Agorata

LEV CHIZHOV[1,2]

@ennucore

July 26, 2022

Abstract

Agorata is an implementation of the idealized economic agent, or, specifically, an aggregator of smart contracts. It combines one or more contracts proposed by the members of the network in a deal which can be evaluated as profitable for Agorata with low risks. Using this approach, Agorata can provide infrastructure for loans, flashloans, bets, derivatives, bridging between chains, and many more financial instruments.

CONTENTS

[1] Moscow Institute of Physics and Technology
[2] Ludwig-Maximilians-Universität München

I. Introduction

There is a variety of standard financial organizations that provide their capital for some simple purely financial deals: banks, brokers, gambling institutions, betting institutions, and many more. Most of them have their own DeFi counterparts, as well as some services that are only possible on the blockchain — for example, flashloans.

All of these organizations can be generalized to an economic agent that has a utility function and accepts a proposed deal if and only if it is the most beneficial in terms of this function. These particular implementations, however, only consider a very limited subset of deals. Also, all of them work only with purely financial deals which include not more than 3 parties[1], including the financial organization.

Technologies of decentralization and smart contracts in particular provide the foundation for a much more fundamental entity — one which goes far beyond standard simple templated purely-financial deals. Instead, this entity can be the implementation of the general economic agent. It can analyze deal proposals and their combinations into more complex deals (for example, combining two proposal in a risk-free deal, like with betting), which it evaluates and participates in. As a result, this entity can serve as a liquidity provider, loan broker, a service for finding a counterparty. It is worth noticing that as the blockchain technologies spread into multiple fields (not only financial), a service like that can become much more versatile and participate in deals including various assets (domain names, art, computational resources are the examples of what can be possible now, future applications will go much further).

The goal of this whitepaper is to present such an entity — Agorata.

II. Overview

As a service, Agorata has the following path of the user:

1. User creates a smart contract, which is a proposal for a deal, using Agorata's smart contract builder, or just uploads the contract code directly to the blockchain. The smart contract builder is a web application that allows to create a smart contract in the following ways:

    - Using one of the predefined templates, which are based on the standard financial deals.
    - Using Agorata's simplified language for smart contracts, A-Lisp, which is based on the Lisp language. It is then compiled to the Fift assebly language.
    - Using the No-Code version for creating a finite state machine smart contract.

2. Agorata builds its own standardized representation of the smart contract.

3. Agorata evaluates the smart contract or a combination of smart contracts in its pool and decides whether it is profitable or not.

4. Agorata accepts the most profitable deal.

[1]An example of such a deal with three parties is a stanard deal on a financial market.

III. Contract representation

i. The structure

A smart contract is considered as an entity with which other entities (users, smart contracts) can interact via messages[2]. For a state of the contract we can determine the messages that can be sent to the contract and for each of them — the response messages and the next state.

The contract is represented as an actor with several states (a finite number of states which can have parameters — i.e., variables in the contract) that can receive several (a finite numer of) types of messages and change the state. The graph of states, messages, and transitions fully describes the system. Each message has a fixed form with parameters and the output state, which parameters depend on the parameters of the message. This dependency is characterized by an algorithm in the form of code in a pure subset of Lisp — A-Lisp[3].

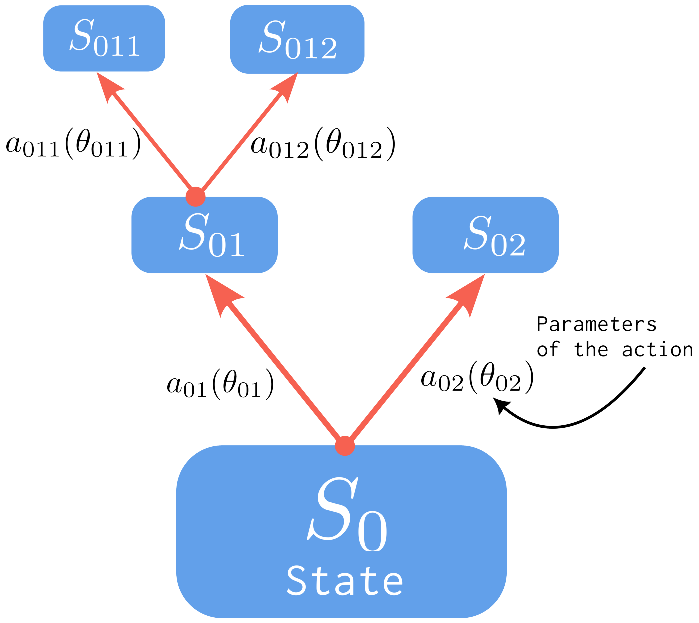


Figure 1: *Deal representation.*

Each action (message) has parameters θ, including:

1. s — the sender of the message.
2. t — the time (block number) at which the message is sent (may be omitted in the representation if it is unimportant for the next states).
3. tokens sent with the message.
4. any other possible information.

[2]The message concept used here is from The Open Network (TON). A message can include tokens, commands, information, code.

[3]The choice of Lisp is motivated primarily by the fact that it is a good unified way to represent an algorithm. There are also ways to use Lisp programs in formal proofs

The parameter space for each action can be constrained — the smart contract can reject some of the messages. For instance, this can lead to the entire space consisting of a small number of elements (the contract rejects everything except for several specific messages, e.g. a specific person sending a specific amount of tokens).

The main parameters of the state are the financial outcomes for the smart contracts.

At the first stages of the project, the states and messages will not have variable parameters. The contract, therefore, will be a finite state machine. For that, the contract will have to check the incoming message on being in the list of allowed messages.

ii. Contract evaluation algorithms

As we will see in the following section, one of the most important parts of the process is finding out whether there exists any action that makes the deal profitable or unprofitable. It is also important to know which actions the agent should make to maximize its output. How do we do that?

The state parameters are represented as functions of action parameters expressed in a formal language. This allows to make all the decision based on mathematical proofs using theorem provers.

Note that for a contract with no parameters of messages and states this kind of an algorithm is not required.

iii. Building the formal representation

At the first stages of the project, the formal representation will be obtained in two ways:

- Pattern matching on the Fift code for standard contracts made from templates
- The creation of the contract performed by the user through the constructor on the Agorata website

Pattern matching will be based on a set of templates with fields. Most prevalent field types will be addresses, token types, and amounts. This algorithm will be able to recognize proposals of many types, including loans, derivative, flashloans, bets. For example, for a loan the paramters are the two involved tokens, sums in them, participating address.

iv. Examples

Figure 2 show graphs for some simple deals.

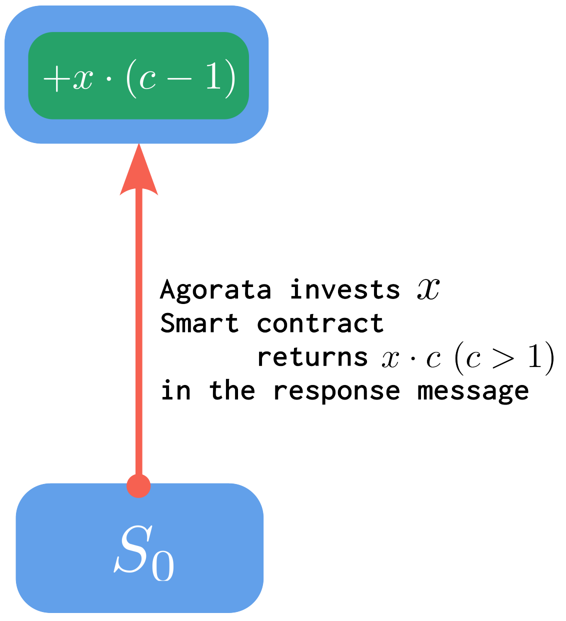
IV. Decision-making algorithm

i. Agent parameters

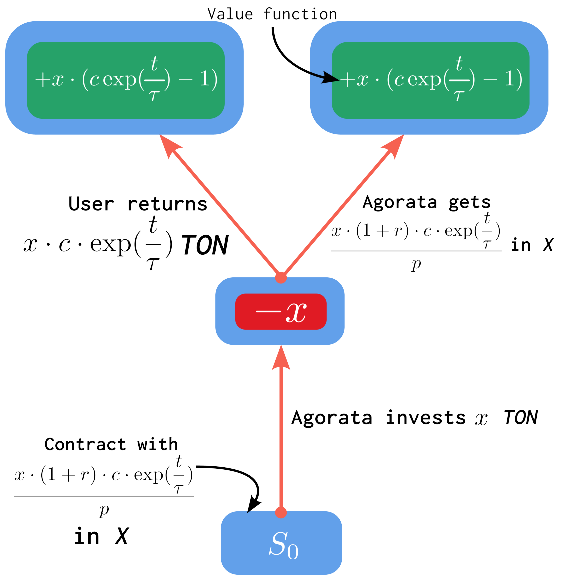
1. T_0 is the maximum deal duration — i.e., the horizon after which the benefits of the deal are not considered.

2. τ is the charachteristic time of discount — $\exp(\frac{1\,year}{\tau}) - 1$ is the minimal yearly rate for a loan.

3. r is the risk parameter for currency switching — the agent will only agree to swap no less than $(1+r)x$ in one currency for x in another.



(a) *Flashloan*


Value function
$+x \cdot (c\exp(\frac{t}{\tau}) - 1)$
$+x \cdot (c\exp(\frac{t}{\tau}) - 1)$
User returns
$x \cdot c \cdot \exp(\frac{t}{\tau})$ TON
Agorata gets
$\frac{x \cdot (1+r) \cdot c \cdot \exp(\frac{t}{\tau})}{p}$ in X
$-x$
Agorata invests x TON
Contract with
$\frac{x \cdot (1+r) \cdot c \cdot \exp(\frac{t}{\tau})}{p}$
in X
S_0

(b) *Loan*

Figure 2: *Graphs for some simple contracts*

Given these parameters, the value function $V((x_i),(t_i))$ (or $V(S)$) can be determined. It takes the sequence of values (x_i, t_i), where x_i are the incoming/outcoming tokens from the agent.

ii. Deal evaluation

The agent considers the worst case of the deal from the perspective of game theory: the best (from the perspective of the value function) actions of the agent and the "worst" actions of the counteragents. This sequence of events is the run through the value function in order to make a decision.

Thus, a deal is considered profitable for the agent (denoted by A) $\overset{def}{\iff} p(S_i) \iff$

$$V(S_i) > 0 \vee (\forall a_{ij}(p(S_{ij}) \vee a.s = A) \wedge \exists a_{ij} : V(S_{ij}) > 0)$$

Here, S_i is state that is analyzed (the state includes the token movements to and from the agent), a_{ij} are the possible actions (messages) in that state. The function $V(S)$ is the value function which analyzes the token movements in terms of profitability.

The algorithm starts at the node S_0, then considers all the possible actions (including the abscence of one) and calls the algorithm at S_{0i}. If there is any action of the counteragent that makes the contract non-profitable for the agent, the deal is considered non-profitable. If there is no possible action and the state includes profit for the agent, the deal is considered acceptable.

V. Applications

i. Flashloans

ii. Loans

iii. Bridges

iv. Bets

v. Derivatives

vi. More complex examples and other usages

Agorata can also be used as a platform for looking for a counteragent for a deal. As a service, it will also feature a constructor of smart contracts from the from the formal language (or its graphical, no-code, representation). This is an extremely useful service since writing programs in FunC is complex, while the logic in most cases is simple.

VI. Conclusion