1 插件

1.1 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。

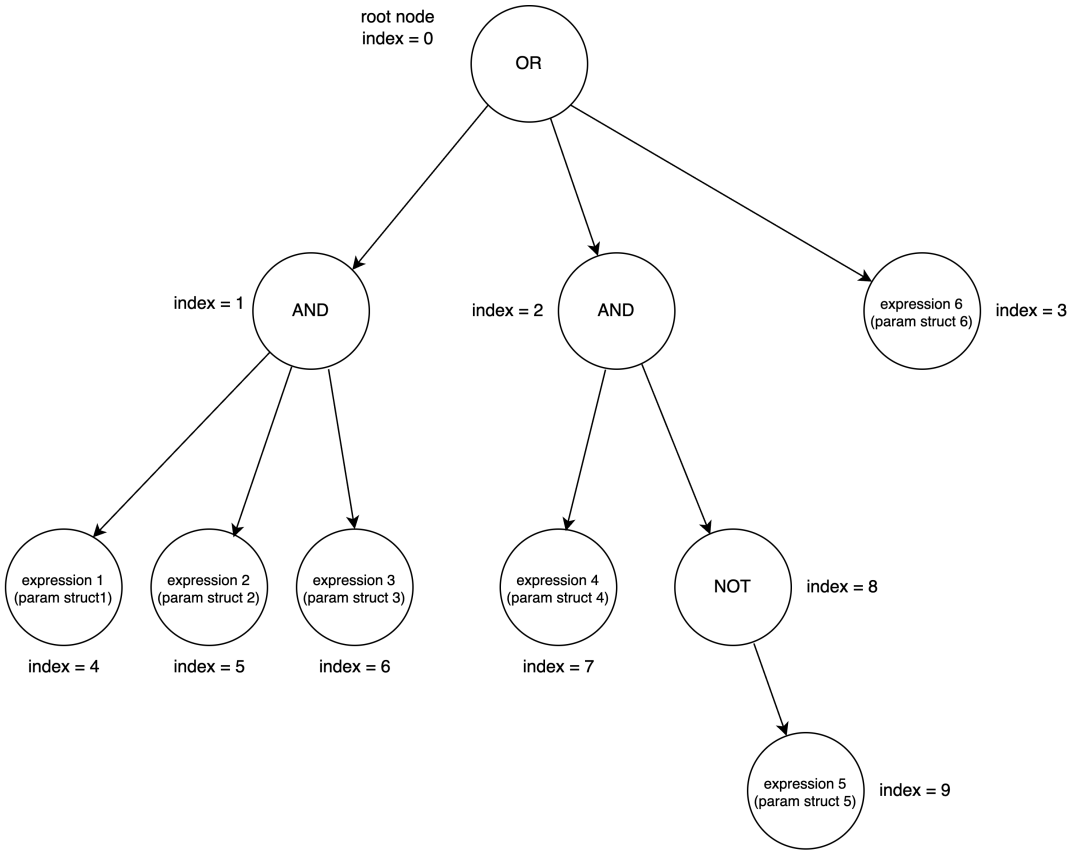
每个插件都有一个条件节点数组，条件节点按顺序存储。根节点对应于索引 0 处的节点，即第一个节点。条件节点数组遵循以下原则：

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 **??** 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。



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 1: 条件节点和表达式树

1.2 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。
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 **??** 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 **??** 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。

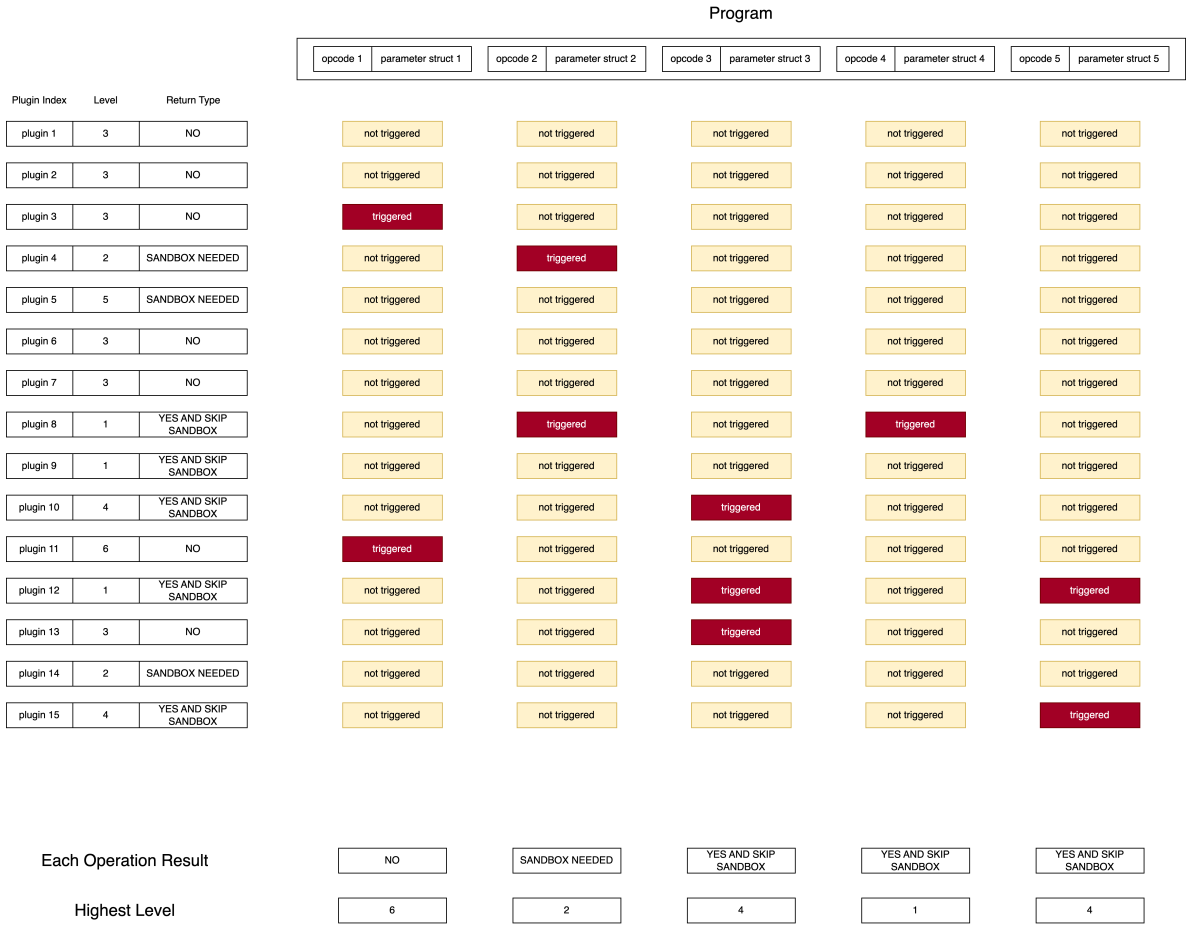


图 2: 带有操作前插件的程序判断

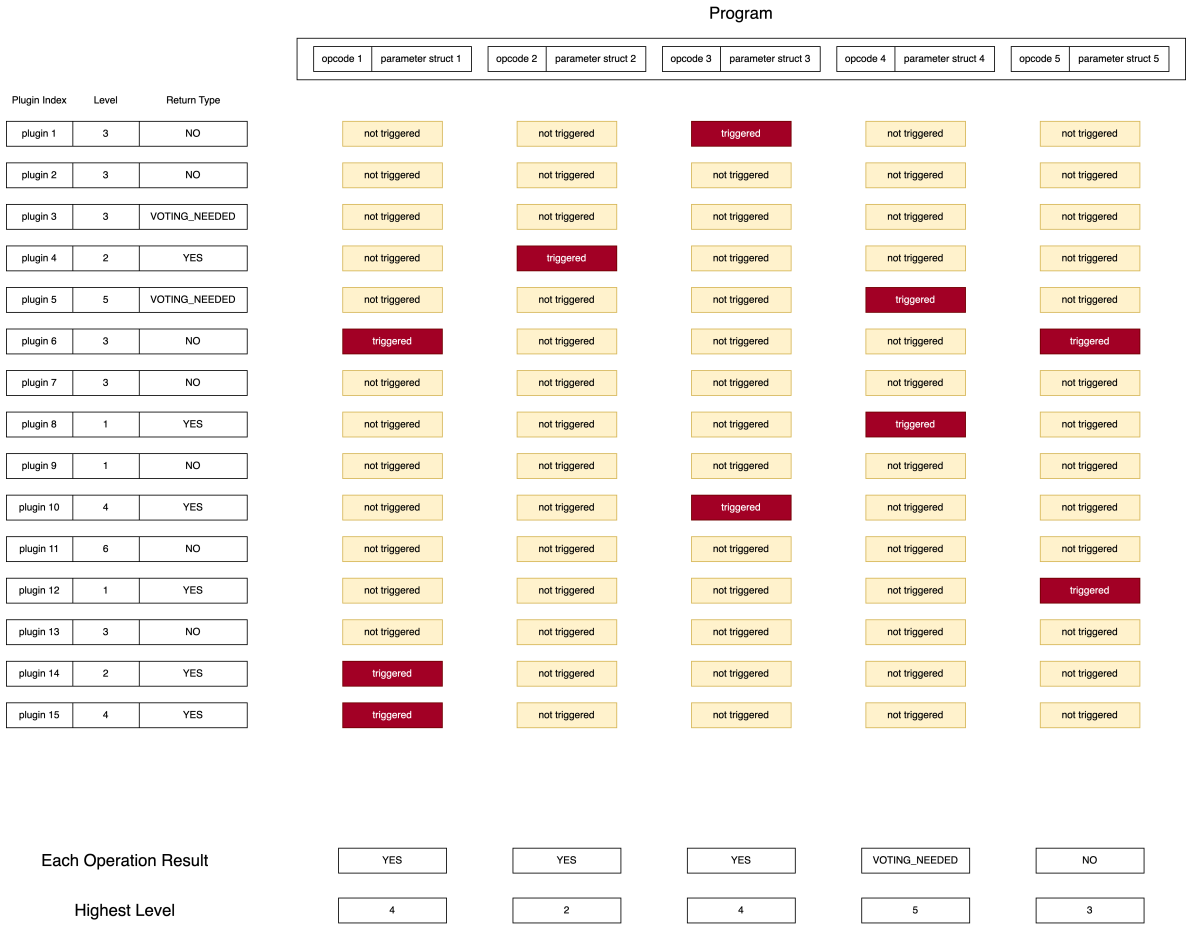


图 3: 带有操作后插件的程序判断