Invariants Classification Characteristics for Checking the Correctness of Computational Processes*

Sergei A. Petrenko[1][0000-0003-0644-1731], Krystina A. Makoveichuk[2][0000-0003-1258-0463], Alexander V. Olifirov[2][0000-0002-5288-2725], Nikolay N. Oleinikov [2][0000-0002-9348-9153]

[1]Innopolis University, Kazan, Russia
s.petrenko@rambler.ru
[2]V.I. Vernadsky Crimean Federal University, Yalta, Russia
christin2003@yandex.ru
alex.olifirov@gmail.com
oleinikov1@mail.ru

Abstract. Selecting the invariant classification characteristics of the program behavior of some secured infrastructure (in this task, into two classes: correct and incorrect execution) is identical to the isomorphism problem of the two systems under some mapping. I order to clarify the necessary and sufficient conditions for the system isomorphism, as well as to determine the isomorphism mapping qualitative and quantitative parameters, a similarity theory of the mathematical apparatus was developed. However, in the late 1980s, the results were applied in the field of modeling, applying the universal digital computers and then transferred to solve a much wider spectrum of problems, including cybersecurity and ensuring the required cyber resilience of the critical information infrastructure.

Keywords: inverse similarity theorem, dynamic control of correctness of calculation programs, the correctness of computing processes.

1 Introduction

The most detailed provisions of the similarity theory were developed concerning the processes, described by the homogeneous power polynomial systems [1, 3]. There are three main theorems in the similarity theory: the direct, inverse, and π-theorem. The similarity theorem, known as "π-theorem", allows identifying the functional relationship between variable processes in relative form. The deductions form the direct theorem and the "π-theorem" of similarity allowed formulating invariant informative fea-

tures for the correct behavior of some critical information infrastructure software [2, 4, 5].

2 Introducing a passport system for programs

Let us consider two processes of p1 and p2, which complete equations have the following form:

$$\sum_{i=1}^{q} \varphi_{ui} = 0 \text{, u = 1, 2, …, r;} \quad (1)$$

$$\sum_{i=1}^{q} \Phi_{ui} = 0 \text{, u = 1, 2, …, r;} \quad (2)$$

Where $\varphi_u = \prod_{j=1}^{n} x_j^{\alpha_{u1}}$ and $\Phi_u = \prod_{j=1}^{n} X_j^{\alpha_{u1}}$ – homogeneous functions of their parameters. The direct similarity theorem states that if the processes are homogeneously similar, then the following system takes place:

$$\frac{\varphi_{ui}}{\varphi_{uq}} = \frac{\Phi_{ui}}{\Phi_{uq}}, \quad (3)$$

u = 1,2, …, r; s = 1,2, …, (q-1).

Expressions

$$\pi_{us} = \frac{\varphi_{ui}}{\varphi_{uq}}, \quad (4)$$

u = 1,2, …, r; s = 1,2, …, (q-1)

are called criteria or similarity invariants and, as a theorem deduction, are numerically equal to all processes belonging to the same subclass of mutually similar processes.

Thus, the direct theorem formulates the necessary conditions for the correlation of the analyzed process with one of the subclasses. Sufficient conditions for the homogeneous similarity of two processes are given in the inverse similarity theorem: if it is possible to reduce the complete processes equations to an isostructural relative form with the numerically equal similarity invariants, then such processes are homogeneously similar [6, 7].

The similarity theorem, known as "π-theorem", allows identifying the functional relationship between variable processes in relative form. The deductions form the direct theorem and the “π-theorem” of similarity allowed formulating invariant informative features for the correct behavior of some critical information infrastructure software [9, 11].

3 Mathematical problem formulation

Imagine the computational process (CP) in the following form ((5), table 1):

$$CP=<T,X,Y,Z,F,\Phi> \quad (5)$$

Table 1. Mathematical problem formulation of a computational process

1	T	The set of times t at which a computational process is observed
2	X, Y	Sets of input and output parameters of the computational process
3	Z	Set of states of the computing process. Every state of the computational process is characterized at each moment of time by the sequence of arithmetic operations at the selected control point k.
4	F	The set of transition operators f_i, reflecting the mechanism of changing the states of the computing process during its execution, including arithmetic operations
5	Φ	The set of output operators ϕ_i, describing the mechanism of the formation of the result during the calculation

We introduce the following notation:

λ - Violation mapping of an arithmetic operation at a specific time t_i for given input parameters;

ψ – Mapping of the computational process regular invariants formation;

μ – Comparative mapping of standard and reference invariants of the computational process;

υ – Mapping of the signal generation about incorrect calculations;

ξ – Mapping of the arithmetic operations recovery, based on reference similarity invariants;

χ –Performed calculation correctness mapping, based on the recovered arithmetic operations.

In order to exclude the possibility of discreet modification, made by the calculation program, it is necessary to perform dynamic control of the executed computational process (Figure 1). Under the dynamic control of the computational program correctness, we will understand the correctness control of the performed arithmetic operations semantics, while their actual execution. Data for dynamic control must first be obtained as a program passport, resulting from its additional static analysis [8, 10].

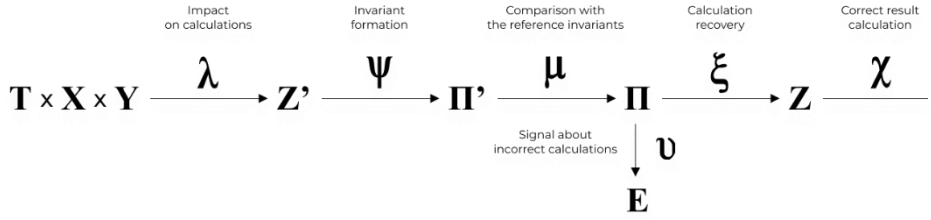

Fig. 1. The mapping diagram of the calculation correctness recovery

Impact on calculations, invariant formation, comparison with the reference invariants, signal about incorrect calculations, calculation recovery, correct result calculation.

In order to form the passport program the following actions are required:

1. Solving the observative problem (the computational process simulation by an oriented program control graph).
2. Solving the problem of presenting calculations by similarity equations on linear graph parts, i.e. to transform the arithmetic operations of the form:

$$z_i(x_1, x_2, \ldots, x_m) = \sum_{j=1}^{p} z_{ij}(x_1, x_2, \ldots, x_m) \tag{6}$$

To dimensionless form:

$$[z_{ij}(x_1, x_2, \ldots, x_m)] = [z_{il}(x_1, x_2, \ldots, x_m)], \; j,l = \overline{1,p} \tag{7}$$

1. Solving the problem of managing the computational process by comparing the semantic invariants with the program passport that means that it is necessary to find the maps:

$$\psi: Z' \rightarrow \Pi'$$
$$\mu: \Pi' \rightarrow \Pi \tag{8}$$
$$\xi: \Pi \rightarrow Z$$

Limitations and assumptions:

1. Considered set of arithmetic operations {+,–,*,/,=}
2. $t_i < t_{max}$, where t_i – computation time recovery, t_{max} – maximum allowable time to recover the correctness of the calculations.

Solving these problems allowed developing a new method to control the computational program semantic correctness, which complemented the known method capabilities to ensure the required cyber resilience of the secured critical information infrastructure [9, 12].

In order to control the software correctness, it was necessary to construct a program control graph.

Let us imagine some computational process in the form of a program control graph: G(B.D).

Where B = {B_i}is set of vertices (linear program part and D = {BxB} set of arcs (control connections) between them.

Here, each linear graph part $B_i \in B$ has its own arithmetic operator sequence, i.e.

$$B_i = (b_{i1}, b_{i2}, ..., b_{il}). \tag{9}$$

An ordered vertex sequence corresponds to each elementary (without cycles) route of the graph input vertex to output vertex:

$$B^k = (B_1^k, B_2^k, ..., B_t^k), \tag{10}$$

where $B^k \subseteq B$ and $B_i{}^k = (b_{i1}^k, b_{i2}^k, ..., b_{il}^k),\ \forall i = \overline{1, p}$ form a sequence of the executed arithmetic operators called a program implementation or a computational process. The arithmetic expression sequence data is the potentially dangerous program fragments.

The computational process algorithm was reduced to the graph representation form to derive the arithmetic expression operators from the control operators (conditional transitions, branching, cycles). As a result, in the control graph, all arithmetic expression operators were grouped on a set of linear program parts — the graph vertices, into which checkpoints (CP) were entered. Here, checkpoints were needed to determine the routing context within which the calculations take place. Moreover, the special systems of defining relations were constructed in the form of similarity equations at each checkpoint for arithmetic operators. The equation system solution allowed to form the matrices of similarity invariants to control the computational process semantics.

4 A similarity equations system development

The studies have shown that the most effective way to control the computation semantics is to test relations, based on theoretically based relations and computation features. Here the key relationships in the approach for detecting the parameters of the incorrect computational process functioning are some invariant, which is understood as the auto modeling (constant) presentation of program execution in the actual operating secured infrastructure conditions. The invariant generation problem, from the different program representations, is non-trivial and poorly formalized. In the program execution dynamics, only semantic invariants remain fully computable (repro-

ducible) (since they do not depend on the specific values of the program variables) [7, 13].

Let us imagine the implementation of B^k of the program control graph as an ordered primary relation sequence, corresponding to arithmetic operators:

$$\begin{cases} y_1 = f_1^k(x_1, x_2, ..., x_N), \\ y_2 = f_2^k(x_1, x_2, ..., x_N, y_1), \\ ... \\ y_M = f_M^k(x_1, x_2, ..., x_N, y_1, y_2, ..., y_{M-1}) \end{cases} \quad (11)$$

Having performed the superposition $\{y_i\}$ on X on the right relation sides, we obtain a relation invariant system according to the displacement:

$$\begin{cases} y_1 = z_1^k(x_1, x_2, ..., x_N), \\ y_2 = z_2^k(x_1, x_2, ..., x_N), \\ ... \\ y_m = z_m^k(x_1, x_2, ..., x_N). \end{cases} \quad (12)$$

The relation $y_i = z_i^k(x_1, x_2, ..., x_N)$ can be presented as:

$$y_i = \sum_{i=1}^{p_i} z_{ij}(x_1, x_2, ..., x_N), \quad (13)$$

where $z_{ij}(x_1, x_2, ..., x_N)$ – a power monomial.

In accordance with the Fourier rule, the summands (13) should be homogeneous in dimensions, i.e.

$$[y_i] = [z_{ij}(x_1, x_2, ..., x_N)] = [z_{il}(x_1, x_2, ..., x_N)], \; j,l = \overline{1, p_i} \text{ or}$$

$$[z_{ij}(x_1, x_2, ..., x_N)] = [z_{il}(x_1, x_2, ..., x_N)], \; j,l = \overline{1, p_i} \quad (14)$$

System (16) is a defining relations system or a similarity equation system.

Using the function $\rho = X \rightarrow [X]$, we associate each $x_j \in X$ with some abstract dimension $[x_j] \in [X]$. Then the summand dimensions (13) will be expressed as

$$[z_{ij}(x_1, x_2, ..., x_n)] = \prod_{n=1}^{N} [x_n]^{\lambda_{jn}}, \; j = \overline{1, p_i} \quad (15)$$

Using (14) and (15), we develop a system of defining relations:

$$\prod_{n=1}^{N}[x_n]^{\lambda_{jn}} = \prod_{n=1}^{N}[x_n]^{\lambda_{ln}}, \quad j,l = \overline{1,p_i} \tag{16}$$

which is transformed into the following form:

$$\prod_{n=1}^{N}[x_n]^{\lambda_{jn}-\lambda_{ln}} = 1, \quad j,l = \overline{1,p_i} \tag{17}$$

Using the logarithm method, as it is usually done, when analyzing the similarity relations we obtain a homogeneous system of linear equations from the system (17)

$$\sum_{n=1}^{N}(\lambda_{j\,n} - \lambda_{l\,n})\ln[x_n] = 0, \quad j,l = \overline{1,p_i} \tag{18}$$

Expression (16) is a criterion for semantic correctness.

Having performed a similar development for $\forall B_i^k \in B^k$, we obtain a system of homogeneous linear equations for k-implementation:

$$A^{\kappa}\omega = 0 \tag{19}$$

Generally, we can assume that the function ρ = X→[X] is surjective and, therefore, the B^k implementation is represented by a matrix $A^k = \|a_{ij}\|$ of size $m_k \times n_k$, which a number of columns are not less than the number of rows, i.e. $n_k \geq m_k$.

We say that the implementation of B^k is representative if it corresponds to the matrix A^k with $m_k \geq 1$, i.e. the implementation allows developing at least one similarity criterion.

Usually, a program corresponds to a separate functional module or consists of an interconnected group of those and describes the general solution of a certain task. Each of the implementations $B^k \in B$ describes a particular solution of the same problem, corresponding to the certain X components values. Since $B^k \cap B^l \neq \varnothing$ $\forall B^k, B^l \in B$ then the mathematical dependencies structure should be preserved during the transition from one implementation to another, i.e. similarity criteria should be common. Then the matrices $\{A^k\}$, corresponding to the implementations $\{B^k\}$, can be combined into one system.

Let the program have q implementations. Denote by A the union of the matrices $\{A^k\}$ corresponding to the implementations $\{B^k\}$, i.e.

$$A = \begin{pmatrix} A_1 \\ \dots \\ A_q \end{pmatrix} \tag{20}$$

The *A* Development can be carried out using selective vertices covering the implementations.

Thus, the matrices A union is part of the program passport and is a database of semantic standards $\{A^k\}$ for the linear program $\{B^k\}$ sections.

The similarity equation example.

Let us consider an assignment operator:

$$p = a*b + c/(d–e) \tag{21}$$

Here, the correct expression must be generated by some selected grammar, which depends on both the possible terms meanings and the chosen operations set. For a context-free grammar, each expression can be matched to an output tree in a unique way. Thus, an output tree can be used as an alternative expression representation.

When constructing a tree by the expression, the order of the calculations plays its role. Obviously, the vertex descendant values are calculated earlier than the ancestor vertex value. Therefore, the operation last performed will take place at the treetop. In order to construct a tree unambiguously, it is necessary to determine the operation calculation order in the expression, taking into account their priorities and the operation order with the same priority, including the case when calculating the same operation (associativity property). Usually, such expressions are calculated from left to right.

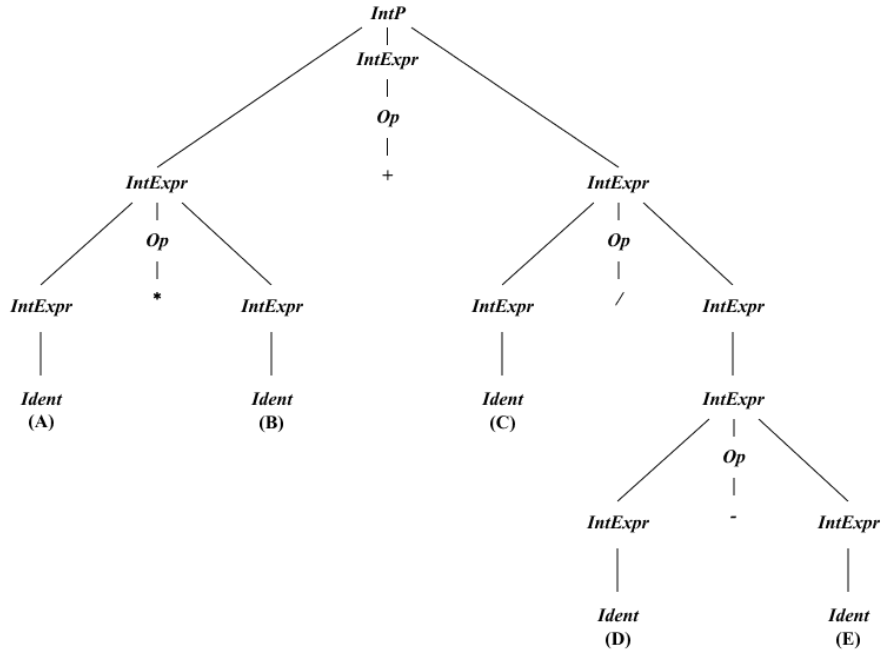

Fig. 2. Arithmetic expression generation tree

The constructed tree will definitely correspond to the specified expression taking into account the calculation order.

We formalize the arithmetic expressions:

Let Op {+,–,*,/} be an arithmetic operations set under consideration.

Terms is a set of terms, consisting of possible objects that can be operation arguments.

Expr is a set of all possible expressions, and $Terms \subset Expr$.

$elem(o,e) \in Expr$ - many other elements, and $o \in Op, e \in Expr$.

Thus, an arithmetic expression is either a term or an operation connecting several expressions.

The expression (20) with the set of terms Terms = {p,a,b,c,d,e} and the binary operations set Op {,-,*,/} will be represented as:

elem: (=,p,(,(*,a,b),(/,c,(-,d,e)))).

The arithmetic operator execution correctness can be assessed using the appropriate semantic function. When applied to expressions, the semantic function $T : a \to [\,a\,]$ assigns to each argument some abstract entity or dimension [a]. Thus, the arithmetic operations, performed on program variables during the program execution are in fact operations on physical dimensions, and the semantics reflections, performed at runtime, are linear mappings. The axiomatic of extended semantic algebra, which defines operations on the variable dimensions, is presented in Table 2.

Table 2. The operations on the program variables dimensions

Operator	Denotation	Correctness condition	Linear equations	Similarity criterion
Addition	R = L + P	[L] = [P]	$[R]^0[L]^1[P]^{-1} = 1$	0 1 -1
Subtraction	R = L – P	[L] = [P]	$[R]^0[L]^1[P]^{-1} = 1$	0 1 -1
Multiplication	R = L * P	[R] = [L][P]	$[R]^1[L]^{-1}[P]^{-1} = 1$	1 -1 -1
Division	R = L / P	$[R] = [L][P]^{-1}$	$[R]^1[L]^{-1}[P]^1 = 1$	1 -1 1
Exponentiation	$R = L^s$	$[R] = [L]^s$	$[R]^1[L]^{-s}[P]^0 = 1$	1 -s 0
Assignment	L = P	[L] = [P]	$[R]^0[L]^1[P]^{-1} = 1$	0 1 -1

where R – the operation result; L, R – left and right operands; [ ] – dimension.

For a correctly running program in the context of this operator, the following relations between the physical dimensions of the terms {p, a,b,c,d,e} should be fulfilled:

$$[p] = [a * b] = [a][b],$$

$$[d] = [e],$$

$$[p] = [c / (d - e)] = [c][d]^{-1} = [c][e]^{-1}, \quad (22)$$

Where [X] – is a physical object X dimension.

A computation model in memory can be represented using the context-free grammars. It allows describing the calculation process structure as a whole. Context-free grammar has the following form:

$$G = (\Sigma, N, R, S), \tag{23}$$

where

$\Sigma = \{identifier, constant, address \ldots register\}$ – a set of assembler terminal symbols/

$N = \{Addition, Subtraction, Multiplication, Division, Appropriate\}$ – a non-terminal character set;

$R = \{AddCommand, SubCommand\, MulCommand, \ldots, DivCommand\}$ – an output rule set;

$S \in \Sigma$ – a starting symbol.

The terminal symbols include arithmetic coprocessor command lexical tokens, including addition, subtraction, multiplication, division, assignment (data transfer) commands. A non-terminal symbol set is a set of lexical tokens, united by a generalizing feature, as well as their combinations, using products. An example of non-terminal symbols is given in the Table 3.

Table 3. Sets of non-terminal symbols

NON_TERMINAL SYMBOLS N	GENERALIZING FEATURE	TERMINAL SYMBOLS Σ
Addition	Addition commands	fiadd | fadd | faddp | …
Subtraction	Subtraction commands	fisub | fsub | fsubr | …
Multiplication	Multiplication commands	fimul | fmul | fmulp | …
Division	Division commands	fidiv | fdiv | fdivr | …
Appropriate	Data transfer commands	fist | fst | fstp | …

The output rule represented by expression (21) determines the use of the “fadd” command. Thus, we will present all possible inference rules in assembly language.

$$\begin{aligned} AddCommand \rightarrow\ & Addition\_Register, Address \\ & \mid Addition\_Register, Register \\ & \mid Addition\_Register, Register \Rightarrow faddp\, st(1), st \\ & \mid \ldots \end{aligned} \tag{24}$$

Where

Addition – a non-terminal set of coprocessor addition commands;

Register – a non-terminal set of coprocessor stack registers;

Address – a memory identifier set or actual memory addresses.

Each output in a context-free grammar, starting with a non-terminal symbol, is uniquely associated with a directed graph, which is a tree and is called an output (parse) tree. An output tree example related to the disassembled expression code, as well as its representation as to the similarity equations in terms of the dimension theory, is shown in figure 3.

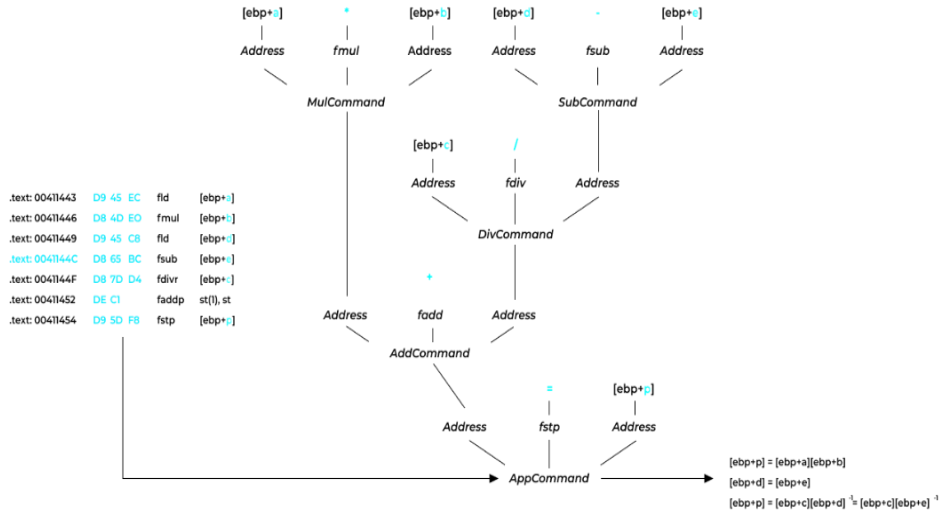

Fig. 3. Calculations representation by similarity equations

The solution to this equation system is a similarity coefficient matrix, constructed as follows:

$$
\begin{array}{lcl}
[ebp+p] & = & [ebp+p]1[ebp+a]\text{-}1[ebp+b]\text{-} \\
[ebp+a][ebp+b] & & 1[ebp+c]0[ebp+d]0[ebp+e]0=1 \\
[ebp+d] = [ebp+e] & \Rightarrow & [ebp+p]0[ebp+a]0[ebp+b]0[ebp+c]0[ebp+d]1 \\
 & & [ebp+e]\text{-}1=1 \\
[ebp+p] & = & \\
[ebp+c][ebp+d]\text{-} & & [ebp+p]0[ebp+a]0[ebp+b]0[ebp+c]\text{-} \\
1=[ebp+c][ebp+e]\text{-}1 & & 1[ebp+d]1[ebp+e]0=1
\end{array}
$$

By taking a logarithm we obtain a homogeneous linear equation system with a coefficients matrix:

$$A^1 = \begin{pmatrix} 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 & 0 \end{pmatrix} \quad (25)$$

In order to organize the similarity relations development, it is necessary to construct a translation grammar for assignment operators of the arithmetic type. The translational (attribute) grammar in addition to the syntax allows describing the action characters, which are implemented as functions, procedures, and algorithms. According to dimensions, these functions should implement algorithmic calculations and the similarity relation development, power monomials, equations and solutions.

Thus, the observation problem solution (control graph) and the computations representation (similarity equation) made it possible to form the image of a system for monitoring destructive software actions on the secured infrastructure, and restoring computation processes based on similarity invariants.

The plan of destructive software impacts control and the computational processes recovery includes preparatory and main stages (Figure 4). The preparatory stage includes the program passport formation in similarity invariants, the main ones are the stages of:

- Similarity invariants formation underexposure,
- Similarity invariants database formation at the checkpoints of the program control graph,
- Validation of the semantic correctness criteria of computational processes,
- Signal generation of the computation semantics violation,
- Partial calculations recovery according to the program passport.

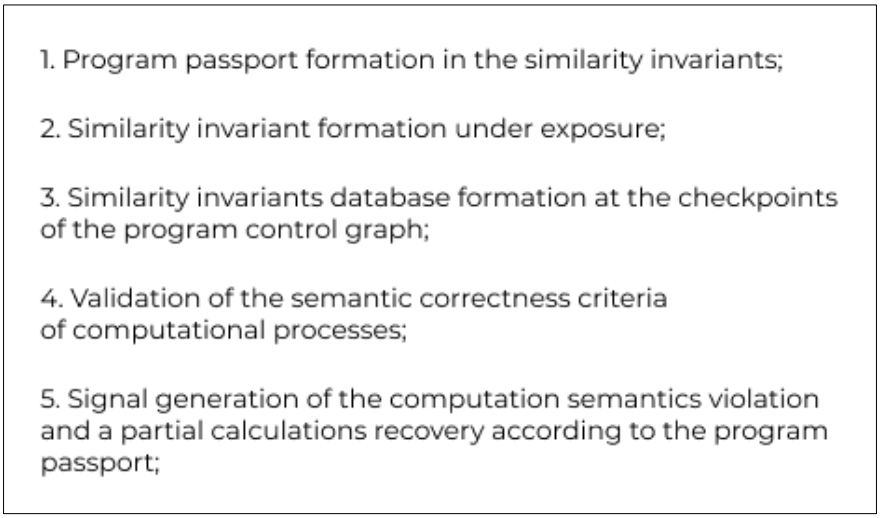

Fig. 4. Distortion control and computation process recovery scheme

A general representation of the information infrastructure that implements correct calculations under the hidden intruder program actions is reflected in Figure 5. We will reveal the stages of the destructive software impacts control and the computation processes recovery in more detail.

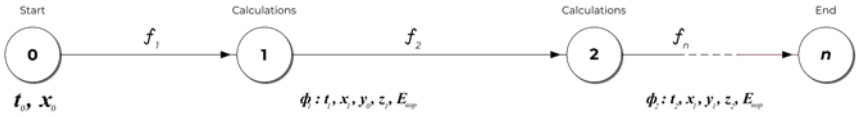

Fig. 5. The correct calculation scheme

Stage 1. The program passport formation in similarity invariants.

In order to implement a dynamic control, it is necessary to use the static verification results in the form of a program passport.

At the stage of a static verification using the disassembled correct calculation code, the program control graph is constructed.

At each checkpoint for each arithmetic operator, a production tree of an arithmetic expression is generated to develop a linear homogeneous equation system in the dimension terms. The result of solving the equation systems for each linear program part is a similarity invariant matrix. The semantic standard database is made up of reference matrices of similarity invariants for each checkpoint (Figure 6).

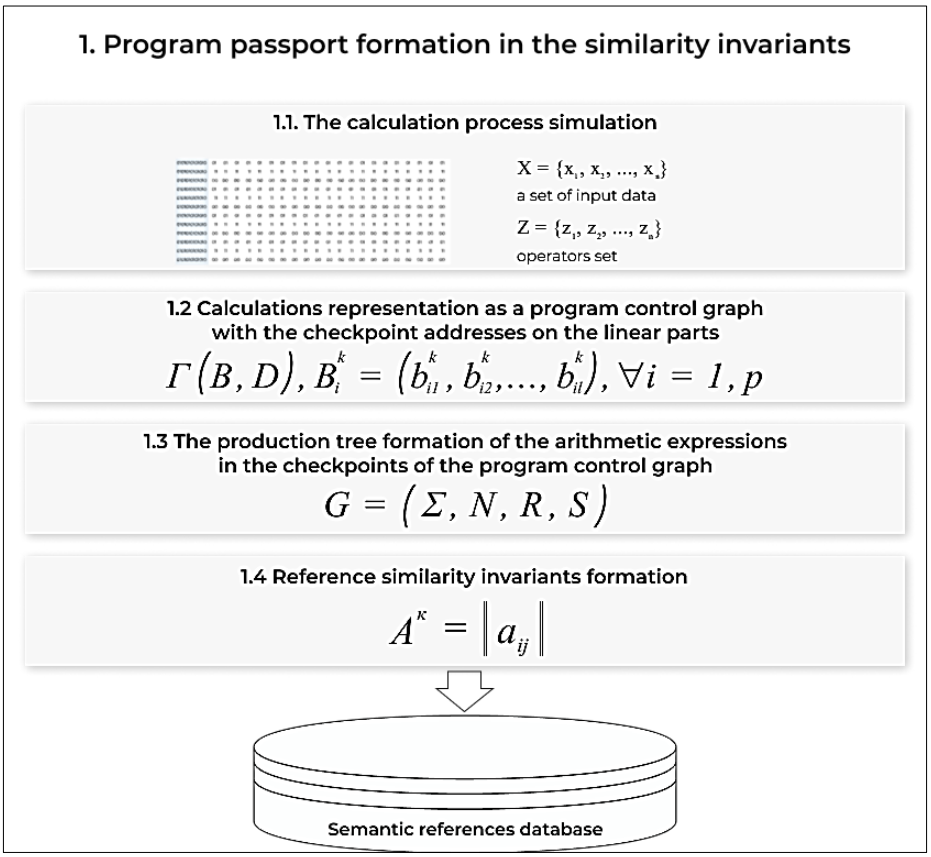

Fig. 6. The passport program formation scheme in the invariant similarity

Stage 2. The similarity invariants formation underexposure

The similarity invariants formation of the computational process, which is subjected to the hidden arithmetic operations impacts, runs according to the same algorithm as the computational process reference invariant formation.

For a given program, a set of checkpoints (CT) is formed, which are embedded in the studied program. The initial program model is the control graph of the computation process in terms of linear program sections. The similarity equations are analyzed and a coefficient matrix is developed in embedded CT for each linear program section, where the calculations take place (Figure 7).

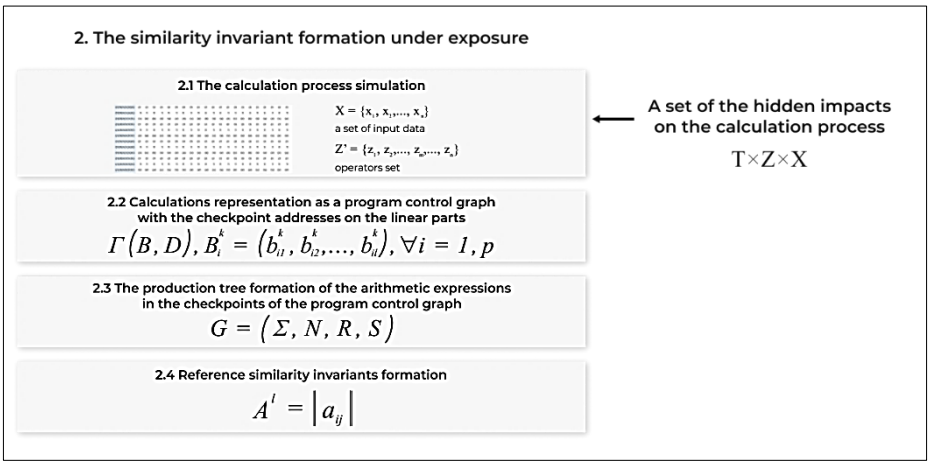

Fig. 7. The similarity invariants scheme underexposure

Incorrect calculations will differ in the state set of the computational process Z, i.e. in arithmetic operator sequence. The incorrect calculations scheme is presented in Figure 8.

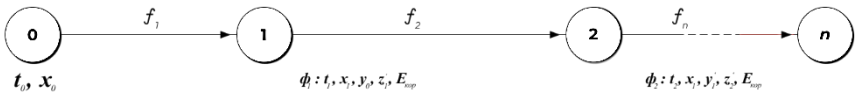

Fig. 8. The incorrect calculations scheme

Stage 3. The similarity invariants database formation at the checkpoints of the program control graph.

At this stage, the similarity invariant matrices constructed for each checkpoint form a similarity invariants database. The scheme of adding matrices to the database is presented in Figure 9.

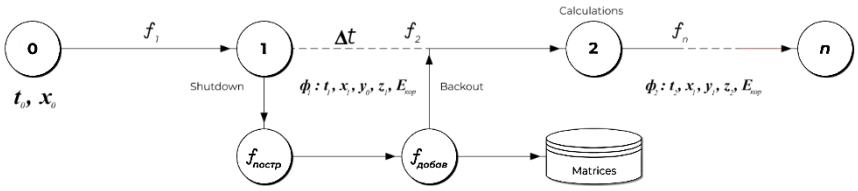

Fig. 9. The similarity invariants database formation scheme

Stage 4. The validation of the semantic correctness criteria of the computational processes.

In order to control the semantic correctness of the performed calculations, it is necessary to check the semantic correctness criterion by the formula (18) applying the reference and standard invariants matrix (Figure 9).

If the validation of this checkpoint has been completed, then proceed to check the criteria in the next CT until the program ends.

Stage 5. The signal generation of the computation semantics violation and the partial calculations recovery according to the program passport.

If the semantic correctness violation of the program execution is detected, that is, if for a given checkpoint $\lambda_{jn} - \lambda_{in} \neq 0$, then a signal is formed and an attempt is made to recover the calculations from the inverse transformation of the reference matrix invariants (Figure 10).

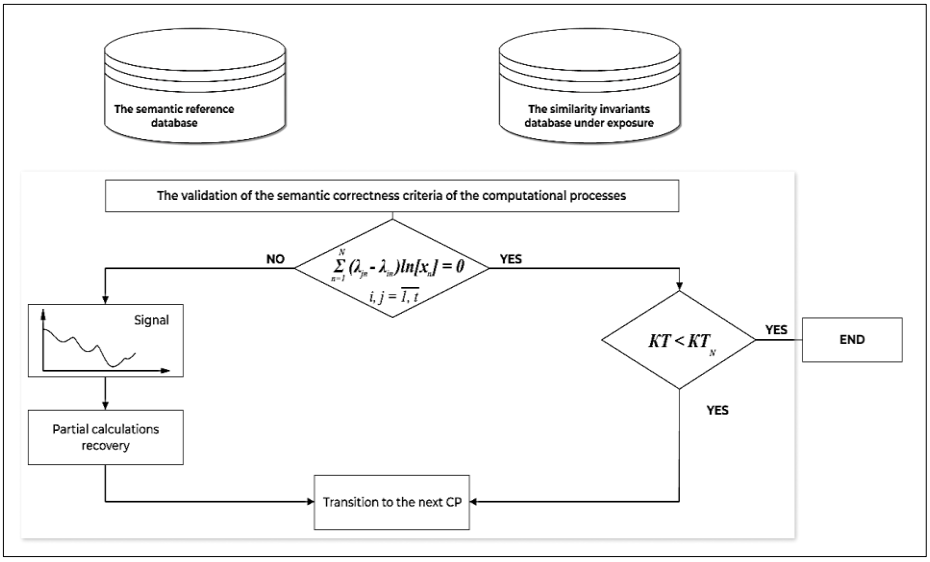

Fig. 10. The computational processes validation scheme

5 Conclusions

This approach allows to determine not only the fact of the calculation semantics violation but also to indicate the specific impact location on the program, using the mechanism for introducing checkpoints Thus, the dimensions and similarity theory application allowed synthesizing new informative features - the so-called similarity invariants for controlling the computational processes correctness. The similarity invariants use made it possible to bring the monitoring system of destructive program actions and the computation processes recovery closer to the controlled computational process semantics.

The obtained results allowed presenting a controlled computational process as a corresponding equations system of dimensions and similarity invariants, and its solution was to analyze the computations semantics under the destructive program impacts on the secured critical information infrastructure.

References

1. A. Fink, R. L. Griswold, and Z. W. Beech, "Quantifying cyber-resilience against resource-exhaustion attacks," in 7th International Symposium on Resilient Control Systems (ISRCS), Denver, CO, 2014.
2. Appliance of information and communication technologies for development. Resolution of the General Assembly of the UN. Document A / RES / 65/141 dated December 20.
3. Bakkensen, L. A., Fox-Lent, C., Read, L. K. and Linkov, I. (2017), “Validating Resilience and Vulnerability Indices in the Context of Natural Disasters”. Risk Analysis, 37: 982–1004. DOI:10.1111/risa.12677.
4. Barabanov A.V., Markov A.S., Tsirlov V.L. Methodological Framework for Analysis and Synthesis of a Set of Secure Software Development Controls, Journal of Theoretical and Applied Information Technology, 2016, vol. 88, No 1, pp. 77-88.
5. Biryukov, D. N., Lomako, A. G. Approach to Building a Cyber Threat Prevention System. Problems of Information Security. Computer systems, Publishing house of Polytechnic University, vol. 2, pp. 13–19, St. Petersburg, Russia, 2013.
6. Biryukov, D. N., Lomako, A. G., Sabirov, T. R. Multilevel Modeling of Pre-Emptive Behavior Scenarios. Problems of Information Security. Computer systems, Publishing house of Polytechnic University, vol. 4, pp. 41–50. St. Petersburg, Russia, 2014.
7. Bongard, M. M. The Problem of Recognition, Fizmatgiz, Moscow, Russia, 1967.
8. Borzykh S., Markov A., Tsirlov V., Barabanov A. Detecting Code Security Breaches by Means of Dataflow Analysis. In CEUR Workshop Proceedings, 2017, Vol-2081 (Selected Papers of the VIII All-Russian Scientific and Technical Conference on Secure Information Technologies, BIT 2017). P. 15-20.
9. Bostick, T. P., Connelly, E. B., Lambert, J. H., & Linkov, I. (2018). Resilience Science, Policy and Investment for Civil Infrastructure. Reliability Engineering & System Safety 175:19–23. DOI: 10.1016/j.ress.2018.02.025
10. Bostick, T. P., Holzer, T. H., & Sarkani, S. (2017). Enabling stakeholder involvement in coastal disaster resilience planning. Risk Analysis, 37(6), 1181–1200. DOI: 10.1111/risa.12737
11. D. N. Biryukov, Cognitive-functional memory specification for simulation of purposeful behavior of cyber systems. Proc. SPIIRAS. 3(40), pp. 55–76 Russia, 2015.

12. D. N. Biryukov, A. P. Glukhov, S. V. Pilkevich, T. R. Sabirov, Approach to the processing of knowledge in the memory of an intellectual system, Natural and technical sciences, No. 11, pp. 455–466, Russia, 2015.
13. E. D. Vugrin and J. Turgeon, "Advancing Cyber Resilience Analysis with Performance-Based Metrics from Infrastructure Assessment," in Cyber Behavior: Concepts, Methodologies, Tools, and Applications, Hershey, PA, IGI Global, 2014, pp. 2033-2055.