DOMAIN-SPECIFIC LANGUAGE FOR MATHEMATICAL EXPRESSIONS

Cristina LÎNIUC, Filip OSOIANU, Eliodor POPOV, Cătălin BARGAN

Technical University of Moldova

Abstract: *In this paper it is analyzed what a domain-specific language means, its main advantages and how to develop a DSL for evaluation of mathematical expressions, how to define its grammar and what rules are applied. There are pointed out the most important parts used to define a domain-specific language. Besides it is discussed on several examples how the grammar was parsed to ANTLR, explained what is ANTLR and showed the parse tree.*

Keywords: *domain-specific language/DSL; ANTLR (ANother Tool for Language Recognition); grammar; production rules; terminal and non-terminal symbols;*

Introduction

Domain-specific languages are becoming more and more important in software engineering due to the fact that they are created to support a particular set of tasks, as they are performed in a specific domain.[2] In other words, DSL is a language built for a certain area and solves a given class of problems. For instance, SQL is a DSL whose main purpose is to be able to manipulate data from the database (insert, extract and modify the data).[1]

The main purpose of a domain-specific language for mathematical expressions is to obtain a correct result of a complicated mathematical expression in a short time and to be more powerful than a calculator. For this, some domains where computing of complex expressions are required were studied. One of them is based on studies in school. To put it differently, pupils sometimes are given to solve exercises and this developed language will be a help for verifying the correct answer of the problems. Another domain is economics where workers manipulate with a big amount of numbers and calculations. By using this DSL, their work will become easier and they will be certain that the answers are correct without having to verify them. The main parts that define a domain-specific language are: grammar, data types, semantic rules, types of assignment, scope rules and how to invoke a method.[3]

1. Grammar of the domain-specific language

Below is represented the set of production rules that states how to generate a program in DSL and what path is followed from start till the end in order that the program to be written correctly. To generate a program, a production rule is applied to an expression by replacing one occurrence of the production rule's left-hand side in the expression by that production rule's right-hand side. The production rules are then applied in any order, until an expression is replaced by a terminal symbol.[6]

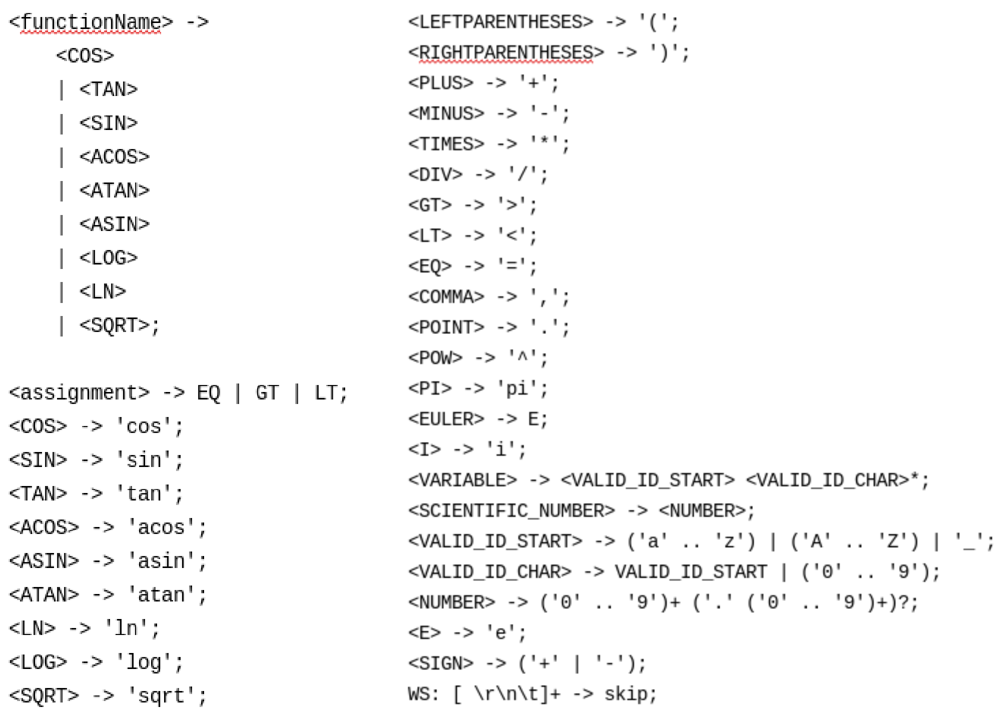
```
Notations: <expression> - non-terminal symbol
           'expression' - terminal symbol
            | - separates alternatives
<equation> -> <expression> <assignment> <expression>;
<expression> -> <multiplyingExpression> ((PLUS | MINUS) <multiplyingExpression>)*;
<multiplyingExpression> ->
   <powExpression> ((TIMES | DIV) <powExpression>)*;
<powExpression> -> <signedAtom> (POW <signedAtom>)*;
<signedAtom> ->
   PLUS <signedAtom>
   | MINUS <signedAtom>
   | <function>
   | <atom>;
```

```
<atom> ->
   <scientific>
   | <variable>
   | <constant>
   | LEFTPARENTHESES <expression> RIGHTPARENTHESES;

<scientific> -> SCIENTIFIC_NUMBER;
<constant> ->  PI | EULER | I;
<variable> -> VARIABLE;
<function> ->
   <functionName> -> LEFTPARENTHESES <expression> (<expression>)* RIGHTPARENTHESES;

<functionName> ->
   <COS>
   | <TAN>
   | <SIN>
   | <ACOS>
   | <ATAN>
   | <ASIN>
   | <LOG>
   | <LN>
   | <SQRT>;

<assignment> -> EQ | GT | LT;
<COS> -> 'cos';
<SIN> -> 'sin';
<TAN> -> 'tan';
<ACOS> -> 'acos';
<ASIN> -> 'asin';
<ATAN> -> 'atan';
```



```
<functionName> ->
    <COS>
    | <TAN>
    | <SIN>
    | <ACOS>
    | <ATAN>
    | <ASIN>
    | <LOG>
    | <LN>
    | <SQRT>;

<assignment> -> EQ | GT | LT;
<COS> -> 'cos';
<SIN> -> 'sin';
<TAN> -> 'tan';
<ACOS> -> 'acos';
<ASIN> -> 'asin';
<ATAN> -> 'atan';
<LN> -> 'ln';
<LOG> -> 'log';
<SQRT> -> 'sqrt';
```

```
<LEFTPARENTHESES> -> '(';
<RIGHTPARENTHESES> -> ')';
<PLUS> -> '+';
<MINUS> -> '-';
<TIMES> -> '*';
<DIV> -> '/';
<GT> -> '>';
<LT> -> '<';
<EQ> -> '=';
<COMMA> -> ',';
<POINT> -> '.';
<POW> -> '^';
<PI> -> 'pi';
<EULER> -> E;
<I> -> 'i';
<VARIABLE> -> <VALID_ID_START> <VALID_ID_CHAR>*;
<SCIENTIFIC_NUMBER> -> <NUMBER>;
<VALID_ID_START> -> ('a' .. 'z') | ('A' .. 'Z') | '_';
<VALID_ID_CHAR> -> VALID_ID_START | ('0' .. '9');
<NUMBER> -> ('0' .. '9')+ ('.' ('0' .. '9')+)?;
<E> -> 'e';
<SIGN> -> ('+' | '-');
WS: [ \r\n\t]+ -> skip;
```

```
<LN> -> 'ln';
<LOG> -> 'log';
```

<SQRT> -> 'sqrt';

2. Semantic Rules

These rules place additional constraints on the the set of valid programs besides the constraints implied by the grammar.[4] A robust compiler will explicitly check each of these rules, will generate an error message describing each violation it is able to find and will highlight it with red.

A program consists of equations. Each equation is constructed from expression assignation and another expression. An expression can be a variable, number or a function. Functions are defined and cannot be added. If a function is called, the function must return a result. If the function has arguments, when calling, there should be the same number of variables as defined. The expression in a return statement must have the same type as the declared result type of the enclosing method definition.

3. Data types

There is only one main type in the DSL - float. It is the only type due to the fact that by mathematical rules, integer numbers are included in the set of real numbers. The program differentiates between integer and float by '.' (dot symbol).

4. Scope Rules

The DSL has very simple scope rules:

- a variable used in future calculations must be declared first;
- variables may be declared everywhere in the program;
- methods cannot be added, only predefined methods are used;

5. Type of assignment

Assignment is permitted only for scalar values, mainly for types declared in the programm. The symbol for the assignment is "=" .

The assignment <expression> = <expression> stores the value from <expression> after assignment into <expression> before it. The both parts of an assignment must have the same data type. Besides, it is possible to assign a method to a variable, it will be the same <expression>, but the DSL will check if it is a variable or one of the functions: addition, subtraction, multiplication, power etc.

6. Method Invocation

Method invocation involves passing argument values to the program, executing it and returning the result. Argument passing is defined in terms of assignment being evaluated from left to right. The program is then executed by executing the statements in sequence. A method that returns a result may be called as part of an expression, in which case the result of the call is the result of evaluating the expression in the return statement when this statement is reached.

Method mainly starts from <equation> -> <expression> <assignment> <expression>.

In order to verify if the grammar was written correctly and the invoked methods show the needed result, ANTLR was used. ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks. From a grammar, ANTLR generates a parser that can build and walk parse trees [5].

Here is an example for evaluating the expression (a - c + a*5 = 32). Figure 1. presents the parse tree in graphical form. <equation> means the start of the program which derives to <expression> <assignment> <expression>. After that, the parser verifies step by step if the expression is a multiplying expression, then goes further verifying if it is a power expression till it recognizes that the expression is a variable. <scientific> means that there is a number in a the program.

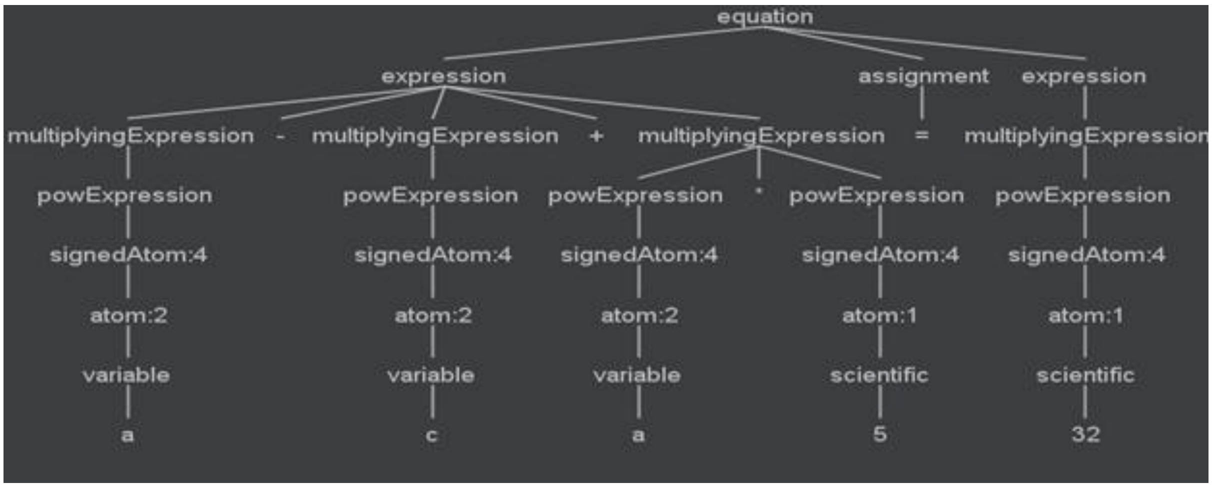


Figure 1. The parse tree of "a - c + a * 5 = 32" expression.

Conclusion

This paper represents the basic ingredients of an approach that uses functional programming as a way of helping students deal with classical mathematics and its applications.

This Domain-Specific language will function like a calculator with a variety of mathematical functions to help process and get the result of complicated expressions.

References

1. https://tomassetti.me/category/language-engineering/domain-specific-languages/
2. https://en.wikipedia.org/wiki/Domain-specific_language
3. http://www.voelter.de/dslbook/markusvoelter-dslengineering-1.0.pdf
4. Course: Formal Languages and Compiler Design
5. https://www.antlr.org/
6. https://en.wikipedia.org/wiki/Formal_grammar