Tokenomics

Introduction 2

Evaluation of the Usefulness of Code 3

Math Definition of the Contribution Measures UCF and USF 4

UCF - Usefulness of Code in Terms of Profits 5

Useful Code Factor per Component 5

Useful Code Factor per Agent 6

USF - Usefulness of Services in Terms of Profits 7

Useful Code Factor per Service 7

Levels of Tokenomics already implemented 8

Level 1: Bonding 8

Math Definition of the Discount factor 8

Product function 10

Respecting inflation schedule 13

Level 2: Locking 16

Inflation distribution - rewards distribution and calculation 18

Inflation schedule 18

Inflation distribution - rewards distribution 21

How will inflation be distributed? 21

How rewards will be distributed? 22

How are rewards calculated? 22

Introduction

Autonolas ecosystem's scarcest resource is agent developers. Hence, tokenomics give due consideration to attracting and retaining developers, as well as facilitating the composability of their code contributions, to enable more than the sum of its parts. Additionally, tokenomics are geared towards attracting investors, or bonders, who can make their capital productive, by pairing it with code. Finally, we introduce a novel model of production which enables the protocol to own its own productive services and generate profits.

Autonolas' tokenomics can hence be broken down into enabling three key goals:

1. **Capital + Code**
   The first goal of Autonolas tokenomics is enabling the pairing of capital and code in a permissionless manner. Autonolas holds the following tenets: code on its own lacks an indication of its value; capital on its own is not productive. When combined, they reinforce each other. By pairing quality code with capital, Autonolas addresses a key problem faced in open-source development: developer time is a scarce resource that is not sufficiently well remunerated. As a by-product, capital allocation creates a signaling mechanism for the quality of contributed code across the protocol.

2. **Composability**
   The second and wider objective of Autonolas tokenomics is to enable and incentivize software composability. In a nutshell, one says that "a platform is composable if its existing resources can be used as building blocks and programmed into higher-order applications."[1] Composability of software components in an ecosystem enables exponential returns. The on-chain protocol extends this composability to the tokenomics, allowing NFTs which represent code and services to be used productively inside the protocol and across DeFi.

3. **Protocol-owned Services (PoSe)**
   The third goal is to create a flywheel which attracts more and more value, and provides truly decentralized services owned by the protocol, operated by the ecosystem and developed by developers around the world. This is achieved by leveraging a novel primitive to web-based business models whereby the protocol

[1] Jesse Walden, *4 eras of blockchain computing: degrees of composability* [11]

itself owns productive services and derives profits from them. Importantly, protocol-owned services provide the level of decentralization necessary for the longevity of the ecosystem in the face of uncertain regulation.

The protocol coordinates these three goals through a tradable utility token, OLA, and a *non-tradable* vote-escrowed virtualized-token, veOLA.

Next, we detail how Autonolas' tokenomics focus on two core roles in the ecosystem, namely, investors and developers. It does so by evaluating the usefulness of code and services organized by the protocol in terms of the profit they return to the protocol. Further components of our tokenomics are finally discussed.

Evaluation of the Usefulness of Code

We want to ensure that the growth of Protocol-owned Liquidity ("Capital") is proportional to the growth of the components and agents ("Code") productively deployed in services in the ecosystem.

To evaluate the code, the protocol can take advantage of several built-in on-chain metrics, most straightforwardly:

- number of services a component/agent is used in
- profits returned to the protocol from a given service

These metrics allow the creation of indexes which can assign each component, agent, and service a direct contribution measure to protocol profits.
We use the **useful code factor per agent / component (UCFa/UCFc)** to measure the utilization of agents and components in services returning profits. In a sense, these measures detect the contribution of an agent or a component to the ecosystem, so these are used to weigh the rewards that the ecosystem will give to agents and components royalties.

We include here the **useful service factor per service (UCFs),** which indicates the usefulness of a specific service in terms of the profits the protocol receives. This measure can be used just as a means of tracking the services' activity or performance in terms of the overall profits the protocol receives.

We use bonding because, as the protocol accrues more value, it has to invest in developing new code, so the protocol needs to own as much liquidity as possible.

Remark. All measures are taken at the end of the epoch (practically, in a transaction processed in some block of the next epoch). Then in the rewards calculation, we can explicitly reference the epoch we need.

Math Definition of the Contribution Measures UCF and USF

Requirements.

The measures must be incentive compatible. That is, it must be non-trivial to affect them, and actions affecting them must lead to incentive-compatible outcomes. Hence, we cannot base measures on components/agents/services which are not either a) featuring in services returning profits to the protocol or b) being assessed by governance. For now, we restrict focus to measures based on a).

Definitions.

Let $\mathcal{C}$ be the set of all components, $\mathcal{A}$ be the set of all agents, and $\mathcal{C}_a$ be the set of components making up the agent $a \in A$. Let C_s be the set of components referenced in the service s. Let $\mathcal{S}$ be the set of all services, and let $\mathcal{A}_s$ be the set of agents making up the service s.
Let $epoch$ be a parameter measured as a fixed number of blocks and set by governance. More precisely, let n be the current block number, and let m be the number of blocks in each $epoch$. Then $n = m \cdot k + d$ where d is the remainder and it takes values from 0 to $(m-1)$. Then the block number n is in the $epoch$ k.

Algorithm notation.
Epoch_length = number of blocks in each epoch
Current_block_num = current block number
Current_block_in_epoch = `Current_block_num % Epoch_length`
$epoch$ = `floor(Current_block_in_epoch)` i.e parameter measured in number of blocks

Definitions.
$s(epoch)$ = generic service which returned profits to the protocol during the specified $epoch$ (so we are considering here: Protocol-owned services and third-party services who are willing to return a fee to the protocol.)

$r_s(epoch)$ = profits returned to the protocol from the service s during the specified $epoch$

Definition.
The following value is used to measure whether the component c has been referenced in a service s that produced profits for the protocol at a specific $epoch$. So $cit_s^c(epoch) \in \{0, 1\},$ and precisely,
$cit_s^c(epoch) = 1$ when the component c is used in the service s during the specified $epoch$ and $cit_s^c(epoch) = 0$ otherwise.

Definition.
The following value is used to measure whether the agent a has been referenced in a service s that produced profits for the protocol at a specific $epoch$. So $cit_s^a(epoch) \in \{0, 1\}$ and precisely,
$cit_s^a(epoch) = 1$ when the agent a is referenced in the service s during the specified $epoch$ and $cit_s^a(epoch) = 0$ otherwise

UCF - Usefulness of Code in Terms of Profits

Useful Code Factor **per Component**

Let $c \in \mathcal{C}$, then we can compute the UCF for the component c by considering the number of times c has been referenced in a service that produced profits for the protocol in the last "settled" epoch as follows. Note that we want the usefulness of the component to be proportional to the profits it contributed to, so we weigh each contribution by the amount of profits returned by the services.

If $\exists s(epoch) : r_s(epoch) \neq 0$, then

$$UCF_c(epoch) = \frac{\sum_{s(epoch)} cit_s^c(epoch) r_s(epoch)}{\sum_{s(epoch)} r_s(epoch)} \quad \text{(Eq. 1)}$$

where the numerator is the sum of all profits across all services which component features in and the denominator is the sum of all profits across all services.
Otherwise,

$$UCF_c(epoch) = 0. \quad \text{(Eq. 2)}$$

So UCF_c can be interpreted as the **relative profit contribution of a component** in an epoch.

Remark.

$$0 \leq UCF_c(epoch) \leq 1, \forall c \in \mathcal{C}, epoch.$$

Remark.

$$0 \leq \sum_{c \in \mathcal{C}(epoch)} UCF_c(epoch) \leq \text{number of profits components}$$

Moreover,

$$\sum_{c \in \mathcal{C}(epoch)} UCF_c(epoch) = 0$$

if and only if all the registered services return no-profits to the protocol. And

$$\sum_{c \in \mathcal{C}(epoch)} UCF_c(epoch) = |\mathcal{C}(epoch)|$$

if and only if the registered components are all referenced in all the services that return profits to the protocol.

Useful Code Factor **per Agent**

An agent is made of components, hence the measure of the usefulness of agent code a could be extracted from the usefulness of the components making up a. Instead, we will take into account the profits returned by services referencing the agent a similarly to what we have done per component, we weigh by the amount of profits returned by the services, as follows.

If

$$\exists s(epoch) : r_s(epoch) \neq 0$$

then

$$UCF_a(epoch) = \frac{\sum_{s(epoch)} cit_s^a(epoch) r_s(epoch)}{\sum_{s(epoch)} r_s(epoch)} \quad \text{(Eq. 3*)}$$

where the numerator is the sum of all profits across all services which agent features in and the denominator is the sum of all profits across all services.
Otherwise,

$$UCF_a(epoch) = 0. \quad \text{(Eq. 4*)}$$

Hence, similarly UFC_a can be interpreted as the **relative profit contribution of an agent** in an epoch.

Remark.

$$0 \leq UCF_a(epoch) \leq 1, \forall a \in \mathcal{A}, epoch$$

Remark.

$$0 \leq \sum_{a \in \mathcal{A}(epoch)} UCF_a(epoch) \leq \text{number of profits agents}$$

Moreover,

$$\sum_{a \in \mathcal{A}(epoch)} UCF_a(epoch) = 0$$

if and only if all the registered services return no-profits to the protocol. And

$$\sum_{a \in \mathcal{A}(epoch)} UCF_a(epoch) = |\mathcal{A}(epoch)|$$

if and only if the registered agents are all referenced in all the services that return profits to the protocol.

USF - Usefulness of Services in Terms of Profits

Next we define the useful factor per service.

Useful Code Factor **per Service**

Let $s \in \mathcal{S}(epoch)$. If $\exists s' \in \mathcal{S}(epoch) : r_{s'}(epoch) \neq 0$, then

$$USF_s(epoch) = \frac{r_s(epoch)}{\sum_{s \in \mathcal{S}(epoch)} r_s(epoch)}.$$

Otherwise

$$USF_s(epoch) = 0.$$

So, USF_s can be interpreted as the **relative profit contribution of a service** in an epoch.

Remark.

$$0 \leq USF_s(epoch) \leq 1, \forall s \in \mathcal{S}(epoch), epoch.$$

Levels of Tokenomics already implemented

From the perspective of a capital investor, the tokenomics model can be broken down into three levels (the third level is out of the scope of this document, so we did not include it here). Each level has its own objective and target audience.

Level 1: Bonding

Summary: Holders of OLA can provide liquidity in a Decentralized Exchange (DEX) and sell their corresponding liquidity provider (LP) tokens to the protocol at a discount for OLA. The process is similar to bonding in Olympus DAO albeit with some important differences.

Objective: This creates a continuous fund-raising market for the protocol and ensures there is ongoing liquidity for the protocol token OLA, thus contributing to price stability. Furthermore, it endows the protocol with third-party assets which it can put to productive use in protocol-owned services or elsewhere.

Audience: Individuals and entities that want exposure to OLA and become ecosystem participants.

Math Definition of the Discount factor

An investor with whitelisted liquidity provider (LP) assets (for example OLA-DAI, OLA-USDC, and OLA-ETH, from a single DEX like SushiSwap or Uniswap to begin with) can bond those assets via the bonding contract at time *t*.
Then at time $t + t_v$ where t_v is the vesting time set by the governance, the investor receives OLA at a discount relative to the price quoted on the relevant DEX.

Remark.

It is fair to assume t_v is on the order of days, likely between 7 and 14. It will be something governance can adjust to finetune tokenomics.

Precisely, assume that an investor wants to bond an LP-assets at a time t whose price is b_p on a relative DEX at time t. Then at time $t + t_v$ the investor will receive a number of OLA tokens whose price is ub_p.

Let $i(t)$ be the interest on b_p that investor will receive when bonding at time t. Then the following holds

$$ub_p(t + t_v) = (1 + i(t))b_p(t).$$

That is, at the time $t + t_v$ the investors will receive a number of OLA tokens whose price is $b_p(t) + i(t)b_p(t)$.

We recall that the discount factor is the value that needs to be multiplied by the future price of an investment to obtain the initial price. So

$$DF(t) = \frac{1}{1 + i(t)}.$$

Remark- Implementation.

The interest rate must be set in such a way $i(t) > 0$. In the circumstance where a negative interest rate would be preferred, governance must vote for a hard-stop of the bonding contract.

Remark - Implementation The parameter t_v will be dynamically defined by governance before the bonding contract will start. And also we can adjust the epoch length. For instance, in the beginning, epochs should probably be much longer, e.g. in the order of weeks, and could potentially go down to days when the protocol is matured.

Remark - Implementation So far, the parameters are defined in terms of LP-tokens. However, for the implementation, input and output values in OLA-tokens are preferred. This explains our argumentation in OLA tokens.

For helping the smart contract, a user interface will “ask” the balance of LP-asset (e.g. OLA-ETH) a user wants to allocate at time t. Then again externally to the bonding contract, it is possible to calculate all the bonded LP-asset at time t, then ‘translate’ this into the corresponding amount of OLA tokens. Once the corresponding value of OLA

tokens bonded at time t is obtained this becomes the input parameter of the bonding contract.

Aim. Towards enabling the goal of pairing capital and code as described in the introduction, we want to build a mechanism allowing that bonded capital in the protocol not to exceed code usefulness.

We can reach our aim by evaluating the potential output of the developed agents and components on-chain. Technically we can consider a *production function PF* that determines the maximum amount of output that can be obtained from a given number of inputs that can be computed on-chain. See paragraph **production function**.

Moreover, the bonding implies an inflationary token model. In particular, we have an inflation schedule that dictates how much we can mint per epoch at a maximum (see paragraph **Inflation schedule and rewards** for more details). Since the bonding implies minting new OLA tokens, we need to take into account that during epoch e, we have a maximum amount that can be bonded. The inflation schedule can be respected by using the fact that investors prefer liquidity today to a small amount of interest on an investment in the future.

Summing up, to control the bonding at the epoch e, we need to consider a direct control factor (DCF), that allows using *PF(.)* in such a way that: the larger *PF,* the greater the return flow of the price, and the smaller *PF,* the smaller the return flow of the price, thus respecting the inflation schedule.

Product function

In Economics, a **production function** relates outputs of a production process to inputs of production. So it is a mathematical function that relates to the amount of output that can be obtained from a given number of inputs. Generally, these inputs are capital and labor.

In our case, we aim to pair code and capital. So the output is the production of code (components and agents) that enables the creation of services that, in turn, will pay back royalties for the produced code and investment for new devs. Hence the production process will be then the creation of the code. For the inputs, we can extract on-chain the following parameters.

Assume that by the end of **epoch e**, the services $s_1, \ldots, s_t$ are all services which return profits to the protocol. Let $R = r_1 + \ldots + r_t$ be the total profits obtained by the protocol during epoch e. These profits will be used as follows.

- R/P: will go to the agents/components code owners (divided among them using ucf_c and ucf_a)
- R/L: will go as rewards for lockers (divided among them using the amount of locked OLA they possess)
- $K(e) = R - (R/P + R/L) = R(1 - (L + P)/(PL))$: will go to the Treasury

In particular, $K(e)$ can be seen as the **capital "**gained" by the protocol during epoch e, so that the latter can be invested in the development of new code from the next epoch.

Moreover, by the end of epoch e, we can count the number of **valuable devs** that produced profitable components or agents.

(Recall that *c* (resp. *a*) is a profitable component (resp. agent) during epoch e if *ucf_c(e)* (resp. *ucf_a(e)*) is non-zero.

Remark.

All ceteris paribus, as K increases, it is possible to invest more to pay developers that, in theory, will bring new valuable code. All ceteris paribus, as the number of valuable devs increases and, in theory, more valuable code can be built.

Product function parameters.

Output = valuable code to enable services that, in turn, will produce profits

Production process = creation of components and agents (and services?)

Inputs =

- $K(e)$ is the capital gained by the protocol in epoch e that can be invested in the development of new code
- $D(e)$ is the number of developers that produced valuable code during epoch e (valuable devs described above)

As first option we can consider a **perfect substitutes production function** where the inputs are substituted at a constant rate

$$f(K, D) = A(k * K + d * D)$$

for some constants $A,\ k,\ d$.

The value A is called factor productivity and measures residual growth in total output that cannot be explained by the accumulation of traditional inputs such as labor and capital. Right now we set A=1, in the future, we will refine our model.

How the parameters $k,\ d$ can be appropriately chosen? We have to set the formers in in such a way that $f(K, D)$ can output the useful code we expect to produce with K, D as inputs.

Let

$$d_a = \text{the number of valuable agents built in one epoch by an agents dev}$$

$$d_c = \text{the number of valuable components built in one epoch by a comp. dev}$$

Remark.

Note that with code we mean both agents and components. To avoid lots of variables, we can say that a valuable unit of code corresponds to n_a profits agents and n_c profits components.

So

unit of valuable code = n_c component or n_a agents.

We start with the following:

unit of valuable code = one component or two agents.

Remark.

Note that this weighting should not be hard coded as $n_c = 1$ and $n_a = 2$ directly on the contract as we might need to tune it.

So let's consider the following.

Notation.

$$d = \text{units of valuable code built by a dev in epoch}$$

$$k = \text{the number of valuable devs can be paid per units of capital per epoch}$$

So that

$d * D(e) =$ the units of valuable code $D(e)$ can build during one epoch.

$k * K(e) =$ the number of devs can be paid during with profits $K(e)$ for one epoch

$d * k * K(e) =$ the units of valuable code can build during one epoch with profits $K(e)$.

In particular,

$$f(K(e), D(e)) = d * k * K(e) + d * D(e)$$

determine the maximal amount of valuable code can be built during the next epoch having capital $K(e)$ and labors $D(e)$.

Remark.

k and d during the first epochs can be guessed based on the Valory devs gains and performance. When the protocol is fully functional, and various epochs are passed, the parameters can be properly estimated using the data of the previous epochs.

Consideration.

When $f(K(e-1), D(e-1))$ is very large, it means that with inputs $K(e-1)$ and $D(e-1)$ it is possible to produce a big amount of valuable code from the epoch e. Producing more valuable code means more returns will be received from the protocol, and so more payments for royalties and rewards will be given.

In particular, it is required a large quantity of liquidity to trade ETH for OLA, and in particular, the bonding should be incentivized.

Another reason for incentivizing bonding in this circumstance is that, the richer the protocol becomes, the more investment must be made. So to properly use the capital for pairing it with code, it must possess as much liquidity as possible to remove some of it for investing in new code.

Respecting inflation schedule

From the inflation schedule, the following value can be considered:

S(e) := total supply of OLA tokens at epoch e

MaxBond(e) := the fraction of S(e) that governance allows bonding during epoch e

Definition.

X(e) := the amount of OLA tokens bonded during epoch e

EffectiveMaxBond(e) := is the fraction of S(e) that governance allows bonding during epoch e plus the possible remainder of MaxBond(e-1) that was used for bonding during epoch e-1.

In formulas, we can write this as follows.

EffectiveMaxBond(e):=MaxBond(e) + Excess(e-1)

Excess(e-1) := EffectiveMaxBond(e-1) - X(e-1)

Def.

Let τ_e be the time corresponding to the beginning of epoch e. Let $\tau_e \leq t < \tau_{e+1}$.

X(t,e) := the amount of OLA tokens bonded during the interval of time $[\tau_e, t)$

B(t,e) := the amount of OLA tokens bonded at time t.

We define the **bonding rate BR(t)** as follows.

BR(t) := 1 - X(t,e)/EffectiveMaxBond(e)

Remark.

When the amount of OLA tokens bonded during the interval of time $[\tau_e, t)$, X(t,e), approaches EffectiveMaxBond(e) (everything also equal), BR(t) approaches 0.

While, when X(t,e) approaches 0 (everything also equal), that is there is a small or no amount of OLA tokens bonded during the interval of time $[\tau_e, t)$, BR(t) approaches 1.

Example. Assume that B(t,e)=0, for every t<t' and that B(t',e)=20. So X(t',e)=B(t',e)=20.

Assume that EffectiveMaxBond(e)=30. So there exists t''>t' such that DF(t'',a,b,c,d)<=1. In other words, there is still the possibility of bonding a certain amount. If it is possible to bond more than 10 at time t'' then we have broken our control mechanism for bonding. So the following remark is crucial.

Remark - implementation

The contract must ensure that at time t'' users are allowed to bond at most EffectiveMaxBond(e)-X(t',e) where X(t',e) is the amount bonded right before the first bonding at time t''.

At the level of smart contract, every bonding tx is sequential by design. And so we can update X(-, e) on each transaction.

Moreover, when BR(t) =0, that is X(t,e)=EffectiveMaxBond(e), for some $\tau_e \leq t < \tau_{e+1}$, in practice we need to create a hard stop.

The hard stop we basically create in practice will correspond with $DF(t, e) = +\infty$.

Remark. How is X(t,e) updated?

In practice, as discussed above, this counter is reset each period, and it is updated, summing up the new amount of capital bonded for each bonding transaction in one epoch.

So that

$$X(\tau_e, e) = B(\tau_e, e),$$

corresponds to the bonding amount in the bonding tx included in the first block of the epoch e.

Now, assume B(t,e)=0, for every t<t' and that B(t',e)=b1. Then X(t',e)=B(t',e)=b1.

Assume B(t,e)=0, for every t<t'' and that B(t'',e)=b2. Then X(t'',e)=b1+b2.

In theory, since t is a continuous variable, we have that

$$X(t', e) = \int_e^{t'} B(t, e)\, dt, \forall\, t' \in [\tau_e, t').$$

And

$$X(e) = \int_e^{e+1} B(t, e)\, dt$$

Direct Control Factor (DCF)

As we pointed out at the beginning, in order to control the bonding at the epoch e, we need to consider a direct control factor (DCF) that allows

- using f(K, D) in such a way the larger f(K, D) the larger the return flow of the price and the smaller f(K, D), the smaller the return flow of the price.
- respecting the inflation schedule.

So for the epoch e, the governance will set

- ➔ $\varepsilon(e)$ = the maximum interest rate that investors can receive at the time $\tau_e \leq t < \tau_{e+1}$ (recall that $\varepsilon(e) \geq 0$, however governance must choose $\varepsilon(e)$ different from 0, see properties of the presented formula for more details)
- ➔ MaxBond(e) = the fraction of S(e) that governance allows bonding during epoch e

We could have a mapping so that governance can already pre-commit to the entire emissions schedule for bonding[2] (and then update it if needs be)

Def.

Let $\tau_e \leq t < \tau_{e+1}$. If X(t,e) < EffectiveMaxBond(e), then the discount factor can be set as

$$DF(t,e) = max\left\{\frac{1}{1+\varepsilon(e)}, \frac{1}{1+(f(K(e-1), D(e-1)))/100}\right\}$$

Otherwise

$$DF(t,e) = \infty$$

Remark- Implementation. The math behind the bonding has to be implemented in a new contract that is not the bonding contract. In such a way, long calculations do not happen in the execution of the bonding contract, and, much more importantly, if the monetary policy of the protocol changes it is not necessary to modify the bonding contract.

Level 2: Locking[3]

Summary: Holders of OLA lock their OLA in return for a *non-tradeable* virtualized token called veOLA. veOLA holders can participate in governance of the Autonolas DAO and are rewarded for their governance contributions..

Objective: This creates alignment between token holders and the protocol objectives. It incentivizes committing to the token OLA by locking it to obtain veOLA and be able to steer the protocol via governance. It reduces short-term decision-making in governance vis-a-vis a model with a tradeable governance token.

Audience: Holders of OLA.

---

[2] If S/b is the fraction of inflation for bonding, then MaxBond and εis pre-committed in such a way MaxBond(e)(1+ε) ≤ S/b

[3] From a practical perspective, and regulatory lense, locking is similar to staking.

Holders of OLA lock their tokens and receive a virtual representation (i.e. a “stake”) called veOLA in return (see Fig. 8). The virtualized token veOLA is a non-tradable token that gives the user a claim against the locking contract. In technical terms, it implements the ERC20 standard without the transfer functionalities. The user must specify the locking duration in months or years when locking. The available set of lock durations is specified by protocol governance. The locking length determines the multiplier with which voting rights are boosted. Voting rights decrease linearly over the locking duration (e.g. like in Curve).

Math formula

User *A* lock at time t_0^A lock a OLA tokes for a locking duration of $A.lock.time$. So that the time of the un-lock time will be

$$A.unlock.time = t_0^A + A.lock.time.$$

Recall that, $A.lock.time \leq Max.lock.time = 4\,years$, so at the maximum the unlock period for user A happens at time

$$A.unlock.time = t_0^A + A.lock.time.$$

At the time $t_0^A \leq t \leq A.unlock.time$, the voting power $v_A(t)$ of user A is weighted in terms of the remaining time to unlocking and in terms of the locked amount. Precisely, if $t = t_0^A + \Delta_A(t)$, then

$$v_A(t) = a \cdot \frac{A.lock.time - \Delta_A(t)}{Max.lock.time}.$$

So that, everything also equal, the voting power is maximum at time t_0^A and decreases as the time approaches the unlock period.

Moreover, everything also equal, $v_A(t)$ increases as the as $A.lock.time$ increases or as *a* increases.

Holders of veOLA can directly participate in on-chain governance (of protocol and protocol-owned services). All holders of veOLA collectively form the voting members of the Autonolas DAO.

Investors and team members will be locked into veOLA from the start according to their vesting schedules.[4]

When the lock phase has passed, the veOLA holder can retrieve their locked OLA. Locking can also be renewed, including before the current lock phase has ended.

The locking mechanism creates “push” and “pull” mechanisms. In the bootstrapping phase of the protocol, "pull" results from the attractive yield from locking OLA and receiving OLA token emissions (in the form of veOLA). After bootstrapping, "pull" can result from earning yield from real revenue sharing. Another less tangible “pull” is that ecosystem participants can only have governance power if they lock. The "push" results from OLA holders not wanting to get diluted. If OLA holders are not locking, then the token emissions and profit shares are taken up by others and a holder’s share of the overall token supply is getting smaller over time.

Inflation distribution - rewards distribution and calculation

Inflation schedule

We have an inflation schedule that dictates how much we can mint per year at a maximum. The inflation schedule is governance controlled, but in the OLA token contract, we created some hard upper bounds on the parameters as forwarding guidance to the governance (i.e. no more than 2% inflation after year 10 & no more than 100% of the 1bn supply until year 10).

We target an initial allocation of 40-50% of the total token supply to the ensemble of DAO treasury (10%), stakeholders (20-30%), and future sales allocation (10%).

It is possible to describe the *inflation schedule* by means of the following three parameters:

- *Initial inflation rate*: the percentage of the initial token supply it is inflated
- *Inflation rate after the first year*: the percentage of the initial token supply produced by the end of the first year that it is inflated
- *Dis-inflation parameter*: the parameter with which the inflation rate is fractionalized every year. This is a positive rational number such that inflation(year)=inflation(year)/m

[4] Amounts will be calculated to target the promised share of the fully diluted token amount after vesting at the end of year 10. It takes into account veOLA staking incentives paid out over the vesting horizon.

- *Long-term inflation rate:* tokens inflated in the long run.

Used Values

- *Initial inflation rate:* 4%
- *Inflation rate after the first year* : 14.2%
- *Dis-inflation parameter* : 1.2 such that i(year)=i(year-1)/m
- *Long-term inflation rate:* no more than 2% after year 10.

A graphical representation of our inflation schedule over 10 years is in Fig 1.

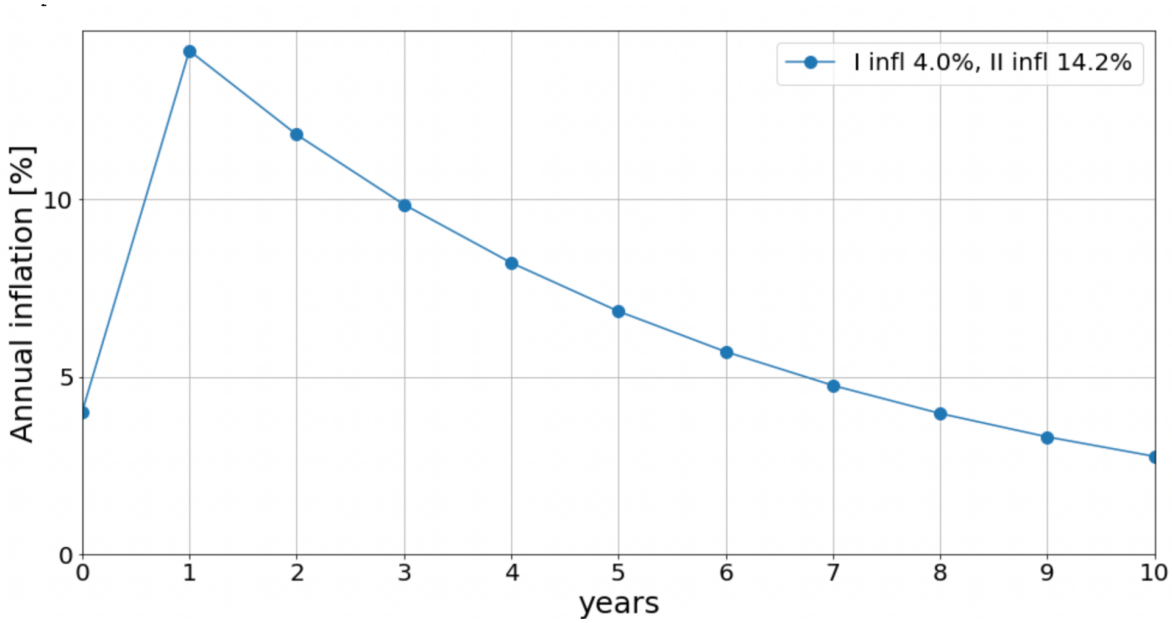

Fig. 1 Inflation schedule

From this Inflation Schedule, we also show in Fig. 2 the token issued over 10 years.

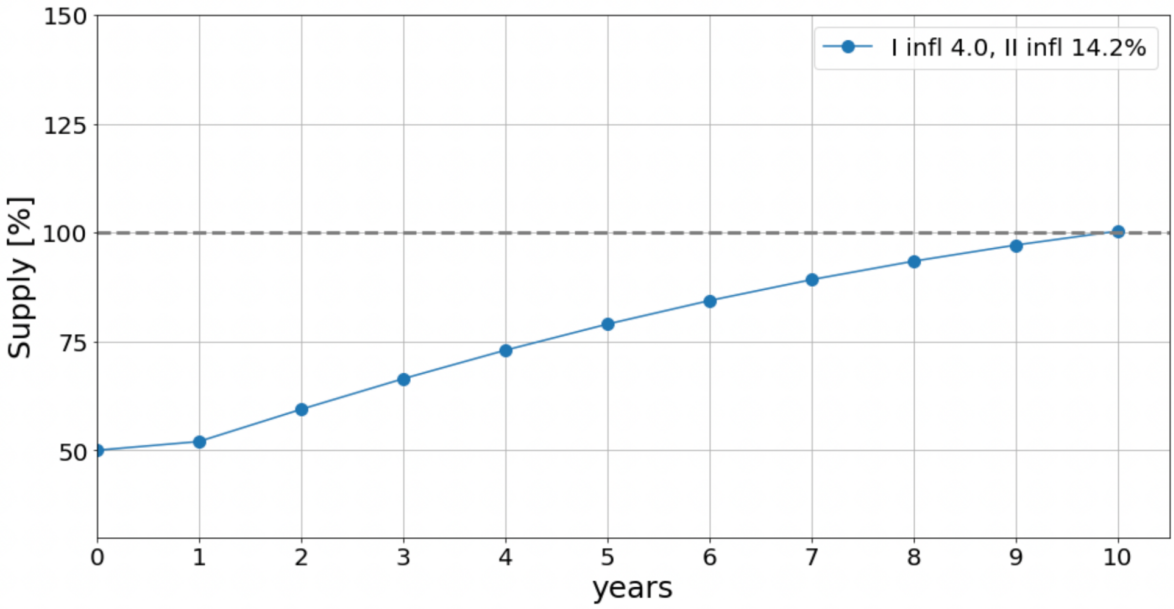

Fig. 2 S-shaped curve of supply

Remark - implementation

On the contract, to simplify the calculation the values are truncated as follows.

Inflation rate for the first 10 years.

inflation(0)= 4%
inflation(1)=14.2%
inflation(2) = 11.8%
inflation(3)= 9.9%
inflation(4)= 8.2%
inflation(5)=6.9 %
inflation(6)= 5.7%
inflation(7)= 4.8%
inflation(8)= 4%
inflation(9)= 3.3%
inflation(10)=2.7%

Supply for the first 10 years.

S(0)= 5e18
S(1)=5.2e18
S(2)= 5.9e18

S(3)= 6.6e18
S(4)= 7.3e18
S(5)= 7.9e18
S(6)= 8.4e18
S(7)=8.9e18
S(8)= 9.3e18
S(9)= 9.7e18
S(10)=e19

Inflation distribution - rewards distribution

The rewards are destined to lockers and agents/components royalties. In the long-term view, these rewards will be covered up with the profits returned from the services to the protocol. However, in the beginning, the rewards are funded by inflation. Up to 10 years, with a profits increase, the inflation decreases as the main source of funding rewards and becomes top up to the reward profits funded. After 10 years, the low inflation will be used mostly for bonding.
Note that at a certain point, part of the profits received by the protocol will be given to the Treasury.

Remark.
To avoid incentive-compatible problems, top-ups are limited to components/agents royalties included in whitelisted services. To begin with, the whitelist will be Autonolas DAO only. Later, governance will vote to whitelist other DAOs that are implicitly expected to behave honestly when it comes to returning profits to the protocol (i.e. they don't just send minimal profits to trigger some top-up, or if they do we assume the services they register are still useful).

How will inflation be distributed?

The minted amount will be used for

- Bonding
- Lockers rewards top-ups
- Agents/components royalties top-ups

The governance parametrizes the inflation amount that is reserved per year for (a) bonding, (b) lockers reward top-ups, (c) agents/components royalties top-ups. However we create some hard upper-bound on the bonding amount as forwarding guidance to governance (i.e. no more than 40-50% of the inflation per year can be used for bonding).

The amount of bonding determines how much gets actually minted. That is if x percent of the (a) gets really for bonding, then also x percent of fraction (b) and (c) will be available for top-ups.

It is worth noting that a mint for an epoch might end up not being used. In particular, if a fraction of (a) was not used during one epoch, it will be added to the mintable amount for bonding during the next epoch.

Remark. The Treasury gets obviously nothing from the mint.

How rewards will be distributed?

Each epoch, we use the returned profits to fund OLA lockers, component/agent royalties, and treasury. The governance can parametrize the split of profits per epoch between lockers, component/agent royalties, and treasury, but is forced to aggregate to 100% of the received profits.

Recall that up to 10 years, the new mint fraction of (b) and (c) of an epoch as described in the previous section will be used to boost the rewards of lockers & component/agent royalties from that epoch.

How are rewards calculated?

First of all, users go to the contract Dispenser to reclaim rewards by owning components/agents corresponding to their code, or veOLA (locked OLA) as proof of locking.

Rewards for lockers

Notation. Locking for period e, **Lock(e)**, means the amount locked at end of epoch e (i.e. the last block of the epoch)

For the moment we consider linear lockers reward, that the reward accumulated during period e, $R_S(e)$, that are designated as rewards for lockers, will be allocated to lockers of period e-1. Precisely, if a user has locked n OLA tokens in period e-1, he will receive

$$(R_S(e) * n)/Lock(e-1)$$

Remark. We use the profits of epoch e for return rewards at lockers during epoch e-1 to avoid a scenario where someone can see the on-chain rewards come in for a period, and then decide to lock OLA to come into the benefit of (some of) those rewards at the next settlement (end of current, start of new epoch).

Remark. Perhaps in the future, we can consider the rewards for the locking weighed on the chosen locking vesting time: the more time you choose to keep OLA locked the more rewards you get. But for now, linear makes more sense because the length of lock is already rewarded in terms of governance power.

Rewards for agents/components royalties

Assume that at the epoch e the total amount of profits designated to fund agents/components royalties is $R_D(e)$. This amount will be split as

$$R_{ag}(e) = r_{ag}\ R_D(e) / (r_A + r_C)$$ i.e. royalties for agents

and

$$R_{comp}(e) = r_{comp}\ R_D(e) / (r_A + r_C)$$ i.e. royalties for components

These amount will be in turn split in a weighted way via UCF_c and UCF_a among owners of agents and components with positive UCF_c and UCF_a as follows.

Let c be a component and let C be the set of the registered component. Let us assume that at least one $UCF_{c'}(e)$ is not zero, for some component c' in the ecosystem.

$$Reward\_for\_component\_c = R_{comp}(e) * UCF\_c(e) / \Sigma_{\{c' \in C\}} UCF_{c'}(e)$$

$$Reward\_for\_component\_c = R_{ag}(e) * UCF\_c(e) / \Sigma_{\{c' \in C\}} UCF_{c'}(e)$$

Remark. We can start by setting $r_{ag} = 1$ and $r_{comp} = 2$, but r_{ag} and r_{comp} can be selected by governance.