

# Platform for Modeling of Algebraic Behavior: Experience and Conclusions

Oleksandr Letychevskyi<sup>1</sup>[0000-0003-0856-9771], Volodymyr Peschanenko<sup>2</sup>[0000-0003-1013-9877],  
Maksym Poltoratskyi<sup>2</sup>[0000-0001-9861-4438] and Yuliia Tarasich<sup>2</sup>[0000-0002-6201-4569]

<sup>1</sup>V.M.Glushkov Institute of Cybernetics of the NAS of Ukraine,  
<sup>2</sup>Kherson State University, Universytets'ka St. 27, 73000, Kherson, Ukraine

oleksandr.letychevskyi@litsoft.com.ua,  
{vpeschanenko, mpoltorackiy, yutarasich}@gmail.com

**Abstract.** In the article the platform for modeling of algebraic behavior is considered. It is intent for formalization of the systems, especially distributed, their modelling and analysis of the properties. Platform is used for the formal verification, model-based testing and security issues detection. Behavior algebra specifications are used as the modeling language. The number of projects is considered in the paper implemented on platform in cybersecurity, blockchain solutions and legal requirements processing.

**Keywords:** insertion modeling, formal verification, algebraic behavior, symbolic modeling, formalization.

## 1 Introduction

The current stage of development of the software systems industry is characterized by a significant complication of the process of their development. Today are widely used cyber-physical systems, blockchain technologies, and, in particular, smart contracts, Distributed ledger technology (DLT), etc.

Cyber-Physical Systems (CPS) are used in space research, transport management, in the energy sector, in production management, in the military sphere, in medicine, in the construction of modern infrastructure, for contactless control of consumer electronics, etc. Typically, CPS are the systems with critical application area. Therefore, when designing such systems, increased demands are placed on their reliability and safety [1].

Distributed ledger technology is information storage technology, the key features of which are the sharing and synchronization of digital data according to the consensus algorithm, the geographical distribution of equivalent copies at different points around the world, the absence of a central administrator [2]. Accordingly, like as any new technology, blockchain has its own vulnerabilities. Thus, all algorithms that are used require the careful analysis and verification, namely, the checking the stability of the system against various attacks, such as Double spending attacks, Grinding attacks,

Transaction denial attacks, Desynchronization attacks, 51% attacks, Selfish-mining, etc.

At the same time, the existing methods of quality control of the developed systems are characterized by incompleteness, high complexity and insufficient reliability. This situation inevitably entails an increase in the number of errors in the development of systems.

As a rule, an analysis of the compliance of a system with the requirements that are imposed on it is carried out either by means of its visual analysis or by a testing method. However, if any of the properties of the system can be formally expressed, for example, in the form of a formula of mathematical logic, then the analysis of this property can be carried out by verification methods.

As a rule, verification is used to analyze the first requirement to the system - its correctness. Note, that this requirement is the main one.

In the first section of this article, a brief description of the current state of theoretical studies in the field of verification and short overview of developed systems for model verification is given. In second section of this article our approach to verification and model testing is presented. The third section of this article describes the Main capabilities and functionality of the platform. In the fourth section a short description of few examples of implemented models is given.

## **2 Related Works**

The problems of modeling and verification of software systems occupy a central position in research on the mathematical theory of programming. This is primarily due to the high relevance of creating a theoretical foundation for the development of reliable software.

In [3-4], the current state of theoretical studies in the field of CPS and their applications in practice is described. The prospects and significance of this direction are justified by the development of roadmaps for the development of this field of research in the USA [5] and in Europe [6].

Since 1970, more than 25 research projects on the use of information technology in legal activity have been developed. The problem of using information technology to legal regulation is not new to the EU, so there are legal policy modeling systems in the EU: EUROMOD, SYSIFF and POLIMOD. In Finland and the Netherlands, JURIX was created by the organization of researchers in the field of law and informatics, who are engaged in computer analysis of legal texts and documents. Thus, article [7] is focused on the method that was developed to model the legislation. The basis for modeling the regulatory framework is UML/OCL. Article [8] presents experience in the analysis of Luxembourg regulatory framework. The idea is also to use a UML approach to model procedural rules. Generally, the model is based on the Domain-Specific Modeling (DSM) approach.

In [9], is also used by UML Activity Diagrams for legal rule formation. After that when models are generated, the resulting model is automatically translated into a

model with using OCL expressions. The resulting OCL legal model can be used for automatic analysis by OCL solvers [9].

Over the past five years found a place the Semantic Web technology and using of ontologies theory in jurisprudence. Thus, tools based on the Semantic Web and ontology theory are generally intended not only to optimize the search for legal information, but also as a tool for clustering, classifying and managing legal knowledge [10].

The paper [11] discusses the approach to constructing agent economic models with using of finite state machines. The authors of this article note, that this approach can be used to model the markets for absolute and monopolistic competition. The article discusses the use of finite state machines for specification and simulation of the Walras model.

The article [12] presents a framework for the analysis and formal verification of Ethereum smart contracts using F\* (functional programming language designed to test programs).

There are various technologies of requirements verification. You can use the traditional method of model checking (model checking) or use deductive methods of symbolic modeling (symbolic modeling) with using an automatic proofing and specialized programs such as provers and solvers.

There are many of tools for simulation modeling of behavior in different domains, such as Maple, MatLab, Powersim, Ithink, Arena for economic modeling, SmartCheck, EtherTrust, ZEUS for smart-contracts, EUROMOD, SYSIFF and POLIMOD for jurisprudence, etc.

All systems are popular and can be used in various scientific fields. But the rapid pace of information technology development, and, in turn, the increasing complexity of software, requires to finding new approaches and solutions. That is why we are working to improve formalization technologies. We use algebraic methods to solve problems more precisely and use more expressive languages.

The study is a continuation of the previous authors works in which were presented using methods of algebraic programming and insertion modeling for verification and simulation of crypto-economics, legal, economics models, hardware systems.

At the articles [13-15] considers the formal methods approach for token economy modeling, analysis and studying of its properties. It uses an insertion modelling technique for verification of token economy and behavior algebra specifications for formalization. The project SKILLONOMY is considered as an example of algebraic approach application. The formalization and properties analysis is considered with usage of insertion modeling platform.

In the paper [16] we present the technology and system where algebraic approach demonstrates formal proving of correctness or irregularity of tax actions for tax payers correspondingly to the law. Given example from practice of taxation illustrates findings in Tax Code and inconsistencies of decisions of Taxation Office. Algebraic Programming System is used for detection of such collisions and proving of incorrectness by automatic reasoning from formalized legal requirements.

An example of describing CPS in the IMS system is presented in [17]. The ways of using algebraic interaction theory and insertion modeling technology to solve problems of analysis and synthesis of cyber-physical systems are shown in the article.

In the papers [18-20] overviews the main concepts of insertion modeling, presents new algorithms developed for symbolic verification, especially a new predicate transformer for local descriptions, and provides a formal description of the method of generating traces from such specifications (which is the key technology used to verify requirements and derive test suites).

### 3 Verification and Model Testing. Our Approach

Model verification is to investigate the behavioral properties of safety or viability. To prove the reachability of the properties, that are checked, can be used the following formal methods of behavior algebra:

- Static - methods for which it is sufficient to prove the feasibility property that is given by the formula in the base language. In other words, there are some values of formula attributes where the formula can be true. Proof of the property of feasibility is achieved by the using of solving machines such as Microsoft Z3 and cvc4.
- Dynamic - Applied to behavioral properties (properties where the formula is represented by expressions of behavior algebra and by semantics of actions, where the formula is represented as an expression in the base language) [21].
- Combined - methods for which the proving of feasibility of formulas is not enough, and it is necessary to prove the reachability of the formula by symbolic modeling.
- Partially dynamic - use the technique of invariants generating [22] in behavior algebra.

A model in the form of algebra of behavior and the property formula, that is verified, are inputs to the verification procedure, and verdict that confirms its reach and a scenario that leading to the corresponding state is its output.

Within the framework of behavior algebra uses such basic verification methods as incompatibility (nondeterminism), incompleteness (deadlocks), compatibility of timing properties, and synchronization problems, critical states (expressing a state of "starvation" when all agents are waiting for receiving of signals, a state of violation of restrictions or other security properties that can be specified by the user), signals racing (when different signal sequences create different states of the environment), properties of liveness, that expressing the reachability of the desired state.

**Model testing.** To create a test set in a suitable testing environment, an algebraic model can serve as a model for traces generating. At each of the development phases can be checked model artifacts. Both design specifications and binary code can be tested. Direct and inverse symbol modeling are the main methods of test generation.

Necessary covering of code lines by tests is determined by both the requirements for the generation of tests and the model where exhaustive testing can be considered, namely, the generation of all possible states of model behavior, coverage of actions in a model, or coverage of all transitions between actions.

The test is the sequence of reception and sending signals between agents. When performing tests, there are two instances - instance that tests and instance that being tested. Test signals are sent to tests instance and then performed a comparison with the expected result.

The following types of testing are considered when using behavior algebra:

- black box method - a set of tests where the code of the instance that being tested is unknown;
- white box method. The process is determined by the symbolic execution of the parallel composition of model and code. When execution, the states of the environment are compared on equivalence.

## **4 Main Capabilities and Functionality of the Platform.**

### **4.1 Main Components**

The platform consists of the following components:

#### **1. Algebraic Programming System (APS)**

APS was developed by the Glushkov Institute of Cybernetics of the National Academy of Science of Ukraine. It was the first system of term rewriting which used the systems of rewriting rules and rewriting strategies separately [23]. The main goal of APS is to create an algebraic program that solves mathematical problems. APS include Proving and Solving Systems.

A Proving System give us a possibilities to prove some trueness of a formula in a theory if the axioms and relation are given. For the resolving equations in Solving Systems in APS are implemented the following theories: enumerated types theory, Boolean logic, linear arithmetic, float arithmetic, string solving, and etc.

#### **2. Insertion Modelling System (IMS)**

Insertion Modelling System is an environment for the development of insertion machines, used to represent insertion models of distributed systems [24]. IMS is an extension of APS.

Insertion modeling focuses on building of models and studying the interaction of agents and environments in complex distributed multi-agent systems. The main notion of IMS is the insertion function, which defines the behaviors of agents and actions of environments. We used the behavior algebra specifications for the formalization for the insertion modeling method [25] and the deductive or symbolic method in IMS based on the such external provers and solvers, as Presburger – omega, Fourier-Motzkin - reallib (our tool), cvc3, z3 and MathSAT.

#### **3. Algebraic Engine**

An Algebraic Engine is an algebraic tool for modeling of system behavior. It is created on the basis of the IMS system.

It has the following features:

- Generation of symbolic behavior scenarios;
- Resolve the problem of reachability of some property (safety, liveness, and security violations). It detects the reachability of a property given as a formula and given as a behavior in a behavioral algebraic expression;
- Provide verification by symbolic modeling (with the usage of slice technique) of protocols, programs, models and other behavioral specifications given in behavioral algebraic expressions;
- Map symbolic modeling to original language specifications.

#### 4. SymTech Platform

SymTech is a system with the use of Symbolic Technology, which involves algebraic and deductive formal methods for the resolution of sophisticated industrial challenges.

The main features of SymTech Platform are:

- testing technologies;
- model-driven development;
- support of the development process of a critical system or system with Quality of Service (QoS);
- verification and validation;
- cybersecurity.

#### 4.2 Functionality of the Platform. Main Window

SymTech (Symbolic Modeling) platform includes a number of system and libraries for implementation of the algebraic formal methods and integration with other program systems.

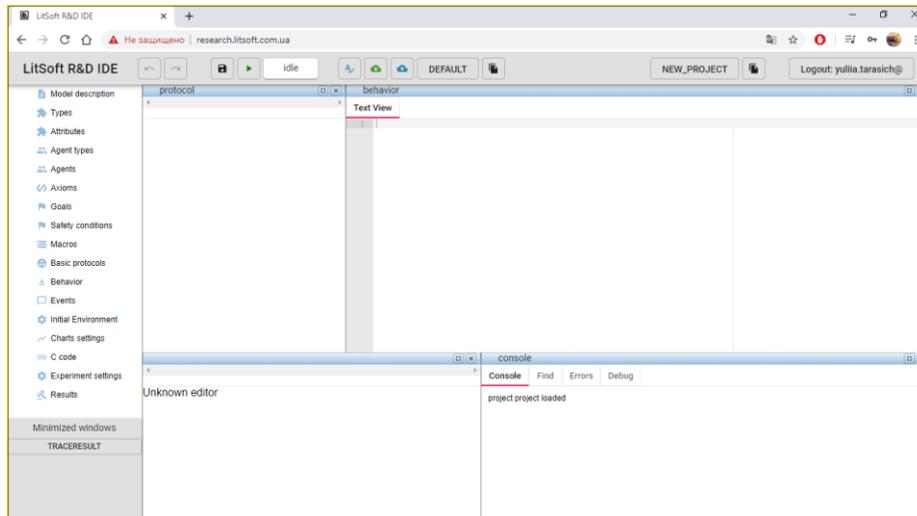
To get started, you must log in to your account on the platform. Once you have logged into your account, you can create a new project, or open a previously created one. After creating a new project, you will see the following arrangement of menus and windows (Fig. 1):

Thus, we have a vertical and horizontal menu for our platform:

- Horizontal menu. Gives us possibilities to make some manipulations with project, such as: back to previous and next action; create, save, start, copy, export or import of project; creation of different experiment with a model, creation a copy of experiment.
- Vertical menu. In this menu we see the list of items that are the key definitions of formalization and are the basis for model formalization and, also a list of special platform tools.

As you can see, the main workspace is occupied by such windows as "protocol", "behavior", "console", etc. (Fig.1). They open by default, but can be closed, minimized or maximized depending on your needs. In these windows, after creation, will

be displayed the key elements of the model, startup results (in console), etc. We also will use them to edit and describe the necessary elements.



**Fig. 1.** SymTech (Symbolic Modeling) platform

Let's consider the main functionality of the platform on the concrete examples.

#### **4.3 Examples of Models Formalization. Requirements for a Model. Coffee Machine.**

In this example we describe the model of how a coffee machine works.

We have an automated machine which interacts with the external environment by means of coins - input data, of certain denomination and two buttons - «Make coffee» and «Cancel». Requirements for the operation of a coffee machine are given below.

##### ***Requirements for the operation of a coffee machine.***

- R1. The coffee machine contains slots for 5, 10, 25, 50 cent coins, display and sensor buttons «Make coffee» and "Cancel".
- R2. The coffee costs 95 cents.
- R3. After inserting coins into the slot corresponding to the amount, the sum will be shown on the display.
- R4. If the total of inserted coins is greater or equal to 95 cents, then after pressing the button «make coffee» a drink is delivered and the corresponding change returned.
- R5. Upon pressing the button "Cancel," after inserting coins, the total amount will be refunded.

The example contains agent – automatic machine which interact with the external environment.

Requirement R1 states that the agent - automatic machine communicates with the environment by means of coins - input data, of certain denomination and two buttons, which can be written as the following, having entered the appropriate integer attribute coin, satisfying the condition:

$$(coin = 5) \vee (coin = 10) \vee (coin = 25) \vee (coin = 50)$$

Requirement R2 says that there is a certain factor that contains attributes of price, which can be written down as the following:

$$CoffeePrice = 95$$

The requirements enclose one more attribute related to the amount of inserted coins. Let's designate it by means of the identifier "SUMMA."

It is also required to define the obvious fact that if no coins were inserted into the automatic machine, then  $SUMMA = 0$ . Thus the agent is formed with variety of attributes which form its type.

For creating a model at the platform we must make next steps:

1. **Add an Agent Types and Agent Attributes.** Add an Agent Types and Agent Attributes. In the vertical menu you can see item "Agent types" (Fig.2 (a)) in that we can add types of agents. After that we can choose needed agent type and in the special window we can add agent attributes (Fig.2 (b)).

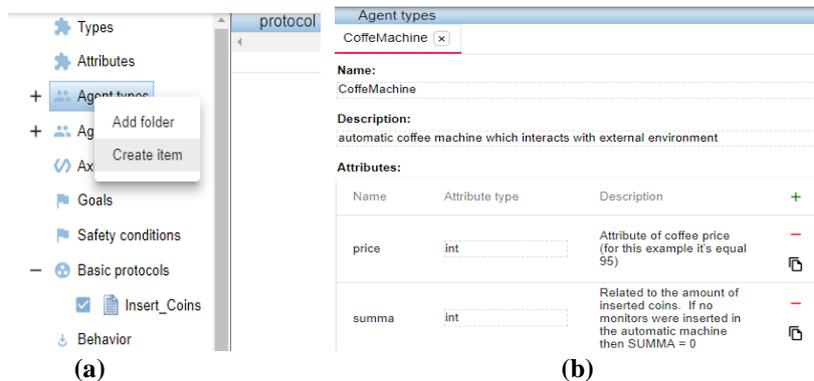


Fig. 2. "Agent types"

2. **Add an Agents.** After the finishing of the describing of all agents types, we must add the instances of agents. In our example, it's only one agent. The agent name is CoffeeMach1 and its type is CoffeMachine (Fig.3(b)).

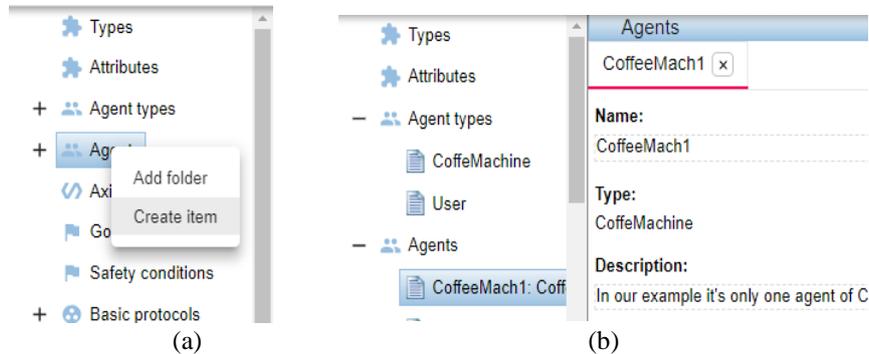


Fig. 3. Agents

It is important to identify the type of agent in those cases when there are only a few of them and the agent expressions shall be used for making these distinctions. In this simple given example, so long as there is only one agent we can consider agent attributes as attributes of the environment.

3. **Add Trigger Events.** The next step is to determine all trigger events (Fig.4). Requirements R3, R4, R5 describe the reaction of the agent to exposure in the environment. They explicitly define trigger events – insert coins, press button “make drink,” push button, “return of the sum.” Thus, preconditions for triggers - presence of inserted amount and the fact that the inserted sum is greater or equal to the price of coffee - are explicitly defined.

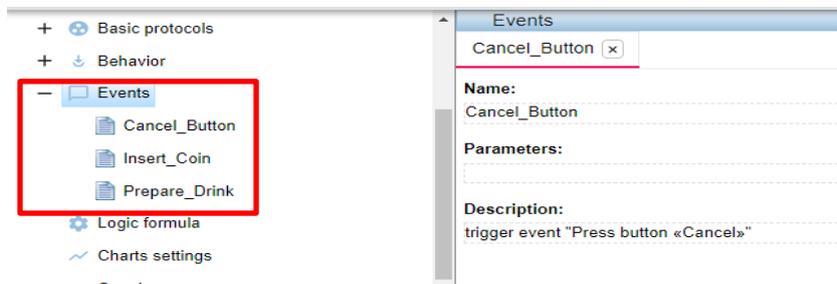
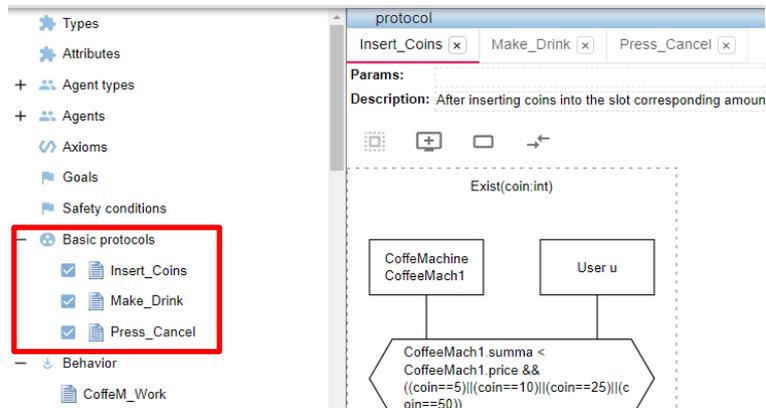


Fig. 4. Trigger events

Such as one of the trigger events is "insert coins," so we declare, for this event, the parameter x of integer type.

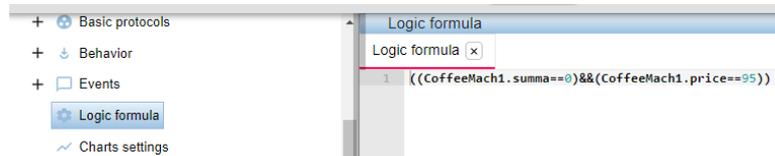
4. **Add Basic Protocols.** All basic protocols are presented in the list of "Basic protocols" in the vertical menu. Any of them must be described on the special tab "Basic protocols". Their MSC-diagrams must be described on this tab too (Fig.5).



**Fig. 5.** List of basic protocols

It should be mentioned, that the attribute coin was not defined at first. The attribute coin in Insert\_Coins protocol is used as a parameter of basic protocol or local attribute whose value is saved only during the execution of basic protocol. After the application of a basic protocol, it becomes indefinite again.

**5. Initial Environment.** In the Initial Environment we describe the initial values (Fig.6)



**Fig. 6.** Initial Environment

**6. Behavior.** Requirements do not clearly define a certain sequence of application of basic protocols. Initially, only the R3 protocol can be applied and the rest after including itself. After the R4 protocol it is possible to repeat preparation of new portion of drink or complete operation. This sequence is represented in the form of the following UCM-diagram (Fig.7)

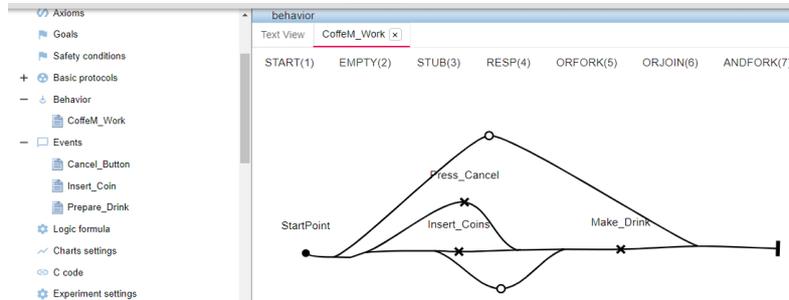


Fig. 7. UCM-diagram CoffeeM\_Work

The reflection of UCM-notation in a text form must be written in the tab "TEXT." For this example, we have the following description (Fig.8).

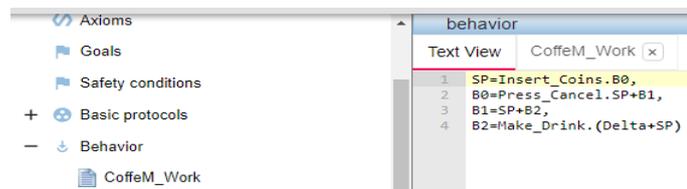


Fig. 8. UCM-notation in a text form

7. **Simulation results.** After starting the program of trace symbolic generation, we get a result, where the behavior of the model is represented by eighteen traces (Fig.9).

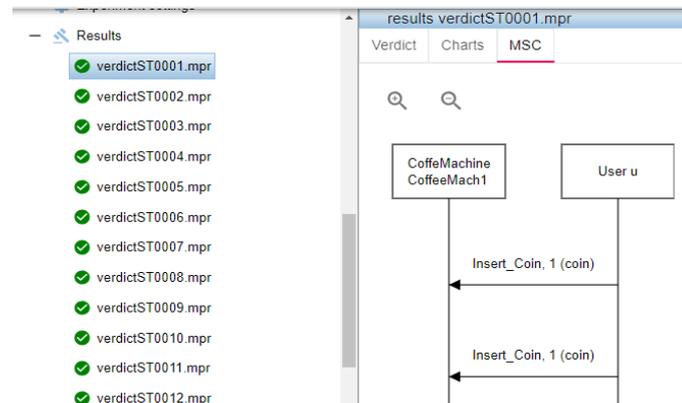


Fig. 9. Simulation results

The text verdict which contains the statistics of trace generation is created after the end of the operation of trace generation. The number of traces, which leads to the

state determined by goal, state the formula such as deadlock states, visited states, states of safety violation and states of possible non determinism is displayed.

The statistics of basic protocol application and the list of not applied basic protocols are provided. The time of trace generation is also documented.

In the given traces information display regarding the symbolic state of the environment, is disabled and there are only those elements illustrating the application of basic protocols, namely messages in the form of arrows and the local actions of an agent.

## **5 Examples of Projects.**

### **5.1 SKILLONOMY Project**

SKILLONOMY is an educational online platform that tokenizes productive activities in the learning process and is focused on gaining monetized online knowledge and skills. The SKILLONOMY ecosystem is built around an IT platform that allows participants to effectively build and administer the relationships that are related to training, investing and sharing experience. Developing the SKILLONOMY project required a set of essential functions of the blockchain for ensuring a stable and efficient system that works [26].

The main purposes of the tokenomics model formalization of the SKILLONOMY project are:

- the search for modeling errors, such as finding failings or possible contradictions;
- the search for effective scenarios of the system in the model, etc.;
- the possibilities for analyzing and predicting the model; and
- the possibilities for analyzing the feasibility of project financing.

The process of the formalization of the tokenomic project consists of the following steps: selection of the agents and definition of their attributes corresponding to the level of abstraction demanded, definition of agents' actions and the design of agents' behavior.

As a result of the simulation of the model and with the selection of different values of attributes, we were able to analyze the behavior of the SKILLONOMY model according to different bitcoin trends.

When we model the specifications, we simulated the activity of all agents that take part in such processes as sending and receiving tokens and selling and purchasing tokens on the stock exchange. In the process of modeling, we were able to monitor the dynamics of attributes and main tokenomic indicators. Moreover, unlike to previous results, in the latest version of model we have modeling the situation when in the system can be present two or more students group and the number of students may increase.

This model allows changing hypotheses to evaluate the risks when selecting the worst conditions in the process of tokenomic modeling. One of most important ad-

vantages of tokenomic modeling is the opportunities to debug the system and to change the algorithm or boundary values of attributes to reach the demanded results.

Description of the model is given in [13-15].

## 5.2 Cybersecurity Projects

The algebraic approach in cybersecurity was demanded over the past two decades with the appearance of efficient solving and deductive tools. Different techniques like symbolic modeling and concolic computations use the algebraic approach that has created more possibilities for methods of detection in cybersecurity.

Our tools anticipate algebraic matching in different modes. The following applications are available and are under active research:

### 1. *Definition of Attacker Behavior*

The project has been started together with the Glushkov Institute of Cybernetics in 2017. The goal of the project is to use the algebraic formal methods in the detection of attack and vulnerability in the software systems.

One of the decisions is the usage of behavior algebra algorithms and the theory of insertion modeling. Types of intruder attacks from CVE database, especially Meltdown and Spectre are presented as behavior algebra expression and detection of vulnerability is defined by resolving of behavior.

### 2. *Vulnerabilities Detection*

The project has started 2016 year with the attempts to formalize the vulnerability “Heartbleed” and try to detect it in Algebraic Programming System.

The results (together with V.Sukhomlinov from Intel) are presented in a paper [27].

The technology of vulnerability formalization has been developed together with the Institute of cybernetics. The technique of vulnerability semantics presentation in behavior algebra expressions has been presented in papers [28, 29]:

The main benefit of an algebraic approach is that we can more accurately detect vulnerabilities. The description of the vulnerability covers a set of its possible scenarios.

The implementation of two or more matching levels can significantly increase the detection efficiency. Thus, matching at the control flow level can be implemented first as a fast procedure, and then, the detected set of scenarios can be modelled symbolically as the more expensive stage.

## 5.3 Tax Code.

We considered the subset of Tax Code and created its formalization that contains all articles belonged to VAT (Value Added Tax) and its dependencies from additional by-laws.

For presentation of algebraic model we use double sort algebra that contains the algebra of behavior and logical language with arithmetic, set-theoretic and logic oper-

ations. Formulas of this language are constructed from operations and predicates over integer, real, enumerated, and symbolic types [30].

Usually in Tax Code we have two kinds of agents that are – tax payer and taxation office. We can consider the set of Tax Payers that interact with each other.

The given approach could be applied to legal requirements that could be formalized by such manner. We can watch the possible scenarios and analyze the environment in system to achieve the right formal presentation. The traces were verified by third person with economic background and corresponding correction were made.

Now the “digital” Tax Code of Ukraine is ready for use in the scope of Algebraic Programming System and could be deployed in interested organization.

Description of the model is given in [16].

## **6 Conclusions**

Behavior algebra and basic protocol specifications are used for the formal description of a model. Algebraic artifacts such as theorems and abstract algorithms provide a basis for formal verification methods. These methods were implemented and successfully applied to industry projects in Motorola, Inc.

Formal verification anticipates the checking of the properties of the system. The properties could be classic, such as the absence of deadlocks and non-determinisms, or subject domain specific defined safety or liveness conditions. We can prove the reachability of demanded properties by using a symbolic modeling method developed in the scope of IMS.

The given approach could be applied to models that could be formalized by such manner. One of the difficulties is the process of formalization. Due to the experience there should be two specialists involved in the process of the formalization – algebraists and specialist in specific area of project.

## **Acknowledgements**

We would like to thank the company LitSoft Enterprise R&D [31] for the opportunity to work with the platform for modeling of algebraic behavior for our research and experiments in the modeling area. We are also grateful to the Glushkov Institute of Cybernetics of NAS of Ukraine for the theoretical and practical results in the field of verification that were used as a basis for our studies of formalization and algebraic modeling in the tokenomics projects area and to the Kherson State University for the active supporting of Insertion Modeling System.

## **References**

1. Lee, E: Cyber Physical Systems: Design Challenges. In: 11th IEEE Int. Symp. on Object Oriented Real-Time Distributed Computing (ISORC), pp. 363-369. Orlando, FL. USA (2008).

2. What is DLT?, <https://www.qpiter.com/what-are-dlts/>, last accessed 2020/03/09.
3. Shi, J., Wan, J., Yan, H., Suo, H.: A Survey of Cyber-Physical Systems. International Conference on Wireless Communications and Signal Processing (WCSP), pp. 1-6, Nanjing (2011).
4. Ashibani, Yosef, Mahmoud, Qusay: Cyber Physical Systems Security: Analysis, Challenges and Solutions. *Computers & Security*. 68. 81–97. (2017).
5. Foundations for Innovation in Cyber-Physical Systems. Workshop Summary Report, <https://www.nist.gov/system/files/documents/el/CPS-WorkshopReport-1-30-13-Final.pdf>, last accessed 2020/03/19.
6. Cyber-Physical European Roadmap & Strategy, [www.cypthers.eu](http://www.cypthers.eu), last accessed 2020/03/09.
7. Tom van Engers, T. et al.: POWER: using UML/OCL for modeling legislation-an application report. In Proceedings of the 8th international conference on Artificial intelligence and law, pp. 157-167 (2001).
8. Soltana, G. et al.: Using UML for modeling procedural legal rules: Approach and a study of Luxembourg's Tax Law. In International Conference on Model Driven Engineering Languages and Systems, pp. 450-466. Springer, Cham (2014).
9. Cabot, J., Clariso, R., Riera, D.: Verification of UML/OCL class diagrams using constraint programming. In: Proc. of 2008 IEEE Conf. on Software Testing Verification and Validation Wrkshp. (ICST'08). pp. 73–80 (2008)
10. Casanovas, P. et al.: Semantic web for the legal domain: the next step. *Semantic Web*, 7(3), pp 213-227 (2016).
11. Gnilomedov, I.: Modeling of economic agents using finite state machines. Scientific reports "Integrated Models, Soft Computing, Probability Systems and Program Complexes in Artificial Intelligence.", Fiz-matlit, Moscow, pp.72-89 (2009).
12. Bhargavan, K. et al.: Formal verification of smart contracts: Short paper In Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, pp. 91-96 (2016).
13. Letychevskiy, O., Peschanenko, V., Poltoratskiy, M., & Tarasich, Y.: Our Approach to Formal Verification of Token Economy Models. In International Conference on Information and Communication Technologies in Education, Research, and Industrial Applications, pp. 348-363, Springer, Cham (2019).
14. Letychevskiy, O., Peschanenko, V., Radchenko, V., Poltoratzkiy, M., Kovalenko, P., & Mogylo, S.: Formal Verification of Token Economy Models. In 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 201-204, IEEE (2019).
15. Letychevskiy, O., Peschanenko, V., Radchenko, V., Poltoratskiy, M., Tarasich, Yu.: Formalization and Algebraic Modeling of Tokenomics Projects. In ICTERI Workshops, pp. 577-584 (2019).
16. Letychevskiy, A., Letychevskiy, O., Peschanenko, V., Poltorackij, M.: An Algebraic Approach for Analyzing of Legal Requirements. In 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), pp. 209-212, IEEE. (2017).
17. Letychevskiy, A., Letychevskiy, A., Jr., Skobelev V., Volkov V. Cyber-physical systems. *Cybernetics and systems analysis*, 53(6), 3–19 (2017).
18. Letychevskiy, A., Letychevskiy, O., Peschanenko, V., Weigert, T. Insertion modeling and symbolic verification of large systems. In International SDL Forum, 3-18, Springer, Cham (2015).
19. Letychevskiy, A., Letychevskiy, O., Peschanenko, V., Huba, A. Generating symbolic traces in the insertion modeling system. *Cybernetics and Systems Analysis*, 51(1), 5-15. (2015).

20. Letichevsky, A., Godlevsky, A., Letychevsky, A., Potiyenko, S., Peschanenko, V. Properties of a predicate transformer of the VRS system. *Cybernetics and Systems Analysis*, 46(4), 521-532. (2010).
21. Letichevsky, A., Letichevskiy, O.: Predicate transformers and system verification. In T. Jebelean, M. Mosbah, N. Popov (eds.): *SCSS 2010*, volume 1, issue: 1, pp. 118-130 (2010).
22. Letichevsky, A, et al.: Invariants in symbolic modeling and verification of requirements. In *Ninth International Conference on Computer Science and Information Technologies Revised Selected Papers*, pp. 1-6., IEEE (2013).
23. Letichevsky, A., Kapitonova, J.: Algebraic programming in the APS system. In *Proceedings of the international symposium on Symbolic and algebraic computation*, pp. 68-75 (1990).
24. Letichevsky, A., Letychevskiy, O., Peschanenko, V. Insertion modeling system. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pp. 262-273, Springer, Berlin, Heidelberg (2011).
25. Letichevsky, A., Kapitonova, J., Volkov, V., Letichevsky, A., Baranov, S., Kotlyarov, V., Weigert, T.: System Specification with Basic Protocols *Cybernetics and Systems Analysis*, 41, 479–493 (2005).
26. SKILLONOMY, <https://skillonomy.org/en/>, last accessed 2020/03/28.
27. Letychevskiy, O., Sukhomlinov, V.: An Algebraic Approach for the Detection of Vulnerabilities in Software Systems. In the *Third International Conference on Electronics and Software Science (ICESS2017)*, pp. 53-60 (2017).
28. Letychevskiy, O., Hryniuk, Y., Yakovlev, V., Peschanenko, V., Radchenko, V.: Algebraic Matching of Vulnerabilities in a Low-Level Code. *The ISC International Journal of Information Security*, 11, special issue, pp. 1-7 (2019).
29. Letychevskiy, O., Peschanenko, V., Radchenko, V., Hryniuk, Y., Yakovlev, V.: Algebraic Patterns of Vulnerabilities in Binary Code. In *10th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Leeds, United Kingdom, (2019).
30. A. Letichevsky, V. B. Kudryavtsey, LG. Rosenberg, "Algebra of behavior transformations and its applications", *Structural theory of Automata Semigroups and Universal Algebra NATO Science Series II. Mathematics Physics and Chemistry*, vol. 207, pp. 241-272, 2005.
31. LitSoft Enterprise R&D, <http://litsoft.com.ua/>, last accessed 2020/06/14.