1 Plugins

1.1 Design of Plugins

Plugin is the law in the DARC Protocol, and all programs and operations within DARC must adhere to the restrictions imposed by all plugins. For an individual plugin, it is required to follow the logic outlined in the pseudo code below:

```
if plugin.condition:
    return plugin.returnType
```

For the DARC protocol, the main difference between before-operation plugins and after-operation plugins lies in their return types. For before-operation plugins, as they determine whether a certain operation should be executed directly, rejected outright, or entered into a sandbox, they have three distinct return types that serve as their final decisions:

1. `NO`. When the condition of a before-operation plugin is triggered, the plugin's decision for that operation is `NO`. This decision indicates that the plugin believes the operation violates its rules, and therefore, it is rejected outright before entering the sandbox for execution.

2. `SANDBOX_NEEDED`. When the condition of a before-operation plugin is triggered, the plugin's decision for that operation is `SANDBOX_NEEDED`. This decision indicates that the plugin cannot determine whether the operation should be accepted or rejected. The plugin is aware that the operation needs to be evaluated in the sandbox by after-operation plugins, and thus, the decision is made to let the operation proceed to the sandbox for further evaluation.

3. `YES_AND_SKIP_SANDBOX`. When the condition of a before-operation plugin is triggered, the plugin's decision for that operation is `YES_AND_SKIP_SANDBOX`. This decision indicates that the plugin has determined that the operation should be approved and does not require execution in the sandbox. Therefore, the operation can proceed directly without going through the sandbox.

For after-operation plugins, since the program has been executed in the sandbox and voting can commence, these plugins can have three return types as their final decisions:

1. `NO`. When the condition of an after-operation plugin is triggered, the plugin's decision for the operation is `NO`. This decision indicates that the plugin believes the operation violates its rules and should be rejected outright.

2. `VOTING_NEEDED`. When the condition of an after-operation plugin is triggered, the plugin's decision for the operation is VOTING_NEEDED. This decision indicates that the plugin believes the operation requires a vote, and the operation needs to initialize a voting item based on the voting rule specified by this plugin.

3. `YES`. When the condition of an after-operation plugin is triggered, the plugin's decision for the operation is `YES`. This decision indicates that the plugin believes, based on its rules, the operation should be allowed to proceed.

Each plugin has a condition node array, where condition nodes are stored in sequence. The root node corresponds to the node at index 0, which is the first node. The condition node array follows the following principles:

1. The type of each node can be a boolean operator or an expression;

2. For boolean operators, the type must be set as one of AND, OR, or NOT;

3. For AND and OR operators, at least two valid child node indices must be specified in the child node list;

4. For the NOT operator, a unique child node index must be specified in the child node list;

5. For expression nodes, valid condition expression parameters consistent with the expression must be set;

6. For expression nodes, the length of their child node list must be 0, meaning no child nodes are allowed.

Figure 1 is an example illustrating how a condition expression binary tree is serialized into a condition node array.

Additionally, a plugin needs to set two parameters: one is the 'level', representing the priority of the plugin within the entire plugin system. For the same operation, the judgment system traverses all plugins, and it is possible that at least two or more plugins are triggered. In such cases, if the levels of the plugins are different, the judgment system will consider the plugin with the higher level as the final determination.

The other parameter is the 'voting rule index,' which points to a specific index in the voting rule array. When the final decision of a plugin is `VOTING_NEEDED`, the plugin automatically requests the DARC protocol to use the voting rule indicated by the voting rule index for initializing the voting item. If the return type of the plugin is not `VOTING_NEEDED`, the voting rule index will be ignored.

1.2 Plugins and the Judgement System

For the DARC protocol, the judgment system needs to undergo two assessments: one through before-operation plugins and another after the program has run completely in the sandbox, followed by evaluation through after-operation plugins. The reason for this design is that, without a sandbox and relying solely on a set of plugins for judgment, it becomes challenging to anticipate the behavior of the program. As a result, it is not possible to protect against modifications to special states in the DARC protocol.

For example, in a DARC instance where shareholder X is required to permanently hold 15% voting rights and 10% dividend rights, when designing a plugin, it is impossible to predict the state of the DARC instance after the execution of operations like mint tokens or burn tokens. This uncertainty poses challenges in ensuring that shareholder X maintains permanent ownership of 15% voting rights and 10% dividend rights. Only by running the operation in the sandbox and then re-evaluating the state of the sandbox, can such modifications be prevented.

In another scenario, if there is a need to ensure that a DARC instance reserves 10,000 native tokens permanently before January 1, 2035, the correct detection and prevention of operations such as paying dividends or withdrawing cash can only be guaranteed by executing these operations in the sandbox. After all operations are completed in the sandbox, the judgment system performs a second evaluation through after-operation plugins. Without a sandbox and relying solely on plugins, the design of such a mechanism would be overly complex.

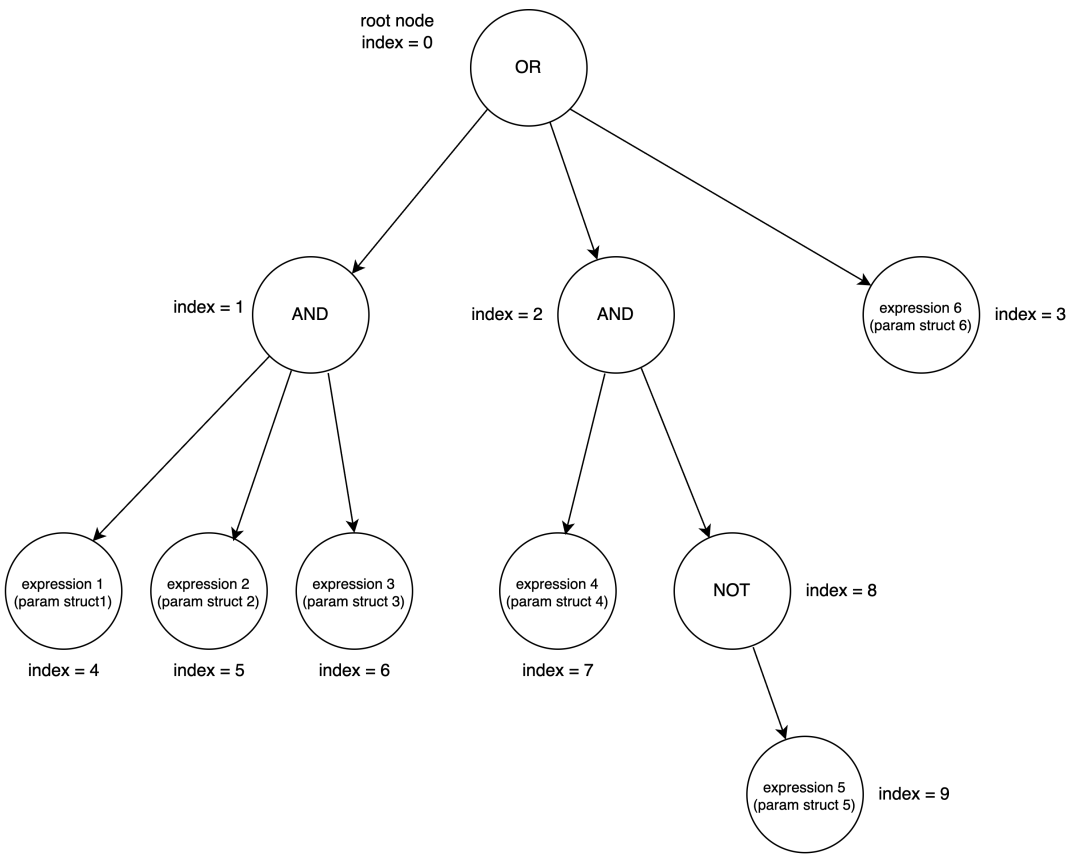
If there are only after-operation plugins and a sandbox without before-operation plugins, it would not be feasible. This is because the running cost of the sandbox is very high. It not only requires the program to run completely in the sandbox but also involves initializing the sandbox by fully replicating the internal state of the entire DARC instance. This process incurs a significant amount of gas fees.

For the majority of simple operations that can be approved without the need for running in the sandbox, it is more cost-effective to have rules established directly in before-operation plugins. These rules can leverage factors such as the operation, operator, timestamp, etc., for a straightforward approval or rejection, preventing them from entering the sandbox and saving costs.

For before-operation plugins, whenever a program is submitted to the DARC protocol, the judgment system sequentially checks each operation. For each operation, the judgment system traverses each before-operation plugin and obtains a single judgment result. Finally, it aggregates the results for all operations to determine the overall result for the entire program. This decision dictates whether the program needs to run in the sandbox (`SANDBOX_NEEDED`), be rejected outright (`NO`), or run directly without the need for the sandbox (`YES_AND_SKIP_SANDBOX`). For before-operation judgment, the following rules are followed:

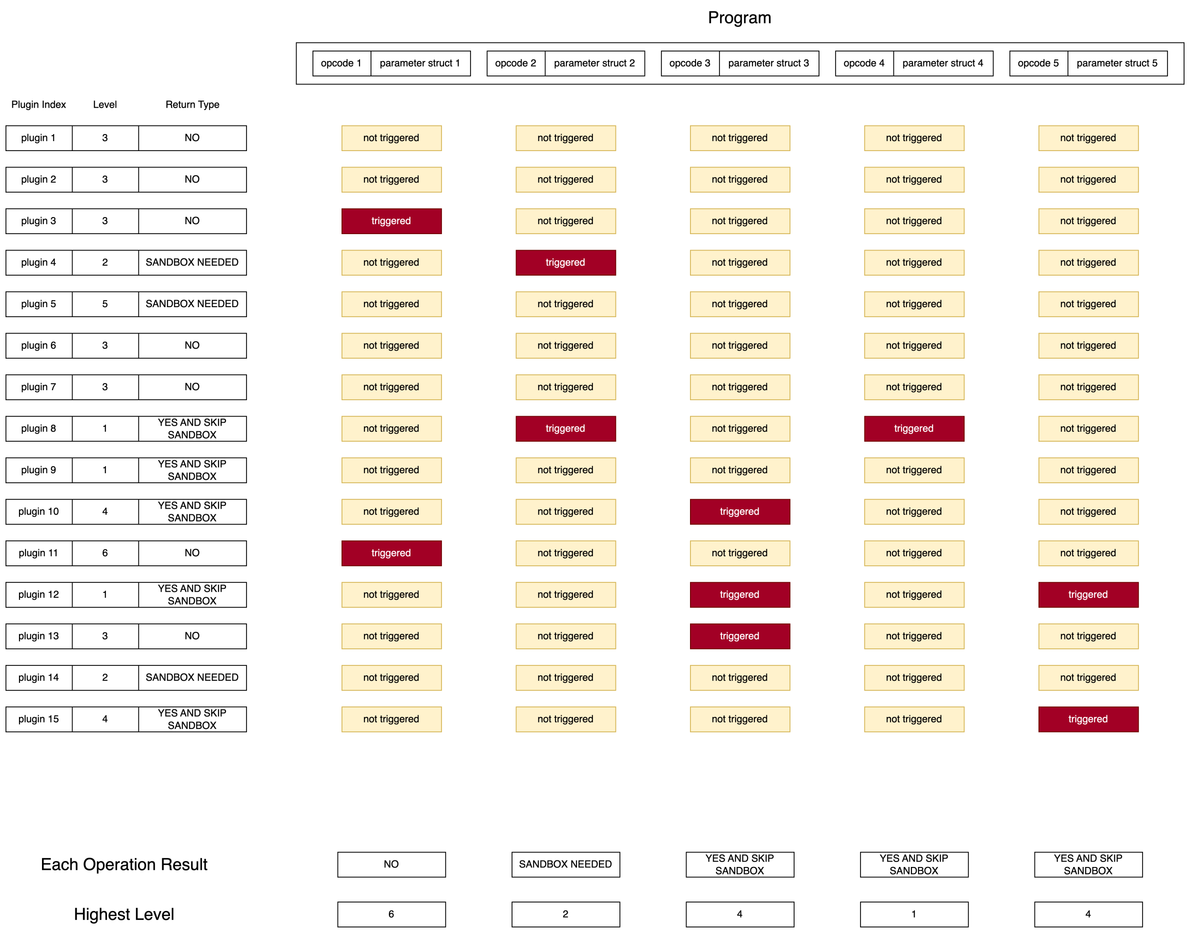
1. If any operation is judged by the judgment system as `NO`, the entire program is rejected with a result of `NO`.

2. If none of the operations is judged by the judgment system as `NO`, and at least one operation is judged as `SANDBOX_NEEDED`, the entire program is required to run in the sandbox, and the result is `SANDBOX_NEEDED`.



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]	[]

Figure 1: Condition Nodes and Expression Tree



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	not triggered	not triggered	not triggered
plugin 2	3	NO	not triggered	not triggered	not triggered	not triggered	not triggered
plugin 3	3	NO	triggered	not triggered	not triggered	not triggered	not triggered
plugin 4	2	SANDBOX NEEDED	not triggered	triggered	not triggered	not triggered	not triggered
plugin 5	5	SANDBOX NEEDED	not triggered	not triggered	not triggered	not triggered	not triggered
plugin 6	3	NO	not triggered	not triggered	not triggered	not triggered	not triggered
plugin 7	3	NO	not triggered	not triggered	not triggered	not triggered	not triggered
plugin 8	1	YES AND SKIP SANDBOX	not triggered	triggered	not triggered	triggered	not triggered
plugin 9	1	YES AND SKIP SANDBOX	not triggered	not triggered	not triggered	not triggered	not triggered
plugin 10	4	YES AND SKIP SANDBOX	not triggered	not triggered	triggered	not triggered	not triggered
plugin 11	6	NO	triggered	not triggered	not triggered	not triggered	not triggered
plugin 12	1	YES AND SKIP SANDBOX	not triggered	not triggered	triggered	not triggered	triggered
plugin 13	3	NO	not triggered	not triggered	triggered	not triggered	not triggered
plugin 14	2	SANDBOX NEEDED	not triggered	not triggered	not triggered	not triggered	not triggered
plugin 15	4	YES AND SKIP SANDBOX	not triggered	not triggered	not triggered	not triggered	triggered
Each Operation Result			NO	SANDBOX NEEDED	YES AND SKIP SANDBOX	YES AND SKIP SANDBOX	YES AND SKIP SANDBOX
Highest Level			6	2	4	1	4

Figure 2: Judgement on program with before-operation plugins

3. If all operations are judged by the judgment system as `YES_AND_SKIP_SANDBOX`, the entire program is approved with a result of `YES_AND_SKIP_SANDBOX`, the entire program can skip the sandbox.

Figure 2 illustrates how individual operations within a program are judged by before-operation plugins, resulting in judgment outcomes.

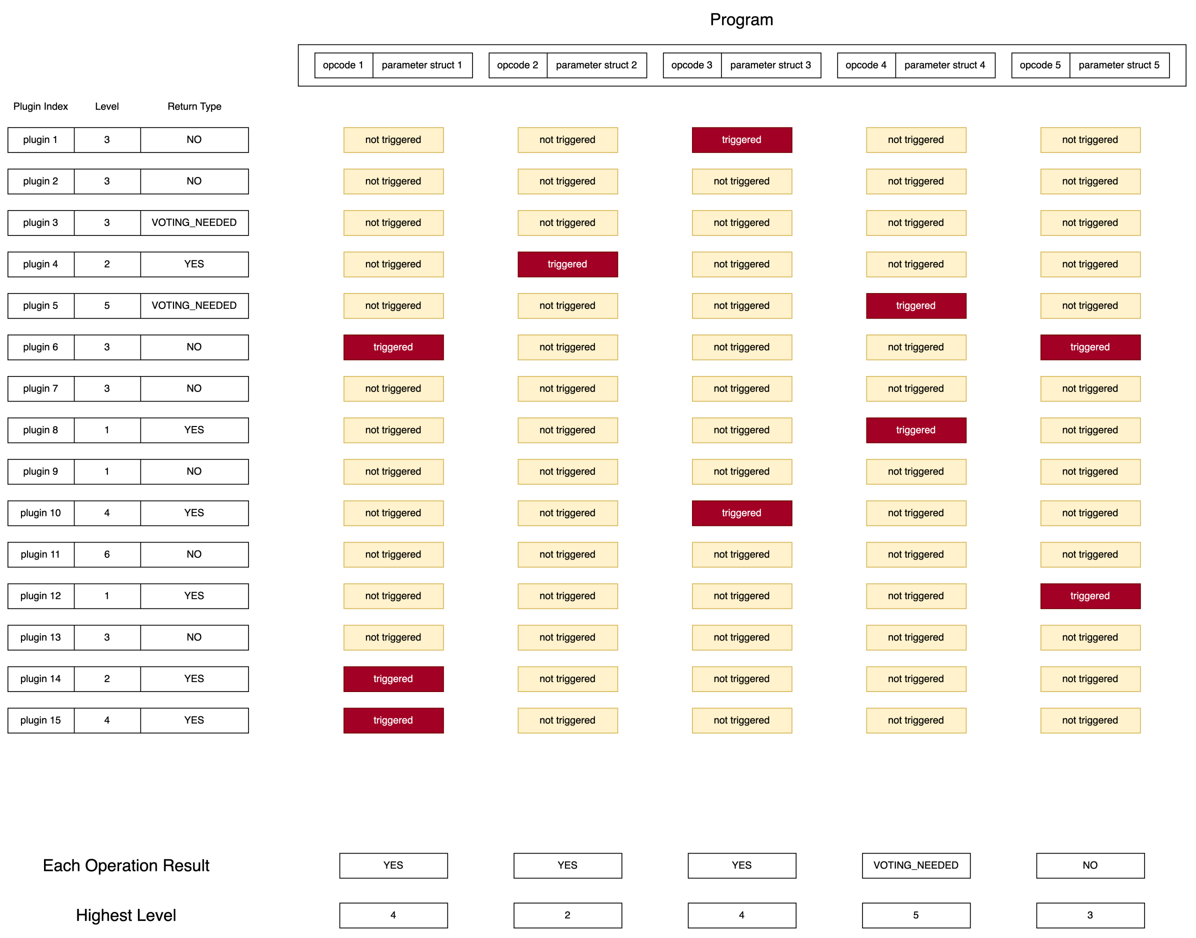
For after-operation plugins, once a program has completed its full execution in the sandbox, the judgment system sequentially examines each operation. For each operation, the judgment system traverses each after-operation plugin, obtaining an individual judgment result. Ultimately, the results for all operations are aggregated to determine the final program result, deciding whether the program needs approval through voting (`VOTING_NEEDED`), should be rejected outright (`NO`), or can proceed directly (`YES`). The following rules apply to after-operation judgment:

1. If any operation is judged by the judgment system as `NO`, the entire program is rejected with a result of `NO`.

2. If none of the operations have been determined as `NO` by the judgement system, and at least one operation has been determined as `VOTING_NEEDED`, the final judgement for the entire program is `VOTING_NEEDED`. This program will be placed into the pending program category, and the DARC protocol must initiate the voting system to decide on approval or rejection.

3. If all operations are judged by the judgment system as `YES`, the entire program is approved with a result of `YES`, the entire program can be executed directly.

Figure 2 illustrates how individual operations within a program are judged by after-operation plugins, resulting in judgment outcomes.



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	not triggered	not triggered	not triggered	not triggered
plugin 3	3	VOTING_NEEDED	not triggered	not triggered	not triggered	not triggered	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	not triggered	not triggered	not triggered	not triggered
plugin 8	1	YES	not triggered	not triggered	not triggered	triggered	not triggered
plugin 9	1	NO	not triggered	not triggered	not triggered	not triggered	not triggered
plugin 10	4	YES	not triggered	not triggered	triggered	not triggered	not triggered
plugin 11	6	NO	not triggered	not triggered	not triggered	not triggered	not triggered
plugin 12	1	YES	not triggered	not triggered	not triggered	not triggered	triggered
plugin 13	3	NO	not triggered	not triggered	not triggered	not triggered	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

Figure 3: Judgement on program with after-operation plugins