https://doi.org/10.52326/ic-ecco.2022/CS.10

Reduction programming in a technological programming environment

Igor Redko [1], ORCID: 0000-0002-3121-1412,
Petro Yahanov[1], ORCID: 0000-0001-7358-9846,
Maksym Zylevich[1], ORCID: 0000-0003-1646-0557
[1] Faculty of Electronics, dept. Chair of Electronic Computing Equipment in National Technical University of Ukraine «Igor Sikorsky Kiev Polytechnic Institute» Kyiv,Ukraine, zila@meta.ua

***Abstract*— This work is aimed at demonstrating, on a representative sample, the usefulness of programming concepts in the state of semantic patterns as relations in a program chain that specify particular types of programs. This is achieved via the use of program descriptors, which act as means of translating composites and basic functions of the technological programming system into their syntactic declarations at the last step of technological programming..**

***Keywords*— concept; concept; monad; composite; composition; programming environment; essential relation; programming system; reduction; descriptor.**

I. INTRODUCTION

According to the traditional individual-subjective paradigm, the understanding of programming comes from the fact that its consequence is dominant, which is most often interpreted as a text in one or another programming language. Programming itself was considered as a tool to achieve the goal. In such a paradigm, programming activity is maximally subjectivized and relies on the skills and abilities of the coder, who notates the software solution into code using programming languages. That is, the programming language acts only as a means of notation of the consequence of programming. Thus, there is no real support for the genesis of programs. Among all the known reasons for such a situation, in our opinion, the main one is an overly simplified understanding of programming that does not meet modern requirements. Therefore, productive modernization of the understanding of programming is a necessary condition for real support of programming. In [1] it is justified that taking into account the active role of the subject (ARS) in such modernization is essential. The following principle plays a key role in this: programming is an activity determined by a program and aimed at creating a program.

Although this understanding of programming differs from the traditional one in its focus on complementing the programming process and its result. It is still too amorphous and therefore needs further productive enrichment. Extending the understanding of the term "program" is key here. The proposed intersubjective paradigm (the name comes from the concept of intersubjectivity, introduced and developed in [2]) is based on the interpretation of the term "program" as a likeness (an outline, as a result of assimilation) of an essential feature. In [3-5] it is substantiated that such an interpretation firstly fully corresponds to the modern pragmatics of programming and secondly it allows further productive enrichment of programming. Directly programming is understood as a complement of two objectively irreducible to each other's modal and real (actual) types of abstractions - the essence - that which can be the subject of consideration, and the entity- the object of consideration, in the sense that the essence is an entity that is (available as a subject of consideration). In this way, we get a productive enrichment of the original premise: programming is an activity conditioned by program similarity (PS). Here, PS is a productive enrichment of essential simile (ES) determined by the program and aimed at creating a program [1, 6]. The content of the PS in the first approximation consists of the mutual complementation of the essence and the entity, oriented towards the creation of the program as a semblance of the essence. The latter, given the mentioned objective irreducibility of these types of abstraction, requires the involvement of the subject in this process, taking into account (objectification) his active role in it. From the above, it follows the importance and necessity of developing an intersubjective understanding of programming, as it is the key to the real technologization of programming.

The practice of programming testifies to the dominance of the "divide and rule" paradigm when

solving problems. The main technique here is the reduction of the holistic understanding which comes down to a general methodical technique - reduction of the complex to the simpler [6]. Therefore, the role of productive reduction mechanisms is essential for the technologization of programming. In [1, 6-8] it is substantiated that the basis of such enrichments is the concept-program active-passive complementarity. Thus, we come to the following explanatory enrichment of the program simile:

$$\begin{cases} \text{concept} = \text{essense that determines the entity} \\ \text{monad} = \text{entity that conditioned by the concept} \end{cases}$$

Works [8-11] show that the conceptual programming platform provides a real objectification of the main factor of productive technologization - the active role of the programming subject. The productive enrichment of software assimilation as a special type of active-passive cause-and-effect relationship and its relativization is carried out quite naturally - the main factor of productive technologization. Accordingly, the technological programming environment (TPE) should naturally be considered as a productive intersubjective enrichment of the mentioned conceptual programming platform of programming analogies to the active-passive complementarity of two objectively irreducible types of abstraction: the closed oracle logic - the integral core of the programming environment and the open diversity of its productive software analogies - technological programming systems (TPS). Any TPS is a consequence of software relativization and a carrier of productive understanding of reduction. This ensures that the active role of the programming subject is taken into account.

The necessity of technological activity, modernization of methods, and its implementation are directly determined by the level of need to objectify the subject's participation in it. In the field of programming, this is manifested in the growth of requirements for software products and the awareness that the main properties of the latter are formed at the stage of their genesis and as a result are determined by the active role of the subject in it [11].

The defining principles of understanding programming technology are formed based on the principles of conditioning, subordination, and separability [4, 5]. The properties and aspects of programs in their complementarity follow from the specified principles. This determines the point of view of what productive programming technology (PT) should look like so that its product meets these basic requirements. In particular, these principles at a general level clearly outline the place and role of productive programming in programming technology. To a first approximation, this can be expressed by the following diagram at Fig.1:

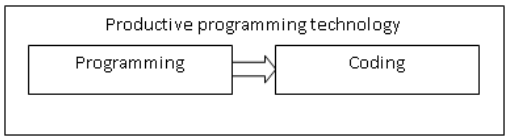

Figure 1. Block diagram of the productive programming technology.

The world practice of programming confirms the fact that despite the constantly growing number of problems and methods of their solution, all of them are subject to the "divide and conquer" paradigm, the main technique of which is reduction - reducing the complex to simpler [6]. Therefore, the role of reduction mechanisms is essential for the technologization of programming. The value of the above is determined by the fact that without an understanding of programming technology, programming technology is impossible.

In [6, 7], the solution to any programming problem is presented as a sequence of performing the stages of productive conceptualization, oracle schematization, composite-composite relativization, and reduction conceptualization. The meaningful essence of this sequence consists of the step-by-step productive enrichment of the solved problem within the framework of intersubjective TPE. The process starts from the subject's general ideas about the problem and ways of solving it and up to its final solution in the TPS- subject-oriented productive enrichment of the TPE. At the same time, the correctness of the solution follows directly from its construction. Many works are devoted to clarifying the content of TPE, the procedure for its creation, and individual steps (for example [1, 5, 6, 7] and their bibliography). Therefore (guided by the principle of reasonable sufficiency) let's allow ourselves a somewhat simplified, thesis to dwell on the construction of TPE and pay more attention to its use for solving problems.

Thus the subject of this work is the TPE mentioned above, its object is programming technology, and the goal is a technological programming system (TPS) based on TPE as a platform for productive programming and its application for problem-solving.

II. REDUCTION OF PROGRAMMING OF TASKS IN A TECHNOLOGICAL PROGRAMMING ENVIRONMENT

It follows from the above that TPS as a subject-oriented closure of TPE is a real subject-oriented programming platform. The closure is a definition of composites as programming concepts, basic object operations, and composite-composite interfaces. In this way, we will build an arithmetic TPS based on the results of compositional programming and studies of the class of computational arithmetic functions and predicates [12, 13]. As a programming platform, we will use composite programming and a nominal model of data, functions, and

operations, as composites - multiplication operations °, branching *IF*, cycling *WD* and the simplest compositions derived from them (in the sense of application operations *Ap* and *n*-ary superposition $S^n|_{n\in N}$), which specify the most used methods of synthesis of some programs from others [14-16], and as basic subject operations – arithmetic operations +,–, 0; logical operations $\vee,\wedge,!$, T, F; relation =,<,>. Parametric operations on nominal data will also be neede $A_{\downarrow}: A_{\downarrow}(a)\big|_{(a\in N)}=\{(A,a)\}$, $A^{\uparrow}: A^{\uparrow}(a)\big|_{(a\in N)}=\{(A,a)\}$, as naming and denaming, respectively and opening and closing parentheses.

In the following, data, functions and operations, unless otherwise specified, mean named data, named functions and named operations, respectively. As for the composite-composite interface, will use the apparatus of serial or, determined by the mentioned composites ° - branched or *IF* – and cycled *WD* reductions [2, 4, 5]. Recall that a tuple of functions $\langle f_1, f_2,\ldots, f_s\rangle$ is ° reduction of function f, if it is a solution of the equation $f = x_1^{\circ}x_2^{\circ}\ldots^{\circ}x_s$ namely $f \equiv f_1^{\circ} f_2^{\circ}\ldots^{\circ} f_s$. A couple of functions f_1, f_2 be *IF* reduction of function f if such a predicate as p exists , that this pair is a solution of the equation $f = IF(p, x_1, x_2)$, namely $f \equiv IF(p, f_1, f_2)$. Also, the function g is *WD* reduction of function f, if there exists a predicate p such that g is a solution of the equation $f = WD(x, p)$, namely $f \equiv WD(g, p)$ [6, 9]. A useful necessary condition for *WD*-reducibility directly follows from the latter.

Theorem. For the function g to be a WD-reduction of the function f, the following equality must hold $g^{\circ} f = f$.

After producing the TPS, will demonstrate the method of programming it using the example of programming the integer division function $div: N\times N \leftrightarrow N$, where $div(a,b)$, is a natural number that $b\times div(a,b)\le a\le b\times(div(a,b)+1)$. To solve this problem, will use the property of this function:

$$div(a,b)|_{a,b\in N\&b>0}=\begin{cases} div(a-b,b)+1, a\ge b,\\ div(a,b)=0, a<b \text{ or } a=0\end{cases}$$

Taking into account the orientation of the described TPS on the nominal data structure and based on the specified property, we can enrich the *div* function with its nominal specification $DIV:\{(A,a),(B,b)\}\to\{(A,\bar{a}),(B,b),(C,c)\}|_{a,\bar{a},b,c\in N}$.

From here it is easy to understand that $DIV \equiv F_1^{\circ}F_2$ where $F_1 = 0(C^{\uparrow})^{\circ}C_{\downarrow}\equiv\{(C,0)\}$,

$F_2:\{(A,a),(B,b),(C,c)\}\to$
$\to\{(A,a-k\times b),(B,b),(C,c+div(a,b))\}$ where
$a,b\in N\ \&\ b\ne 0,\quad k:(k+1)\times b<a<k\times b.$ This specification is an oracle scheme [6] due to the composite of multiplication.

The F_1 obviously, does not require further detailing and is a so-called "cell reset" $C-\{(C,c)\}\to\{(C,0)\}$. From F_2 it follows directly from the definition that its WD-reduction will be a function $G:\{(A,a),(B,b),(C,c)\}\to\{(A,a-b),(B,b),(C,c+1)$, where $P:\{(A,a),(B,b)\}\to\begin{cases}T, if\ a\ge b\\ F, if\ a<b\end{cases}$ - nominal specification of the corresponding predicate. That mean $F_2 = WD(G,P)$. This specification is also an oracle scheme. But due to the WD composite. Without going into insignificant details, F_2 can be represented somewhat simplified as follows:

$$F_2 = WD(\left(A_{\downarrow}\left(A^{\uparrow}-B^{\uparrow}\right)\right)^{\circ}\left(C_{\downarrow}\left(C^{\uparrow}+1\right)\right), P\left(A^{\uparrow},B^{\uparrow}\right)$$

respectively

$$DIV\equiv\left(0\left(C^{\uparrow}\right)^{\circ}C_{\downarrow}\right)^{\circ}$$
$$^{\circ}(WD\left(\left(A_{\downarrow}\left(A^{\uparrow}-B^{\uparrow}\right)\right)^{\circ}\left(C_{\downarrow}\left(C^{\uparrow}+1\right)\right), P\left(A^{\uparrow},B^{\uparrow}\right)\right).$$

The simplification is that the given expression is not a compositional term in its "pure form". Several meta-expressions are deliberately used along with the means inherent in the constructed TPS $0\left(C^{\uparrow}\right), A_{\downarrow}\left(A^{\uparrow}-B^{\uparrow}\right), C_{\downarrow}\left(C^{\uparrow}+1\right), P\left(A^{\uparrow},B^{\uparrow}\right)$.

The goal pursued by this is twofold. First, these expressions are mnemonically more familiar and at the same time, their representation in terms of TPS is not difficult to obtain. Secondly, their use makes it possible to demonstrate an essential feature of the proposed programming technology – its ability to take into account ARS. Strictly speaking, the activity of the programming subject is not limited to an exhaustive list of tools of any traditional programming system. On the contrary, the subject of programming actively influences the core of TPS, both in terms of the evolution of its concept and at the stage of encoding the solution. And the meta-expressions are examples of such influence. Below, what has been said will be reflected in the description of the corresponding definer.

As a result of the first stage of technological development, namely reduction programming, the above-described specification was obtained in the given system. Its correctness follows from the construction of the

program. After receiving the specification, coding can be done.

III. CODING AS A SEMANTIC-SYNTACTIC TRANSITION

Most programming languages are the only means of syntactic notation of programming results. The productive technology of programming is meaningfully an implementation of the complementarity of the above-mentioned basic principles of programming - genetics (conditionality), subordination and separability, and targeted creation of a software product [17-19]. It is a micro-conveyor of stages, where the "programming" stage realizes the subordination of semantics to pragmatics and its result is a program - a subject-driven outline of a problem solution in the form of a corresponding semantic (composite-compositional) term. [20, 21]. The stage of "encoding" refers directly to the semantic-syntactic transition from the semantic specification of the solution (program) to its code in the form of a corresponding syntactically correctly written text in a specified programming language. It has already been noted that the semantic-syntactic transition can be automated due to the derivation of the syntactic aspect of programs from the semantic aspect (principle of subordination). The core of this process is the corresponding definer of the programming language [22-26]. Let's apply this to the *DIV* function programming example discussed above.

As an example, consider the part of the definer of the system given above. The definer data is sufficient to demonstrate the creation of the program. It presents the corresponding composites and functions with their syntax notations in a Pascal-like manner (tables 1 and 2).

TABLE I. PROGRAMMING AND CODING PATTERNS

The concept (patterns) of programming	***The concept (patterns) of coding***
…	**…**
F	F
(F)	F
$F_1 \circ F_2$	$begin\, F_1; F_2\, end$
$IF(F_1, F_2, F_3)$	$if\ F_1\, then\, F_2\, else\, F_3\ \ F_1\, F_2$
$F \circ X_{\downarrow}$	$X^{\uparrow} := F$
$X^{\uparrow} \circ S \quad X^{\uparrow} \circ S \circ Y_{\downarrow}$	$X^{\uparrow} + 1 \quad Y := X + 1$
$WD(F_1, F_2)$	$while\, F_2\, do\, F_1\, end$
$P(A^{\uparrow}, B^{\uparrow})$	$(A > B)\, or\, (A = B)$
$F_1[F_2]$	$F_1\ ; F_2$
$\text{meta}\begin{bmatrix} 0(C^{\uparrow}) \\ A_{\downarrow}(A^{\uparrow} - B^{\uparrow}) \\ C_{\downarrow}(C^{\uparrow} + 1) \end{bmatrix}$	$\begin{bmatrix} 0(C) \\ A := A - B \\ C := C + 1 \end{bmatrix}$
`…	**…**

TABLE II. BASIC FUNCTIONS AND THEIR CODES

Basic functions	***Basic function codes***
…	**…**
0	0
+	+
–	–
∧	*and*
∨	*or*
!	*not*
<	<
=	=
>	>
$X^{\uparrow}$	X
$X_{\downarrow}$	X
`…	**…**

In the presented tables, the notation *F*, possibly with indices, F_i, i=1,2,3,... and only these are used as non-terminal symbols or non-terminals. Similarly, terminal characters $X^{\uparrow}, X_{\downarrow}, X$ can also be used with subscripts: $X_i^{\uparrow}, X_{i\downarrow}, X_i, i = 1, 2, 3...$

Through them, the recursiveness of constructions is ensured [22-26]. Concepts of programming and coding presented in Table 1 represent correctly written words in the combined alphabet of terminal symbols and non-terminal symbols. Table 2 lists terminal symbols for basic operations and their corresponding Pascal-like codes.

Let's turn to the above program. The previously used additional markup of the program demonstrates its inherent hierarchical structure. It is due to the step-by-step implementation of oracle updates in the programming system, starting from the *DIV* oracle and ending with the oracle-free one, that is, the compositional term locked in the programming system. Moving along this hierarchy, following the definer fragment specified in Tables 1 and 2, we recursively build a Pascal-like program code (Table 3).

https://doi.org/10.52326/ic-ecco.2022/CS.10

TABLE III. EXAMPLES OF PROGRAMS

Program	***Templates used***	***Updates of non-terminals***
$DIV \equiv \left(0\left(C^{\uparrow}\right)^{\circ}C_{\downarrow}\right)^{\circ}$ $^{\circ}(WD\left(\left(A_{\downarrow}\left(A^{\uparrow}-B^{\uparrow}\right)\right)^{\circ}\left(C_{\downarrow}\left(C^{\uparrow}+1\right)\right),P\left(A^{\uparrow},B^{\uparrow}\right)\right)$	$F_1{}^{\circ}F_2$	$F_1 \Leftarrow 0\left(C^{\uparrow}\right)^{\circ}C_{\downarrow}$ $F_2 \Leftarrow WD(\left(A_{\downarrow}\left(A^{\uparrow}-B^{\uparrow}\right)\right)^{\circ}\left(C_{\downarrow}\left(C^{\uparrow}+1\right)\right),P\left(A^{\uparrow},B^{\uparrow}\right)$ $DIV \Leftarrow begin\, F_1; F_2\, end$
$F_1 \equiv 0\left(C^{\uparrow}\right)^{\circ}C_{\downarrow}$	$F_{11}{}^{\circ}F_{12}$ $X_{\downarrow}$ $X^{\uparrow}$	$F_{11} \Leftarrow 0(C)$ $F_{12} \Leftarrow C_{\downarrow}$ $F_1 \Leftarrow begin\, F_{11}; F_{12} end$
$F_2 = WD(\left(A_{\downarrow}\left(A^{\uparrow}-B^{\uparrow}\right)\right)^{\circ}\left(C_{\downarrow}\left(C^{\uparrow}+1\right)\right),P\left(A^{\uparrow},B^{\uparrow}\right)$	$WD(F_1,F_2)$	$F_{21} \Leftarrow \left(A_{\downarrow}\left(A^{\uparrow}-B^{\uparrow}\right)\right)^{\circ}\left(C_{\downarrow}\left(C^{\uparrow}+1\right)\right)$ $F_{22} \Leftarrow P\left(A^{\uparrow},B^{\uparrow}\right)$ $F_2 \Leftarrow while\, F_{22}\, do\, F_{21} end$
$F_{21} \Leftarrow \left(A_{\downarrow}\left(A^{\uparrow}-B^{\uparrow}\right)\right)^{\circ}\left(C_{\downarrow}\left(C^{\uparrow}+1\right)\right)$	$F_{11}{}^{\circ}F_{12}$ (F) $X_{\downarrow}$ $X^{\uparrow}$	$F_{31} \Leftarrow A_{\downarrow}\left(A^{\uparrow}-B^{\uparrow}\right)$ $F_{32} \Leftarrow C_{\downarrow}\left(C^{\uparrow}+1\right)$ $F_{21} \Leftarrow begin\, F_{31}; F_{32} end$
$F_{22} \Leftarrow P\left(A^{\uparrow},B^{\uparrow}\right)$	meta $P(F_1,F_2)$	$F_{22} \Leftarrow (A>B or A=B)$
$F_{31} \Leftarrow A_{\downarrow}\left(A^{\uparrow}-B^{\uparrow}\right)$	meta $A_{\downarrow}(A^{\uparrow}-B^{\uparrow})$ $F^{\circ}X_{\downarrow}$ (F)	*begin* $F_{31} \Leftarrow A := A-B$ *end*
$F_{32} \Leftarrow C_{\downarrow}\left(C^{\uparrow}+1\right)$	meta $C_{\downarrow}(C^{\uparrow}+1))$ $F^{\circ}X_{\downarrow}$ (F)	*begin* $F_{32} \Leftarrow C := C+1$ *end*
$DIV \equiv F_1{}^{\circ}(WD\left(F_{21}{}^{\circ}F_{22}\right)$	(F) $F_{11}{}^{\circ}F_{12}$ $F^{\circ}X_{\downarrow}$ $WD(F_1,F_2)$ meta $\begin{bmatrix} 0\left(C^{\uparrow}\right) \\ A_{\downarrow}(A^{\uparrow}-B^{\uparrow}) \\ C_{\downarrow}(C^{\uparrow}+1) \end{bmatrix}$	*begin* $while(A>B or A=B) do$ *begin* $F_{31} \Leftarrow A := A-B$ $F_{32} \Leftarrow C := C+1$ *end* $DIV \Leftarrow F_{32}$ *end*

IV. CONCLUSIONS

The fundamental role of productive reduction in the technologization of programming is shown.

It is substantiated that the new paradigm of programming should be based on the activation of the role of the programming subject, in which programming is considered as an activity determined by the program.

It is confirmed that programming technology uses reduction methods as a means of transforming an information resource into a software product in intersubjective TPE.

The reduction determined by the concept plays a fundamental role in the technologization of programming. The concept of the software product determines its semantics, and the syntactic notation of the programming results determined by the program is completed by one of the programming languages chosen by the programming subject.

With the help of reductive programming, a program specification was obtained in the given system, the

correctness of which follows from its construction. Based on the received specification, the program code is obtained with the help of definers.

A representative example demonstrates the use of programming concepts in the form of semantic templates as links in a program chain that determine certain classes of programs. A program definer is used, which acts as a means of translating composites and basic functions of TPS into their syntactic representation.

The use of meta-expressions in program construction substantiates the objectivism of the active role of the subject and determines the place and significance of this activity in obtaining the result. Metaexpressions do not belong to the toolkit determined by the intersubjective programming environment but are the product of the TPS programmer's conceptualization of the means of achieving the final goal within the limits of personal competence..

REFERENCES

[1] I. Redko, P. Yahanov, “Conceptual model of the technological environment of programming”, KPI Science News, vol.1, no.1, pp. 18-26, 2020. DOI: 10.20535/kpi-sn.2020.1.197953.

[2] E. G. Husserl, “Logical Studies. Cartesian Reflections”, Minsk, Belarusia, 2000.

[3] I. Redko, “Pragmatic foundations of descriptive environments”, Programming issues, no 3, pp 2-25, (in Russian), 2005.

[4] I. Basarab, N. Nykytchenko, V. Redko. “Composite databases”. Kyiv: Lybid , 1992. p. 192.

[5] D. I. Redko, I. V. Redko, P. O. Yahanov, T. L. Zakharchenko. “Compositional basis in programmer activity”, System research and information technologies, vol. 4, pp. 83-96, 2016.

[6] I. Redko, P. Yahanov and M. Zylevich, "Reduction conceptualization of oracle schemes,", System research and information technologies, vol.1, no1, pp.21-33, 2021. doi: 10.20535/SRIT.2308-8893.2021.1.02.

[7] I. Redko, P. Yahanov and M. Zylevich, "Reduction conceptualization of oracle schemes", 2020 IEEE 2nd International Conference on System Analysis & Intelligent Computing (SAIC), Kyiv, Ukraine, pp. 125-130, 2020. DOI: 10.1109/SAIC51296.2020.9239204.

[8] U. Kovaliv, Literary encyclopedia vol 2, Kyiv, Ukraine: Acadenia (in Ukrainian), 2007.

[9] I. Redko, “Descriptive foundations of programming”, Kibernetika i sistemnyj analiz, no.1, pp3-19, (in Russian), 2002.

[10] I. Redko, “Foundations of descriptive science”, Kibernetika i sistemnyj analiz, no.5, pp16-36, (in Russian), 2003.

[11] V. N. Redko, N. V. Grishko, I. V. Redko, “Descriptive systems: conceptual basis”, Problem of Programming, vol. 2-3, pp. 75-80,2006.

[12] V. Redko, “Semantic structures of programs”, Programming, vol. 3, pp 3-13, 1979.

[13] D. Bui, V. Redko, “Primitive program algebras”, Programming, vol. 5, pp 1-7, 1984.

[14] V. Redko, “Program compositions and compositional programming”, Programming, pp. 3-24, (in Russian), 1978.

[15] V. Redko, “Definitors and the method of definitor processing”, Kibernetika i sistemnyj analiz, no.6, pp52-56, (in Russian), 1974.

[16] I. Basarab, N. Nykytchenko, V. Redko. “Composite databases,” Kyiv, Ukraine: Lybid (in Russian), 1992.

[17] V. N. Redko, “Foundations of programmology,” Kibernetika i sistemnyj analiz, vol. 1, pp. 35-57, (in Russian), 2000.

[18] V. N. Redko, “Basics of compositional programming,” Programming, vol 3, pp. 3-13, (in Russian), 1979.

[19] F. Brooks, "The Mythical Man-Month: After 20 years," in IEEE Software, vol. 12, no. 5, pp. 57-60, 1995, doi: 10.1109/MS.1995.10042.

[20] V. N. Redko, N. V. Grishko, I. V. Redko, “Explicit programming in the environment of logical-mathematical specifications”, UkrPROG’98,pp71-76, (in Russian), 1998.

[21] V. N. Redko, N. V. Grishko, I. V. Redko, “Explicit programming in an integration environment”, Problem of Programming, vol. 2, pp. 59-65, (in Russian), 2004.

[22] D. I. Redko, I. V. Redko, P. O. Yahanov, T. L. Zakharchenko. “Compositional basis in programmer activity System research and information technologies, vol. 4, pp. 83-96, 2015.

[23] I. Redko, “Pragmatic foundations of descriptive environments”, Programming issues, no 3, pp 2-25, (in Russian), 2005.

[24] G. Leibniz, Compositions, Moscow, Russia: Mysl, (in Russian), 1982.

[25] V. Redko, “Interpreted languages and interpreters”, no.5, pp15-21, (in Russian), 1969.

[26] V. Redko, G. Trubchaninov, “Syntactic definitions: structural approach”, Programming, no.5, pp. 9-20, (in Russian), 1977.