International Journal of Innovative Studies in Sciences and Engineering Technology (IJISSET)

ISSN 2455-4863 (Online) www.ijisset.org *Volume: 3 Issue: 11 | November 2017*

A Grammar-based Methodology for Producing Mathematical Expressions

Sahereh Hosseinpour [1], Mir Mohammad Reza Alavi Milani [2], Huseyin Pehlivan [3]

[1] *Computer Engineering Department, Ataturk University, Erzurum, Turkey*
[2] *Computer Engineering Department, Ataturk University, Erzurum, Turkey*
[3] *Computer Engineering Department, Karadeniz Technical University, Trabzon, Turkey*

Abstract: *There are situations that one needs to write various kinds of mathematic questions, such as practicing tests, school exams, and function optimization algorithms. Some systems may use a database of mathematic equations where all expressions are written once and then it is being used many times. Solutions or answers of such questions might be stored, but it will not be a dynamic system, and no variety in expressions. There are various ways for random generation of mathematic expressions. Depending on involving operators and operands, variables, different types of generators can be implemented. Unfortunately, it is not possible to control those kinds of expressions that require satisfying conditions; hence there are methods that can control limited part of generating expressions. This paper addresses a grammar based methodology to automatically generate template-based mathematical expressions such as first-degree and quadratic equation, polynomials, and limits.*

Keywords: *Grammar-based method, Mathematical Expression, random generation, dynamic template, educational system*

1. INTRODUCTION

Different systems have been developed for expression generation using template in Natural Language Generation (NLG). YAG [1] produces Template Based strings in the form of real time and general-purpose. D2S (Data-to-Speech) [2] has been developed for different applications such as rout description, music, soccer report, and also for different languages including English. EXEMPLARS [3] is an object oriented, rule-based framework which supports dynamic text generator, and is a superset for JAVA, can be used templates of HTML/SGML. XtraGen [4] is XML and JAVA-based software system for NLG which can be easily integrated with other applications.

Randomness is an interesting topic for scholars, since past times. This study includes random number, random graphs, and more recently random natural languages generating. So, different methods have been proposed for random numbers generation [5-8]. Random numbers have different applications in different sciences such as cryptography [9-11], Computer Simulation [12, 13], and even Animal Sciences [14]. Because of random numbers generation importance, different methods have been proposed to produce random and pseudorandom numbers, such as methods that using chaotic functions [15] or electronic noises [16]. Also, Random generation is of great interest in other fields including E.N.GILBERT [17] studies which is concerns with random graphs and related probabilities. Random production is considered in Natural Language Processing (NLP), also, and different systems have been developed under the title of NLG, as measures taken by Langkilde [18], using stochastic techniques for NLG. These systems are divided into two categories real and Template Based. Kees Van Deemter [19] has compared these categories.

Tillman Bechar [20] presented a generation method for template-based NLG, using TAG. He proceeded on random language generation through integrating Basic Tree Nodes. Of course, template random generation is not limited to string random generation. It has also other applications; for example, Amruth N. Kumar [21] used templates to produce problems and mainly programs. Test case generation is another application of random generation. Takahide Y. et.al [22] designed a tool for Just-In-Time (JIT) compilers as runtime test. The issue of random generation is also presented in designing automatic tests. Various systems have been generated to help teachers generate question files in the internet [23-25]. In [26], Joao et.al generated a system to produce automatic mathematical tests with simple answers in which some structures have been designed to generate tests. In [27], Ana Paula designed a system for automatic generation of mathematics exercises based on Constraining Logic Programming (CLP). Such systems provide facilities for automatic generation of tests in environments such as Internet and virtual training systems.

One of the most important issues in designing mathematical tests is automatic generation of

mathematical expressions (AGMEs). For this purpose, we need to design templates which are appropriate templates for questions. AGMEs will be examined in this article. Moreover different generation methods of mathematical expressions and random space (number of generated expression) will also be examined in this article. Using Extended BNF grammars, we can develop a grammar for AMGEs. However, since we need to generate expressions with special characteristics in issues such as automatic tests, our work mainly focuses on generating mathematical issues with special characteristics and generating production templates. Produce of expression in proposed method is not completely random, and based on the subject can be design templates that generate mathematical expression with specific circumstances.

numerical and analytical are two main methods for solving mathematical problems. There are approximations for numerical solution of problems and different methods have been proposed for them. Analytical solving of problems in comparison with numerical solution is complicated, so some of problem haven't analitycal solution. In this paper we present a method to automatically generate expression which have analytical solutions.

In this article, a Context-Free Grammar (CFG) is suggested to prepare templates and develop classes of object-oriented type to generate Mathematical Expressions with underlying characteristics. In the literature, CFG are generally used to verify the syntax of given input data. On the contrary, this work uses CFGs to produce input data itself. We will also give a brief explanation of random expression, and then a grammar is presented to generate General Mathematical Expressions (GMEs). Generating random expressions and discuses about domain of this problem are two cases that presented in this article. Afterwards, we proceed on Mathematical Expressions generation with desired characteristics. Then we present a grammar entitled "Template-Context Grammar", and then examples of templates using this grammar will be implemented for some Mathematical Expressions.

2. GENRATION METHODOLOGY FOR MATHEMATICAL EXPRESSIONS

There are Simple and basic methods for randomly generating mathematical expressions. Such as, generating a mathematical expression involving numbers as operand and these characters, +, - ,* as operators which can be developed in Algorithm 1.

```
FormulaStr ← generate a random number
while True
Operator ← select a random operator from {'+' , ' -' , '*'}
Operand ← generate a random number
FormulaStr ← FormulaStr &  Operator & |Operand|
```

Algorithm 1: *Statement for generate a random basic arithmetic expression*

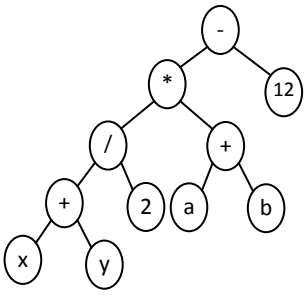
According to Algorithm 1 arithmetic expressions such as "2^3+5*3-2+3+10" can be generated randomly. Similar to the mentioned method in this algorithm, by adding some modifications to this method, polynomial expression can also be generated as in Algorithm 2.

```
polyExp ← ' '
n←a random number as length of polynomial
repeate for n times
        CoeffN ←generate a random number
        if CoeffN>0 : polyExp←polyExp & '+' & CoeffN & '* x'^n
        if CoeffN<0 : polyExp←polyExp & '-' & |CoeffN|& & '* x'^n
        decrease n by 1
```

Algorithm 2: *Generating a random polynomial expression*

Using statements of Algorithm 2, polynomials such as $3*x^5-15*x^3+8*x^2+30$ can be generated. Note that in this example, an expression with the power of four has the coefficient of zero. Of course, the method discussed in this section is appropriate for generating Specific Mathematical Expressions (SMEs) like polynomials. Mathematical Expression, other than above mentioned cases in this section, can be placed in parentheses or have similar mathematical functions, division operators, sin, cos etc. These issues make their generation almost impossible by this method.

To generate mathematical functions in general, we need a special structure, with recursive characteristic. In [28], Todd V. proposed a structure for mathematical expressions to transfer expressions as function arguments. Moreover, mathematical expressions can be displayed in the form of binary tree structure. This structure known as Expression Tree has an inherent tree-like structure. For example, Algorithm 3 indicates tree-structure of " $\frac{x+y}{2} * (a + b) - 12$".



Algorithm 3: *Tree representing the expression* $\frac{x+y}{2} * (a + b) - 12$

Binary tree-structure can be appropriate for illustrating mathematical expressions. Using this structure, we can simulate all kinds of operators and mathematical functions, and prioritize operators when defining the expression. Expression retrieval from this structure with infix traverse can be easily done. Mathematical expression can be also indicated in the form of object-tree in tree structure. For example object tree expression for expression $\frac{x+y}{2} * (a + b) - 12$ can be written as follows.

Minus (*Times* (*Divide* (*Plus* (*Var* ('x') , *Var* ('y')) , *Num* (2)), *Plus* (*Var* ('a'), *Var* ('b'))) , *Num* (12))

This structure can be easily evaluated in a recursive manner. Every node in this structure has an operator or an operand. Each child node is an operand for its parent node, and structurally it can also be an operator. Thus making mathematical expressions in this structure is easy to evolve and evaluate.

There are several types of mathematical expressions that can be mentioned to some of them as follows:

- Arithmetic expressions that contain only numbers and operators.
- Mathematical expressions containing variables (operations such as derivatives, limits, etc.)

Equality expressions (equations).

Also mathematical expressions can be solved in two forms of numerical or analytical. Numerical approch have any approximations error, and have been developed variety method for the numerical solution via computer. In this paper proposed a method for generate mathemathical expression that suite for numerical and analytical solution. Proposed methodology has three main steps as following:

- Developing a grammar.
- Create a template according grammar.
- Developing a template class via programming language.

3. DEVELOPING A CONTEXT-FREE GRAMMAR

To work with mathematical expressions in Computer Algebra Systems, defining a framework is necessary for identifying and/or generating mathematical expressions. BNF is useful to define context-free grammars in languages, because it has simple notations, recursive structures, and is widely available supported by many compiler generation tools such as YACC [29], LEX [30], and JavaCC [31, 32]. Ryan et al in [33] developed a BNF grammar for identification and production of mathematical expressions. Inspired proposed grammar in [33], we have developed an Extended-BNF grammar in Algorithm 4 for mathematical expressions.

G={*N*,*T*,*P*,*S*}
N={*expr*,*op*,*func*,*var*,*number*,*digit*}
T={*X*,*Sin*,*Cos*,*Tan*,*Log*,*Exp*,*Sqrt*,+,-,*,/,^}
S={*expr*}
Productions :
<*expr*>::= <*expr*> <*op*> <*expr*> | (<*expr*> <*op*> <*expr*>)
|<*func*>(<*expr*>) |<*var*> |<*Number*>
<*op*>::= ' +' | '-' | '*' | '/' |'^'
<*func*>::= ' *Sin* ' | *'Cos'* | *'Tan'* | *'Log'* | *'Exp'* | *'Sqrt'*
<*var*>::= 'X'
<*number*>::= '-'? <*digit*> + ('.' <*digit*>+)?
<*digit*>::=[' 0'-'9']

Algorithm 4: *An EBNF grammar for mathematical expression*

Using grammar of Algorithm 4, all mathematical expression can be identified. In this grammar five operators and six functions have been placed. Other functions can be added to the grammar if needed. Note that in this grammar, the priority of operators has not been taken into consideration, while mathematical expressions have different priority levels based on different operators. Since in this article we aim to use this grammar as mathematical expression generator, we do not need consider priority levels. Hence present grammar can be used for mathematical expression generation, using its random grammar rules.

A general and structural template can be defined for generation of different and desired mathematical expression, using Template-Context Grammar.

Definition 1: A template context grammar is a system G = <N, T, P, S>, where N and T are disjoint finite nonempty sets (non-terminal and terminal alphabets), S ∈ N (the start symbol), P is a nonempty finite set of rules in the form u→v, I where u ∈N, v ∈ (N ∪ T)* and I is a positive integer number, denoting the number of iterations for the current rule.

This type of grammar has a structure similar to programmed grammars. The only difference is that right side of rules in this grammar is composed of two sections. First part is similar to CFG grammars, and second part involves an integer number, identifying the number of frequency for this rule. Using such kind of grammar, we can generate mathematical expressions with identified features. It is, worth to mention that by

using such types of grammar, we can generate general frames for expressions, and some features including domain of random numbers can be considered as parametric for coefficients, number of expressions etc. at the time of implementation. In the following sections, we will explain template generation method and its implementation in Java language, using grammar.

4. GRAMMAR MANIPULATION

In order to implement mathematical expression generation using Template-Context Grammars, some functions can be developed for each non-terminal of grammar. This function calls other functions, according to grammar rules. Frequency of each grammar rule (part two in right side of rules) is implemented through placing the grammar inside a loop. As an example, to implement a rule in the form of u→v ,I, supposing that equivalent function of u and v are uFunc and vFunc respectively, pseudo code of Algorithm 5 can be proposed.

```
Public String uFunc :
    Str ← ""
    Repeate for  I  times
        Str ← Str & vFunc
    Return Str
```

Algorithm 5: *Pseudo code for implement of a template-context rule*

Algorithm 5 is appropriate for rules having only one term in their right hand side. For those rules in the form of u→(v1,v2,..,vn),I each call should be randomly accomplished from different rules. Pseudo-code of Algorithm 6 indicates the above mentioned case.

```
Public String uFunc :
    Str ← ""
    Repeat for I times
        n ← Generate a rule index randomly
        Str ← Str & v_nFunc
    Return Str
```

Algorithm 6: *Pseudo code for implement of a template-context multi rule*

Using pseudo-code of Algorithm 6, we can develop appropriate function for all considered grammar rules. Another important point in designing template is adding special characteristics and generating more limited expressions in accordance with user's application. To implement such limitations, we can consider variables using object-oriented concept and based on it we can change some generated functions in accordance with type of limitation. Doing so, we can develop classes for all kinds of considered frame. Most limitations occur while generating numbers. In accordance with underlying domain of a frame, we can do this using Rand function.

When implementing object oriented, we can use some variables to make limits, parametric and generate considered mathematical expressions through initializing these parameters. Doing so, we can develop classes for each kind of considered frames. In what follows some examples of frame and implementations appropriate with it, will be presented.

5. AUTOMATIC EXPRESSION PRODUCTION

In this proposed CFG grammar, a random mathematical expression can be generated starting with a non-terminal and random selection from grammars in which our non-terminal has been placed in left side. The pseudo-code shown in Algorithm 7 indicates mathematical expression method, based on the grammar of Algorithm 4.

```
AST ← <expr>
repeate while a Non-Terminal exist in AST
    E ← select a NonTerminal symbol in AST
    subAST ←select a rul that begin by E, randomly
    AST ←put subAST in place E of AST
inorder traverse AST and get mathematical expression
```

Algorithm 7: *Pseudo code for generate a mathematical expression randomly*

According to the pseudo-code in Algorithm 7, we can generate a mathematical expression with random characteristics. Number and operators, and size of generated numbers are unpredictable in this expression. For example it is possible to choose the rule <expr>:= <number> at initial stages. In this situation, through selecting a number instead of <number>, it will be generated mathematical expression without any operator. Similarly, it is possible to generate a mathematical expression by only one variable, like "f(x) = x". On the other hand, rules of grammar should be chosen such that the length of generated mathematical expression is extremely long or even infinite. Stochastic Grammars [34] can be used to control generated mathematical expressions. With initialize the probabilities for each rule in the beginning, and changes during generation process; we can generate more limited mathematical expressions. The restriction that can be created through this is the restriction on the string length. So that, at the time of

generation and arriving to identified level of tree, the probability of generating for all rules except rules that will be generate leave node (rules that generate number or variable) reaches zero, and preventing tree development and increase in its length. Length restriction can be applied through achieve a certain number of generated nodes. By doing so, long mathematical expressions will not be generated. The pseudo-code in Algorithm 8 is a changed form of that in Algorithm 7 pseudo-code through limiting tree product at n^{th} node.

AST ← <*expr*>

index←0

repeate while a NonTerminal exist in *AST*

E ← select a NonTerminal symbol in *AST*

if *index*<*n* then

subAST ←select a rul that begin by *E*, randomly

else *subAST* ←select a rul from<*expr*>::=<*number*>|<*var*>

AST ←put *subAST* in place *E* of *AST*

update *index* according selected rule /* increase by 1 or 2 */

inorder traverse *AST* and get mathematical expression

Algorithm 8: *Modified pseudo code with level restriction for generate a mathematical expression*

Using the pseudo code in Algorithm 8, it is possible to generate longest limited mathematical expression; still some other limitations should be taken into consideration when generating mathematical expressions. For example, generate a quadratic equation or an ambiguity expression for limit question.

6. SIZE EVALUATION OF PRODUCTION SPACE

Mathematical expression generated by the provided grammar in Algorithm 4, can be selected randomly or can be selected from infinite number of expressions. If we limit generated expressions by identified mechanisms such as maximum nodes or maximum level of trees, it is possible to convert infinite space of the problem to finite space. Statistical space of problem can be calculated in two methods.

Method 1: suppose that applied limit of generating mathematical expressions is related to the number of nodes. In this case we can do as following, to calculate possible number of states. Number of trees produced by n nodes, is calculated using equation 1.

$$S_n = \frac{1}{n+1}.\binom{2n}{n} = \frac{2n!}{n!(n+1)!} \qquad (1)$$

Considering that underlying generated trees have a maximum of n nodes, total number of trees generated by n nodes can be calculated using equation 2.

$$S_{total} = \sum_{k=1}^{n} \frac{2k!}{k!(k+1)!} \qquad (2)$$

The obtained number from equation 2 is the number of possible trees. While there are various combinations for each tree, based on operators or functions that can be located in nods, and number or variables that located in leaves. In the following, statistical population space at limit of number of tree levels will be calculated more accurately.

Method 2: Statistical space of problem can be considered based on restriction of tree level. Moreover in each produced tree structure, different integrations can be generated through considering operators, functions, variables, and interval of random numbers. Suppose that mathematical expressions which are to be generated have the following characteristics:

- Number of variable : *v*
- Number of figures (interval of numbers produced randomly): *n*
- Number of operator : *m*
- Number of mathematical function (such as sin, cos, etc.) : *f*

In this case number of possible states for a node will be equal to equation 3:

$$S_0 = v + n \qquad (3)$$

Such number is equal to the number of tree states at 0 levels. To calculate number of states at 1st level, different states developed from a node, should be taken into consideration. These states are shown in Fig. 1.

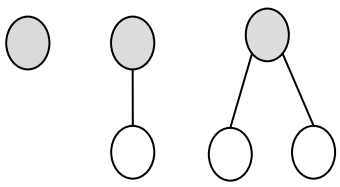


Fig. 1: *The possible scenarios of binary tree in first level.*

Nodes of 1st degree can have f state and quadric nodes can have m state. Moreover like tree, leave can be considered with level 0. Therefore, the total number of possible states for the tree level 1 is equivalent to equation 4:

$$S_1 = S_0(1 + f + m.S_0) \qquad (4)$$

For the second level of tree, we can consider states displayed in Fig. 1, in a way that these states can be used as substitution for leave of level 1. Fig. 2 indicates this issue.

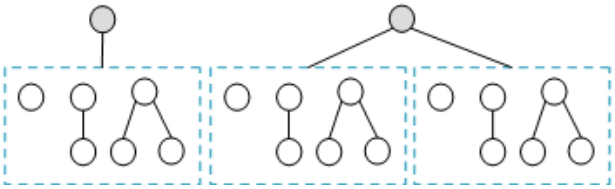


Fig. 2: *The possible scenarios of binary tree in second level*

Considering Fig. 2, we can use equation 5 for calculating the number of possible mathematical expressions of tree with level two.

$$S_2 = S_1(1 + f + m.S_1) \qquad (5)$$

Generalizing equation 5, we can obtain equation 6 for calculating the number of producible mathematical expressions by tree with level n.

$$S_n = \begin{cases} v + n & n == 0 \\ S_{n-1}(1 + f + m.S_{n-1}) & n > 0 \end{cases} \qquad (6)$$

For example, Table 1 shows the number of possible expressions for some levels of the tree. For Table 1, number of operators is five, number of functions is eight, number of variables is one, and number interval is ranges from -50 to + 50.

Table 1: *Number of possible mathematical expression generating in various level of tree*

Level of Tree	*Number of States*
Level 0	S0=1+101=102
Level 1	52938
Level 2	14,012,635,662
…	…
Level 6	6.743220307892116 e+172
Level 7	2.273551006038432 e+ 346

7. METHODOLOGY ILLUSTRATION

7.1. Generate SMEs Randomly

As seen in previous section, generation space of mathematical expressions in structure tree is a large and probability of generating expressions with special characteristics is low. To produce expressions in which identified characteristics are considered, two methods can be used. In the first method, through changing the grammar generator of mathematical expressions, we can design expressions appropriate with underlying type. This method is difficult and complex by the current grammars. The other method is designing templates for automatic generation of expressions in which considered characteristics have been placed. We will discuss more about both generation methods in the following.

7.2. Generate SMEs via Grammar

In this method a separate grammar generator should be designed for each expression. Generated grammars are of EBNF type. While designing them, considered expression characteristics should be taken into account. In following sections some examples from this type of expression generator have been presented.

7.2.1 First Degree Equation

When generating first degree equation, both sides of equation should have expressions with maximum degree 1. To do so, EBNF grammar can be developed in a way that the possibility of generating equations with degrees higher than one will be diminished. Such states usually occur while using the operator '*'. Hence in the case of seeing operator '*', we can send operands to rule (<expr'>) in which generation of variable is impossible. Algorithm 9 lists grammars related to this grammar.

```
G={N,T,P,S}
 N={expr, expr',S, var, number, digit}⊆ Σ
T={x,+,-,*}⊆Σ
P:
< S >::= <expr> ' =' <expr> | <expr> ' / ' <expr> '=' <expr' >
<expr>::= <expr> ' +' <expr>
          | <expr> ' -' <expr> |  number | var
<expr>::= <expr>  ' *'  <expr'>  | <expr' > ' *' <expr>
<expr'>::= <expr'> ' +' <expr'>
          |<expr' >  ' -' <expr'>
          | <expr'> ' *' <expr'> |  Number
<var>::= 'x'
<number>::= '-'? <digit> + ('.'<digit>+)?
<digit>::=[' 0'-'9']
```

Algorithm 9: *EBNF grammar for first degree equation generation*

Note that grammar in Algorithm 9 has been considered only for division operator at a simple state. If we want to use division operator generally, then grammar design will be more complex.

7.2.2 Quadratic Equation

Similar to generation of first degree equations, we can write a grammar for generation or identification of quadratic equations. In the grammar of Algorithm 10 presented for quadratic equations no division operator will be generated.

G={N,T,P,S}
N={*expr*,*expr'*,*expr''*,*S*,*var*,*number*,*digit*}
T={*x*,+,-,*,/,^}
P:
< *S* >::= <*expr*> '=' <*expr*>
<*expr*>::= <*expr*> ' +' <*expr*>
| <*expr*>' -'<*expr*> | *number* | *var* |*var*^{2}
<*expr*>::= <*expr'*> ' *' <*expr'*>
|<*expr*>' *'<*expr''*>|<*expr''* >' *'<*expr*>
<*expr'*>::= <*expr'*>' +'<*expr'*>
|<*expr'* >' -'<*expr'*>|<*expr'* >' *'<*expr''*>
| <*expr''*>' *'<*expr'*> | *number* | *var*
<*expr''*>::= <*expr''*>' +'<*expr''*>
|<*expr''* >' -'<*expr''*>
| <*expr''*>' *'<*expr''*> | *Number*
<*var*>::='*x*'
<*number*>::= '-'? <*digit*> + ('. '<*digit*>+)?
<*digit*>::=[' 0'-'9']

Algorithm 10: *EBNF grammar for quadric equation*

Although all expressions generated in this way are of quadratic equation, still there is no identified control of length, number of operators, and expression sophistication (complexity). To apply limitation at expression generation, special methods such as limiting generated tree level, using probable grammars, and changing their probabilities can be used depending on tree level etc.

8. PRODUCTION OF MATHEMATICAL EXPRESSION VIA TEMPLATES

Another method to generate identified mathematical expressions is using templates which have specific characteristics. In these templates, random cases should be identified. While processing templates, underlying places are replaced with random values to generate random expressions. For example to produce a polynomial with identified number, we should generate a template that involves intervals for the number of expressions, their degree, and coefficients. Algorithm 11 shows a simplified template for polynomials.

Polynomial:
Number of Monomials: (N_a, N_b)
Polynomial Degree : (D_a, D_b)
Coefficient of Monomials : (C_a, C_b)

Algorithm 11: *A simple template structure for polynomials*

However, using this method is not appropriate, since we need to define different structure for expressions of different types. Moreover, different processing should be considered for generating different mathematical expressions. To use template methods in generating mathematical expressions we need a special and unique method. The method should be universal and capable of defining templates for all kinds of expressions with identified features. Proposed structure of this article is tree structure and a grammar can be developed for its identification and generation. Suggested grammar has special characteristic, which is entitled Template-Context Grammar in this article. This grammar is a grammar of developed structure from Programmed Grammars [35] class. Structure of these grammars will be explained in the following.

8.1 Examples of Mathematical Expression Templates

In this section we are developed two examples that show how we can create template classes for mathematical expression generators.

8.1.1 Polynomial Template

In this section, using the suggested grammar structure we would like to develop a grammar which can be used as template for generating polynomials. Considering that polynomials are composed of different terms added to or subtracted from each other, and each term (monomial) has its own identified structure, a grammar such as the one in Algorithm 12 can be suggested.

G={N,T,P,S}
N={*E*,*T*, *T'*, *R*, *R'*, *F*,*num*, *digit*,*S*}
T={*x*,+,-,*,^}
P:
S →*T E* ,1
E →(+*T* | -*T*) , *R*
T →(*F* | *F* "*" *T'*) ,1
T'→("*x*" | "*x*" *R'*),1
R →*num*,1
R'→*num*,1
F →*num* , 1
num→ '-'? *digit* + ('.' *digit*+)?
digit→[' 0'-'9']

Algorithm 12: *A template-context grammar for polynomials*

Considering the grammar in Algorithm 12, we can generate random polynomials. Although, apparently, these polynomials are expressed generally, there is limiting probability in it. Characteristics which can limit polynomials are number of terms, coefficients of each term and polynomial degree. In the time of implementation, considering a simple process and putting parameters to generate numbers related to each section, we can formulate considered polynomial.

In java programming language, a class can be developed which can be appropriate for generation of

desired polynomial. In this class, some parameters are considered to determine three above mentioned limits: interval of number of terms, coefficients, and degree of polynomial. Algorithm 13 indicates codes related to polynomial generator class, based on grammar of Algorithm 12.

```
Class polynomialTemplate {
Ranges termsRange = new Ranges( );
Ranges coeffRange = new Ranges( );
Ranges degreeRange = new Ranges( );

Public String polynomialGenerate ( ) {                    /*
Equivalent to the start symbol (S). */
        return term( ) & expr( );
}

Public String expr( ) {                                   /*
Equivalent to the E symbol. */
    String str ;
    int n = termRange.min + (int) (Math.random(
    )*(termRange.max- termRange.min));

    for ( int i=0; i< n ; i++ ){
        int t = (int)(Math.random( )*2);
        if(t==0) str = str & "+" & term( ) ;
        else if (t==1) str = str & "-" & term( ) ;
    }
    return str;
}

Public String term( ) {   /* Equivalent to the T symbol. */
    int t = (int)(Math.random( )*2);
    int f = degreeRange.min + (int) (Math.random(
    )*(degreeRange.max- degreeRange.min));
    if ( t==0 )    return Integer.toString ( f );          /*
    T → F  */
    else    return Integer.toString ( f ) & "*" & term2 ( );    /*
    T → F * ( 'x' | 'x^' R)   */
}

Public String term2 ( ) {   /* Equivalent to the T' symbol. */
    String str;
    int t = (int) (Math.random( )*2);
    if ( t==0) str = "x";
    else {
        int d = coeffRange.min + (int) (Math.random(
        )*(coeffRange.max- coeffRange.min));
        str = "x^" & Integer.toString ( d );
    }
    return str;
}
}
```

Algorithm 13: *Polynomial generator class in Java*

The class shown in Algorithm 13 is able to generate all kinds of parametric polynomials. Using this class and determining value of parameters *termsRange*, *coeffRange*, and *degreeRange*, each template will be definable. As an example, to define a template to generate polynomials whose number of terms is between three and five, with polynomial degree of 2-6 and polynomial coefficient of 5-10 we can act as Algorithm 14.

```
polynomialTemplate PolyTmp1 = new polynomialTemplate ( );
PolyTmp1. termsRange.min = 3; PolyTmp1. termsRange.max = 5;
PolyTmp1. degreeRange.min = 2; PolyTmp1. degreeRange.max =
6;
PolyTmp1. coeffRange.min = -5; PolyTmp1. coeffRange.max = 10;
```

Algorithm 14: *An example template for polynomial template class*

The example shown in Algorithm 14 is a sample polynomial in which a node involving random polynomial with underlying characteristics can be generated through calling function *PolyTmp1.polynomialGenerate*(). By doing so, a defined class can be considered as Dynamic Template.

8.1.2 Quadric Trigonometric Equation Template

As another example, we can consider quadric trigonometric equation generation. According to proposed grammar, a grammar can be developed for such kind of equations as template. This grammar can be seen in Algorithm 15.

G={N,T,P,*S*}
V={*E*,*E'*,*T*, *T'*, *R*, *R'*, *F*,*num*, *digit*,*S*}
T={*sin x*, *cos x* ,*tan x*, *cotan x*, +, -, *, ^, =}

P:
S →(*E* "=" *E* | *E* "/" *E* "=" *F*),1
E →*TE'* ,1
E' →(+*T* | -*T*) , *R*
T →(*F* | *F* "*" *T'*) ,1
T'→(*T''* | *T''* "^2"),1
T''→("*sin x*" |"*cos x*"| "*tan x*" | "*cotan x*"), 1
R →*num*,1
F →(*N* | √*N*), 1
N →(*num* | *num* / *num*), 1
num→ '-'? *digit* + ('.' *digit*+)?
digit→[' 0'-'9']

Algorithm 15: *A template-context grammar for quadric trigonometric equations*

According to the grammar in Algorithm 15, the generated equations can be limited to two intervals: Number of terms for each side of equal, and coefficients of each term. It is, possible for this special example to consider other characteristics during implementation. For example, a parameter can be considered to determine the type of triangles function which

generating term will be involve only that function and no other triangle functions will appear.(it is of course possible to choose triangle functions randomly, before each call).

9. CONCLUSIONS

In this paper, we have presented Template-Context Grammar and object oriented classes, designed to support practical template based mathematical expression generations. By using this grammar type a grammar can be developed for generating mathematical expressions. Designed grammar can change to classes during implementation, which has limiting parameters. For example it is possible to implement number of terms or maximum degree of polynomial in a class as parameter. By doing so, we can design dynamic templates for to generating mathematical expressions. These templates are in the form of classes. By designing different frames for the type of term, we can have a data base of templates. This data base can be used for generating automatic tests or other systems which require mathematical expressions with special format.

REFERENCES

[1] McRoy, S.W., S. Channarukul, and S.S. Ali. "YAG: A template-based generator for real-time systems". in Proceedings of the first international conference on Natural language generation, 2000. Vol 14. Association for Computational Linguistics.

[2] Theune, M., et al., "From data to speech: a general approach". *Natural Language Engineering*, Vol 7, No 1, 2001. pp. 47-86.

[3] White, M. and T. Caldwell. "A practical, extensible framework for dynamic text generation". in *Proceedings of the Ninth International Workshop on Natural Language Generation (INLG). 1998*.

[4] Stenzhorn, H. "XtraGen: a natural language generation system using XML-and Java-technologies". in *Proceedings of the 2nd workshop on NLP and XML,* 2002. Vol 17. Association for Computational Linguistics.

[5] Schellekens, D., B. Preneel, and I. Verbauwhede. "FPGA vendor agnostic true random number generator". in *International Conference on Field Programmable Logic and Applications, 2006. FPL'06.* IEEE

[6] Tkacik, T.E., "A hardware random number generator", in *Cryptographic Hardware and Embedded Systems-CHES 2002* 2003, Springer. pp. 450-453.

[7] Jun, B. and P. Kocher, "The Intel random number generator". *Cryptography Research Inc*. white paper, 1999.

[8] Cohn, C.E., "RANDOM NUMBER GENERATOR", *Google Patents* , 1972.

[9] Jakimoski, G. and L. Kocarev, "Chaos and cryptography: block encryption ciphers based on chaotic maps". *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 2001. Vol 48, No 2: pp. 163-169.

[10] Kocarev, L., "Chaos-based cryptography: a brief overview". *Circuits and Systems Magazine, IEEE, 2001*. Vol 1, No 3: pp. 6-21.

[11] Lagarias, J., "Pseudorandom number generators in cryptography and number theory". *Cryptology and computational number theory*, 1990. Vol 42: p. 115-143.

[12] Ferrenberg, A.M., D. Landau, and Y.J. Wong, "Monte carlo simulations: Hidden errors from ''good''random number generators". *Physical Review Letters*, 1992. Vol 69, No 23: pp. 3382.

[13] Coates, R.F., G.J. Janacek, and K.V. Lever, "Monte Carlo simulation and random number generation". *IEEE Journal on Selected Areas in Communications,* 1988. Vol 6, No 1: pp. 58-66.

[14] Schaeffer, L., "Application of random regression models in animal breeding". *Livestock Production Science*, 2004. Vol 86, No 1: pp. 35-45.

[15] Stojanovski, T. and L. Kocarev, "Chaos-based random number generators-part I: analysis [cryptography]". *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications,* 2001. 48(3): p. 281-288.

[16] Epstein, M., et al., "Design and implementation of a true random number generator based on digital circuit artifacts", in *Cryptographic Hardware and Embedded Systems-CHES 2003*, Springer. pp. 152-165.

[17] Gilbert, E.N., "Random graphs". *The Annals of Mathematical Statistics*, 1959: pp. 1141-1144.

[18] Langkilde, I. and K. Knight. "Generation that exploits corpus-based statistical knowledge". in *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th*

International Conference on Computational Linguistics, 1998-Vol 1. Association for Computational Linguistics.

[19] Van Deemter, K., E. Krahmer, and M. Theune, "Real versus template-based natural language generation: A false opposition?", *Computational Linguistics*, 2005. Vol 31, No 1: pp. 15-24.

[20] Becker, T. "Practical, template-based natural language generation with TAG". in *Proceedings of TAG*. 2002.

[21] Kumar, A.N., "Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors". *Technology, Instruction, Cognition and Learning (TICL) Journal*, 2006. Vol 4, No 1.

[22] Yoshikawa, T., K. Shimura, and T. Ozawa. "Random program generator for Java JIT compiler test system". in *Proceedings Third International Conference on Quality Software, 2003*. IEEE.

[23] Melis, E., et al., "ActiveMath: A generic and adaptive web-based learning environment". *International Journal of Artificial Intelligence in Education (IJAIED)*, 2001. Vol 12: pp. 385-407.

[24] Hoffman, D., et al., "Two case studies in grammar-based test generation". *Journal of Systems and Software*, 2010. Vol 83, No 12: pp. 2369-2378.

[25] Klai, S., T. Kolokolnikov, and N. Van den Bergh. "Using Maple and the web to grade mathematics tests". in *Proceedings of International Workshop on Advanced Learning Technologies, IWALT 2000*. IEEE.

[26] Almeida, J.J., et al. "Math exercise generation and smart assessment". in *8th Iberian Conference on Information Systems and Technologies (CISTI), 2013*. IEEE.

[27] Tomás, A.P. and J.P. Leal, "A CLP-based tool for computer aided generation and solving of maths exercises", in *Practical Aspects of Declarative Languages*2003, Springer. pp. 223-240.

[28] Veldhuizen, T., "Expression templates". *C++ Report*, 1995. Vol 7, No 5: pp. 26-31.

[29] Johnson, S.C., "Yacc: Yet another compiler-compiler". Bell Laboratories Murray Hill, NJ , 1975, Vol. 32.

[30] Levine, J.R., T. Mason, and D. Brown, "Lex & yacc", *O'Reilly Media, Inc* , 1992.

[31] Viswanadha, S. and S. Sankar, "Java compiler compiler (JavaCC)-The java parser generator". *Java. net, https://javacc. dev. java. net/*, accessed Aug, 2009. 23.

[32] Kodaganallur, V., "Incorporating language processing into java applications: A JavaCC tutorial. Software", IEEE, 2004. Vol 21, No 4: pp. 70-77.

[33] Ryan, C. and M. O'Neill, "Grammatical evolution: A steady state approach". *Late Breaking Papers, Genetic Programming,* 1998: pp. 180-185.

[34] Newmeyer, F.J., "Grammar is grammar and usage is usage". *Language*, 2003: pp. 682-707.

[35] Rosenkrantz, D.J., "Programmed grammars and classes of formal languages". *Journal of the ACM (JACM)*, 1969. Vol 16, No 1: pp. 107-131.

AUTHORS' BIOGRAPHY

Sahereh Hosseinpour: She is working as a lecture and Phd. student, Dept. of Computer Engineering at Ataturk University in Turkey.

Dr.M.Mohammad R. A. Milani: He is working as a Asst. Professor, Dept. of Computer Engineering at Ataturk University in Turkey.

Dr.Huseyin PEHLIVAN: He is working as a Asst. Professor, Dept. of Computer Engineering at Karadeniz Technical University in Turkey.