Generation of Test Tasks in Systems of Computer Mathematics for Educational Purposes

Michael Lvov, Hanna Shmarova

Kherson State University, Uníversitets'ka, 27
73000, Kherson, Ukraine
lvov@ksu.ks.ua, hanna.shmarova@gmail.com

Abstract. The study of exact sciences includes not only lectures but also the active forms of learning (practical classes, laboratory work and so on). Thus, it is necessary to control not only declarative knowledge but also procedural knowledge of the students (the knowledge of methods for solving problems). This article describes functional requirements, mathematical models, and algorithms for the development of the modules of procedural knowledge testing in the systems of computer mathematics for educational purposes. We present methods of generation of test tasks implemented in the web application "Test on Mathematics".

Keywords: Systems of computer mathematics for educational purposes, generation of test tasks, procedural knowledge, computer software, template.

Key Terms: Educational process, computer software.

1 Introduction

There are two approaches to solve the problem of the generation of specific test tasks: saving a series of similar tests in the database and using algorithms for automatic generation of tests implemented as system procedures. Each of these approaches has its advantages and disadvantages. In the first case, it is necessary to spend a lot of time to fill the database, and tests are repeated. In the second case, the large amount of time is spent on the implementation of algorithms for automatic generation, but each test is unique. The weighted approach is to:

1. Develop a single common model for each sufficiently wide class of tests, as well as models and algorithms for generating conditions and answers for this model.
2. Develop common CASE-technologies for description subclasses of test tasks based on a single common model.
3. Develop joint mechanisms for storing and calling of algorithms for generation of specific test tasks.

2 Test Tasks-Templates

The expression $F(x_1, \dots, x_m; a_1, \dots, a_k)$ in signature $\sum_{SD}$ in subject domain SD is called expression template.

Variables $x_1, \dots, x_m$ are called variable templates or metavariables. The common range of values for metavariables is set Var ($x_i \in Var$). It follows that in each particular case, each of the variable templates represents one of the variables.

Variables $a_1, \dots, a_k$ are called coefficient templates or parameters. The range of values for parameters is numerical sets $(a_j \in Num)$. It follows that in each particular case, each of the coefficient templates is a number.

Example 1

$a_1x_1 + a_2x_2$ is expression template.

Specific instances of this template: $2a + 3x, -\frac{2}{3}u + 5x$ and so on. Suppose $F(x_1, \dots, x_m; a_1, \dots, a_k)$ is expression template and $x_j \in Var, a_j \in Num$, then

$$Sub^{v_1,\dots,v_m}_{x_1,\dots,x_m} Sub^{c_1,\dots,c_m}_{a_1,\dots,a_k} F(x_1, \dots, x_m; a_1, \dots, a_k) = F(u_1, \dots, u_m; c_1, \dots, c_k)$$

is called specialization (a special case or instance) of $F(x_1, \dots, x_m; a_1, \dots, a_k)$.

Expression templates define the common view of the model and common view of the answer for test task. It follows that the basis of determining the test task T is pair $< F_{Task}, F_{Ans} >$, where F_{Task}, F_{Ans} are expressions templates (Fig.1), defined on common lists of metavariables (X_{var}) and parameters (A_{Coef}).Then

$$T(X_{var}, A_{Coef}) =< F_{Task}(X_{var}, A_{Coef}), F_{Ans}(X_{var}, A_{Coef}) >$$

In tests on equivalent expressions transformation $F_{Task} =_{SD} F_{Ans}$, where " $=_{SD}$ " means semantic equality in domain SD.

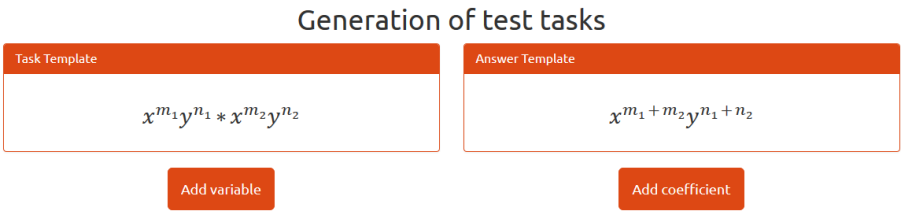

Fig. 1. Forms for displaying task and answer templates in the interface of web application "Test on Mathematics"

Example 2

$$T(x, y; m_1, n_1, m_2, n_2) =< x^{m_1}y^{n_1} \cdot x^{m_2}y^{n_2}, x^{m_1+m_2}y^{n_1+n_2} >$$

In addition to the signature of the domain system function $Val(F)$ is used in the definition of a template of test task. It is defined on the set of expressions of the sub-

ject domain. System interpreter of function $Val(F)$ computes a special canonical form of the expression. Function $Val(F)$ is used in algorithms for generation and verification of test task.

3 Descriptions of Variables and Coefficients in Samples

Set Var is common range of values for metavariables and basic set. Description of metavariables conforms to the following rules:

$$VarDescription ::=< VarList >\in< VarType >|.VarDescription;$$

$$< VarList >\in< VarType >$$

$VarList$ is a list of variables separated by commas,

$$VarType ::= Var|[VarID..VarID]|VarID|\{VarSet\}|VarType \cup VarType.$$

Var is a basic set of variables;
$VarID$ is one letter;
$[VarID..VarID]$ is a segment with type Var, initial and final value of segment are various letters without indices, or different letters with the same index, or the same letter with different indices (Fig.2);
$\{VarSet\}$ is a set of variables separated by commas;
$VarType \cup VarType$ are two combining types of descriptions.

Variables VarList (i.e., the left side of the definition) are metavariables. Variables that are included only in $VarType$ are instances. Both metavariables and instances of variables can be used in the template. Metavariables and instances of variables should be denoted by different letters to avoid the complication of semantics.

For example:

- $X, Y \in Var$ are metavariables are defined in the Var;
- $A, B, C \in [a..d] \cup [u..z]$ are metavariables take value $a, b, c, d, u, v, w, x, y, z$;
- $A \in [a..d]; B \in [u..z]$ are metavariables A and B have different domains;
- $F = ax + by + c;\ a, b, c \in [A..R].\ a, b, c$ are metavariables, x, y are instances of variables;
- $a_1,\ a_2,\ a_3,\ a_4,\ a_5,\ a_6 \in \{u, v\};\ u, v \in [a..z]$ are metavariables take value u and v, that take as values small Latin letters.

Numerical algebras $Nat \subset Int \subset Rat$ are common ranges of values for parameters and basics sets (Fig.3). Parameters with type Rat are defined by the value of their numerator and denominator:

$$r \in Rat, Num(r) \in [MinNum, MaxNum], Den(r) \in [MinDen, MaxDen]$$

with default constraints:

$$MinNum \leq MaxNum, MinDen \leq MaxDen, Num(r) \in Int, Den(r) \in Nat,$$

$$GCD\big(Num(r), Den(r)\big) = 1.$$

Descriptions for test tasks' templates are used in a procedure for generation instances of tests. In the simplest case, the procedure $GetTest$ has specification:

$$TaskTest\ P \coloneqq GetTest(TemplateTest\ T),$$

where $T =< F_{Task}, F_{Ans} >$.

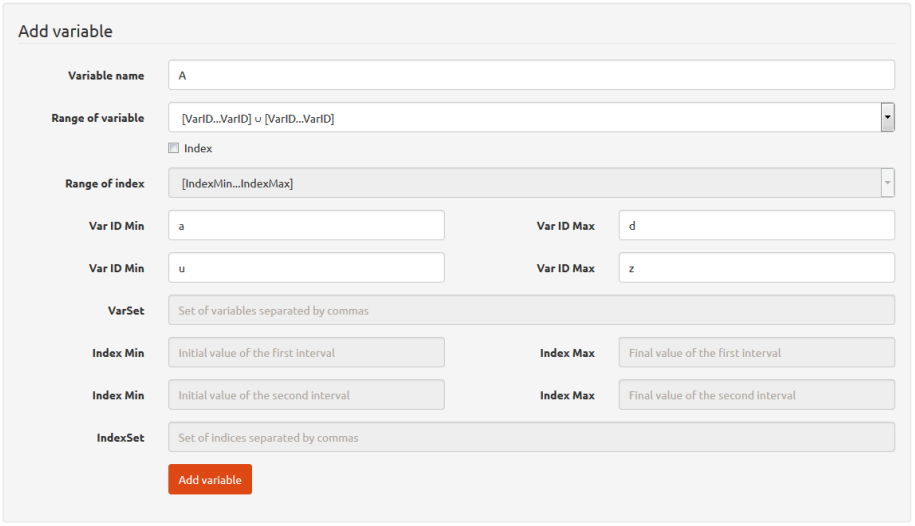

Fig. 2. Form for adding variable with a given range of values in the interface of web application "Test on Mathematics"

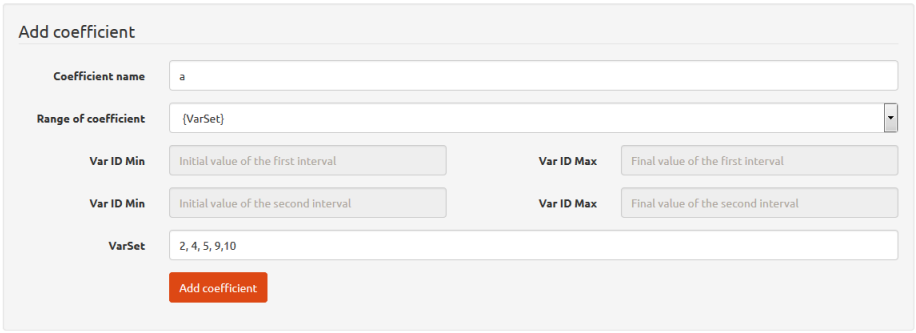

Fig. 3. Form for adding parameter with a given range of values in the interface of web application "Test on Mathematics"

Example 3. Apply the formulas of abridged multiplication:

$$T =< (a+b)^2, a^2+2ab+b^2 >$$

Suppose the user describes following template for this test task:

$$T_1 =< (X+Y)^2, X^2+2\cdot X\cdot Y+Y^2 >,$$

$$T_2 =< (a\cdot X+b)^2, a^2\cdot X^2+2\cdot a\cdot b\cdot X+b^2 >,$$

$$T_3 =< (a\cdot X+b\cdot Y)^2, a^2\cdot X^2+2\cdot a\cdot b\cdot X\cdot Y+b^2\cdot Y^2 >,$$

$$T_4 =< (a\cdot X^m+b\cdot Y^n)^2, a^{2\cdot m}\cdot X^{2\cdot m}+2\cdot a^m\cdot b^n\cdot X^m\cdot Y^n+b^{2\cdot n}\cdot\cdot Y^{2\cdot n} >.$$

X, Y are metavariables, a, b, m, n are parameters. Using conditions leads to a reduction of templates' number and extension of test tasks' classes. Thus, test task T_4 describes all classes of test tasks with templates T_1, T_2, T_3 in conjunction with additional conditions $= 1, b = 1, m = 0 \vee m = 1, n = 0 \vee n = 1$. Description of the conditions corresponds to the syntax:

$$CondList ::= Cond;|Cond, CondList;$$

$$Cond ::= AtomCond|AtomCond\ \&\ (Cond)|AtomCond \vee (Cond),$$

$$AtomCond ::= VarID = Const|VarID = VarID.$$

In this example condition has the form:

$$\Phi_{Cond} = (a = 1, b = 1, m = 0 \vee m = 1, n = 0 \vee n = 1).$$

The semantics of condition description is:

- Each atomic condition divides all set of tests into two classes. First class is a class, where this condition is satisfied, second class is a class, where it fails (in this case corresponding value is randomly generated from the range of values, that are defined by parameter descriptions): $VarID \coloneqq Const|VarID \coloneqq Random()$.
- Independent conditions are listed with separation by commas. All particular cases are defined by all possible combinations of conditions.
- Conjunction unites independent conditions that describe the same particular case.
- Disjunction combines dependent conditions, each of which defines different individual cases.

Test generation algorithm returns a single particular test in each of its call. Therefore, the condition is reduced to canonical form

$$\Phi_{Cond} = \Phi_1 + \Phi_2 + \ldots + \Phi_N$$

where Φ_j defines one particular case. Let's use logical variables that are corresponding to atomic conditions:

$$A = (a = 1), B = (b = 1), M_0 = (m = 0), M_1 = (m = 1), N_0 = (n = 0),$$

$$N_1 = (n = 1).$$

Then various tests are described via a Zhegalkin polynomial:

$$G = (A + 1)(B + 1)(M_0 + M_1 + 1)(N_0 + N_1 + 1)$$

Each monomial describes one particular test. It is necessary to write the polynomial in standard form G for describing Φ_{Cond} in canonical form:

$$G = ABM_0N_0 + ABM_0N_1 + \ldots + A + B + M_0 + M_1 + N_0 + N_1 + 1.$$

Presentation of tests' set for given template via polynomial G is used in the classification of tests according to their computational complexity. The complexity of the test is determined by a degree of the monomial (the number of boolean variables). The smaller is the degree of monomial, the more complex is a case of the test. In given example, the simplest tests are tests $ABM_0N_0, ABM_0N_1, ABM_1N_0, ABM_1N_1$. The most difficult test is test with all randomly generated parameters.

Semantics for the $GetTest$ procedure:

1. Randomly substitute one of the variables that are listed in the description of the metavariables into the T template instead of each metavariables x_j (Fig.4).
2. Randomly substitute one of the values that are listed in the description of the parameters into the T template instead of each parameters a_j(Fig.5).
3. Calculate the function Val in the resulting instance (Fig.6).
4. Return the result of calculations in the form $P =< F_{Task}, F_{Ans} >$.

Variables

Variable name	Conditions for variable	Conditions for index	Value
x	[a...f]	[...]	e
y	[k...z]	[...]	z

Showing 1 to 2 of 2 entries Previous 1 Next

Fig. 4. Table for displaying the description of the variables and their generated values in the interface of web application "Test on Mathematics"

Coefficients

Coefficient name	Conditions for coefficient	Value
m1	[1...10]	9
m2	[-10...-1]	-10
n1	[1...7]	5
n2	[2...8]	2

Showing 1 to 4 of 4 entries Previous 1 Next

Fig. 5. Table for displaying the description of the parameters and their generated values in the interface of web application "Test on Mathematics"

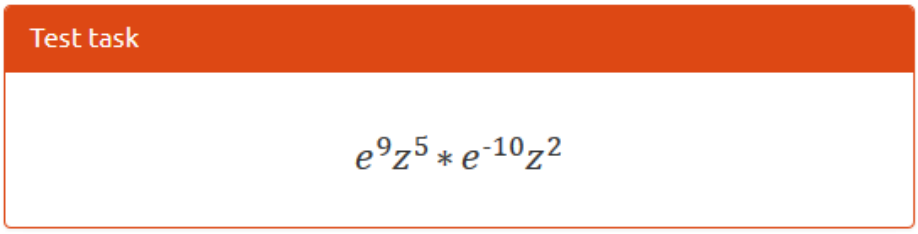

Fig. 6. An example of a specifically generated test task based on template in the interface of web application “Test on Mathematics”

4 Conclusion

In this article, we describe functional requirements, mathematical models, and algorithms for the development of the modules of procedural knowledge testing in the systems of computer mathematics for educational purposes.

Using of generation of test tasks saves time because there is no need to spend a lot of time to fill a database with conditions and answers for each task. The generator provides a large number of similar tasks based on a single template and each instance of the test is unique.

The testing system based on the results of this research is implemented in the web application “Test on Mathematics”. The system of computer mathematics for educational purposes “Test on Mathematics” will reduce the burden on teachers via automatic generation of the tests and verification of the answers. Thus, it is easy to control procedural knowledge during the absence of student (during his illness, during quarantine, etc.).

References

1. Lvov M.S. Proektirovanie logicheskogo vyvoda kak poshagovogo reshenija zadach v matematicheskih sistemah uchebnogo naznachenija / M. S. Lvov // Upravljajushhie sistemy i mashiny. – 2008. –No1. – S.25–32.
2. Lvov M.S. Koncepcija proghramnoji systemy pidtrymky matematychnoji dijaljnosti./ M.S.Lvov // Komp'juterno-orijentovani systemy navchannja: Zb. nauk. praci/ K.:NPU im. M.P.Draghomanova.– Vyp. 7.–2003.– S.36–48.
3. Lvov M.S. Koncepcija, arhitektura i funkcional'nost' gibkoj raspredelennoj programmnoj sredy uchebnogo naznachenija dlja srednej shkoly. Rabochee mestometodista /M. Lvov //Upravljajushhie sistemy i mashiny. –2009. –No 6. –S. 71-78.
4. Lvov M.S. Raspredelennye programmnye sredy uchebnogo naznachenija. Podsistema upravlenija uchebnym processom / M.S. Lvov // Upravljajushhie sistemy i mashiny. – 2010. –No1. – S. 66 – 71.
5. Lvov M.S. Matematicheskie testy v sistemah komp'juternoj matematiki uchebnogo naznachenija. / M.S.Lvov // Upravljajushhie sistemy i mashiny. –2011. – No6. – S.60–67

6. Lvov M.S. Matematychni modeli ta metody pidtrymky khodu rozv'jazannja navchaljnykh zadach z analitychnoji gheometriji/ M.S.Lvov // Iskustvennyj intellekt. –No1. – 2010. –C. 86–92.