Implementation of Recursive Structural Parser for Symbolic Computation using Mathematical Pseudo Language and Features of Java

K Sudipta Achary¹, Motahar Reza²
School of Computer Science and Engineering
National Institute of Science and Technology
Palur Hills, Berhampur, India, 761008

ABSTRACT

Symbolic Computation is a Computer Algebra System (CAS) which is developed to compute the symbolic mathematical expressions consisting simple or complex calculations. Symbolic Computation provides the interface to provide input expression in a typical mathematical format. Mathematical problems related to algebra, logarithm, roots, sum & product of series etc can be solved using this package.

Use of Object Oriented programming on JAVA platform provides an efficient method to such calculations. GUI components like Panels, Applets, Menus, and Buttons from AWT package are used to make the GUI simple and user friendly. The GUI is made in such a way that it takes input in a typical mathematical format which is helpful for anyone to compute expressions without using complex instructions or commands. This paper represents a strong yet simple way to solve mathematical problems by representing expressions using Symbols and characters with the concept of pseudo language.

Keywords

Computer Algebra System (CAS); Symbolic Computation; Structural Parsing; Recursive Parser; Mathematical Pseudo Language;

1. INTRODUCTION

Java was first developed as a programming language for developing web applications by Sun Microsystems. It provides interpreter for byte code (Java Virtual Machine), a library, an object hierarchy, abstract window tool kit and swing components along with the compiler to build graphical user interface [1]. It also provides programming conventions for event driven inter component operations. It is a cross-platform language so that the applications can be run in any platform or any operating system.

Symbolic computation [2, 3] is a Computer Algebra System which takes the mathematical expression using alphabets, numeric characters and symbolic notations and parses the expression using a recursive parser and finally gives the output. A Mathematical Pseudo Language [4] is used to interpret the symbolic expressions and convert it into a string which is actually given to the parser.

There are some CAS tools which are already developed. They are MAPLE [5], MATHEMATICA [6], MATLAB, Mu PAD etc.

This article discusses Java as symbolic computation developing tool [7]. Java can be an efficient language to implement algebraic algorithms. We discuss the use APIs

which convert the symbolic notation into a string which can be parsed. The architecture of the Symbolic Computation tool is also explained. This tool is designed in such a way that any mathematical module can be integrated easily.

In section 2 the Mathematical Pseudo Language (MPL) is discussed. Section 3 describes the requirements of Symbolic Computation and it also shows how java is useful to develop such type of applications. The required algorithms are explained in section 4. The result is discussed in Section 5 and it is validated by comparing the output with existing CAS (MAPLE and MATHEMATICA).

2. MATHEMATICAL PSEUDO LANGUAGE

Mathematical pseudo language (MPL) is an algorithmic language that is used to describe the concepts and algorithms. MPL algorithms are readily expressed in the programming languages of MAPLE, MATHEMATICA, and Mu-PAD. In this Symbolic computation the MPL algorithm is implemented. It acts as an interpreter of Symbolic notations. It converts the expression in Symbolic Notation to an expression string. A parser can't recognize the symbolic expression. So the pseudo language is introduced to prepare a string which can be parsed.

MPL mathematical expressions are constructed with the following symbols and operators.

- Integers and fractions
- Alphabets which can be used as variables that are used in various functions
- Operators +, -, *, /, ^ (Power), and ! (Factorial)
- Function forms that are used for mathematical functions $\sin(x)$, $\text{indefint}(x^2+\sin(x))dx$, $\log[b](\text{expr})$, $\text{permu}(n, r)$ etc.

3. SYMBOLIC COMPUTATION IN JAVA

3.1 Objects as functions

The view of mathematical objects as functions is powerful. The difficulty is to transmit the data across the components and that can be achieved by using different programming language.

3.2 Storage management

Languages like Lisp, Java and Maple provides automatic garbage collection where C and C++ require explicit garbage collection. The most challenging thing for component technology is to combine the different memory models in a single computation.

3.3 Exceptions

The most familiar way of handling exceptions is try/throw/catch mechanism in Object oriented programming languages. Exception handling must be included between the components.

3.4 Legacy code

It is always recommended to start the computation from the bottom layer because all the functionalities can be custom made. However, such design leads to monolithic systems which must keep pace with new algorithmic advances. There remains the legacy code, often written in a different programming language.

3.5 Algorithmic shortcuts

One of the important paradigms in Symbolic Computation is the evaluation technique. Some libraries can be used for the same. Sometimes the libraries can be modified also. Building Symbolic Computation is better in Java as compared to other programming language like C, C++ etc.

3.6 Generic Programming

The main purpose of generic programming is the ability to write algorithms with variable types. The template feature of C++ is designed for the first instantiation method. Dynamic evaluation is used in Maple's domain package. C++ has virtual class member functions for this purpose, which has been adopted by abstract method in Java.

Originally Java was thought as architecture independent programming language. With the byte code, virtual machine, and the automatic garbage collection its execution resembles that of Maple, Mathematica and Lisp.

3.7 Plug and Play

Java is a downward compatible language that means any program written in C or in other programming languages can be invoked by a Java program via its native method mechanism. The java runtime environment decides if a native code library is available and chooses between a Java and a more efficient native implementation.

There is an alternate way facilitated by Java's networking capability. The java and non-java components can communicate via sockets or pipeline.

3.8 Memory Management

Java provides dynamic memory management facility. Automatic garbage collection provides a convenience to the programmer, but it has some condition. The virtual machine must determine if the object is still in use. C++ can provide garbage collection through constructors and destructors but it need some library package to be imported. However the efficient memory management is required for Symbolic Computation as it deals with large data structure.

3.9 Additional Features

Java pioneered standard object hierarchy and library for building graphical user interface in an operating system independent fashion. It also provides more features like serialization and Big Numbers.

Serialization: This is a feature of converting any object into serial stream, ready for writing in file or transmitting over a network. In other object oriented language the programmer has to mention explicitly for serialization. But Java can control the objects by using default serialization by JVM.

Big Numbers: In Java a large value can be dealt by using variables. This is beneficial in cryptography application. It can also be used to solve large mathematical expression.

4. ALGORITHM

Symbolic computation takes the input in two ways. First is from the user directly and takes the special symbols i.e. (Summation, Integration, Theta, exponential etc) from the symbol panel. Three basic swing components are used to represent the input.

- JTextField
- JLabel
- JPanel

The Symbolic Computation package is divided into five basic modules.

- Add Symbol
- Listener
- Extract Input
- Parser
- Extract Output

4.1 Add Symbol

The special symbols as mentioned earlier are added to input in a standard mathematical expression. And the special symbols are grouped with related inputs. i.e. (The integration symbol is grouped with its upper limit and lower limit).

Add Symbol Algorithm

- Take the coordinate pointer to add symbols
- Set the dimension of symbols
- Add the symbols to input panel
- Update the panel
- Increment the coordinate pointer

4.2 Listener

Listener is used to handle the events generated either by Key Board or Symbol Palette. Two types of Listeners are there.

- Symbol Listener
- Constant Listener

Symbol Listener handles the events generated by Symbol Palette. And Constant Listener handles the events generated by Key Board. Both the listener uses the Java APIs.

Figure 1 Constant Listener

Figure 2 Symbol Listener

Constant Listener Algorithm

- Read key from keyboard
- If it is valid key
 - If it is valid char
 - If it is '^'
 - Increment the Y axis of the coordinate pointer
 - If it is '/'
 - Decrement the Y axis of the coordinate pointer and draw '—' symbol
 - else add Panel
 - if ASCII value is 10 then solve the expression
 - if ASCII value is 8 then remove the newly added character

4.3 Extract Input

Extract Input is used before parsing. It scans the input from the standard mathematical notations and converts the notations to a string which is to be parsed by the Parser.

Figure 3 Symbolic Input and Expression String

sum[x,1,100](x^2+x+1)+nroot[3](125)+log[2](10)+permu[5,4]

Extract Input Algorithm

- Scan the component
- Set the input string as 'null'
- If it is text field then add the content of that text field into input string
- If it is label then update the input by adding the Label value
- If it is panel then check for special symbol
- If it contains Σ
 - Add sum[delim, upper_limit, lower_limit] to input
- If it contains P
 - Add permu[n,r] to input
- If it contains C
 - Add combi[n,r] to input
- If it contains $\int$
 - Add inte[upper_limit, lower_limit] to input string
- If it contains $n\sqrt{}$
 - Add root[n] to input string
- If it contains logbx
 - Add log[b](x) to input string

4.4 Parser

A parser takes the input expression and tokenizes the input into different segments, evaluates them finally gives the result. Here the structural parser scans the mathematical expression represented in symbolic notations and converted it into a string which can be accepted by the parser. Then the expression is tokenized and each token is passed to the respective parser module.

Each parser module expects a double value. If the input is of type double then it evaluates the corresponding value and returns that value. If it finds an expression then it calls the main parser recursively and if it gets the double value then evaluates it otherwise it sends back the string as constant expression.

Example

If the input is sin(1+2+3+x) then the main parser calls parse sine module. Again the input is not a double value so the expression 1+2+3+x will be given to the main parser recursively and the output is 6+x. Again the output of the expression is not a double value so it returns sin(6+x) as final answer.

Parser Algorithm

- Extract the input from the Panel or take a String
- Tokenize the input string and store in a array input []
- For i -> 0 to length of input
 - If input[i] is a valid character
 - Add to output
 - If input[i] is an expression
 - Solve using expression tree and add to output
 - If input [i] starts with integration
 - Call parseIntegration Module
 - If input [i] starts with differentiation
 - Use parsedifferentiation method
 - If input[i] starts with "sum"
 - Call parseSum method and add the result to output
 - If input[i] starts with "permu"
 - Call parsePermutation method and add the result to output
 - If input[i] is any trigonometric function
 - Call respective parser and update the output
 - If input[i] is of logarithm type
 - Call the logarithm parser
 - Else accept the constants and expressions and update the output
- Separate the numeric and non numeric indexes of output and store in different arrays
- Calculate the final value from array containing numeric values

- Concatenate the elements from non-numeric array
- Finally merge the numeric final value and the constant string and show the result

The parser is called as recursive structural parser. It scans the expression in symbolic structure (Mathematical notations) and converts it into an expression string which can be understood by any program (parser). And it is recursive in nature. As mentioned earlier in some functions if the input is not numeric then the parser assumes that the input is an expression and calls the main parser and evaluates the expression recursively.

4.5 Extract Output

It takes the result which comes after processing the input by parser and represents the final output using Symbolic notations.

Sample Output:

$$x^3 / 3 + x^2 / 2 + x * (1) + K$$

Figure 4 Output using Symbolic Notation

5. RESULT VALIDATION & DISCUSSION

This computer algebra system is developed purely in Java language. The package uses Swing components, Event handler, and some classes from java.lang and java.util package. It provides object view for each mathematical function. As java provides automatic garbage collection and big numbers this algebra system can handle large expressions and huge value.

This package was developed by keeping existing algebra systems like MAPLE, Mathematica, Mat-Lab etc. in mind.

It takes the Symbolic expression and converts into a string. The string is tokenized and each token is given to respective object. After processing each token the result is combined and improved. And finally the result is shown using symbolic notations.

It also saves the input expression in its own format (.sc) like (.mw) in Maple and (.nb) in Mathematica.

It has been mentioned that it was developed by keeping existing CAS in mind. So this CAS is compared with some existing CAS.

In this paper the input is verified with Maple and Mathematica with various test cases. The following test cases show that the outputs in all three CAS are equivalent.

Input 1

Output in Symbolic Computation CAS

$$25.34104863489161 + x^3 / 3 + a / \ln(a) + K + \sin(15 + x) + \sin(a) + K$$

Figure 5 Output in CAS

Output in Maple:

$$\frac{1}{3}x^3 + \frac{a^x}{\ln(a)} + 625^{(1/4)} + 13 + \sin(15 + x) + e^2 + \sin(a)$$

Figure 6 Output in Maple 10

Output in Mathematica:

$$25.3891 + a \cos a + \frac{x^3}{3} + \frac{a^x}{\log[a]} + \sin[15 + x]$$

Figure 7 Output in Mathematica 6.0

Input 2

$$\sin\left(\sum_{x=1}^{10} (x^2 + x + 1)\right) + \int_3^9 (x^2 + x + y) dx + \log\left(\log\left(\log\left(\frac{1}{10}\right)\right)\right) + \frac{5}{3}P + \frac{d}{dy}(y^2 + y + c)$$

Figure 8 Test Case Input

Output 2 in Symbolic Computation CAS

$$331.10 + 6\pi(y) + 1.10 + 2.10\pi y$$

Figure 9 Output in CAS

6. CONCLUSION

There is no end to such type of calculus problems. To build software that exhaustively covers all kind of calculus related problems would be a task next to impossible. That is why in this paper a set of classes are selectively picked up and made application to solve problems belonging to those classes.

Figure 10 Complete Operation of CAS

This application was developed keeping in mind the other software like Lingo, Maple, Mat-Lab, Mathematica etc and features of Java by which the functionalities can be achieved. Since the paper named above require extensive knowledge of the subject and learning of certain languages, hence we tried our best. The application however can be extended in order to make it efficient by toning down its properties adding new functionalities and wide range of inputs.

7. REFERENCES

- [1] H. Schildt, Complete Reference Java – McGraw Hill
- [2] Abdali, S. K., Cherry, G. W., and Soiffer, N. A Proc. 1986 Symp. Symbolic Algebraic Comput. Symsac'86 (New York, N. Y., 1986), B. W. Char, Ed., ACM pp. 24-30
- [3] Díaz, A., Hitz, M., Kaltofen, E., Lobo, A., and Valente, T. Process scheduling in DSC and the large sparse linear systems challenge. J. Symbolic Comput. 19, 1-3 (1995), 269-282
- [4] Cohen, Joel S. (2003). Computer Algebra and Symbolic Computation: Mathematical Methods. AK Peters, Ltd.
- [5] Maple 10, Maple Inc. <http://www.maplesoft.com>
- [6] Mathematica 6.0, Wolfram Research www.wolfram.com/Mathematica
- [7] Symbolic Computation in Java :an Appraisalment, Laurent Bernardin, Bruce Char, Erich Kaltofen
- [8] Díaz, A., and Kaltofen, E. FoxBox a system for manipulating symbolic objects in black box representation. In Gloor [22], pp. 30-37
- [9] Mac Tech | The journal of apple Technology <http://www.mactech.com/articles/mactech/Vol.07/07.05/MathParser/index.html>
- [10] Iglio, P., and Attardi, G. Software components for computer algebra. In Gloor [22], pp. 62-69
- [11] Kaltofen, E. Challenges of symbolic computation my favorite open problems. J. Symbolic Comput. 29, 6 (2000), 891-919. With an additional open problem by R. M. Corless and D. J. JeFFrey
- [12] Lakshman Y. N., Char, B., and Johnson, J. Software components using symbolic computation for problem solving environments. In Gloor [22], pp. 46-53.
- [13] Pavlović, M.N., Symbolic computation in structural engineering, Computers and Structures, Volume 81, Number 22, September 2003, pp. 2121-2136(16)
- [14] Bernardin, L. A Java framework for massively distributed symbolic computing. Mathematics and Computers in Simulation (1999)
- [15] Howard, J.C., Practical applications of symbolic computation, IPC Science and Technology Press, UK, 1980
- [16] EUROSAM 84: International symposium on symbolic and algebraic computation: J Fitch (ed) Springer-Verlag (1984) ppp 396.