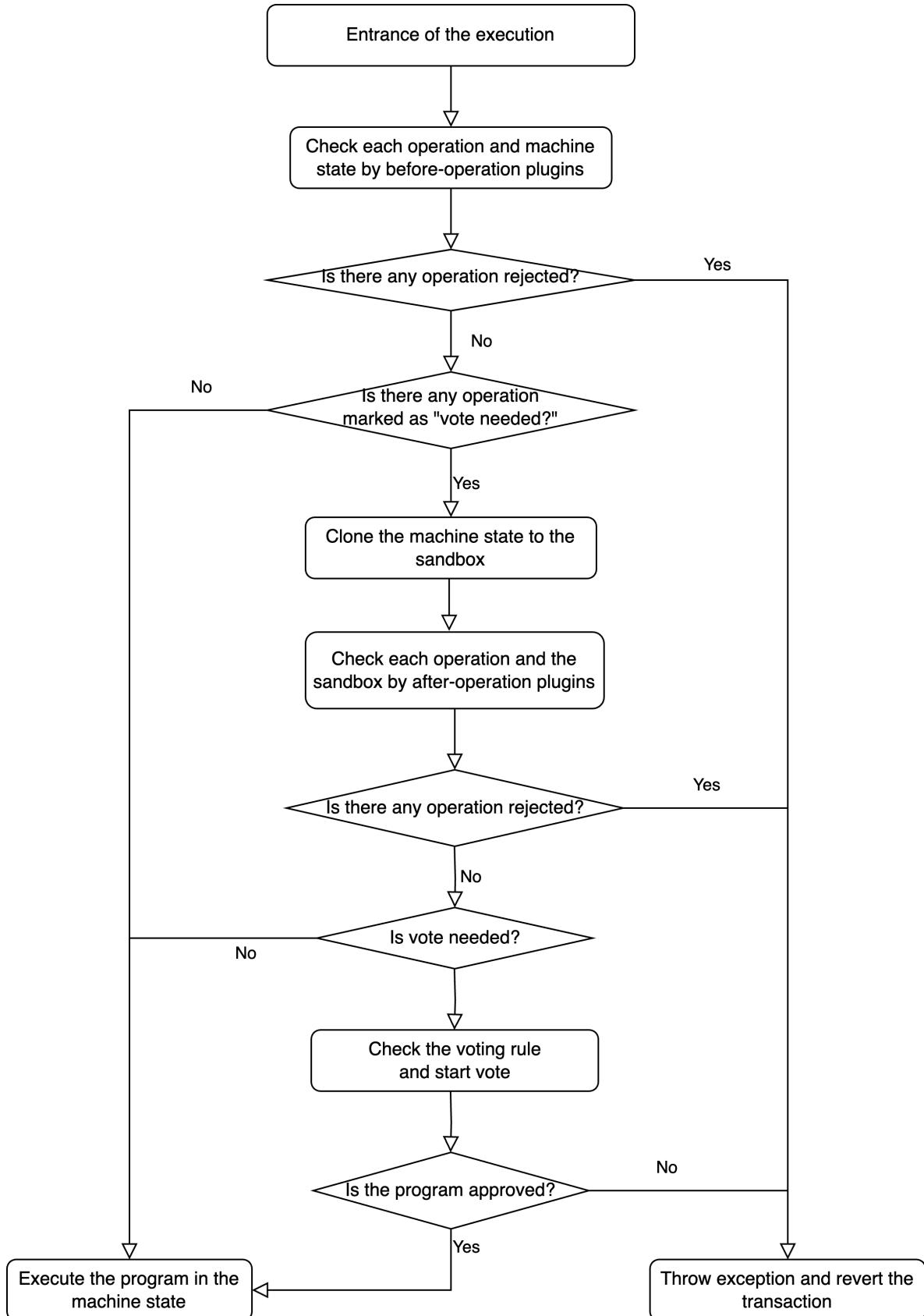


图 2: 执行引擎的工作流程。

3.4.2 成员资格

成员资格是一个从地址到级别的映射，使 DARC 能够层次化地管理不同地址。DARC 的治理可以确保不同级别的人被分配到不同的级别。

我们在 DARC 协议中引入成员资格的原因是，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：

```
/**  
 * 每个代币拥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、雇员、  
 * 董事会成员、特殊代理等。  
 */  
mapping (address => MemberInfo) memberInfoMap;  
  
/**  
 * DARC协议的股票代币拥有者信息  
 * 此结构用于存储每个代币拥有者的角色索引号  
 */  
struct MemberInfo {  
    bool bIsInitialized;  
    bool bIsSuspended;  
    string name;  
    uint256 role;  
}
```

3.4.3 可提取现金和分红

可提取现金和分红是两个独立的余额哈希映射。操作者可以从这两个余额中提取原生代币（如以太坊），如果余额不为零。对于可提取现金，向地址余额添加可提取代币的唯一方式是执行 BATCH_ADD_WITHDRAWABLE_BALANCE 操作，带有目标地址和代币金额。如果操作被批准，相应金额将被添加到余额中。对于分红，向余额添加可提取代币的唯一方式是执行 OFFER_DIVIDENDS 操作，不带任何参数。如果操作被批准，所有拥有分红权重大于 0 的代币持有者将收到相应的分红金额。

要将原生代币从 DARC 提取到地址，操作者可以执行 WITHDRAW_CASH_TO 操作，将可提取现金从余额转移到目标地址，或在操作被批准的情况下，执行 WITHDRAW_DIVIDENDS_TO 操作，将分红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相应操作的记录，每个操作都有其最新的时间戳。当程序成功执行后，程序中包含的操作的时间戳将在操作历史中更新。通过操作历史，开发者可以为单个地址或 DARC 中的所有成员设置每个操作之间的最短时间间隔。

规则 1 是一个使用操作历史限制操作者在一段时间内执行相同操作的示例：

规则 1：我们每月提供 10 ETH 作为雇员级别和经理级别地址（4 级和 5 级成员）的工资。

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，操作是添加可提取余额，操作的总金额小于或等于 10 ETH（或 1000000000000000000 wei），并且操作者在超过 30 天（或 2592000 秒）的时间内进行了相同的操作，则该操作将被批准，且该操作无需进行沙箱检查。

这里是一个为规则 1 设计的 By-law 脚本中的插件示例：

```
const plugin_Rule_1 =  
{  
    condition: operation_equals(BATCH_ADD_WITHDRAWABLE_BALANCE)  
    & total_add_withdrawable_balance_LE(1000000000000000000)  
    & 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,
note: ""
}

```

规则 1 没有限制 BATCH_ADD_WITHDRAWABLE_BALANCE 操作的目标地址。这是因为允许操作者向自己的可提取现金中添加 10 ETH，也允许他们向其他地址添加，只要他们在 30 天内添加的可提取现金总额不超过 10 ETH。

3.4.5 插件

插件是 DARC 协议中的基础机制，因为所有规则和规章都需要在 DARC 插件系统中定义、转译并保存。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，提交给 DARC 的每个程序都需要通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，则程序将被拒绝。

如果所有操作都被批准且不需要沙箱检查，DARC 将直接执行程序；如果一个或某些操作需要在沙箱中检查，DARC 将首先将机器状态克隆到沙箱，然后在沙箱内执行整个程序，并根据执行结果和沙箱状态做出决定。

每个插件由以下项目组成：

- 返回类型：当条件被触发时插件的决策，包括 YES、VOTING_NEEDED、NO、YES_AND_SKIP_SANDBOX 和 SANDBOX_NEEDED。
- 等级：当条件被触发时当前插件的优先级。当操作触发多个插件时，插件系统将使用具有最高等级的插件的返回类型作为最终决策。由于每个等级只允许具有相同返回类型的插件，这将确保每个操作都从插件系统获得确定的判断结果。
- 条件节点：conditionNodes 是表示此插件表达式二叉树的条件节点数组。每个节点可以是一个带参数的条件表达式，用于验证某个标准，或一个逻辑运算符，将两个或多个子节点（条件表达式或逻辑操作）合并为单个条件表达式。
- 投票规则索引：如果条件表达式标准被触发并且插件的返回类型是 VOTING_NEEDED，则指向某个特定投票规则的索引号。如果需要投票过程，判断系统将遍历所有 VOTING_NEEDED 插件并创建一个带有选定投票规则的多项投票过程。
- 注释：插件设计者为将来目的留下的注释。
- 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，操作者在成功部署后不能移除或修改插件。禁用某个插件的唯一方式是将布尔标志 bIsEnabled 从 True 设置为 False。如果操作被允许，也可以通过将其设置回 True 来启用插件。
- 布尔标志 bIsInitialized：指示插件是否成功初始化的布尔标志。如果 False，判断系统将跳过此插件。
- 布尔标志 bIsBeforeOperation：指示插件是操作前还是操作后插件数组的布尔标志。虽然操作前插件和操作后插件存储和管理在两个单独的数组中，但在判断系统遍历每个数组时将对此布尔值进行二次检查。这个字段是必需的，以确保在通过 By-law 脚本构建插件并通过 DARC 程序入口发送时，每个插件对象结构保持一致。

插件的结构在 Solidity 中以以下结构设计：

```

struct Plugin {
    /**
     * 当前条件节点的返回类型
     */
    EnumReturnType returnType;
}

```

```

 * 限制的等级，从0到uint256的最大值
 */
uint256 level;

/**
 * 条件二进制表达式树向量
 */
ConditionNode[] conditionNodes;

/**
 * 如果返回类型为VOTING_NEEDED，则当前插件的投票规则id
 */
uint256 votingRuleIndex;

/**
 * 插件注释
 */
string note;

/**
 * 指示插件是否启用的布尔值
 */
bool bIsEnabled;

/**
 * 指示插件是否被删除的布尔值
 */
bool bIsInitialized;

/**
 * 指示插件是操作前插件还是操作后插件的布尔值
 */
bool bIsBeforeOperation;
}

}

```

3.4.6 投票规则数组

投票规则数组是存储 DARC 协议中所有投票规则的数组。当程序在沙箱中执行并被插件判断系统检查时，一个或多个操作可能处于待定状态，等待最终投票结果。在投票过程中，所有成员都被允许在有效的投票期间投票。每个投票规则包含启动投票项所需的所有必要要求。

3.5 沙箱

DARC 的沙箱与机器状态管理器的存储设计完全相同，包括代币系统、成员资格、可提取现金、分红、操作历史、插件和投票规则。对于不能被操作前插件批准的每个程序，DARC 机器状态管理器将首先将机器状态克隆到沙箱，然后在沙箱内执行程序，并根据执行结果和沙箱状态做出决定。

3.6 投票状态

在机器状态管理器中，投票状态是管理所有与投票相关属性的独立模块。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、累积的投票权力和每个地址的投票记录。

4 法规脚本

法规脚本是一种专为 DARC 协议设计的、具有类 JavaScript 语法的编程语言。用户可以使用法规脚本执行多种任务，包括：

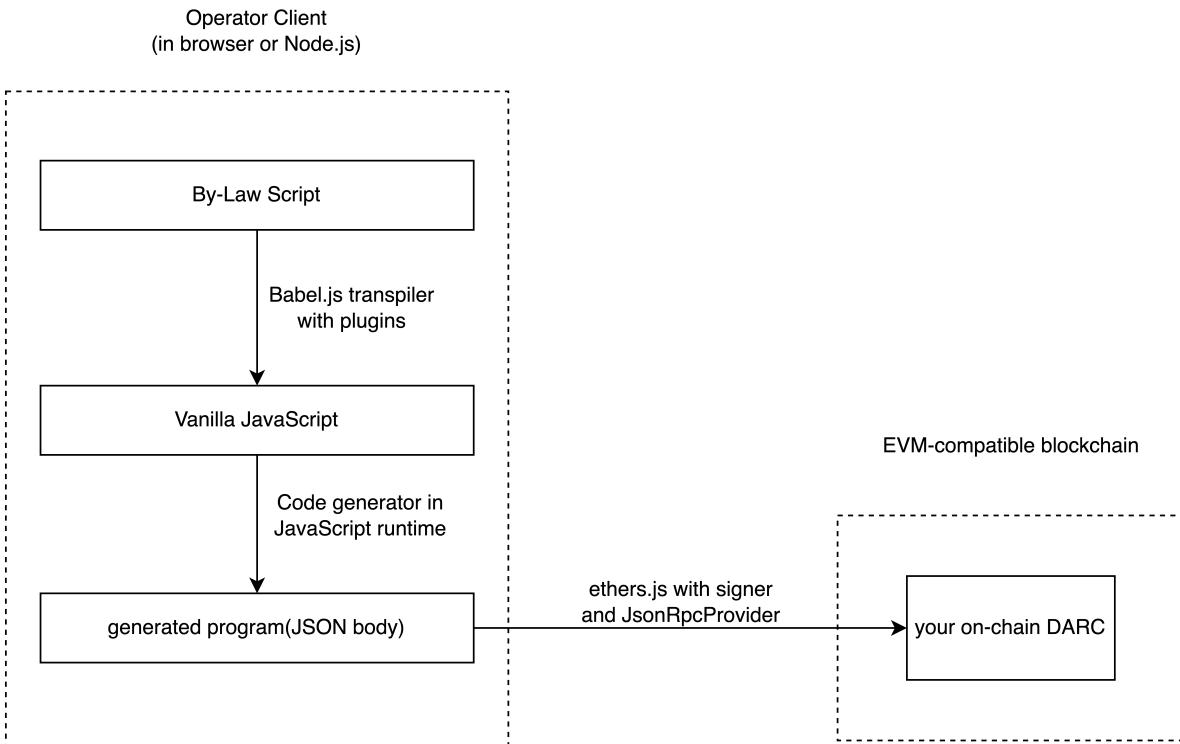


图 3: 编译和执行法规脚本的过程

- 操作: 用户可以使用法规脚本发起一系列操作。这些操作包括铸造和销毁新代币、代币转移、代币购买、启用或禁用插件以及资金提取等行动。
- 插件设计: 法规脚本允许用户使用二元表达式树设计插件, 结合逻辑运算符和带参数的 DARC 判断表达式。这些插件可以定义一个返回类型, 并在必要时指定一个投票规则。
- 投票: 法规脚本简化了当前操作的投票过程。

法规脚本的语法与原生 JavaScript 语法非常相似, 包括变量、常量、赋值、基本数据类型、函数、类和其他各种特性。除了基本的 JavaScript 语法外, 法规脚本还支持操作符重载, 这对于描述插件的条件节点特别有用。

图 3 展示了法规脚本在 DARC 协议中的编译和执行过程。在操作者客户端, 法规脚本的程序需要被转译为原生 JavaScript。随后, 它在本地 JavaScript 运行时使用代码生成器执行。在整个程序以 JSON 体的形式生成后, 它将成为一个包含操作码和参数列表的操作列表。此时, 程序已准备好从客户端作为函数调用提交给 EVM 兼容区块链上的 DARC 应用二进制接口 (ABI)。

要在 DARC 协议中执行程序, DARC 软件开发包 (SDK) 访问 DARC 智能合约的地址, 并通过调用入口函数并附上钱包签名启动执行过程。这一通信过程通过 EVM 兼容区块链的 JSON RPC 服务器进行。

4.1 转译器

用户运行法规脚本后, 转译过程的第一步是将法规脚本转换为原生 JavaScript。这一初步转换使用 Babel.js [git] 进行前端语法分析, 并利用操作符重载插件生成结果原生 JavaScript 代码。

使用操作符重载插件的主要意义在于它能够转译用户设计的条件节点中逻辑运算符的语法。用户创建插件来逻辑连接并描述各种条件表达式使用的逻辑运算符。当这些插件使用操作符重载功能进行转译时, 插件中的每个条件都被转换为一个对象构造的树状结构, 使用纯函数和参数表示。

此外, 转译器还支持其他高级 ECMAScript 语法和语法糖, 使用户使用起来更为方便。

4.2 代码生成器

一旦用户成功从法规脚本的转译生成了原生 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。

用户生成的转译原生 JavaScript 代码包括一个或多个操作命令函数，每个函数都包含操作码及其对应的参数。执行时，每个这样的操作段都存储在一个数组中，最终形成一个程序对象。随后，代码生成器利用这个程序对象创建一个完整的程序 JSON 体，符合 DARC ABI 入口规范。

一旦程序 JSON 体成功生成，用户可以使用 ethers.js 根据 ABI 将此程序体发送至 EVM 兼容区块链上的 DARC 协议。这个过程确保了程序在 DARC 内完整地执行。

图 4 展示了一个完整的编译过程。

5 多级别代币系统

多级别代币系统是 DARC 协议的核心机制，用户可以设计具有不同投票权重和分红权重的不同级别的代币。通过插件作为限制，多级别代币系统可用于代表组织中的不同组成部分或资产。

5.1 普通股

普通股是股份公司机制的中心元素，作为筹集资本和分配所有权的手段。普通股股东通常享有投票权，允许他们在股东大会上对重要公司决策发表意见。此外，普通股股东可能会收到分红，这是公司利润分配给股东的一部分。投票和分红的双重属性使普通股成为股东参与公司治理和财务回报的关键工具。

如果一个代币级别被分配了 1 的投票权重和 1 的分红权重，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，那么这个代币级别可以被视为 DARC 中的普通股。

要实例化这样一个代币级别，首先必须在法规脚本中建立它，分配 1 的投票权重和 1 的分红权重。成功执行此脚本后，DARC 协议将初始化一个具有 1 的投票权重和分红权重的 0 级别代币。

```
batch_create_token_classes(  
    ["TOKEN_0"], // 代币符号字符串  
    [0], // 代币级别  
    [1], // 投票权重  
    [1] // 分红权重  
)
```

考虑到公司事务的复杂性，创建了两个用于不同目的的投票规则：

投票规则 1：要批准一个操作，必须满足以下条件：超过 50% 的有效投票权力支持该操作，并且这次投票需要是绝对多数模式，意味着它必须包括至少 50% 的总代币投票权力。投票规则 1 的投票时间为 7 天 (604800 秒)。程序获批后，操作者需要在 1 小时 (3600 秒) 内执行程序。

投票规则 2：要批准一个操作，必须满足以下条件：超过 75% 的有效投票权力支持该操作，并且这次投票需要是相对多数模式，意味着它必须包括至少 75% 的有效投票权力。投票规则 2 的投票时间为 1 天 (86400 秒)。程序获批后，操作者需要在 1 小时 (3600 秒) 内执行程序。

以下是投票规则 1 和 2 的初始化法规脚本。当添加一个带有投票规则索引 1 或 2 的新插件时，插件将确保当条件被触发且插件优先级最高时，将根据选定的投票规则启动投票过程。

```
batch_add_voting_rule([  
    // 添加规则1  
    {  
        // 仅允许0级别代币进行投票  
        votingTokenClassList: [0],  
  
        // 批准门槛: 50%  
        approvalThresholdPercentage: 50,  
  
        // 投票持续时间: 604800秒  
        votingDurationInSeconds: 604800,  
  
        // 执行待定持续时间: 3600秒  
        executionPendingDurationInSeconds: 3600,  
    }])
```

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
    }
]);
```

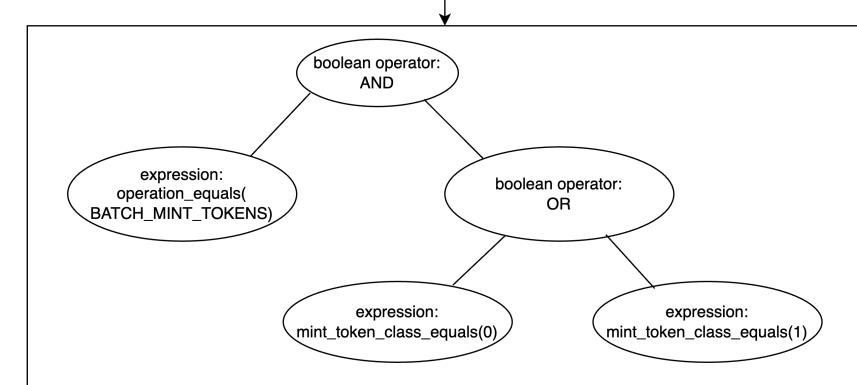
transpiler with operator overloading

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



Generated Program
(Plugin Condition Node)

codegen runtime

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过程

```

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

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

    // 是否是绝对多数
    bIsAbsoluteMajority: true
  },

  // 添加规则2
  {
    // 仅允许0级别代币进行投票
    votingTokenClassList: [0],

    // 批准门槛: 75%
    approvalThresholdPercentage: 75,

    // 投票持续时间: 86400秒
    votingDurationInSeconds: 86400,

    // 执行待定持续时间: 3600秒
    executionPendingDurationInSeconds: 3600,

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

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

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

```

5.2 A/B 类股

A 类和 B 类股是公司财务的基本工具。A 类提供投票权和分红优势，通常由创始人持有。B 类具有限制的投票权，用于在保留控制权的同时筹集外部资本。这种双重类别结构平衡了治理需求和增长需求，对股东的决策制定和财务利益有所影响。

首先初始化两个代币级别：0 级别具有 1 的投票权重和 1 的分红权重，1 级别具有 100 的投票权重和 1 的分红权重。

```

batch_create_token_classes(
  ["TOKEN_0", "TOKEN_1"],    // 代币符号字符串
  [0, 1],                  // 代币级别
  [1, 100],                // 投票权重
  [1, 1]                   // 分红权重
);

```

接下来，我们将铸造 10000 个 0 级别代币和 200 个 1 级别代币。这将使 0 级别代币的总投票权力为 10000，1 级别代币为 20000。这样，DARC 的总投票权力为 30000，其中 66.7% 在 1 级别代币。通过这种方式，共同创始人和早期投资者将控制 DARC 的大多数投票权力，允许所有 0 级别和 1 级别代币持有者参与投票过程。

我们假设 `addr1` 和 `addr2` 总共持有 10000 个 0 级别代币，每人 5000 个，每个 `addr3`、`addr4`、`addr5` 和 `addr5` 持有 50 个 1 级别代币。首先在法规脚本中将代币铸造给这些地址。

```

batch_mint_tokens(
    [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以限制 0 级别和 1 级别代币的总供应量分别为 10000 和 20000。

投票规则 1: 要批准操作，0 级别和 1 级别代币的总投票权力的 90% 必须投 YES。投票过程需要是绝对多数模式。投票规则 1 的投票时间为 7 天 (604800 秒)。程序获批后，操作者需要在 1 小时 (3600 秒) 内执行程序。

操作前插件规则 1: 如果操作是 BATCH_MINT_TOKENS 并且铸造的代币级别是 0 或 1，操作需要在沙箱中执行后再做决定。

操作后插件规则 1: 如果操作是 BATCH_MINT_TOKENS 并且铸造的代币级别是 0 或 1，操作需要经过投票规则 3 的批准。

操作前插件规则 2: 如果操作是 BATCH_DISABLE_PLUGIN 并且索引在操作前插件 1、操作前插件 2 或操作后插件 1 之前，拒绝操作。此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 级别或 1 级别代币改变股东结构时，操作需要经过 90% 的总投票权力的批准。这将防止 1 级别股东稀释 0 级别代币的价值，并保护那些主要是零售投资者的 0 级别代币持有者。

操作前插件 2 是对上述其他插件的保护。当任何操作者试图禁用这三个插件中的任何一个时，操作将被拒绝。这将永久锁定该机制并保证插件不能被禁用。

```

batch_add_voting_rules([
    // 添加操作前插件1
    {
        // 允许0级别和1级别代币进行投票
        votingTokenClassList: [0, 1],

        // 批准门槛: 90%
        approvalThresholdPercentage: 90,

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

        // 执行待定持续时间: 3600秒
        executionPendingDurationInSeconds: 3600,

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

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

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

```

然后向 DARC 添加三个插件。

```

batch_add_and_enable_plugins([
    // 添加操作前插件, 索引0
    {
        returnType: SANDBOX_NEEDED, // 需要沙箱
        level: 255, // 等级255
        condition:
            operation_equals(BATCH_MINT_TOKENS) &
            (

```

```

        mint_token_class_equals(0) | mint_token_class_equals(1)
    ),
    votingRuleIndex: 0, // 不需要投票规则索引
    note: "操作前插件1",
    bIsBeforeOperation: true // 插件将作为操作前插件添加
},
// 添加操作后插件，索引0
{
    returnType: VOTING_NEEDED, // 需要投票
    level: 258, // 等级258
    condition:
        operation_equals(BATCH_MINT_TOKENS) &
    (
        mint_token_class_equals(0) | mint_token_class_equals(1)
    )
    votingRuleIndex: 0, // 投票规则1，索引 = 0
    note: "操作后插件1",
    bIsBeforeOperation: false // 插件将作为操作前插件添加
},
// 添加操作前插件1
{
    returnType: NO, // 拒绝
    level: 257, // 等级257
    condition:
        operation_equals(BATCH_DISABLE_PLUGIN) &
    (
        disable_before_op_plugin_index_equals(0) |
        disable_before_op_plugin_index_equals(1) |
        disable_after_op_plugin_index_equals(0)
    )
    votingRuleIndex: 0, // 不需要投票规则索引
    note: "操作前插件2",
    bIsBeforeOperation: true // 插件将作为操作前插件添加
}
]);

```

5.3 优先股

优先股，也称为非投票股，是一种在公司中不享有投票权但提供财务利益如分红优先权、清算优先权和稳定性的所有权份额。它可以是可转换的或可赎回的，并且因其稳定的收入而受到青睐。然而，它不提供对公司决策的发言权，使其成为寻求资本保值并对公司事务有限控制的收入型投资者的选择。在 DARC 协议中，当一个代币被初始化为具有 1 的分红权重和 0 的投票权重时，它可以被归类为非投票股。

5.4 带印花税的股票

带印花税的股票是在购买或出售时需缴纳政府税收的证券，这种税收称为“印花税”。这种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，具体取决于当地规定。人们在交易股票时支付印花税是因为它作为政府收入的来源，也可以作为调节金融市场和抑制过度交易的工具。印花税的具体原因和税率因地区而异。

在 DARC 协议中，我们有能力创建一个机制，强制代币持有者在转让不同级别的代币时支付交易费，称为“印花税”。例如，我们可以将 0 级别的代币指定为 DARC 协议中的标准股票，每次代币转移应用固定的 1000 wei 印花税。

首先，我们必须实现一个插件来禁用所有 BATCH_TRANSFER_TOKENS 操作，当被转让的代币属于 0 级别时。接下来，我们应该开发一个插件，仅通过 BATCH_PAY_TO_MINT_TOKENS 操作限制转让 0 级别的

代币，前提是每个代币的交易费大于或等于 1000 wei。此外，用户需要将 BATCH_PAY_TO_MINT_TOKENS 操作的 dividendable 标志设置为 0（假），确保交易费（印花税）不会被视为 DARC 的分红收入。

```
batch_add_and_enable_plugins([
    // 添加操作前插件1: 禁用0级别代币的转让
    {
        returnType: NO, // 拒绝
        level: 255, // 等级255
        condition:
            operation_equals(BATCH_TRANSFER_TOKENS) &
            transfer_tokens_level_equals(0)
        votingRuleIndex: 0, // 不需要投票规则索引
        note: "禁止转让0级别代币",
        bIsBeforeOperation: true // 插件将作为操作前插件添加
    },
    // 添加操作前插件2:
    // 允许支付转让0级别代币,
    // 每个代币的交易费 >= 1000 wei
    {
        returnType: YES_AND_SKIP_SANDBOX, // 允许并跳过沙箱
        level: 256, // 等级256
        condition:
            operation_equals(BATCH_PAY_TO_TRANSFER_TOKENS) &
            (
                pay_to_transfer_tokens_level_equals(0) &
                transaction_fee_per_token_GE(1000) &
                pay_to_transfer_tokens_dividendable_flag_equals(0)
            )
        votingRuleIndex: 0, // 不需要投票规则索引
        note: "允许支付转让0级别代币，交易费 > 1000 且分红标志为0",
        bIsBeforeOperation: true // 插件将作为操作前插件添加
    }
]);
});
```

5.5 董事会

在 DAOs 中，通常通过使用治理代币进行投票来达成决策。然而，当每个决策都涉及成千上万的代币持有者时，这个过程可能变得缓慢且低效。此外，缺乏对待执行提案智能合约的规定或预定义规则意味着，批准的投票可以潜在地授权 DAO 内进行广泛的行动。

相比之下，在传统的股份公司中，日常运营和事务是由董事会而不是依靠所有股东的决策来监督的。董事会以其专业知识、战略眼光、快速决策能力以及维护公司治理稳定性和一致性而著称。虽然股东投票对于问责制和代表性至关重要，但董事会制度提供了一个结构化和见多识广的决策方法，确保公司的长期利益得到周到的优先考虑和有效管理。

以下是一个示例，展示了 DARC Y 实验如何使用多级别代币系统设计其董事会。存在四种类型的代币，每种都在组织内扮演着独特的特性和角色，如表 4 所示。

0 级别 (A 类代币): 这些代币总供应量为 400，拥有 1 的投票权重和 1 的分红权重。它们可能代表 DARC Y 协议中的普通利益相关者。

1 级别 (B 类代币): B 类代币的总供应量为 60，拥有 10 的显著投票权重，使其在决策中具有影响力。它们还具有 1 的分红权重，暗示着可能的收入分配份额。

2 级别 (独立董事): 只有一个独立董事代币，设计上没有投票权重和分红权重，表明其在治理结构中的独特角色。

3 级别 (董事会): 董事会由五个代币组成，具有 1 的投票权重但没有分红权重，表明其在决策中的主要功能。

以下是设计董事会机制的一些指导原则：

原则 1：董事会应由最多 5 名成员组成，所有董事会决策必须得到超过 2/3 成员的批准。

原则 2: 如果一个地址持有超过 2/3 的 0 级别代币，并且不持有任何 1 级别、2 级别或 3 级别代币，它可以代表零售投资者的利益，并可能被选为 A 类董事会成员。如果没有任何地址持有超过 2/3 的这些代币，董事会将没有 A 类代表。

原则 3: 董事会应始终包括一名独立董事，独立董事有权选择其继任者。不允许将独立董事从董事会中移除。

原则 4: 董事会应由最多 3 名 B 类董事会成员组成，允许任何持有超过 5% B 类代币的地址添加。现有董事会成员可以提名 B 类董事会成员，需得到至少 2/3 所有董事会成员的批准。

原则 5: 现有董事会成员有权提议移除 B 类董事会成员。这样的提案可以在以下条件下获批：(1) 被移除的地址持有的 B 类代币数量小于或等于总供应量的 5%，或 (2) 该地址不持有超过 2/3 的 0 级别代币总供应量或独立董事代币，并且提案获得超过 2/3 的董事会投票权力的支持。

原则 6: A 类代币董事、B 类代币董事和独立董事应保持独立于代币持有量。这意味着 (1) A 类代币董事只能持有 A 类代币和董事会代币，禁止持有 B 类代币或独立董事代币；(2) B 类代币董事仅允许持有 B 类代币和董事会代币，不得持有 A 类代币或独立董事代币；(3) 独立董事只允许持有独立董事代币。

接下来，我们将设计以下插件以实现上述定义的规则和流程：

操作前插件规则 1: 如果操作者地址持有超过 2/3 的 0 级别代币，并且不持有任何 1 级别、2 级别或 3 级别代币，它可以为自己铸造 1 个 3 级别代币而无需沙箱检查。此插件规则用于提名 A 类代币董事会成员。

操作前插件规则 2: 当操作者地址打算将 2 级别代币转让给另一个目标地址，且目标地址不持有任何 0 级别、1 级别或 3 级别代币时，此操作可以在不需要沙箱检查的情况下获批。此插件便于独立董事职位的转让。

操作前插件规则 3: 在情况下，操作者地址持有 1 个 2 级别代币和 0 个 3 级别代币并希望为自己铸造一个 3 级别代币，此操作可以在不需要沙箱检查的情况下获批。此插件用于提名独立董事。

操作前插件规则 4: 涉及铸造或销毁 2 级别代币的操作需要被拒绝。此插件限制独立董事的数量。

操作前插件规则 5: 如果操作者地址打算为目标地址铸造 1 个 3 级别代币，且目标地址持有超过 5% 的 1 级别代币总供应量，并且操作者地址持有 1 个 3 级别代币，此操作必须进行沙箱检查。此插件用于提名 B 类代币董事会成员。

操作前插件规则 6: 如果操作者地址打算从目标地址销毁一个 3 级别代币，且目标地址持有的 0 级别代币总供应量小于或等于 2/3，持有的 1 级别代币小于或等于 5%，且拥有 0 个 2 级别代币，此操作可以在不需要沙箱检查的情况下获批。此插件便于移除不合格的董事会成员。

操作前插件规则 7: 如果操作者地址打算从目标地址销毁一个 3 级别代币，且目标地址持有的 0 级别代币总供应量小于或等于 2/3，持有的 1 级别代币超过 5%，且拥有 0 个 2 级别代币，并且操作者地址至少持有 1 个 3 级别代币，此操作必须进行沙箱检查。此插件用于移除 B 类代币董事会成员。

操作后插件规则 1: 如果操作者地址打算从目标地址销毁一个 3 级别代币，且目标地址持有的 0 级别代币总供应量小于或等于 2/3，持有的 1 级别代币超过 5%，拥有 0 个 2 级别代币，并且操作者地址至少持有 1 个 3 级别代币，此操作必须通过投票规则 1 进行投票。此次投票涉及所有董事会成员，需要绝对多数模式，批准率超过 66%。此插件用于 B 类董事会成员选举。

接下来是法规脚本中实现的插件。

```
const plugin_before_op_1 = {
    returnType: YES_AND_SKIP_SANDBOX,
    level: 255,
    condition: 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,
    note: "操作前插件1",
    bIsBeforeOperation: true
},
const plugin_before_op_2 = {
    returnType: YES_AND_SKIP_SANDBOX,
    level: 255,
```

```

condition: operation_equals(BATCH_TRANSFER_TOKENS)
    & transfer_token_level_equals(2) ,
votingRuleIndex: 0,
note: "操作前插件2",
bIsBeforeOperation: true
},

const plugin_before_op_3 = {
    returnType: YES_AND_SKIP_SANDBOX,
    level: 255,
    condition: operation_equals(BATCH_MINT_TOKENS)
        & operator_owns_num_of_token_equals(2, 1)
        & number_of_token_mint_equals(1)
        & level_of_token_mint_equals(3) ,
votingRuleIndex: 0,
note: "操作前插件3",
bIsBeforeOperation: true
},

const plugin_before_op_4 = {
    returnType: NO,
    level: 257,
    condition: (operation_equals(BATCH_BURN_TOKENS)
        & level_of_token_burned_equals(2)) |
    (operation_equals(BATCH_MINT_TOKENS)
        & level_of_token_mint_equals(2)) ,

votingRuleIndex: 0,
note: "操作前插件4",
bIsBeforeOperation: true
},

const plugin_before_op_5 = {
    returnType: SANDBOX_NEEDED,
    level: 256,
    condition: 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_owns_num_of_tokens_greater(1,12),
votingRuleIndex: 0,
note: "操作前插件5",
bIsBeforeOperation: true
},

const plugin_before_op_6 = {
    returnType: YES_AND_SKIP_SANDBOX,
    level: 255,
    condition: 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),
votingRuleIndex: 0,
note: "操作前插件6",
bIsBeforeOperation: true
}

```

```

    },

const plugin_before_op_7 = {
    returnType: SANDBOX_NEEDED,
    level: 256,
    condition: operation_equals(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),
    votingRuleIndex: 0,
    note: "操作前插件7",
    bIsBeforeOperation: true
},

const plugin_after_op_1 = {
    returnType: VOTING_NEEDED,
    level: 253,
    condition: operation_equals(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_owns_num_of_tokens_equals(3,1),
    votingRuleIndex: 1,
    note: "操作后插件1",
    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_before_op_6,
    plugin_before_op_7,
    plugin_after_op_1
]);

```

5.6 公司债券

公司债券是企业为筹集资金而发行的债务证券。投资者向公司借出资金，以换取定期利息支付和到期时本金的返还。它们为公司提供了筹集资本的手段，并为投资者提供了可预测的收入来源。

在 DARC 协议的背景下，BATCH_PAY_TO_MINT_TOKENS 和 BATCH_BURN_TOKENS_AND_REFUND 命令便于在特定时间框架内以指定价格购买和赎回债券代币，使任何地址都可进行购买。购买和退款这些债券代币的两条规则如下：

规则 1：在 2020 年 1 月 1 日至 2020 年 2 月 1 日期间，任何地址都可以使用 BATCH_PAY_TO_MINT_TOKENS 以每个代币 10000 wei 的价格购买不超过 100 个债券代币（2 级别）。债券代币的总供应量不得超过 10000。

规则 2：地址有权使用 BATCH_BURN_TOKENS_AND_REFUND 以每个代币 15000 wei 的价格在 2030 年 1 月 1 日后赎回债券代币。

```

batch_add_and_enable_plugins([
{
    returnType: YES_AND_SKIP_SANDBOX,
    level: 253,

```

```

        condition: operation_equals(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), // 总代币供应量
        votingRuleIndex: 0,
        note: "",
        bIsBeforeOperation: true
    },
    {
        returnType: YES_AND_SKIP_SANDBOX,
        level: 253,
        condition: operation_equals(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
        votingRuleIndex: 0,
        note: "",
        bIsBeforeOperation: true
    },
]);

```

5.7 产品代币和非同质化代币

当我们将投票权重和分红权重都设置为 0，并强制要求操作者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。

当我们进一步配置几个级别的代币以允许每笔交易仅支付铸造一个代币，禁止铸造和销毁操作，并禁止在某一级别已存在一个代币的情况下进行额外的支付铸造交易。同时，我们允许这些代币的持有者自由将它们转让给其他地址，此时，这些特定级别的代币可以被视为 NFTs（非同质化代币）[WE]。NFTs 可以用作并被视为代表特定物品或内容的所有权或真实性证明的独特数字资产，在 DARC 协议中使用。

6 插件

6.1 插件的设计

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，它遵循下面伪代码中概述的逻辑：

```

if plugin.condition:
    return plugin.returnType

```

对于 DARC 协议，操作前插件和操作后插件之间的主要区别在于它们的返回类型。对于操作前插件，由于它们决定某个操作是否应该直接执行、直接拒绝或进入沙箱，因此它们有三种不同的返回类型作为它们的最终决定：

1. NO。当操作前插件的条件被触发时，插件对该操作的决定是 NO。这个决定表明插件认为操作违反了它的规则，因此在进入沙箱执行之前就被直接拒绝。
2. SANDBOX_NEEDED。当操作前插件的条件被触发时，插件对该操作的决定是 SANDBOX_NEEDED。这个决定表明插件无法确定操作应该被接受还是被拒绝。插件知道需要在沙箱中由操作后插件评估该操作，因此做出让操作进入沙箱以进行进一步评估的决定。
3. YES_AND_SKIP_SANDBOX。当操作前插件的条件被触发时，插件对该操作的决定是 YES_AND_SKIP_SANDBOX。这个决定表明插件已经确定操作应该被批准，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。

对于操作后插件，由于程序已在沙箱中执行并且可以开始投票，这些插件也可以有三种返回类型作为它们的最终决定：

1. NO。当操作后插件的条件被触发时，插件对操作的决定是 NO。这个决定表明插件认为操作违反了它的规则，应该被直接拒绝。
2. VOTING_NEEDED。当操作后插件的条件被触发时，插件对操作的决定是 VOTING_NEEDED。这个决定表明插件认为操作需要投票，并且操作需要根据该插件指定的投票规则初始化投票项。
3. YES。当操作后插件的条件被触发时，插件对操作的决定是 YES。这个决定表明插件根据它的规则认为操作应该被允许进行。

每个插件都有一个条件节点数组，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或表达式；
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，必须在子节点列表中指定至少两个有效的子节点索引；
4. 对于 NOT 运算符，必须在子节点列表中指定一个唯一的子节点索引；
5. 对于表达式节点，必须设置与表达式一致的有效条件表达式参数；
6. 对于表达式节点，其子节点列表的长度必须为 0，意味着不允许有子节点。

图 5 是一个例子，说明了如何将一个条件表达式二叉树序列化为条件节点数组。

此外，插件需要设置两个参数：一个是“级别”，代表插件在整个插件系统中的优先级。对于同一个操作，判断系统遍历所有插件，可能至少有两个或更多插件被触发。在这种情况下，如果插件的级别不同，判断系统会将级别更高的插件视为最终决定。

另一个参数是“投票规则索引”，它指向投票规则数组中的一个特定索引。当插件的最终决定是 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，则会忽略投票规则索引。

6.2 插件与判断系统

对于 DARC 协议，判断系统需要进行两次评估：一次通过操作前插件，另一次在程序完全在沙箱中运行后，然后通过操作后插件进行评估。这样设计的原因是，如果没有沙箱并且仅依赖一组插件进行判断，预测程序的行为将变得非常困难。因此，无法保护 DARC 协议中特殊状态的修改。

例如，在一个 DARC 实例中，要求股东 X 永久持有 15% 的投票权和 10% 的分红权时，设计插件时无法预测铸造代币或销毁代币等操作执行后 DARC 实例的状态。这种不确定性给确保股东 X 永久拥有 15% 的投票权和 10% 的分红权带来了挑战。只有在沙箱中运行操作然后重新评估沙箱的状态，才能防止这种修改。

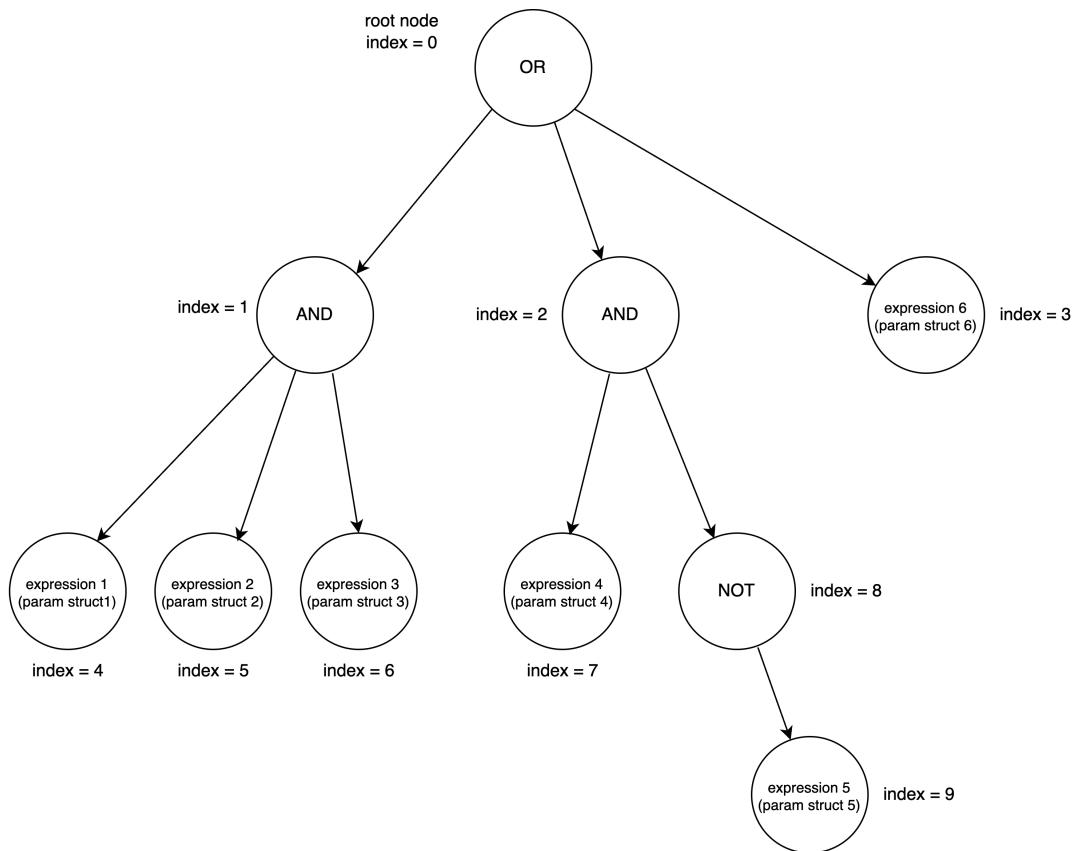
在另一个场景中，如果需要确保一个 DARC 实例在 2035 年 1 月 1 日之前永久保留 10000 个原生代币，只有在沙箱中执行这些操作后，才能通过操作后插件进行第二次评估，从而确保正确检测和防止支付分红或提取现金等操作。如果没有沙箱并且仅依赖插件，设计这样的机制将过于复杂。

如果只有操作后插件和沙箱而没有操作前插件，那将是昂贵且低效的。这是因为沙箱的运行成本非常高。它不仅需要在沙箱中完全运行程序，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 Gas 费用。

对于大多数可以在不需要在沙箱中运行的简单操作，直接在操作前插件中建立规则更为节省成本。例如，小股东和零售投资者之间的股份交易、客户进行日常交易、董事会成员执行常规支付操作以及员工为自己发放薪水和股票激励——这些众多的日常和高频活动可以定义为操作前插件。这种方法有助于为大多数日常操作节省 Gas 费用。

对于操作前插件，每当一个程序提交给 DARC 协议时，判断系统会顺序检查每个操作。对于每个操作，判断系统遍历每个操作前插件，并获得一个单一的判断结果。最后，它汇总所有操作的结果，以确定整个程序的最终结果。这个决定决定了程序是否需要在沙箱中运行 (SANDBOX_NEEDED)、被直接拒绝 (NO) 还是可以直接运行而不需要沙箱 (YES_AND_SKIP_SANDBOX)。对于操作前判断，遵循以下规则：

1. 如果任何操作被判断系统判定为 NO，整个程序将以 NO 的结果被拒绝。



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

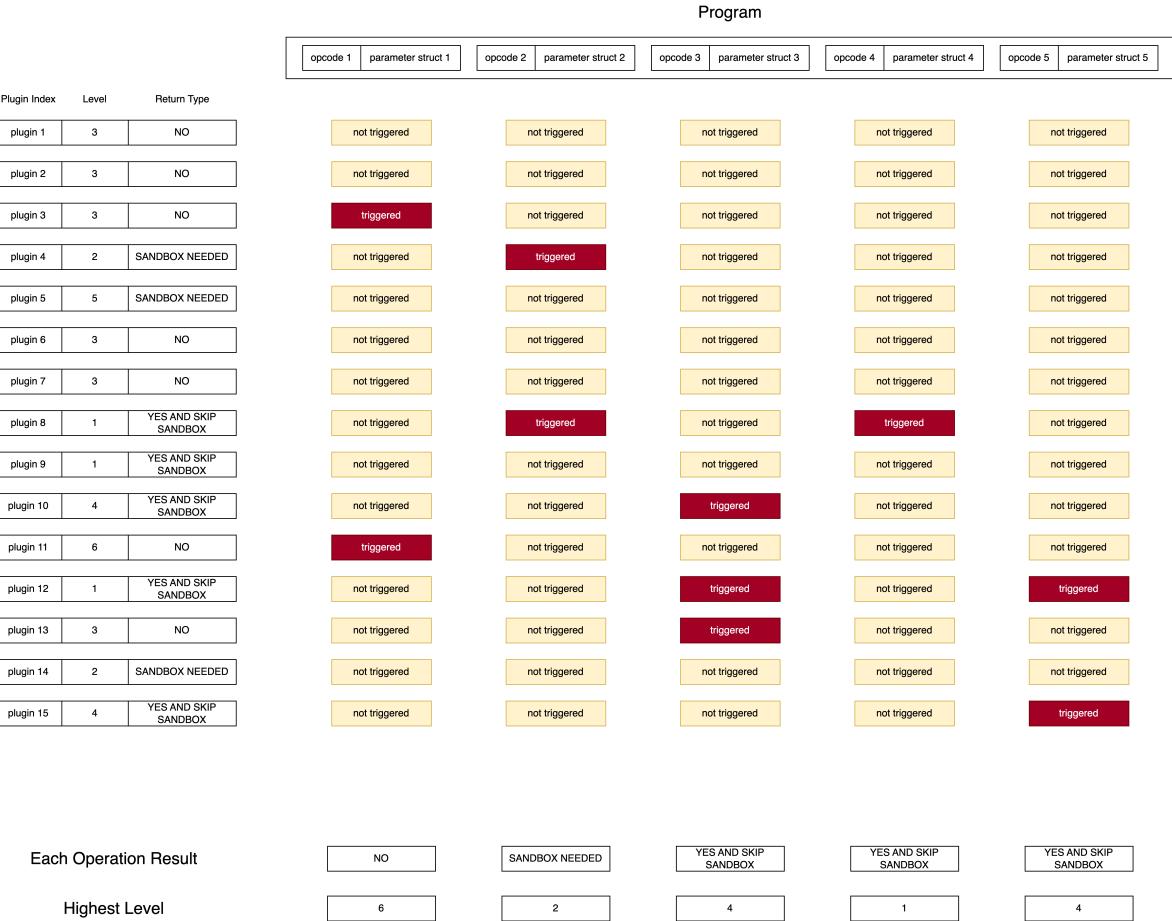


图 6: 带有操作前插件的程序判断

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 SANDBOX_NEEDED，整个程序需要在沙箱中运行，结果为 SANDBOX_NEEDED。
3. 如果所有操作都被判断系统判定为 YES_AND_SKIP_SANDBOX，整个程序将以 YES_AND_SKIP_SANDBOX 的结果被批准，整个程序可以跳过沙箱。

图 6 说明了程序内的单个操作如何通过操作前插件被判断，得出判断结果，决定程序是否需要通过投票批准 (VOTING_NEEDED)、直接拒绝 (NO) 或可以直接进行 (YES)。对于操作后判断，遵循以下规则：

1. 如果任何操作被判断系统判定为 NO，整个程序将以 NO 的结果被拒绝。
2. 如果判断系统没有将任何操作判定为 NO，并且至少有一个操作被判定为 VOTING_NEEDED，整个程序的最终判断为 VOTING_NEEDED。这个程序将被归类为待决程序，DARC 协议必须启动投票系统来决定批准或拒绝。
3. 如果所有操作都被判断系统判定为 YES，整个程序将以 YES 的结果被批准，整个程序可以直接执行。

图 7 说明了程序内的单个操作如何通过操作后插件被判断，得出判断结果。

7 投票

7.1 投票过程和状态

如图 8 所示，您可以将每个 DARC 协议视为一个有限状态机 (FSM)，它具有三种状态：

			Program				
Plugin Index	Level	Return Type	opcode 1 parameter struct 1	opcode 2 parameter struct 2	opcode 3 parameter struct 3	opcode 4 parameter struct 4	opcode 5 parameter struct 5
plugin 1	3	NO	not triggered	not triggered	triggered	not triggered	not triggered
plugin 2	3	NO	not triggered				
plugin 3	3	VOTING_NEEDED	not triggered				
plugin 4	2	YES	not triggered	triggered	not triggered	not triggered	not triggered
plugin 5	5	VOTING_NEEDED	not triggered	not triggered	not triggered	triggered	not triggered
plugin 6	3	NO	triggered	not triggered	not triggered	not triggered	triggered
plugin 7	3	NO	not triggered				
plugin 8	1	YES	not triggered	not triggered	not triggered	triggered	not triggered
plugin 9	1	NO	not triggered				
plugin 10	4	YES	not triggered	not triggered	triggered	not triggered	not triggered
plugin 11	6	NO	not triggered				
plugin 12	1	YES	not triggered	not triggered	not triggered	not triggered	triggered
plugin 13	3	NO	not triggered				
plugin 14	2	YES	triggered	not triggered	not triggered	not triggered	not triggered
plugin 15	4	YES	triggered	not triggered	not triggered	not triggered	not triggered

Each Operation Result	YES	YES	YES	VOTING_NEEDED	NO
Highest Level	4	2	4	5	3

图 7: 带有操作后插件的程序判断

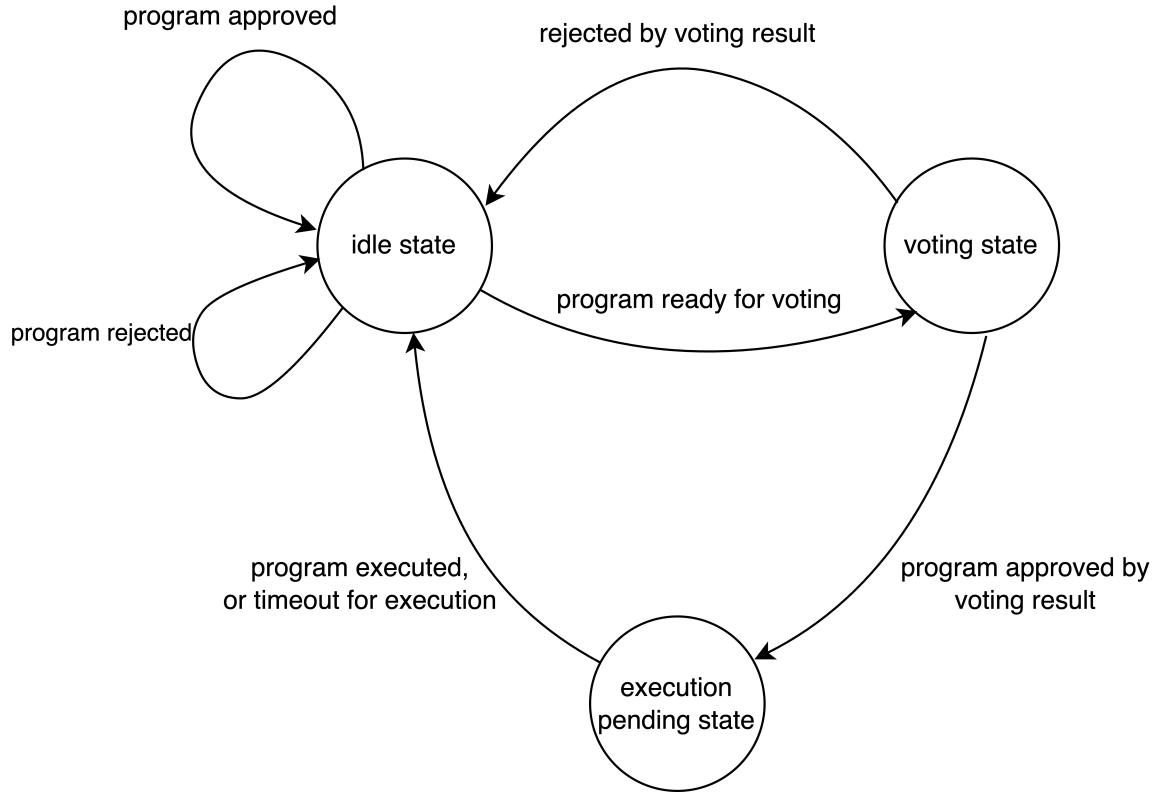


图 8: DARC 的有限状态机

1. 空闲状态：在此状态下，DARC 协议可以接受任何程序。当程序被接受并执行后，DARC 协议转回到空闲状态。同样，如果程序被拒绝，DARC 协议返回到空闲状态。当程序包含需要投票的操作时，DARC 协议转换到投票状态。

2. 投票状态：在 DARC 协议的投票状态期间，所有用户被允许且只被允许执行包含单一投票操作的程序。如果程序包含多个操作，程序将被拒绝。同样，如果程序包含的单一操作不是投票，程序也将被拒绝。如果当前投票被批准，系统将自动转换到等待执行状态。此外，投票状态有一个最长时间限制。如果在此时间限制内，每个投票项的支持票数未总体超过定义的阈值，当前的投票轮被视为不成功，系统将自动返回到空闲状态。只有当所有投票项在指定时间限制内获得批准时，程序才会被 DARC 协议批准。

3. 等待执行状态：当一个程序被 DARC 协议投票并批准后，它可以在等待执行状态中被执行。等待执行状态也有一个最长时间限制。如果程序在此限制内被执行，它将被成功执行，系统将自动恢复到空闲状态。如果执行没有在此限制内发生，程序将失败，DARC 协议将丢弃程序，自动返回到空闲状态。

7.2 投票规则

每个投票规则由以下项目组成：

- 投票代币类别列表：允许为此投票项投票的代币类别索引列表。它包含至少一个有效的代币类别索引号。列在此数组中的所有代币都被视为有效投票代币。持有任何包含在此数组中的代币的所有代币持有者都被允许为此项投票。
- 批准阈值百分比：以百分比表示的批准阈值。
- 布尔标志 `bIsAbsoluteMajority`：一个布尔标志，指示投票模式是绝对多数还是相对多数。
- 投票持续时间（秒）：此投票过程的最大持续时间，以秒为单位。如果投票过程超过此参数的时间，投票过程将被终止，投票结果将被设置为失败。

- 执行等待持续时间（秒）：在投票过程结束后，操作者执行批准程序的最大持续时间，以秒为单位。如果程序被批准，操作者必须在定义的持续时间结束前执行此程序；否则，程序将被中止，DARC 将重置为空闲状态。
- 布尔标志 isEnabled：一个必须设置为 True 的布尔标志，如果投票规则被成功初始化并启用。
- 注释：存储在投票规则中的字符串，包含此投票规则的额外信息、评论或外部 URL。

```
struct VotingRule {

    /**
     * 投票代币类别索引列表
     */
    uint256[] votingTokenClassList;

    /**
     * 投票政策的批准阈值百分比
     */
    uint256 approvalThresholdPercentage;

    /**
     * 投票政策的投票持续时间（秒）
     */
    uint256 votingDurationInSeconds;

    /**
     * 投票政策的执行等待持续时间（秒）
     */
    uint256 executionPendingDurationInSeconds;

    /**
     * 投票政策是否启用
     */
    bool isEnabled;

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

    /**
     * 投票政策是绝对多数还是相对多数。
     */
    bool bIsAbsoluteMajority;
}
```

7.3 投票类型

通常，在 DARC 协议的投票系统中，有两种类型的投票方法：

- 绝对多数：在绝对多数投票中，批准需要超过一个固定阈值，以总投票权重的百分比表示。如果批准票的组合权重超过预定阈值，投票被认为是成功的。例如，总权重为 1000，阈值设置为 70%，如果批准票的总权重超过 700，投票被认为是成功的。
- 相对多数：在相对多数投票中，批准是相对于总投票权重的百分比。与绝对多数不同，总投票权重是实际投票权重的百分比，而不是预定义的绝对值。例如，如果总权重为 1000，但只有 300 权重在投票中被投出，相对多数要求批准的投票权重超过投出票的 70% ($0.7 * 300 = 210$)，投票才被认为是成功的。

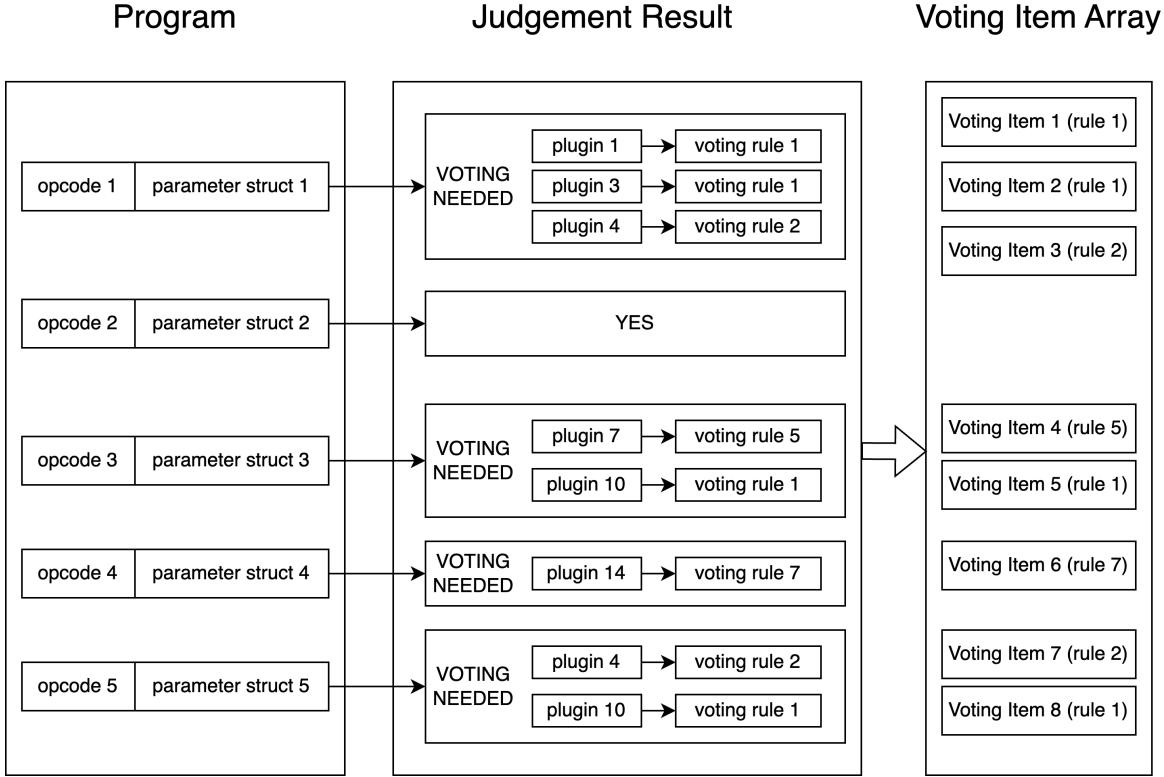


图 9: 投票规则和项

我们提出“绝对多数”和“相对多数”投票选项，是因为这两种模式在平衡投票结果时提供了不同的侧重点。

关于绝对多数，其优势在于批准的投票权重只需要占总可能投票的一部分，提供更多灵活性。它适用于重大决策的情况，其中一个显著多数足以确定结果，以促进更广泛的共识。其应用场景适合于重要决策，其中至关重要的是获得大多数人的支持。使用绝对多数作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要构成当前投票中的最大份额，提供了更高的便捷性和灵活性。这种方法适用于相对多数足以确定结果的情况。它的适用性扩展到需要更灵活和快速决策过程的情况。相对多数有助于在不需要固定百分比的总可能投票权重的情况下更迅速地达成共识。

通过提供绝对和相对多数选项，我们增强了投票规则以满足不同协议和组织的多样化治理需求的适应性。选择适当的投票模式有助于更好地满足特定决策场景的共识和效率需求。

7.4 投票机制

对于每个程序，在收到来自操作前插件的批准并在沙箱内完全执行后，由所有操作后插件引导的判断系统，评估每个操作。如果对特定操作的所有插件的最高级别判断标记为“VOTING_NEEDED”，则收集与该插件相关的所有投票规则。一旦判断系统汇编了程序中每个操作对应的所有投票规则，它会生成一个投票项数组，为其各自的条目分配每条规则。

对于每个操作，可能同时触发一个或多个不同的插件，所有这些插件的返回类型为“VOTING_NEEDED”。在这种情况下，将按顺序收集指向该操作的每个插件的投票规则。另一方面，对于每个程序，不同的操作可能会被同一个插件依次触发。在投票项数组中，每个操作的对应投票规则将生成一个新的投票项。

图 9 是一个示例，说明 DARC 协议是如何基于判断结果初始化投票项数组的。

一旦投票项数组被初始化，作为有限状态机，DARC 协议转换到投票状态。假设在投票项数组中有 N 个投票项，所有操作者都将被允许投票，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。对于每个操作者，在每个投票状态中，他们可以且必须只投票一次，确保投票操作中的参数是一个长度为 N 的布尔数组。如果操作者尝试执行超过一次的投票操作或提供一个长度不为 N 的布尔数组，投票程序将被自动拒绝。

在 DARC 协议中，每个投票项都有两个计数器：一个用于“YES”票，另一个用于“NO”票。每当一个操作者执行有效的投票操作时，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代币级别 $C = [c_1, c_2, \dots, c_n]$ ，其中 n 是代币级别的数量。每个级别 c_i 有一个相应的投票权重，表示为 w_i 。假设在投票过程中，一个操作者选择对这个投票项进行投票，操作者的投票权重 W 通过以下公式计算：

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

在此公式中， W 代表操作者对此投票项的总投票权重，求和遍历所有允许投票的代币级别 c_i 。对于每个级别 i ， v_i 是此操作者在该级别拥有的代币数量。

在每个操作者提交有效的投票操作后，每个投票项将把相应的投票权重增加到 YES 或 NO 计数器中。如果操作者在可投票代币集合中持有零代币，对于特定的投票项，既不会影响 YES 计数器也不会影响 NO 计数器。在投票过程完成后，DARC 协议将根据以下标准确定投票是否结束：

1. 如果所有投票项中至少有一个在绝对多数模式下运行的投票项收到足够数量的 NO 票，整个投票过程将提前终止，导致投票失败。随后，DARC 协议转换到空闲状态。例如，如果一个投票项的批准阈值为 66%，以绝对多数模式运行，并且在截止日期前收集到超过 34% 的 NO 票，整个投票过程将立即结束，导致投票失败并转换到空闲状态。
2. 当所有投票项仅在绝对多数模式下运行，并且每个投票项都获得了超过批准阈值的 YES 票时，整个投票过程将提前结束，表明投票成功。随后，DARC 协议转换到等待执行状态。
3. 当投票状态不符合标准 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，每个在绝对多数模式下的投票项都收到了等于或超过总投票权重乘以阈值的 YES 票，并且对于每个在相对多数模式下的投票项，该项获得了等于或超过提交的投票权重乘以阈值的 YES 票，投票被批准，DARC 协议转换到等待执行状态。如果任何投票项未能满足此标准，投票被认为是不成功的，DARC 协议恢复到空闲状态。

在进入等待执行状态后，当前投票状态中存储的程序已被批准且准备执行。程序必须在新的等待执行截止日期前完成。成功执行后，DARC 协议转换到空闲状态。然而，如果在执行过程中发生错误或异常，或者如果在截止日期前没有人执行程序，DARC 协议也会恢复到空闲状态，并且程序将无法继续执行。

8 成员资格

在每个公司中，除了董事会成员、股东、高管等成员外，还有许多员工、合同工、实习生、客户、供应商等。他们可能不持有任何一级代币或公司股份，但他们的工资、订单、订阅费和其他功能需要批量管理。因此，成员资格是另一套用于 DARC 协议中的便捷工具。

表 5 是 DARC X 咨询公司的一个分层成员资格结构示例。在这个表中，我们定义了 6 个级别：联合创始人、重要股东、高管、经理、员工和离职员工。我们可以基于这个成员资格表为 DARC X 定义一些插件。

规则 1：我们需要限制联合创始人在未通知高管的情况下出售大量股票。如果操作者是一个拥有“联合创始人”角色的地址，并且操作是出售超过 100000 级别-0 代币，这个操作需要得到所有董事会成员的批准。

规则 2：我们为员工级别和经理级别的地址（级别 4 和 5）提供每月 10 ETH 的工资。

现在我们分析上述规则并设计 By-law 脚本中的插件。

对于规则 1，我们需要确保如果当前操作是“转移代币”，代币的级别是 0，并且代币数量超过 100000，并且地址在成员资格表中角色级别等于 1，程序将被暂停并等待投票过程。投票过程被定义并保存为投票规则 1，要求所有持有级别-3 代币的持有者在 1 小时内投票。总共有 5 个级别-3 代币，每个董事会成员持有 1 个代币。如果所有 5 个董事会成员在 1 小时内投票同意，此操作将被批准。否则，整个程序将被拒绝，操作将失败。

这里是为规则 1 设计的 By-law 脚本中的示例插件：

```
const plugin_Rule_1 =
{
  condition: (operation_name_equlas(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，操作是增加可提现余额，操作的总金额小于或等于 10 ETH（或 1000000000000000000 wei），并且操作者在超过 30 天（或 2592000 秒）内做了同样的操作，这个操作将被批准，并且可以跳过此操作的沙盒检查。

这里是为规则 2 设计的 By-law 脚本中的示例插件：

```

const plugin_Rule_2 =
{
    condition: operation_equals(BATCH_ADD_WITHDRAWABLE_BALANCE)
    & total_add_withdrawable_balance_LE(1000000000000000000)
    & 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,
    note: ""
}

```

9 分红

可分红资金池将包含万分之 X 的一部分，可用于分红分配。对于可分红资金池中的总金额 T 以及当前 DARC 协议内各个代币级别的分红权重之和，我们可以按照以下方式确定每个分红单位的分红金额：

$$u = \frac{XT}{10000W}$$

这里， u 代表每个分红单位的分红金额， T 是可分红资金池中的总金额， W 代表当前 DARC 协议内所有可分红代币跨不同代币级别的分红权重之和。

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 D_i ，那么用户 i 在本轮中可以接收的总分红金额 U_i 为：

$$U_i = \frac{D_i}{10000}u$$

对于 DARC 协议，可以采用三种分红分配方法：

基于交易的分红分配：第一种方法是基于可分红交易的数量分配分红。这通过设置分红交易周期计数器， N ，到一个合理的值来实现。一旦接收到至少 N 个可分红交易，指令 OFFER_DIVIDENDS 就变得可执行。执行后，可分红资金池和可分红交易周期计数器被重置为零，开始新的计数。这个过程重复，等待下一个 N 交易和随后的分红分配操作。

基于时间的分红分配：第二种方法是基于时间分配分红。将 N 设置为 1，可以引入一个额外的插件，允许在整个 DARC 实例中，OFFER_DIVIDENDS 被调用的时间不少于上一次调用后的 S 秒。以这种方式，分红可以每 2 周、4 周、3 个月、6 个月或 1 年安排一次。这种方法更接近传统公司的分红。

基于资金池的分红分配：第三种方法涉及根据可分红资金池中的总金额分配分红。将 N 设置为 1，一个额外的插件使得一旦可分红资金池中的金额超过一个指定的阈值，就可以执行 OFFER_DIVIDENDS。根据这种设计，分红分配可以在获得 Y 个可分红本币代币后触发。

用户可以选择上述任何一种方法，或者设计插件来创建基于不同 DARC 组织结构和方法的替代、实用和合理的分红分配模式。

需要注意的是，对于每个收到的可分红交易执行 OFFER_DIVIDENDS 可能会导致大量的 Gas 费用浪费。这是由于潜在的时间复杂度为 $O(MNP)$ ，其中 M 代表代币级别的数量， N 是每个级别中的代币数量， P 是代币持有者的总数。因此，实现一个高效的分红分配机制以节省 Gas 费用是必要的。

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。协议不保障现金分红，OFFER_DIVIDENDS 只执行计算，将每个代币持有者即将获得的分红存储在可提取的分红余额中。当操作者提取分红时，如果 DARC 协议没有足够数量的本币代币，这是不可能的。为了保护特定或所有代币持有者的分红权利，必须设计额外的插件。

10 紧急代理

在 DARC 协议中，用户经常遇到各种问题，例如：

1. 设置不正确的插件参数、触发条件或执行错误的代币操作，使得恢复变得不可能。
2. 锁定 DARC 协议中的某些关键操作，导致协议内功能永久不可用。
3. 人为争议导致组织无法运作，这些争议无法通过插件解决，但可能通过使用文件、文本、证据等手动诉讼来解决。
4. 发现 DARC 协议的漏洞或面临攻击，需要紧急暂停和恢复。
5. 解决可能出现的其他潜在技术问题或争议。

在 DARC 协议中，用户可以为紧急情况指定一个或多个紧急代理。在不可预见的情况下，用户可以邀请紧急代理进行干预。紧急代理作为一个超级管理员，拥有在 DARC 协议内执行任何操作的权限，不受插件限制。这个角色对于解决 DARC 协议内紧急或未解决的问题至关重要。

1. `addEmergency(emergencyAgentAddress)`：此命令用于通过提供紧急代理的地址来添加紧急代理。一旦添加，紧急代理获得超级管理员权限，允许他们在 DARC 协议中执行任何操作，不受插件的限制。
2. `callEmergency(emergencyAgentAddress)`：此命令用于通过指定要调用的紧急代理的地址来调用紧急代理。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()`：此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 DARC 协议的正常操作。

11 可升级性

一旦在 EVM 兼容区块链上部署了编译后的智能合约二进制文件，它就变得不可变，对二进制文件的任何修改都不可能。OpenZeppelin [Ope] [ethc] 使用基于代理的方法进行更新。具体方法涉及部署一个代理智能合约以及所有实现智能合约，并设置一个管理员地址。当需要更新实现合约时，管理员可以直接修改代理，将代理重定向到新的实现合约地址。

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利，可能会覆盖所有现有资产、插件和信息，导致管理员的权力超过所有不同级别的代币持有者。在一个公司已经建立了公司结构的情况下，拥有一个拥有修改所有法律方面、处理所有公司股份和管理资产权限的超级管理员将是一个危险的提议。

程序入口作为 DARC 协议的统一和唯一入口点。从旧的 DARC 实例 X 升级到新的 DARC 实例 Y 时，过程涉及为旧的 DARC 实例 X 配置升级地址，并设置新的 DARC 实例 Y 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在 DARC 实例 X 中执行程序，这些程序将被委托并在新的 DARC 实例 Y 中执行。

在 DARC 协议中，有三个与升级相关的操作：

- `upgradeToAddress(targetAddress)`：将当前 DARC 实例升级到 targetAddress。所有提交给当前 DARC 实例的程序将被委托并在 targetAddress 处的 DARC 实例中执行。

- `confirmUpgradedFromAddress(sourceAddress)`: 允许当前 DARC 实例从 sourceAddress 处的 DARC 实例进行升级。当程序从 sourceAddress 代理提交时，允许它在当前 DARC 实例中以程序提交者的操作者身份运行。
- `upgradeToTheLatest()`: 如果当前 DARC 实例 A 已经升级到新的 DARC 实例 B，并且新的 DARC 实例 B 也已经升级到更新的 DARC 实例 C，执行此函数将直接将 DARC 实例升级到 DARC 实例 C。换句话说，在这个 DARC 实例 A 中执行的程序将直接在 DARC 实例 C 中委托和执行，绕过 DARC 实例 B。这个操作将被不同地解释。假设一个 DARC 实例 A 已经升级到 DARC 实例 B，在这样的场景下，如果用户在 A 中执行一个仅有一个 `upgradeToTheLatest()` 操作的程序，程序将在 DARC 实例 A 中执行，而不是被委托到 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：

在第一个场景中，用户已经部署了 My DARC 1，并且 My DARC 1 没有升级到任何其他 DARC，需要以下三个步骤：

1. 在区块链上部署新版本的 My DARC 2 并完成所有配置。
2. 在 My DARC 2 中执行 `confirmUpgradedFromAddress(MyDARC1Addr)`，允许 My DARC 2 接受程序并作为 My DARC 1 的代理。
3. 在 My DARC 1 中执行 `upgradeToAddress(MyDARC2Addr)`，完成从 My DARC 1 到 My DARC 2 的升级。

图10展示了直接从 My DARC 1 到 My DARC 2 的整个升级过程。

在第二个场景中，用户已经部署了 My DARC 1，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)`，使 My DARC 3 能够接受程序和来自 My DARC 1 的代理。
3. 在 My DARC 1 中执行 `upgradeToAddress(MyDARC2Addr)`。由于这个程序将在 My DARC 1 的代理下在 My DARC 2 中运行，My DARC 2 的 `upgrade_address` 将指向 My DARC 3 的地址。
4. 在 My DARC 1 中执行 `upgradeToTheLatest()`，根据 My DARC 2 的地址直接将 My DARC 1 指向 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展示了直接从 My DARC 1 到 My DARC 3 的整个升级过程。

12 讨论

DARC 协议在商业和加密世界的未来方向和应用，对于重塑组织结构和治理机制具有重大潜力。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 协议提供了建立更加健壮和透明的企业实体的机会。通过利用 DARC 的自我调节和可编程性，企业可以潜在地简化其治理流程、增强合规性，并确保长期可持续性。DARC 通过其插件强制执行严格的规则和规定，可以导致创建更加有责任心和效率高的企业结构，类似于传统的股份公司，但增加了基于区块链的治理的优势。

此外，By-law 脚本的灵活性为创建多样化的公司结构开启了大门，包括 A/B 股份、有限责任公司 (LLCs)、C 型公司、非营利基金会等。这种适应性使 DARC 成为一个多用途平台，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需求。因此，DARC 有潜力成为广泛商业实体的基础框架，为企业治理和运营提供了一个新的范例。

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 (DAOs) 继续获得关注，DARC 作为一个受规范和可适应的替代品脱颖而出。它支持多代币系统、分红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 DARC 成为传统企业结构和自治 DAOs 之间的桥梁，为去中心化组织提供了一个受规范和可编程的基础。

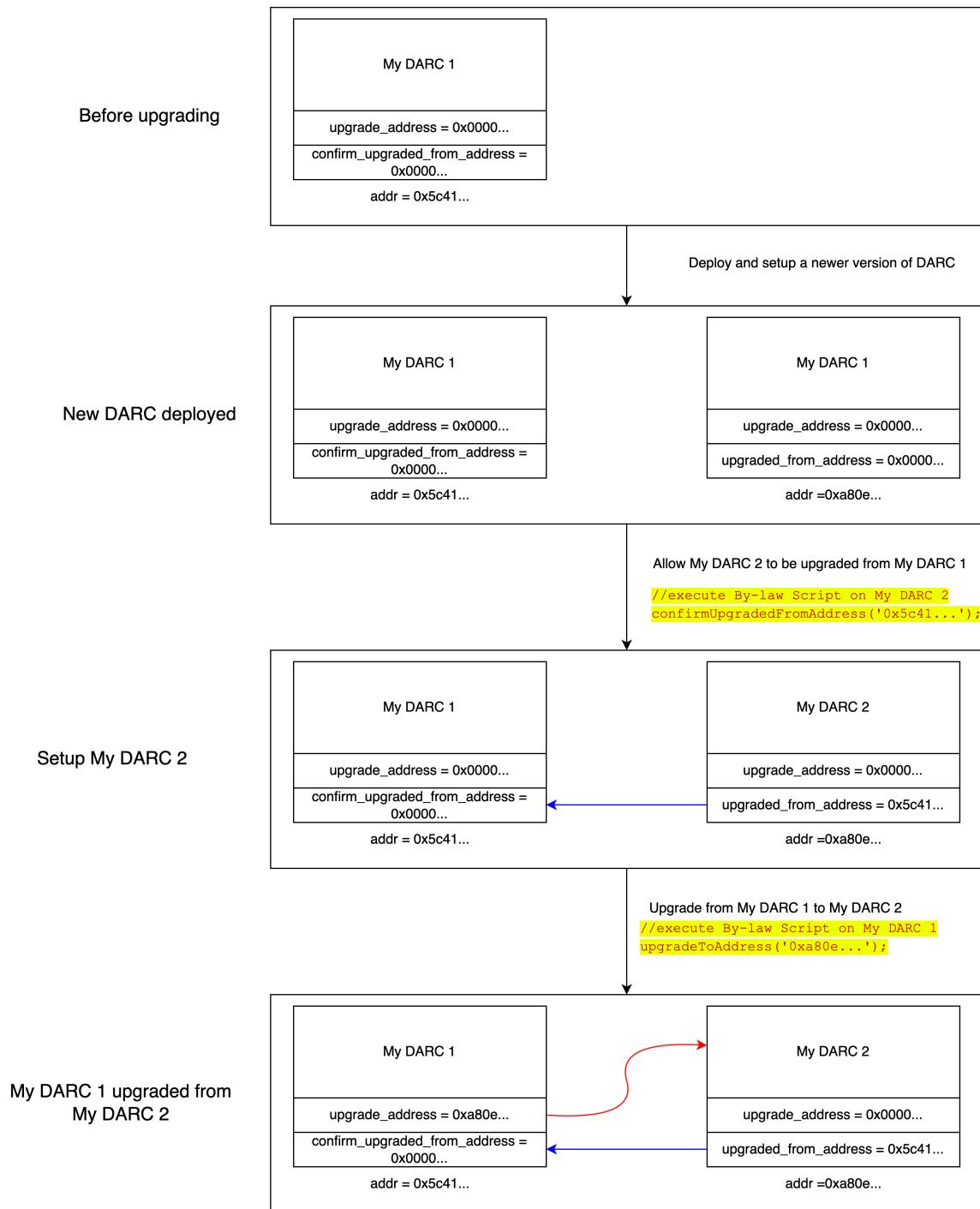


图 10: 从 My DARC 1 升级到 My DARC 2

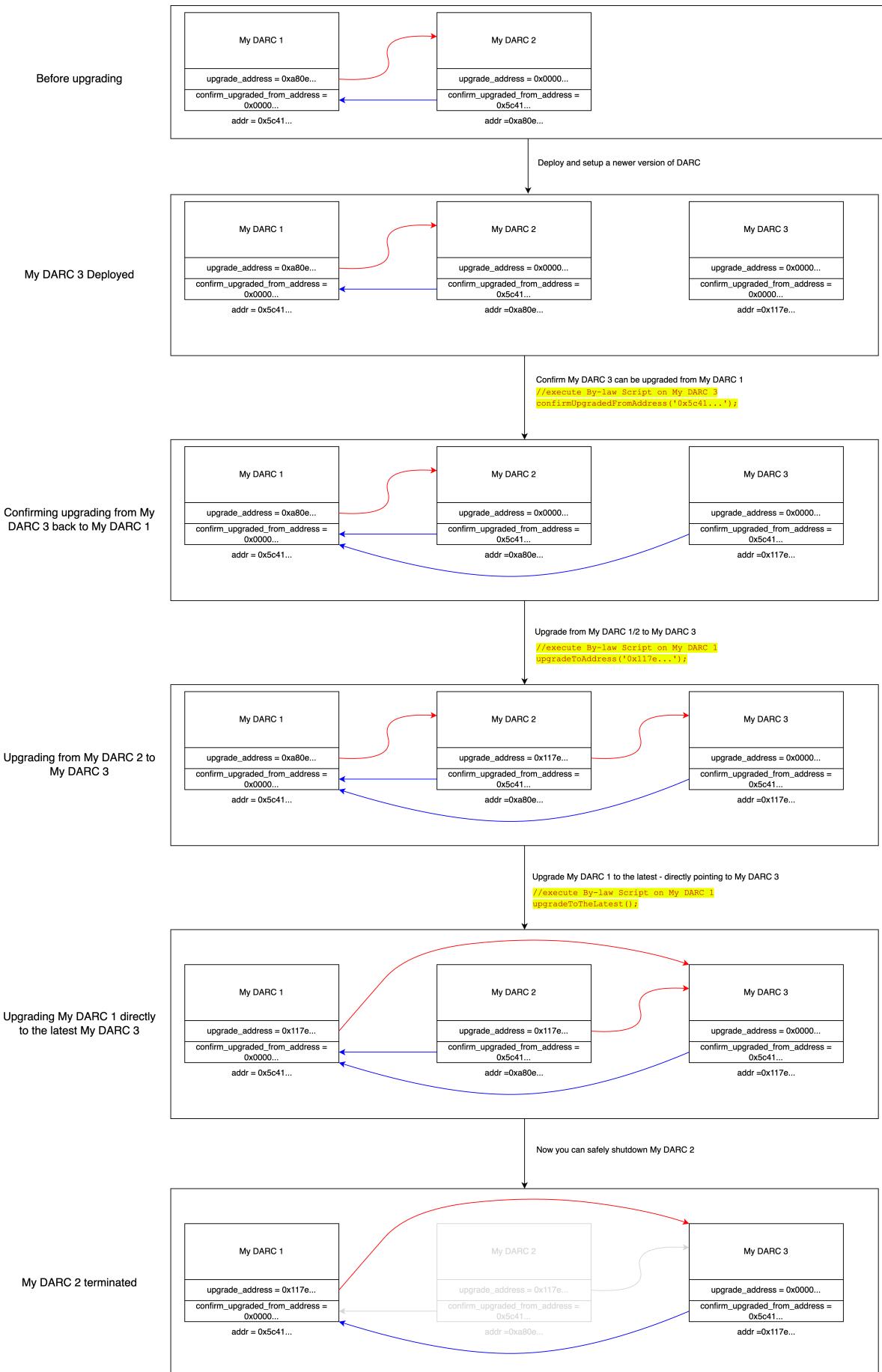


图 11: 从 My DARC 1 升级到 My DARC 2 再到 My DARC 3

展望未来，将 DARC 与其他区块链基础设施和智能合约编程语言如 Rust、Move 和 Plutus 的整合，呈现了进一步优化和扩展的激动人心的途径。这可能导致更高效和多样化的 DARC 协议实现的发展，扩大其在多样化的区块链生态系统中的应用和影响范围。

总之，DARC 协议的未来对于革新企业治理、商业运营和去中心化组织结构充满希望。它建立受规范、自我治理的公司的潜力以及适应多样化公司结构的能力，使 DARC 成为商业和加密世界中的变革力量，提供了一种新的治理、合规和可持续性的方法。

参考文献

- [B+] Vitalik Buterin et al. Ethereum white paper.
- [etha] Documentation — docs.ethers.org. <https://docs.ethers.org/v6/>. [Accessed 10-12-2023].
- [ethb] EIP-170: Contract code size limit — eips.ethereum.org. <https://eips.ethereum.org/EIPS/eip-170>. [Accessed 11-12-2023].
- [ethc] ERC-1967: Proxy Storage Slots — eips.ethereum.org. <https://eips.ethereum.org/EIPS/eip-1967>. [Accessed 11-12-2023].
- [git] GitHub - babel/babel: Babel is a compiler for writing next generation JavaScript. — github.com. <https://github.com/babel/babel>. [Accessed 10-12-2023].
- [Jen16] Christoph Jentzsch. Decentralized autonomous organization to automate governance. *White paper*, November, 2016.
- [Ope] “Proxy Upgrade Pattern”, OpenZeppelin Docs, <https://docs.openzeppelin.com/upgrades-plugins/1.x/proxies>.
- [sol] Solidity Solidity 0.8.23 documentation — docs.soliditylang.org. <https://docs.soliditylang.org/en/v0.8.23/>. [Accessed 10-12-2023].
- [WE] Jacob Evans Nastassia Sachs William Entriken, Dieter Shirley. ERC-721: Non-Fungible Token Standard — eips.ethereum.org. <https://eips.ethereum.org/EIPS/eip-721>. [Accessed 11-12-2023].

附录 1：指令操作码的参考设计

```
/*
 * @notice opcode枚举用于表示DARC协议的指令。
 */
enum EnumOpcode {

    /**
     * @notice 无效操作
     * ID: 0
     */
    UNDEFINED,

    /**
     * @notice 批量铸造代币操作
     * @param ADDRESS_2DARRAY[0] address[] toAddressArray: 需要铸造新代币的地址数组
     * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: 需要铸造新代币的代币类别索引数组
     * @param UINT256_2DARRAY[1] uint256[] amountArray: 需要铸造的代币数量数组
     *
     * ID: 1
     */
    BATCH_MINT_TOKENS,
```

```

/**
 * @notice 批量创建代币类别操作
 * @param STRING_ARRAY[] string[] nameArray: 需要创建的代币类别名称数组
 * @param UINT256_2DARRAY[0] uint256[] tokenIndexArray: 需要创建的代币类别的代币索引数组
 * @param UINT256_2DARRAY[1] uint256[] votingWeightArray: 需要创建的代币类别的投票权重数组
 * @param UINT256_2DARRAY[2] uint256[] dividendWeightArray: 需要创建的代币类别的分红权重数组
 *
 * ID:2
 */
BATCH_CREATE_TOKEN_CLASSES,

/**
 * @notice 批量转移代币操作
 * @param ADDRESS_2DARRAY[0] address[] toAddressArray: 需要转移代币到的地址数组
 * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: 需要转移代币的代币类别索引数组
 * @param UINT256_2DARRAY[1] uint256[] amountArray: 需要转移的代币数量数组
 *
 * ID:3
*/
BATCH_TRANSFER_TOKENS,

/**
 * @notice 批量从地址A转移到地址B的代币操作
 * @param ADDRESS_2DARRAY[0] address[] fromAddressArray: 需要从中转移代币的地址数组
 * @param ADDRESS_2DARRAY[1] address[] toAddressArray: 需要转移代币到的地址数组
 * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: 需要转移代币的代币类别索引数组
 * @param UINT256_2DARRAY[1] uint256[] amountArray: 需要转移的代币数量数组
 *
 * ID:4
*/
BATCH_TRANSFER_TOKENS_FROM_TO,

/**
 * @notice 批量销毁代币操作
 * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: 需要销毁代币的代币类别索引数组
 * @param UINT256_2DARRAY[1] uint256[] amountArray: 需要销毁的代币数量数组
 *
 * ID:5
*/
BATCH_BURN_TOKENS,

/**
 * @notice 批量从地址A销毁代币操作
 * @param ADDRESS_2DARRAY[0] address[] fromAddressArray: 需要从中销毁代币的地址数组
 * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: 需要销毁代币的代币类别索引数组
 * @param UINT256_2DARRAY[1] uint256[] amountArray: 需要销毁的代币数量数组
 *
 * ID:6
*/
BATCH_BURN_TOKENS_FROM,

/**
 * @notice 批量添加成员操作
 * @param ADDRESS_2DARRAY[0] address[] memberAddressArray: 需要作为成员添加的地址数组
 * @param UINT256_2DARRAY[0] uint256[] memberRoleArray: 需要添加的成员角色数组

```

```

* @param STRING_ARRAY string[] memberNameArray: 需要添加的成员名称数组
*
* ID:7
*/
BATCH_ADD_MEMBERSHIP,

/** 
* @notice 批量暂停成员操作
* @param ADDRESS_2DARRAY[0] address[] memberAddressArray: 需要暂停的成员地址数组
*
* ID:8
*/
BATCH_SUSPEND_MEMBERSHIP,

/** 
* @notice 批量恢复成员操作
* @param ADDRESS_2DARRAY[0] address[] memberAddressArray: 需要恢复的成员地址数组
*
* ID:9
*/
BATCH_RESUME_MEMBERSHIP,

/** 
* @notice 批量更改成员角色操作
* @param ADDRESS_2DARRAY[0] address[] memberAddressArray: 需要更改角色的成员地址数组
* @param UINT256_2DARRAY[0] uint256[] memberRoleArray: 需要更改的成员角色数组
*
* ID:10
*/
BATCH_CHANGE_MEMBER_ROLES,

/** 
* @notice 批量更改成员名称操作
* @param ADDRESS_2DARRAY[0] address[] memberAddressArray: 需要更改名称的成员地址数组
* @param STRING_ARRAY string[] memberNameArray: 需要更改的成员名称数组
*
* ID:11
*/
BATCH_CHANGE_MEMBER_NAMES,

/** 
* @notice 批量添加紧急代理操作
* @param Plugin[] pluginList: 插件数组
*
* ID:12
*/
BATCH_ADD_PLUGINS,

/** 
* @notice 批量启用插件操作
* @param UINT256_ARRAY[0] uint256[] pluginIndexArray: 需要启用的插件索引数组
* @param BOOL_ARRAY bool[] isBeforeOperationArray: 标志数组, 指示插件是否为操作前插件
*
* ID:13
*/
BATCH_ENABLE_PLUGINS,
*/

```

```

* @notice 批量禁用插件操作
* @param UINT256_ARRAY[0] uint256[] pluginIndexArray: 需要禁用的插件索引数组
* @param BOOL_ARRAY bool[] isBeforeOperationArray: 标志数组, 指示插件是否为操作前插件
* ID: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
* ID:16
*/
BATCH_SET_PARAMETERS,

/**
* @notice 批量添加可提现余额操作
* @param address[] addressArray: 需要添加可提现余额的地址数组
* @param uint256[] amountArray: 需要添加的可提现余额数量数组
* ID:17
*/
BATCH_ADD_WITHDRAWABLE_BALANCES,

/**
* @notice 批量减少可提现余额操作
* @param address[] addressArray: 需要减少可提现余额的地址数组
* @param uint256[] amountArray: 需要减少的可提现余额数量数组
* ID:18
*/
BATCH_REDUCE_WITHDRAWABLE_BALANCES,

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

/**
* @notice 批量支付以铸造代币操作
* @param ADDRESS_2DARRAY[0] address[] addressArray: 需要铸造代币的地址数组
* @param UINT256_2DARRAY[0] uint256[] tokenClassArray: 需要铸造代币的代币类别索引数组
* @param UINT256_2DARRAY[1] uint256[] amountArray: 需要铸造的代币数量数组
* @param UINT256_2DARRAY[2] uint256[] priceArray: 每个代币类别铸造的价格
* @param UINT256_2DARRAY[3] uint256[1] dividendableFlag: 标志, 指示支付是否可分红。1为是 (支付购买
* ID:20
*/
BATCH_PAY_TO_MINT_TOKENS,

```

```

/**
 * @notice 支付一些现金以转移代币（可用作商品币）
 * @param ADDRESS_2DARRAY[0] address[] toAddressArray: 需要转移代币到的地址数组
 * @param UINT256_2DARRAY[0] uint256[] tokenClassArray: 需要转移代币的代币类别索引数组
 * @param UINT256_2DARRAY[1] uint256[] amountArray: 需要转移的代币数量数组
 * @param UINT256_2DARRAY[2] uint256[] priceArray: 每个代币类别转移的价格
 * @param UINT256_2DARRAY[3] uint256[1] dividendableFlag: 标志，指示支付是否可分红。1为是（支付购买）
 * ID:21
 */
BATCH_PAY_TO_TRANSFER_TOKENS,

/**
 * @notice 添加一组地址作为紧急代理
 * （可用作具有新唯一代币类别的产品NFT）
 * @param ADDRESS_2DARRAY[0] address[] 需要添加为紧急代理的地址数组
 * ID:22
 */
ADD_EMERGENCY,

/**
 * @notice 从合约的现金余额中提现现金
 * @param address[] addressArray: 需要提现现金到的地址数组
 * @param uint256[] amountArray: 需要提现的现金数量数组
 * ID:23
 */
WITHDRAW_CASH_TO,

/**
 * @notice 调用紧急代理处理紧急情况
 * @param UINT256_2DARRAY[0] address[] addressArray: 需要调用的紧急代理索引数组
 * ID:24
 */
CALL_EMERGENCY,

/**
 * @notice 使用给定的abi调用合约
 * @param address contractAddress: 需要调用的合约地址
 * @param bytes abi: 需要调用的函数的abi
 * ID:25
 */
CALL_CONTRACT_ABI,

/**
 * @notice 支付一些现金
 * @param uint256 amount: 需要支付的现金数量
 * @param uint256 paymentType: 需要支付的现金类型，0为以太币/matic/原生代币
 * 1为USDT, 2为USDC（目前仅支持0），3为DAI等
 * @param uint256 dividendable: 标志，指示支付是否可分红,
 * 0为否（支付投资），1为是（支付购买）
 * ID:26
 */
PAY_CASH,

```

```

* @notice 计算分红并提供给代币持有者
* 通过将分红添加到每个代币持有者的可提现余额中
*
* ID:27
*/
OFFER_DIVIDENDS,

/***
* @notice 从可提现分红余额中提取分红
* @param address[] addressArray: 需要提取分红到的地址数组
* @param uint256[] amountArray: 需要提取的分红数量数组
* ID:28
*/
WITHDRAW_DIVIDENDS_TO,

/***
* @notice 设置所有转移操作的批准通过地址
* @param address: 需要设置所有转移操作批准的地址
* ID:29
*/
SET_APPROVAL_FOR_ALL_OPERATIONS,

/***
* @notice 批量销毁代币并退款
* @param UINT256_2D[0] uint256[] tokenClassArray: 需要从中销毁代币的代币类别索引数组
* @param UINT256_2D[1] uint256[] amountArray: 需要销毁的代币数量数组
* @param UINT256_2D[2] uint256[] priceArray: 每个代币类别销毁的价格
* ID:30
*/
BATCH_BURN_TOKENS_AND_REFUND,

/***
* @notice 永久地将存储IPFS哈希添加到存储列表中
* @param STRING_2DARRAY[0] address: 需要设置所有现金提取操作批准的地址
* ID:31
*/
ADD_STORAGE_IPFS_HASH,

/***
* 以下两个操作可以在投票等待过程中使用
*/

/***
* @notice 为投票等待的程序投票
* @param bool[] voteArray: 每个程序的投票数组
* ID:32
*/
VOTE,

/***
* @notice 执行已经投票并获批的程序
* ID:33
*/
EXECUTE_PROGRAM,

```

```

    /**
     * @notice 紧急模式终止。紧急代理在此操作后不能做任何事情
     * ID:34
     */
    END_EMERGENCY,

    /**
     * @notice 将合约升级到新的合约地址
     * @param ADDRESS_2DARRAY[0][0] 新合约的地址
     * ID:35
     */
    UPGRADE_TO_ADDRESS,

    /**
     * @notice 接受当前DARCs从旧合约地址升级
     * @param ADDRESS_2DARRAY[0][0] 旧合约的地址
     * ID:36
     */
    CONFIRM_UPGRADED_FROM_ADDRESS,

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

    /**
     * @notice 批量支付转移代币操作
     * ID:38
     */
    op_BATCH_PAY_TO_TRANSFER_TOKENS
}

```

附录 2：程序和操作的参考设计

```

    /**
     * 操作的参数或操作数
     */
    struct Param {
        uint256[] UINT256_ARRAY;
        address[] ADDRESS_ARRAY;
        string[] STRING_ARRAY;
        bool[] BOOL_ARRAY;
        VotingRule[] VOTING_RULE_ARRAY;
        Plugin[] PLUGIN_ARRAY;
        MachineParameter[] PARAMETER_ARRAY;
        uint256[][] UINT256_2DARRAY;
        address[][] ADDRESS_2DARRAY;
    }

    /**
     * 要执行的操作，包括操作者地址、操作码和参数
     */

```

```

struct Operation {
    address operatorAddress;
    EnumOpcode opcode;
    Param param;
}

< /**
 * 要执行的程序，包括程序操作者地址和操作数组
 */
struct Program {
    address programOperatorAddress;

    /**
     * @notice operations: 要执行的操作数组
     */
    Operation[] operations;
}

```

附录 3：插件的参考设计

```

< /**
 * 条件节点类型
 */
enum EnumConditionNodeType { UNDEFINED, EXPRESSION, LOGICAL_OPERATOR, BOOLEAN_TRUE, BOOLEAN_FALSE}

< /**
 * 逻辑操作符类型
 */
enum EnumLogicalOperatorType {UNDEFINED, AND, OR, NOT }

enum 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。
     */
    UNDEFINED,

    /**
     * 操作被批准，但必须在沙盒中执行以检查操作
     * 在当前机器状态下是否有效。
     * 只有BEFORE操作插件系统可能返回SANDBOX_NEEDED。
     */
    SANDBOX_NEEDED,

    /**
     * 操作被拒绝，在此级别应该被拒绝。
     * BEFORE和AFTER操作插件系统都可能返回NO。
     */
    NO,

    /**
     * 决定待定，应在此级别创建投票项。
     */
    PENDING
}

```

```

 * 只有AFTER操作插件系统可能返回VOTING_NEEDED。
 */
VOTING_NEEDED,

```

```

 /**
 * 操作被批准，应跳过沙盒检查。
 * 只有BEFORE操作插件系统可能返回YES_AND_SKIP_SANDBOX。
 */
YES_AND_SKIP_SANDBOX,

```

```

 /**
 * 操作最终在此级别被批准。
 * 只有AFTER操作插件系统可能返回YES。
 */
YES
}

/**
 * 条件节点表达式参数
 */
struct NodeParam {
    uint256[] UINT256_ARRAY;
    address[] ADDRESS_ARRAY;
    string[] STRING_ARRAY;
    uint256[][] UINT256_2DARRAY;
    address[][] ADDRESS_2DARRAY;
    string[][] STRING_2DARRAY;
}

/**
 * 条件节点结构
 */
struct ConditionNode {
    /**
     * 当前条件节点索引
     */
    uint256 id;

    /**
     * 当前条件节点的类型
     */
    EnumConditionNodeType nodeType;

    /**
     * 当前条件节点的逻辑操作符
     */
    EnumLogicalOperatorType logicalOperator;

    /**
     * 当前条件节点的条件表达式
     */
    EnumConditionExpression conditionExpression;

    /**
     * 当前条件节点的子节点列表
     */
}

```

```

    uint256[] childList;

    /**
     * EXPRESSION节点参数数组
     */
    NodeParam param;
}

/**
 * 插件结构
 */
struct Plugin {
    /**
     * 当前条件节点的返回类型
     */
    EnumReturnType returnType;

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

    /**
     * 条件二进制表达式树向量
     */
    ConditionNode[] conditionNodes;

    /**
     * 如果返回类型为VOTING_NEEDED，当前插件的投票规则id
     */
    uint256 votingRuleIndex;

    /**
     * 插件备注
     */
    string note;

    /**
     * 指示插件是否启用的布尔值
     */
    bool bIsEnabled;

    /**
     * 指示插件是否已删除的布尔值
     */
    bool bIsInitialized;

    /**
     * 指示插件是操作前插件还是操作后插件的布尔值
     */
    bool bIsBeforeOperation;
}

```

概念	DARC	法律公司实体
章程	1. 设计所有与章程相关的核心插件。 2. 设计并运行一个包含以下内容的法规脚本程序： 2.1 添加并启用所有插件到 DARC。 2.2 添加紧急代理。 2.3 添加档案信息。	编写并签署章程文件
发行股票	设计并运行一个包含以下内容的法规脚本程序： 1. 初始化代币类别和信息。 2. 铸造普通股代币给股东。 3. 铸造董事会成员代币给董事会成员。	发行股票证书
投资	1. 设计并运行一个包含以下内容的法规脚本程序： 1.1 铸造普通股代币给股东。 1.2 禁用不必要的之前的股票相关插件。 1.3 添加并启用新的协议插件。 1.4 支付特定数量的代币。 2. 投票并批准程序。 3. 执行批准的程序。	发行股票证书
雇佣	设计并运行一个包含以下内容的法规脚本程序： 1. 如有必要，添加一个新的成员资格级别。 2. 添加新插件以允许具有某种成员资格的操作者每月提现现金。 3. 添加新插件以允许具有某种成员资格的操作者每年铸造 RSU 代币。 4. 将新员工添加到成员资格中。 5. 允许经理级别的操作者解雇员工（从成员资格级别中移除员工）。	签署雇佣合同，支付工资和期权/RSUs
购买	设计并运行一个包含以下内容的法规脚本程序： 1. 向特定地址支付现金 (外部拥有的账户、另一个 DARC、或其他智能合约)。	电子资金转账 (电子支票、电汇等)
分红	设计并运行一个包含以下内容的法规脚本程序： 1. 提供分红。	通过支付 每季度或每年支付分红
股票交易	设计并运行一个包含以下内容的法规脚本程序： 1. 将代币从一个地址转移到另一个地址。	私下股票交易， 或在股票交易所 或场外交易 (OTC) 市场上交易
接受支付	向 DARC 发送或转移原生代币，或从其他 DARCs 或智能合约中提取原生代币。	接受来自客户、供应商和其他方的支付
治理	设计并运行一个包含以下内容的法规脚本程序： 1. 设计投票规则并添加到 DARC。 2. 添加插件以定义需要投票的场景。 当程序处于挂起状态且需要投票时进行投票。	董事会会议， 股东会议， 以及其他公司治理
争议和诉讼	1. 添加并启用紧急代理。 2. 召唤紧急代理以解决争议。 3. 联系紧急代理，提供必要的信息，并支付费用。 4. 紧急代理操作者接管 DARC 管理，解决问题或恢复状态。	法律行动， 仲裁和调解

表 3: DARC 与法律公司实体的操作和管理比较

级别	名称	总供应量	投票权重	分红权重
0	A 类代币	400	1	1
1	B 类代币	60	10	1
2	独立董事	1	0	0
3	董事会	5	1	0

表 4: DARC Y 中的代币分配结构

级别	角色
1	联合创始人
2	重要股东
3	高管
4	经理
5	员工
6	离职员工

表 5: 一个分层成员资格结构

地址	角色	名称	是否暂停
0x0AC..03	1 (联合创始人)	Ann	否
0x156..21	2 (重要股东)	Banana Capital	否
0x918..1B	3 (高管)	Tom	否
0x4E1..90	4 (经理)	Jack	否
0x510..0B	5 (员工)	Bob	否
0x113..C7	6 (离职员工)	Tim	否

表 6: DARC X 的成员资格表