Restricted Move – a Smart Contract Description Language to Create and Control Finance Instruments on DFinance Blockchain Platform

R. E. Spiridonov
Saint Petersburg Electrotechnical University “LETI”, St. Petersburg, Russia
E-mail: Rom@nSpiridonov.ru

Abstract. Smart contracts created to issue tokens and control other digital financial assets on the majority of blockchain platforms require knowledge of programming language for specific platforms. This fact imposes a restriction on attracting financial specialists from conventional markets. This paper presents the concept of a Restricted Move language allowing the users to form a smart contract description in financial English, as well as its further development into a high-level digital asset management language, which can be extended by users and can open new opportunities to automate their market operations.

1. Introduction
Smart contracts were first implemented on the Ethereum blockchain platform as functions for EVM compiled into bytecode and stored inside the blockchain, allowing the user to perform various actions to create custom tokens [1]. However, writing smart contracts is still inaccessible to a wide class of users due to the complexity of their creation and organization, requirements for the specific development tools and debugging.

Another problem is the need to take into account a number of tasks to ensure the proper level of security of smart contracts that could be hacked in order to steal funds or compromise access to other wallets. It is also worth noting that blockchain platforms are closed within themselves and that they lack compatibility with each other in most cases, which makes it impossible to perform operations between different networks without an intermediary that is not automated and requires human participation.

All the abovementioned problems lead to creation of a new language that is accessible for a wide class of users familiar with financial markets and aware of all the possibilities of modern financial instruments when they are transferred to the world of crypto currencies and automated management of digital assets. It is proposed to make an addon to the Move language called the Restricted Move the basis for such smart contract description language.

It goes without saying that simplification of smart contracts is not an absolutely new task and the attempts to solve this problem have already been made [2]. However, most of them use the Ethereum blockchain and a complex programming language Solidity that is closer to C++ than its counterparts that are easier to master and use. On the other hand, various blockchain platforms already have additional languages, which simplifies the task of smart contracts creation. However, they also require that the user should possess programming skills and should be able to formulate thoughts in
declarative logic for functional instructions or in a set of calls to define parameters for a finite state machine.

2. The Concept of a Smart Contract Description Language

The main aim of creating a new smart contract description language is to attract as many classical finance professionals as possible to crypto world. They have to be closely familiar with this new instrument to use it in everyday tasks. For this reason, we opted to base it on natural language constructions set with some addons.

Firstly, a language should allow the creation of digital assets, like finance derivatives and digital tokens. On the other hand, it should grow up into a financial management language that would allow describing finance strategies and specific market conditions and actions.

2.1. Language Requirements

Similar to any high-level programming language, Restricted Move consists of the following construction parts:

- Alphabet;
- Syntax and rules of program design;
- Data actions (input-output);
- Data structures.

However, in this case it is problematic to determine the entire familiar set that characterizes a programming language due to the external simplicity and semantics of the phrases for describing financial instruments in English, as well as due to the presence of uncertainty about the size of the entire set of language elements – the language has to be able to be supplemented with new phrases and opportunities.

Let us consider a few examples of descriptions that create new tokens in financial English:

Create 1 non-fungible Token as an American Call Option contract, representing 1 Nike Jordan Air, expiring November 30th.

Create 100 Tokens as an American Put Option contract, representing 100k USD each, balanced with 50% S&P, 25% European Sovereign Debt, 15% BTC and 10% Real Estate, not expiring.

Or a collateralized debt position with debt risk option on it in one description:
Take a CDP Loan of 2000 XFI using 1 ETH as loan-to-value 23%. Create a Debt Risk Option with a price of 10 XFI till January 21, 2021.

In such examples, you can use typical subquery constructions that determine the tool creation parameters. The entire sequence of such constructions can be formalized as a sequential call of functions with an indication of their arguments.

To be more similar to natural languages, it has to be allowed to break the standard description sequence. To do it we need to add the ability to start a description from different parts of the smart contract request to the Restricted Move standard. For example, we first describe some of the limitations or properties of the finance tool, and then the tool itself.

An additional requirement for Restricted Move is to store the entire language structure inside the blockchain. A number of restrictions and possibilities for the language syntax follows from this requirement. For example, upgrading language versions will automatically provide backward compatibility with the previous versions, however, there is a limitation on updating separate blocks – we can only completely redefine them.

2.2. Low-Level Language Structure

To store all language description in blockchain, we will describe all language terms as separate blocks that can be combined with each other based on the predetermined rulesets. All blocks can be grouped into two types: Data Nodes and Flow Nodes.
2.2.1. Data Nodes. These are the nodes to input, compute or show smart contract data. They are similar to base types used in programming: string, number, enum or date. At the first glance they could appear simple, however they can contain complex rules to validate their values or call external API to fetch some data, exchange rate, for example. In this type of nodes, data models are defined to describe values for a smart contract parameters and options. All data nodes contain a label value to show in a resulting description sentence.

2.2.2. Flow Nodes. These are the nodes to control description flow and sentence construction from Data Nodes. In the simplest version of the language, two kinds of nodes are sufficient - for Series and Parallel flows of the Data Nodes. However, to make sentence formation more flexible, we can also add a Set block. Set blocks are similar to Series but they do not require strict following of sequence and can be empty. All Flow Nodes can be nested in each other without any limitations.

With help of these node types, we can represent the required part of token creation query as follows:

![Diagram of token creation query]

Token fungibility property has additional nesting just for illustration of this possibility. To get the same result, we can use a Data Node with an option selected from the list.

2.3. High-Level Language Structure
At the low level, the language structure looks obvious, however how would it be possible to obtain smart contract description from these nodes and how would it look in a sentence? What words would it contain? High-level structures have to be defined to answer these questions.

2.3.1. Contract Description. In view of the requirements to keep the language constructions near-natural, it was decided to make the predefined sentences for the base finance instruments. After carefully considering these sentences, each one of them was divided in two parts: the Required and the Optional.

In the Required part of the sentence, there is an instrument definition or a finance operation description. In the Optional part of the sentence, there are some available parameters that can be omitted without any restrictions. For example, in the process of creating a new token, we could insert a description of what it should represent as a digital asset. However, this part can be omitted.

To construct an interface and autocomplete functions for smart contract descriptions, Contract Description blocks name property has to be added.

2.3.2. Contracts Container. To make a complex product that, for example, allows adding a collateralized debt position and a debt risk option, users should be able to combine some sentences with each other. Moreover, there is an option to make custom products that are lacking at the real-world finance markets. It can achieved by uniting the optional parts from different Contract Description in one Container.
2.4. Language Security
Since smart contracts work with real currencies, the security of the solutions needs to be taken into account. It could be achieved through input validation rules in Data Nodes and with validation of Contract Description nodes inside Contracts Containers.

After validation of the result query, a transaction can be formed, signed and sent to the blockchain.

3. Conclusion
In this article we have presented a concept for a smart contract description language and defined the language requirements and structure. The task of creating a special language for describing smart contracts for the financial industry is solvable. The stated requirements for storing the structure of the language and providing opportunities for its extension from blocks are realizable and are now undergoing practical testing, the results of which will be published later.

The approach to building the Restricted Move language from individual blocks allows gradual building up of the power of the language and adding new syntax to describe other financial instruments or complicate the internal logic of blocks in order to use a series of synonyms when forming phrases. In the future, it is possible to introduce new high-level blocks that will recognize phrases intelligently and parse them into separate blocks.

References
[1] Vitalik Buterin. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. 2013. URL {https://ethereum.org/en/whitepaper/}.
[2] F. Franz, T. Fertig, A. E. Schütz and H. Vu, "Towards Human-readable Smart Contracts," 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Seoul, Korea (South), 2019, pp. 38-42, doi: 10.1109/BLOC.2019.8751309.
[3] M. Wohrer and U. Zdun, "From Domain-Specific Language to Code: Smart Contracts and the Application of Design Patterns," in IEEE Software, vol. 37, no. 5, pp. 37-42, Sept.-Oct. 2020, doi: 10.1109/MS.2020.2993470.
[4] E. Regnath and S. Steinhorst, "SmaCoNat: Smart Contracts in Natural Language," 2018 Forum on Specification & Design Languages (FDL), Garching, 2018, pp. 5-16, doi: 10.1109/FDL.2018.8524068.
[5] V. K. Dwivedi and A. Norta, "A Legally Relevant Socio-Technical Language Development for Smart Contracts," 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Trento, 2018, pp. 11-13, doi: 10.1109/FAS-W.2018.00016.