EAI Endorsed Transactions
on e-Learning

Research Article **ICST.ORG**

Automation of Algorithmic Tasks for Virtual Laboratories Based on Automata Theory

Evgeniy A. Efimchik[1,*], Mikhail S. Chezhin[1], Andrey V. Lyamin[1]

[1]ITMO University, Saint Petersburg, Russia

Abstract

In the work a description of an automata model of standard algorithm for constructing a correct solution of algorithmic tests is given. The described model allows a formal determination of the variant complexity of algorithmic test and serves as a basis for determining the complexity functions, including the collision concept – the situation of uncertainty, when a choice must be made upon fulfilling the task between the alternatives with various priorities. The influence of collisions on the automata model and its inner structure is described. The model and complexity functions are applied for virtual laboratories upon designing the algorithms of constructing variant with a predetermined complexity in real time and algorithms of the estimation procedures of students' solution with respect to collisions. The results of the work are applied to the development of virtual laboratories, which are used in the practical part of massive online course on graph theory.

Keywords: Assessment, Virtual Learning Environments, Massive Open Online Course.

Received on 10 November, 2015, accepted on 13 January, 2016, published on 10 March, 2016

doi: 10.4108/eai.10-3-2016.151123

1. Introduction

The development of information technologies in education resulted in a wide distribution of electronic teaching instruments, virtual laboratories (VL) being one of them. VL are electron media making possible the creation and study of visual models of the real phenomena. This is rather an extended determination taking into account the fact that both real laboratory installations (the laboratory is called distant in this case) and various mathematic and imitation models may form the basis of the models. A large number of information systems differing by aims, application methods, and program structure comply with it. A special feature of all VL may be their orientation to the formation and checking of practical skills and experiences. To determine the application field of the models and methods given in this work we shall give a short classification of VL.

One of the features of VL may be the type of tests embedded in them: there are both algorithmic tests requiring fulfillment of a rigid sequence of actions and logical methods of solving and tests associated with creative activity requiring accomplishment of intuitive jumps, objects recognition, and use of heuristic solving methods [1].

Another feature may be the program architecture of VL. Autonomic VL are a united supplement [2,3], and the functions of distributed VL are divided between several individual modules interacting between each other with the help of special Remote Laboratory Control Protocol (RLCP) [4] or other network technologies [5].

An important marker is the presence and method of automatic check of student's solution, since this property directly influences applicability of VL during independent

[*]Corresponding author. Email: efimchick@cde.ifmo.ru

work with electronic practical courses of electronic information and education media. The automatic check is especially important for massive open online courses. Automatic estimation is often carried out by a method of testing the black box: student's solution is represented as a system, which can be acted upon and its reaction can be compared with what was expected [1,6,7]. One more example of commonly used practice in the systems of massive online courses may be the instrument of peer assessment: after completing his own test the student must check several solutions of other course participants selected randomly [8]. The method of checking depends on the character of the test: the black box test is convenient and therefore widely used method of automatic check of the tests concerning description of a designated algorithm in some program or modelling language and a peer assessment is used when student must present a work badly amenable to automatic analysis – an essay, figure, or abstract.

One more property of VL is its wide application. Multi-purpose VL [1,3], which can be used to carry out studies on various topics, are usually substantially more scaled and complex than specialized VL developed for solving problems concerning one topic [2].

In [4] a model of a distributed VL of a standardized structure with automatic checking students' solutions, which represents VL as a connected up modulus of electronic information-education system, is given. This model allows creation of unified medium of fulfillment for multiple VL on the basis of AcademicNT system. Such a method appeared to be appropriate for control over the software of small specialized VL intended mainly for working with algorithmic tests. We emphasize that student is not required to describe the very algorithm, but he must reproduce its actions correctly for a given variant. Preparation of the test variants in the automatic way appeared to be an important problem – it was decided to reject the traditional bank of variants owing to its inherent drawbacks. In this case, variants of tests must have a predetermined complexity to ensure equal conditions for all students in the estimation of achieved education results. The complexity of the test variant in this context is a quantitative characteristic reflecting the number of operations needed to be fulfilled for obtaining a correct solution. It is necessary to distinguish the complexity of the test variant from difficulty of the test. The difficulty is associated with mastering the algorithm of solving the test and is expressed by a percent from the number of students being tested from a representative selection, who fulfilled this test correctly. Nevertheless, under condition of limited time the complexity of the test influences its difficulty. Even knowing the algorithm of solving the test you can fail to meet the schedule of its fulfillment, if a given variant has an excessive complexity.

2. Automata Model of Reference Algorithm

In this work a method is proposed for formal determination of variants complexity of for algorithmic tests based on automation model of *reference algorithm*. Let us assume that there is a certain algorithmic test t, which must be solved with the help of reference algorithm a, and there is a great number of variants V, for it:

$$V = \{v_1, v_2, v_3, \dots, v_n\}. \qquad (1)$$

Each element v_i is a particular variant of the test with specific data.

As an example let us concern ourselves with an algorithm for Turing machine, which increases an integer by a unit. The starting number is on the tape, written in binary digits from left to right, in other cells this is an empty symbol, and the head points to the eldest order of the number. Then with the aim of increasing the number by a unit we must fulfill the following sequence of actions:

- Move to the right till you meet the empty symbol;
- Shift to the left;
- If symbol in the current cell equals '1', change it for '0' and move to the left;
- If the value of the current cell equals '0' or an empty symbol, write down '1' into the cell and complete the work.

In this case the test t requires the actions of the reference algorithm a to be reproduced, and starting state of Turing tape is a variant of test v_i. Here we should emphasize once more that student must not write the program for Turing machine but must reproduce the above described algorithm correctly. He gains access to Turing tape, the possibility to accomplish the requests for reading a symbol from a current cell and the commands for shifting the head and writing the symbol into a cell.

To develop VL with automated processes of constructing the test variants and estimation of the student solutions for a model test t with the help of algorithm a we suggest to advance a special automation model M. As such model we propose to use a combination of determined final automaton with an output (controlling automaton) and a data depositary, which it interacts (the control object) with. This model is the development of a model of automated object (AO).

The AO model consists of three main components: controlling automaton, object of control and external medium. At every step of the work the controlling automaton forms a new state of the control object (calculating state) on the basis of external medium action, of current state of the control object and of the state of controlling automaton (controlling state). Applying this model to algorithmic tests, we find that at every step the controlling automaton forms a record of correct solution s, based on the data of test variant v and intermediate results of previous steps fixed in the record of the solution. The test variant is the object of external medium, and the state of the control object must be considered as the record of the test solution including intermediate results. This means that after completing the work of controlling automaton the control object must contain all information about the transfers carried out by controlling automaton and about the sequence of control states attended by it. Then with the help of the advanced model M we can obtain a correct solution for each variant of the test v. In other words, there is a reflection ρ_M of a great number of the test variants V to a multitude of solutions S ($\rho_M : V \to S$).

Figure 1 represents the AO for the algorithm of increasing an integer by a unit described above. The solution being formed is contained in the control object C. The variant of the test V contains starting state of Turing tape, which can be obtained with the help of the getTape inquiry.

A special attention must be paid to the structure of the control object, since it must be designed in such a manner that in the resulting calculating state all the intermediate results were given. Hence, the object contains Turing tape, which starting state can be found with the help of the setTape command. The control over the tape is performed by the head shift commands (left and right), the record of symbols (write) and an inquiry for reading the symbol in the current cell (read). Moreover, in order to save the intermediate results a journal of accomplished commands is added to the control object. Each time a shift of the head of the symbol record is performed, a corresponding record is added to the journal. Such a journal may be presented as one more Turing tape, let us call it the tape of command journal, with the aim of distinguishing it from the data tape. As the automaton fulfills command to shift the head of the data tape or to record a symbol, a symbol is written into the current cell of the command journal, which designate this command, then the head of the tape of the command journal is shifted to the right. The structure of the control object is given in Fig. 2. Table 1 contains description of commands and inquiries of both test variant and solution being formed in control object.

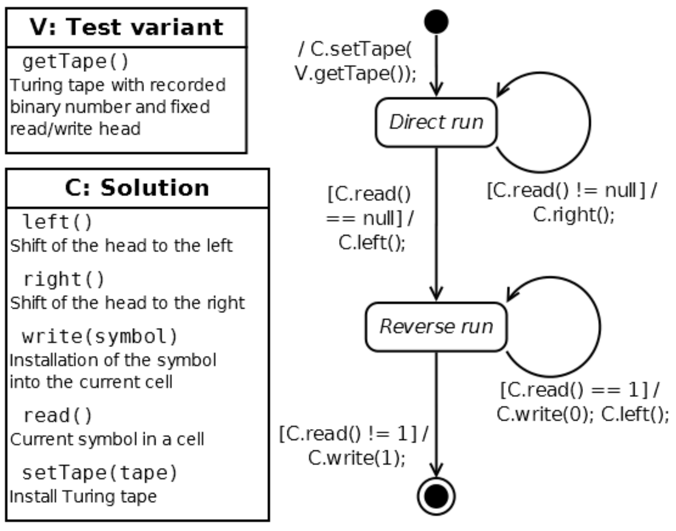
V: Test variant
getTape()
Turing tape with recorded binary number and fixed read/write head
C: Solution
left()
Shift of the head to the left
right()
Shift of the head to the right
write(symbol)
Installation of the symbol into the current cell
read()
Current symbol in a cell
setTape(tape)
Install Turing tape
/ C.setTape(V.getTape());
Direct run
[C.read() == null] / C.left();
[C.read() != null] / C.right();
Reverse run
[C.read() == 1] / C.write(0); C.left();
[C.read() != 1] / C.write(1);

Figure 1. Automation model of the algorithm of increasing an integer by a unit for Turing machine

Table 1. Commands of the test variant and the solution being formed

Object	Command	Description
V: Test variant	getTape()	Turing tape with recorded binary number and fixed head
C: Solution	left()	Shift of the head to the left
	right()	Shift of the head to the right
	write (symbol)	Installation of the symbol into the current cell
	read()	Current symbol in a cell
	setTape (tape)	To install Turing tape

The model is loaded into the controlling automaton, the operation of copying the state of Turing tape from the test variant into the solution being formed is included into the initiating stage in this case.

Assuming that student must adhere to the reference algorithm a and its representation with the control object initiated according to data of the variant and also an interface for interacting with it. We can reason that in the case, when the student reproduced the actions of the algorithm correctly, his solution as a resulting calculating state of the given control object must coincide with the correct solution. Thus, after adding a unit to “11011” the data tape must contain “11100” and the tape of the commands journal – the sequence “right, right, right, right, right, left, write ‘0’, left, write ‘0’, left, write ‘1’”.

The aforesaid interface must be designed in such a manner that student could interact with the control object with the help of the same commands as the controlling automaton. The student has an access to the same commands, which are used in the automation model in Fig. 1 – the head shift and symbol recording. Moreover, the student only sees the content of the current cell, as well as the controlling automaton, hence the variant solution is not evident for him, and he must follow the given algorithm.

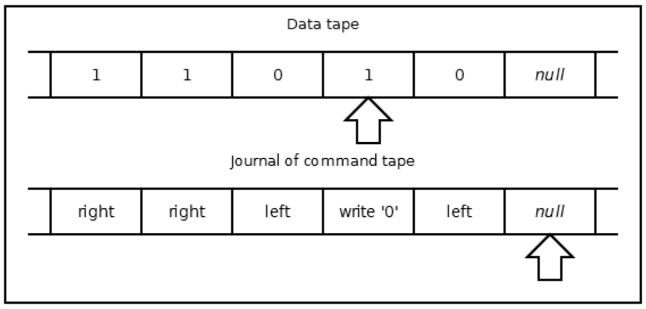

Figure 2. Structure of the control object of automation model of the algorithm of increasing an integer by a unit for Turing machine

3. Consideration of Collisions

The special feature of automation models under consideration is the following: external actions on AO are known to be determined first of all by the data of the test variant. Moreover, some algorithms as known allow a selection from several equivalent alternatives (collision) during fulfillment, and then great number of correct solutions may exist for one variant of the test. In this case student's solution may be correct, but not coinciding with the solution constructed with the help of automation model. To avoid such situation it is necessary that the automation model takes into account student's choice made in the situation of collision. A correct solution is constructed with the help of the controlling automaton as long as situation of collision emerges. Then the information about the made choice is selected from student's solution and is added to input action on the automaton at the next step of the work. Then the solution record is constructed by usual manner up to the emergence of the next collision. Therefore, we can state that each component of the input action x_E on the controlling automaton really consists of the component x_{EV} resulting from the test variant and the component x_{EU} resulting from the student's choice fixed in the solution suggested by him: $X_E \subseteq X_{EV} \times X_{EU}$. Therefore, in general case, the reflection ρ_M of great number of test variants to a multitude of solutions given as M must take into account student's choice in the situations of collision and it is determined as function $\rho_M : V \times U \rightarrow S$, where U is the multitude of the choices made by student in the situations of collision.

In the example considered above the collisions are not encountered, however they can appear, if the test was changed a little and, for example, let the student set the head into any number order besides the eldest from the beginning. This means that the model of the control object given in Fig. 2 must be updated by adding one more element – a number variables, which the information about the very number order the student set the head must be written into. In this case a modification of the automation model of the algorithm given in Fig. 1 must be made: one more element appears in the model – student's solution *U*. The model *U* and the model of the solution *C* coincide, however they have various sets of commands. It is possible to find out which number order the student set the head into with the help of getInitPosition() inquiry of the student solution *U*. A command for setting the head getInitPosition() is added to the solution *C*, it is excited at the stage of initiation. The modified automation model is given in Fig. 3.

4. Formal Determining of Complexity of Algorithmic Tests

The complexity of the test variant c_v can be determined as the number of transfers completed by the controlling automaton if the complexity of completing the transfers is the same.

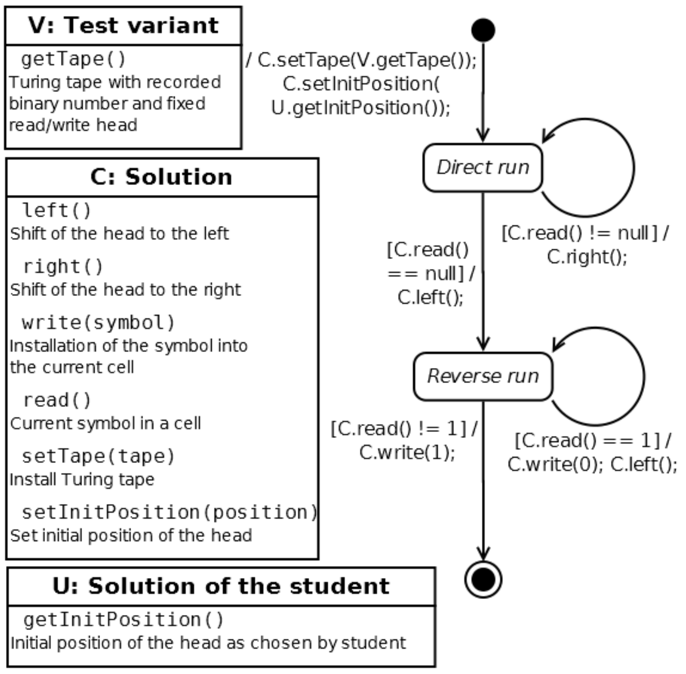
V: Test variant
getTape()
Turing tape with recorded binary number and fixed read/write head
C: Solution
left()
Shift of the head to the left
right()
Shift of the head to the right
write(symbol)
Installation of the symbol into the current cell
read()
Current symbol in a cell
setTape(tape)
Install Turing tape
setInitPosition(position)
Set initial position of the head
U: Solution of the student
getInitPosition()
Initial position of the head as chosen by student
/ C.setTape(V.getTape()); C.setInitPosition(U.getInitPosition());
Direct run
[C.read() != null] / C.right();
[C.read() == null] / C.left();
Reverse run
[C.read() != 1] / C.write(1);
[C.read() == 1] / C.write(0); C.left();

Figure 3. Automation model of the algorithm of increasing an integer by a unit for Turing machine taking account of collisions

This value can be obtained as the number of terms *q* of the sequence Y_v of controlling states visited by controlling automaton in the process of constructing correct solution for the test variant *v*:

$$Y_v = ({}_{r_i})_{i=1}^{q},$$
$$y_{r_i} = \delta\left(y_{r_{(i-1)}}, x_i\right), \qquad (2)$$
$$c_v = q,$$

where $\delta : X \times Y \rightarrow Y$ is the transfers function of controlling automaton of model *M;* x_i is an input action on the controlling automaton formed under the influence of variant *v* in the *i* – cycle; r_i is the index of visited controlling state.

In the case when the complexity of completing the transfers cannot be considered the same, it is necessary to determine function *f* of complexity of completing the transfer c_i to the state y_{r_i} from the state $y_{r_{i-1}}$ under the action x_i .

Then the resulting complexity of the test variant will be equal to the sum of complexities of transfers made by the controlling automaton:

$$Y_v = ({}_{r_i})_{i=1}^{q},$$
$$y_{r_i} = \delta\left(y_{r_{i-1}}, x_i\right),$$
$$c_i = f\left(y_{r_{-1}}, y_{r_i}, x_i\right), \qquad (3)$$
$$c_v = \sum_{i=1}^{q} c .$$

Using such a procedure, we can determine complexity of any existing variant of the test, but this cannot solve the problem of automatic construction of the test variants of a designated complexity. It is necessary to examine the automation model of algorithm for each test t in order to answer the question, which properties of the test variants do influence the complexity of the solution and how they influence. The result of such examination must be the function of the kind $c = c(v)$, making possible the calculation of the complexity as a function of properties of the test variant. Let us call it *complexity function*.

In the example described above (without admission of the collisions) the number of transfers completed by the controlling automaton depends on two factors: l – the total number of the number orders and z – the ordinal number of the last number order equal to '0'. Automation model at the stage of direct run passes all the number orders and at the stage of reverse run completes one more transfer in order to return to the last order and passes to the first order not equal to '1', hence the resulting complexity is: $c = l + 1 + (l - z) = 2l + 1 - z$. For example, if a number «1000000000» is written in Turing tape, the complexity of adding a unit to it will be $c = 2 \times 10 + 1 - 10 = 11$. Really, the stage of reverse run includes only one transfer. For the variant with number «101111» the complexity is also equal to 11: $c = 2 \times 6 + 1 - 2 = 11$, since in the stage of reverse run there will be five transfers.

5. Automatic Variant Constructing and Estimation of the Student Solution

Having the complexity function, we can compose the algorithm of constructing the variants of algorithmic test with a designated complexity. Thus, if it is necessary to create the variant of the test for reproducing the above described algorithm of the increment of binary number with a designated complexity C, we can use the algorithm given in Fig. 4. With the help of the pseudorandom numbers generator (GPRN) we select the length of number L. It is evident that it must be greater than half of the complexity and less than the complete complexity minus 1. Function randInt is used for this purpose and returns one of the integers being in the range from the first argument to the second argument (exclusively). The choice of a number is based on GPRN, the sequence generated by it obeys the even distribution. With the help of complexity function the second argument Z – the ordinal number of the last number order equal to '0' is calculated (for convenience sake we use the element numeration starting with a unit). We write '1' into all the orders residing to the right from this order and also to the first number order, we write the values obtained with the help of randBinaryDigit function into the orders between the first and the last. This function using GPRN brings back either '0' or '1' with equal possibility. For example, the variant with complexity 20 is required. We select the length of the number from the interval [11, 19], for example, 16. Then the ordinal number of the last order equal to zero will be $z = 2l + 1 - c = 2 \times 16 + 1 - 20 = 13$.

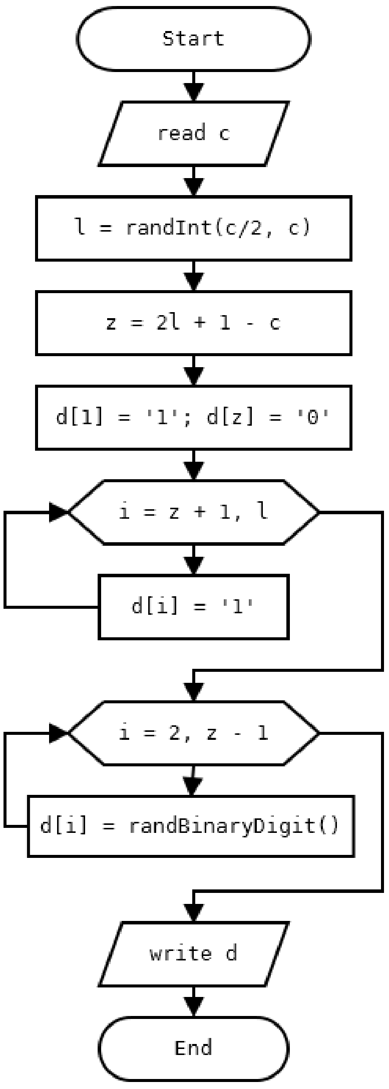

Figure 4. Algorithm of constructing the test variants

This means that for the orders from fourteenth to sixteenth symbol '1' is chosen. The first symbol also must be '1' in order to get a correct record of binary positive integer, and the thirteenth order is '0'. For the orders from the second to the twelfth any symbol can be chosen. The resulting variant of the test will contain a binary line d corresponding to the binary number, for example, «1101000110100111».

The estimation of the student solution is based on the procedure of *verification* – the solution is considered correct if it coincides with the solution obtained with the help of automation model. Both the student solution and the solution obtained with the help of reference algorithm represent the state of the control object – a structured system, which may be examined. This allows the stages and components of the solution to be separated and the order of element-by-element comparison of the solutions to be determined. Such a method makes possible not only checking the correspondence of the student solution to the correct solution, but also determination of the place and nature of an error in the automatic mode, which in its turn allows to accompany the comments of the estimation results. Moreover, stage-by-stage verification allows the concept of partial correctness of the solution to be introduced – if several steps of solution are made correctly, instead of a dichotomous scale "correct/not correct" we can use the scale of estimation with several marks or based on an fuzzy logic.

In the studied example a correct state of the data tape does not guarantee that the student exactly followed necessary algorithm. This problem is accomplished by the tape of the command journal – the student solution must be considered correct if the sequence of student's actions coincides with the sequence fulfilled by the controlling automaton. Otherwise, we can determine at what step the divergence occurred, in order that a commentary could be added to the result of estimation. If the majority of commands is accomplished correctly, or in case of a wrong state of the command tape the state of the data tape is correct, student's solution may be considered partially correct. Then some kind of special series of rules can be used to determine the mark. For example, if the values in the interval [0, 1] are used, where 0 is a completely wrong solution and 1 is a completely correct one, the following rules of estimation may be used for partially correct solutions:

- Basic mark is 0;
- If the state of the data tape is correct, 0.25 is added to the basic mark;
- If more than a half of commands corresponds to the correct solution, the basic mark is increased by $0.25 \times r / n$, where r is the number of correctly fulfilled commands, and n is the total number of commands accomplished by the controlling automaton.

Such rules, for example, allow us to guarantee that a partially correct solution will be accompanied by the mark of less than 0.25. Nevertheless, this is only an example, under other conditions and requirements the rules of estimating partially correct solutions may differ.

One more interesting problem is accounting of collisions upon composing the variants and estimating the solutions. Collisions may influence the complexity of the test variants. If, as was mentioned above, the collisions are admitted in the example under consideration allowing student to fix the head into any number order at the beginning, the function of complexity will lose its sense, since it is unknown in advance how many commands will be accomplished at the stage of direct run. In this case, the algorithms of composing the test variants and estimating the solutions given earlier lose their meaning and require correction.

It is evident for a student, if he is familiar with necessary algorithm, that the most efficient way is to fix the head into the extreme right number order. Then the stage of direct run will take only one step. The stage of the reverse run will remain unchanged in doing so. Assuming that the student will always act in the most efficient manner for himself we obtain a new function of complexity: $c = 1 + 1 + l - z = 2 + l - z$. This is the minimal possible number of steps for obtaining the variant solution. It is clear that the student can also choose another variant of the head location, which will result in an increase in the complexity of accomplishing the variant. But, first, such a situation is impossible to be foreseen, and, second, such a choice is illogical and may testify for a weak familiarity with the algorithm.

The influence of collisions on the algorithm of composing test variants is small – it is sufficient to change the limits of generation for the length of the number, which now must be greater than or equal to $c - 2$ and to use a new formula for calculating z.

Collisions also influence the procedure of verification. First, it will be necessary to modify the control object in order to record in what exactly number order the student fixed the head. Second, upon composing a correct solution with the help of controlling automaton we have to take into account student's choice – the head must be fixed in the same order as in the student's solution upon initialization of the data tape.

6. Conclusion

To summarize we can point out that automation model of the reference algorithm allows the complexity of the variants of algorithmic tests to be determined formally. Nevertheless, to solve the problem of automatic construction of the test variants with designated complexity it is necessary to determine the complexity function, which characterizes the dependence between the variant complexity and its properties. Automation of composing variants with equal complexity and estimation based on verification procedure allows VL to be created for algorithmic tests functioning in the completely automatic mode, which makes possible their use in the preparation of massive online courses.

Automation model of reference algorithm allows us to correlate student's solutions with a corresponding correct solution making possible to check all the intermediate results and an exact answer to the question, whether the student presented a correct solution, and also to indicate the place of an error. This model gives us the method of formal determination of the variant complexity of the algorithmic test. The significance of this method consists in the fact that on its basis the algorithms of constructing the test variants with designated complexity are created. However, the field of application of this model is limited by algorithmic tests only.

References

[1] Lyamin, A.V., Vashenkov, O.E. (2009) Virtual environment and instruments for student olympiad on cybernetics. In Proceedings of 8th IFAC Symposium on Advances in Control Education, Kumamoto, Japan, pp. 95-100.

[2] Tao, J., Jing-ying, Z., Lang, W. (2014) The thermal simulation of electromechanical platform system. In Proceedings of Transportation Electrification Asia-Pacific (ITEC Asia-Pacific), 2014 IEEE Conference and Expo, Beijing, pp. 1-4.

[3] Jaffry, D. (2014) Best Practices for Implementing Modeling Guidelines in Simulink, http://www.mathworks.com/company/newsletters/articles/best-practices-for-implementing-modeling-guidelines-in-simulink.html. Mathworks

[4] Efimchik, E.A., Lyamin, A.V. (2012) RLCP-Compatible Virtual Laboratories. In Proceedings of The International Conference on E-Learning and E-Technologies in Education (ICEEE 2012), Lodz, Poland, pp. 59-64.

[5] Le Xu, Dijiang Huang, Wei-Tek Tsai (2014) Cloud-Based Virtual Laboratory for Network Security Education. IEEE Transactions on Education, vol. 57, iss. 3, pp. 145-150. IEEE Press, New York (2014)

[6] Fu, Q., He, K., Ma, X. (2005) Research on Experimental Skills Assessment Based on Computer Simulation Technology. China Distance Education, Beijing, pp. 68-69.

[7] Rodríguez-del-Pino, J. C., Rubio-Royo, E., Hernández-Figueroa, Z. J.(2012) A Virtual Programming Lab for Moodle with Automatic Assessment and Anti-plagiarism Features. In Proceedings of the International Conference on e-Learning, e-Business, Enterprise Information Systems, & e-Government, Las Vegas.

[8] Sterbini, A., Temperini, M. (2013) Peer-assessment and grading of open answers in a web-based e-learning setting. In Proceedings of Information Technology Based Higher Education and Training, Antalya, Turkey, pp. 1-7.