Automata

Tony Jin
nijynot@gmail.com

December 18, 2020
WORKING DRAFT 1

Abstract

N/A

1 Introduction

Automata is a protocol for tokenizing GovernorAlpha [1] delegations or -votes and are denominated as dTokens and vTokens respectively. dTokens can be exercised to delegation to an Autonomous Proposal while vTokens can be exercised to vote on any active proposal. Minting dTokens or vTokens is done by depositing the underlying governance ERC-20 [2] token and 200% of the deposit amount is returned.

Automata enables new ways to participate and engage in on-chain governance. Liquidity providers earn a fee (when dTokens or vTokens are burned) and governance participants can leverage higher amounts of influence on proposals by paying a one-off premium.

This paper will outline the mechanics of tokenizing delegations and votes by pool sharding. Future work outside the scope of this paper is discussed at the end.

2 Pairing ERC-20 tokens

By using a pairing function, $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, an ordered pair can represent the balance $b \in \mathbb{N}$ of a pairing ERC-20 token.

In this paper, we will use Matthew Szudzik's Elegant Pairing Function [3] which is defined by

$$\pi(x, y) = \begin{cases} x^2 + x + y, & \text{if } x \geq y \\ x + y^2, & \text{otherwise} \end{cases}$$

where $x, y > 0$ and $x, y \in \mathbb{N}$. Similarly, there's the inverse unpairing function given by

$$\pi^{-1}(z) = \begin{cases} (\lfloor\sqrt{z}\rfloor, z - \lfloor\sqrt{z}\rfloor^2 - \lfloor\sqrt{z}\rfloor), & \text{if } z - \lfloor\sqrt{z}\rfloor^2 \geq \lfloor\sqrt{z}\rfloor \\ (z - \lfloor\sqrt{z}\rfloor^2, \lfloor\sqrt{z}\rfloor), & \text{otherwise.} \end{cases}$$

Transfering tokens from one account to another requires re-calculation of the balance and incurs higher gas fees than usual. As such, a more desireable pairing function on Ethereum would be a function that satisfies an isomorphic property

$$\pi(x+y, z+w) = \pi(x, y) + \pi(z, w)$$

where $x, y, z, w \in \mathbb{N}$ and such that the function is bijective. The isomorphism optimization is deemed out of scope for this paper we refer to Section 4.1 for more details.

3 Mechanism

3.1 dTokens

3.1.1 Minting dTokens

dTokens are minted proportionally to the amount of the underlying ERC-20 token deposited. The amount of dTokens minted can be calculated by

$$T_d = T_u \cdot c,$$

where T_d is dTokens, T_u is underlying tokens deposited and c is collateral factor.

3.1.2 Burning dTokens (Exercising)

Upon burning dTokens, delegations will be sent to an Autonomous Proposal [4] and the function `propose()` can be called, if and only if

$$\alpha + \alpha_s \geq \rho,$$

where α is amount of delegations in Autonomous Proposal, α_s is the amount of scheduled dTokens and ρ is GovernorAlpha's `proposalThreshold()` value.

If $\alpha + \alpha_s < \rho$, then the dTokens will be locked in an Scheduler contract until $\alpha + \alpha_s \geq \rho$ is satisfied and `propose()` can be called.

3.1.3 Fee

Definition 3.1 (Unit fee)**.** Let b be amount of dTokens burned and R_t be the total supply of locked ERC-20 tokens. The unit fee is then defined as

$$u = \frac{b}{R_t},$$

which represents dTokens earned per locked ERC-20 token.

Let $(b_k)_{k=0}^{n}$ be a sequence of burned dTokens, let $(R_{t_k})_{k=0}^{n}$ be the total supply of locked ERC-20 tokens at the timepoint of a burn and let $(u_k)_{k=0}^{n}$, where $u_k = \frac{b_k}{R_{u_k}}$ be a sequence of unit fees.

Definition 3.2 (Single-variate fee)**.** Let $(u_k)_{k=0}^{n}$ be a sequence of unit fees and c be sum of that sequence $\sum_{k=0}^{n} u_k$. The single-variate fee function $\phi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is then defined as

$$\phi(t, x) = t\left(c - \sum_{k=0}^{x} u_k\right),$$

where t is number of locked ERC-20 tokens and $x \in \mathbb{N}$.

Remark. The single-variate fee is only useful for a single deposit or redemption of ERC-20 tokens and its earned fee. Support for trading liquidaty and additional deposits into the pool requires a multi-variate fee calculation equation.

Definition 3.3 (Multi-variate fee)**.** Let $(u_k)_{k=0}^{n}$ be a sequence of unit fees and c be sum of that sequence $\sum_{k=0}^{n} u_k$.

The multi-variate fee function $\Phi : \mathbb{N}^m \times \mathbb{N}^m \to \mathbb{N}$ is then defined as

$$|\mathbf{t}| = \sum_{i=0}^{m} t_i$$
$$\Phi(\mathbf{t}, \mathbf{e}) = |\mathbf{t}|\left(c - \frac{1}{|\mathbf{t}|}\sum_{i=0}^{m}\left(t_i \cdot \sum_{s=0}^{e_i} u_i\right)\right),$$

where m is discrete amount of deposits, $|t| = \sum_{k=0}^{m} k_i$, $t = (t_0, ..., t_m) \in \mathbb{N}^m$ is vector of amount of ERC-20 tokens locked and $e = (e_1, ..., e_m) \in \mathbb{N}^m$ is indicies for the partial unit fee sum.

Remark. The multi-variate definition is derived from adding two single-variate fees together. To convince the reader of an additive property of the fee function, we propose the following lemma.

Lemma 3.1. *Let $(u_k)_{k=0}^{n}$ be a sequence of unit fees.*

If $\Phi(\mathbf{t}, \mathbf{e})$ and $\Phi(\mathbf{v}, \mathbf{f})$ are two multi-variate fees, then there exists sum of the two fees which can be described as

$$\Phi(\mathbf{t} + \mathbf{v}, \mathbf{g}) = \Phi(\mathbf{t}, \mathbf{e}) + \Phi(\mathbf{v}, \mathbf{f}),$$

where $0 \leq \mathbf{g} \in \mathbf{N}$.

Proof. Let $\mathbf{w} = \mathbf{t} \oplus \mathbf{v}$ be the concatenation of the two vectors and $\mathbf{j} = \mathbf{e} \oplus \mathbf{f}$.

$$\begin{aligned}
\Phi(\mathbf{t}, \mathbf{e}) + \Phi(\mathbf{v}, \mathbf{f}) &= |\mathbf{t}|\left(c - \frac{1}{|\mathbf{t}|}\sum_{i=0}^{m}\left(t_i \sum_{s=0}^{e_i} u_i\right)\right) + |\mathbf{v}|\left(c - \frac{1}{|\mathbf{v}|}\sum_{i=0}^{n}\left(v_i \sum_{s=0}^{f_i} u_i\right)\right) \\
&= (|\mathbf{t}| + |\mathbf{v}|)c - \left(\sum_{i=0}^{m}\left(t_i \sum_{s=0}^{e_i} u_i\right) + \sum_{i=0}^{n}\left(v_i \sum_{s=0}^{f_i} u_i\right)\right) \\
&= (|\mathbf{t}| + |\mathbf{v}|)c - \left(\sum_{i=0}^{m+n}\left(w_i \sum_{s=0}^{j_i} u_i\right)\right)
\end{aligned}$$

□

3.1.4 Liquidity

After depositing the underlying ERC-20, liquidity tokens are sent to an address. The amount of liqudity tokens sent are determined by two variables, the amount of ERC-20 tokens locked and a partial sum of a unit fee sequence.

Definition 3.4 (Liquidity tokens)**.** Let $\Phi(\mathbf{t}, \mathbf{e})$ be a multi-variate fee. Then the liquidity tokens is defined by

$$l = \pi\left(|\mathbf{t}|, \textstyle\sum_{i=0}^{m}\left(t_i \sum_{s=0}^{e_i} u_i\right)\right),$$

for some pairing function $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

Remark. To calculate the amount of locked ERC-20 tokens or the fee accrued, apply the unpairing function to the liquidity tokens l.

3.1.5 Collateral factor

The collateral factor of 200%[1] exists to allow for liqidity providers to manage their risk and to increase efficiency of the underlying tokens. Given that not all liquidity providers might sell their dTokens on the market, other liqidity providers can take a higher risk profile by selling their whole dToken stake.

As delegations only get delegated when the `propose()` function can be called, delegation calls will never overlap and significant lockup of delegations will not occur. This implies a higher throughput of delegating, and undercollateralization of the underlying ERC-20 becomes possible.

3.2 vTokens

Minting, burning, fee calculation and liquidity tokens are calculated similarly to dTokens as for vTokens.

As GovernorAlpha does not support fractional voting[2], a more complex smart contract is required for exercising of vTokens.

3.2.1 Pool sharding

Sharding the pool into strictly smaller subsets allows for exercising for every number vTokens in $\mathbb{N}^{[1,n]}$. By splitting up the pool into pool shards where the maximum amount of ERC-20 tokens it can hold being

$$2^m, \quad \text{where } m \in \mathbb{N}^{[1,n-1]},$$

it becomes possible to represent every number as a combination of the pool shards.

Each pool shard has it's own fee distribution. This incentives the market to find an equilibruim for the amount of pool shards for a given 2^m. Pool sharding of vTokens is incompatible with the dToken pool and will need to be kept separate unless fractional voting is implemented in GovernorAlpha.

[1]The collateral factor can be adjusted by the Automata's governance system.

[2]Fractional voting allows an account to vote with a number less than the total votes available on the account.

4 Future work

4.1 Isomorphism

Both an isomorphic pairing function and fee function would reduce the gas fees when minting or burning d/vTokens. In other words, functions that satisfy

$$f(x + y) = f(x) + f(y)$$

and is bijective.

4.2 Fractional votes on GovernorAlpha

If voting with a fraction of your votes was possible, it'd reduce unnecessary complexities with Automata's delegation and voting. Delegating requires sending of the ERC-20 tokens to a separate contract first before delegating. Voting on Automata can only be done by sharding the pool into smaller discrete pool such thats a combination of the shards will cover the end-users' desired vote amount.

References

[1] Compound. *GovernorAlpha*. `https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/GovernorAlpha.sol`.

[2] Fabian Vogelsteller and Vitalik Buterin. *EIP-20: ERC-20 Token Standard*. `https://eips.ethereum.org/EIPS/eip-20`.

[3] Matthew Szudzik. *An Elegant Pairing Function*. `http://szudzik.com/ElegantPairing.pdf`.

[4] Compound. *Autonomous Proposals*. `https://github.com/compound-finance/autonomous-proposals`.