www.ijaret.org Volume 1, Issue V, June 2013

INTERNATIONAL JOURNAL FOR ADVANCE RESEARCH IN ENGINEERING AND TECHNOLOGY

WINGS TO YOUR THOUGHTS…..

An Approach for Syntax Tree Construction using Formal Specification Language

Kamleshwari Verma[1], Toran Verma[2]

[1]Research Scholar, RCET Bhilai, Dist-Durg(C.G.), India
kamleshwarikv@gmail.com

[2]Reader(CSE), RCET Bhilai, Dist-Durg(C.G.), India
vermatoran24@gmail.com

Abstract—*A Compiler is a language translator which translates high level language to low level language, this is the task of a compiler. Removing the bugs of source program does not assure that the generated target code is completely error free because there may be bugs in compiler also. So it is more important to verify the Compiler itself. Lexical analyzer is a first phase and main part of compiler used for scanning input stream of characters and grouping into tokens. In this paper formal construction of syntax tree is described from the regular expression to verify the lexical analyzer and this is the aim of paper. If the syntax tree will be constructed successfully for the given regular expression then the lexical analyzer will be verified. Formal specifications can be rigorously validated and verified leading to the early detection of specification errors. The widely used formal specification language is VDM(Vienna Development Method), Z notation and B. Here the specification is described using Model oriented formal specification languages Vienna Development Method (VDM) because Vienna Development Method provide support for concurrency control and also using VDM Software requirement specification can be automatically converted into computer source code. Z notation does not support for concurrency control and also software requirement specification can not be automatically converted into computer source code. B method also does not support for concurrency control.*

Keywords—*Compiler Verification, Formal Specification, Lexical Analyzer, Regular Expression, VDM.*

1. INTRODUCTION

Compiler is a combination of 6 phases. The first phase of a compiler is Lexical Analyzer. It is very important phase for Compiler and just because of these verifying the Lexical analyzer. If we verify the compiler which compiles the program then there is a chance of bugs and error in the target code will be very less. Now a day, compiler construction is considered as an advanced research area due to the size and complexity of the code generated. It is believed that design and construction of a verified compiler will remain a challenge of twenty first century. Although there exists much work in this area but it needs further investigation because the bugs in the compiler can lead to an incorrect machine code even the source program is verified to be correct. Formal methods are mathematical-based techniques used for specification, analysis, proving and verification of software and hardware systems [1].
The process of formal verification means applying formal techniques to verify the properties of a system. Formal verification of software targets the source program in which the semantics of a language gives the precise meanings to the program to be analyzed. On the other hand, program verification does not provide any guarantee that the executable machine code is correct as described by the semantics of the source program. This is because compiler may lead to an incorrect target program because of bugs in the compiler itself and can invalidate the guarantees ensured by the formal techniques. It concludes that verification of a compiler is much more important than verification of a source program to be compiled.
In this paper, formal construction of syntax tree is described directly from the regular expression to verify the lexical analyzer. Lexical analyzer is an important part of compiler which scans input stream of characters making groups into tokens. Tokens are sequences of characters having meanings in collective format. Few preliminary results of this research were presented in [2] by formalizing some important concepts of context-free grammar useful for parsing analysis. In this paper, regular expression (input program) is described by defining all of its possible symbols and operators. Relationship among the components of expression is specified to prove its well-defined-ness. The regular expression is augmented by joining a special symbol at the end of the program. An abstract syntax tree is defined based on the regular expression including its internal and terminal nodes. So here formal specification of the algorithm is described using Vienna Development Method.

2. LITERATURE REVIEW

Various methods have been cited in the literature for improving the accuracy and efficiency of compiler for example: Nazir Ahmad Zafar work on the field of Compiler verification in 2012 and on the basis of Z Notation Specification formal Construction of syntax tree is done for given regular expression[3]. Fawaz Alsaade and Nazir Ahmad Zafar (2012) give the concept of DFA (Deterministic Finite Automata) formal construction on the basis of regular expression.The aim of that paper was also verification of compiler[4]. Michael Bebenita, Florian Brandner and Manuel Fahndrich (2010) improves the runtime performance of JavaScript Program on .NET plateform where a trace based JIT(Just In Time) compiler is made for CIL(Common Intermediate Language)[5]. Gr. Thurmair, V. Aleksić and Chr. Schwarz (2009) give the concept of Large-scale lexical analysis where accuracy of around 98%, for large scale data is achieved[6]. Danny Dub´e and Anass Kadiri (2006) give the concept of Automatic construction of parse trees for lexemes where DFA is constructed for lexeme[7].

3. LEXICAL ANALYZER

Lexical analyzer is an important part of compiler which scans input stream of characters making groups into tokens. Tokens are sequences of characters having meanings in collective format. Few preliminary results of this research were presented in [2] by formalizing some important concepts of context-free grammar useful for parsing analysis.

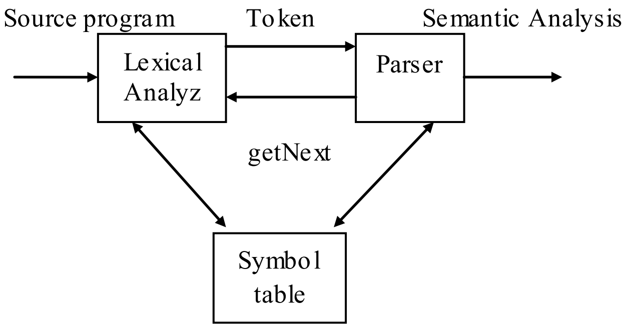

Figure 1 : Lexical Analyzer

Compiler verification is a branch of software engineering which deals to prove that compiler behaves exactly as its language specification. Testing and formal methods are two most common techniques for validation and verification in development of a compiler. Compiler testing has various disadvantages similar to other computer programs testing. For example, it is hard to prove that compiler is completely error-free and optimized. The primary objective of writing a compiler is to prove that it is correct and error-free. There exists much research work referring that many tested compilers have bugs and errors in the code. [8]
There are two primary methods for implementing the lexical analyzer. The first method is a hard coded program to perform the scanning tasks and the second method uses regular expression and automata theory to model the scanning process. In the first method a main loop in the program reads characters one by one from the input program and uses a switch statement to process it. The output of the procedure is a sequence of tokens from the source program.

In the second method, the source program is read character by character beginning with the start state. After reading each character, the transition function is used to move from current state to the next state. If the final state is reached, it is checked if the token read is reserved word it is passed to the token stream as output. If it is not a reserved word, its name is put in the symbol table if does not exit already. Once a final state is reached an associated action is performed and the same process is continued. If we are not able to reach a final state an error is encountered and error handling routine is called upon. In this method, input is a program which is a regular expression and output is a collection of tokens identified by the finite automata. In this paper a part of verification of lexical analyzer to construct syntax tree from regular expression is described.
There are various other applications of automata theory in addition to compilers construction and verification. Software engineering and maintenance, pattern identification, robotics and speech recognition are some examples of it [9]. In software engineering, test cases can be generated if the system is described by models using automata theory [10].

4. PROPOSED METHODOLOGY

4.1 Vienna Development Method:

Formal methods are mathematically based techniques that can be applied throughout the development of a system to precisely describe a system and involve the use of refinement techniques and proof obligation at each stage to ensure the correctness, completeness and consistency of specification. The formal specification languages are based on set theory and first order predicate calculus, but this mathematical background was initially not fully formalized.[11]The representation used in formal methods is called a formal specification language. The language is "formal" in the sense that it has a formal semantics and as a result can be used to express specifications in a clear and unambiguous manner. A formal specification language can be used to specify the task at hand in a clear and concise manner. As formal methods and formal specification language has sound mathematical basis, it provides the means of proving that specification is realizable, complete, consistent and unambiguous. Even the most complex systems can be modeled using relatively simple mathematical objects, such as sets, relations and functions.[11]
A formal specification language is usually composed of three primary components or in mathematical term we can say that it consists of two sets syntax and semantics and a set of relation.[11] One of the longest-established Formal Methods for the development of computer-based systems is termed as Vienna Development Method (VDM).It was originated at IBM's Vienna Laboratory [12] in the 1970s.The Z, B and VDM are model based languages, which usually model a system by representing its state as a collection of state variables, their values and some operations that can change its state. All are based on set

theory and mathematical logic. VDM was used in programming language description and compiler design. The main goal was to develop the language's fundamental features and to establish some formal semantics. Z notation is a strongly typed [13] mathematical, specification language. But Z notation not support for concurrency control[14].VDM provide support for concurrency control using VDM++[14]. Writing concurrent programs is a challenging task. While it is well known that shared resources must be protected from concurrent accesses to avoid data corruption, guarding individual resources is often not sufficient. Sets of semantically related actions may need to execute in mutual exclusion to avoid semantic inconsistencies.While databases have native support for such 'transactional' constructs, most concurrent programming languages lack adequate mechanisms to handle this task[15].

4.2 Specification:

Regular Expression

Regular expressions are an important notation for specifying lexeme patterns. The formal specification of a regular expression is assumed using four variables.

a) symbol :

symbol is a collection of all alphabets and operators. It represents internal nodes in the syntax tree.

b)terminals : terminal is a finite set of alphabets representing children in the syntax tree.

c)operators : operators having values concatenation, alternation and repetition.

d)Re : The fourth one component is regular expression representing *re* and is a sequence of alphabets and operators.

The *Symbol*, *Terminal* and *Operator* are sets at an abstract level of specification over which operators, for example, union, intersection and complement cannot be defined.

[*Symbol*]; *Terminal*== *Symbol; Operator*== *Symbol*

Formal definition of regular expression is given below using following schema :

Operators: *,+,or

Symbols: lp (left parenthesis)

rp (left parenthesis)

Res : res is a power set of schema RE.

ExtendedRE : supposed that the hash symbol does not of the regular expression.

In the schema ExtendedRE, a special symbol # is joined at the end of the input string to produce augmented string.It is supposed that the hash symbol does not exist in the set of symbols of the regular expression.

```
If re ≠ ϵ
Then
        Left symbol ≠ */or
        First element ≠ )
        Right most symbol ≠ or/lp
If re ≥ 1
Then excluding first element & for any element
If *
  Then left is terminal/rp
If re ≥ 2
Then for any element excluding first and last element
If it is or
Then
        Left ≠ or OR lp
        Right ≠ or of rp
If re ≥ 2
Then for any element excluding last element
If it is lp
Then
        Right element ≠ rp,*,or
If re ≥ 2
Then for any element excluding first element
If it is rp
Then
        left element ≠ lp OR or
```

Syntax Tree:

For constructing syntax tree 3 important function :

a) Nullable
b) First position
c) Last position

The internal node of a tree consists of 6 variables.

I) node
II) left
III) right
IV) firstpos
V) rightpos
VI) nullable

Relation between Regular Expression and Syntax Tree is:

- Leaf is either a terminal or null string.
- Each internal node has two well defined children. One of these might be null but both can not be null string.
- The set of identifiers(leafs) of the tree is same as the set of terminals of the language.
- The set of identifiers of parent is based on its children.

Operator Specification :

Three types of nodes are assumed:

Alternation

Concatenation

Repetition

Node Type := OR/CON/STAR

Nullable consists of 4 components:

I) node itself
II) left child
III) right child
IV) node type

i). Nullable

- If node= Alternation
  Then
  It is nullable if and only if one of its children is nullable.
- If node= Concatenation
  Then
  It is nullable if and only if both of its children are nullable.
- If node= Repetation
  Then
  It is nullable if and only if its child is nullable.

ii). Left Position

- If node Type= Alternation
  Then
  Firstpos=firstpos of(left child union right child)
- If node Type= Concatenation,
  If its left child is nullable
  Then
  Firstpos=firstpos of(left child union right child)
  If its left child is not null
  Then
  Firstpos=firstpos of left child
- If node Type= Repetation
  Then
  Firstpos=left position of its child

iii). Right Position

- If node Type= Alternation
  Then
  Lastpos=lastpos of(left child union right child)
- If node Type= Concatenation and right child=null
  Then
  Lastpos=lastpos of(left child union right child)
  If right child=not null
  Then
  Lastpos= lastpos of right child
- If node Type= Repetation
  Then
  Lastpos=last position of its child

4.3 Rules for Syntax Tree Construction:

The techniques which I have implemented and analyzed for syntax tree construction are as follows:

--- Steps for Syntax Tree Construction:

1. Regular Expression is given as a input to the System.
2. Make the augmented Regular Expression.(By adding # symbol at the end of the input string)
3. Syntax Tree construction is done using 3 function.
   a) Nullable
   b) First Position
   c) Last Position
4. Relation between Regular Expression and Syntax Tree is given using following Rule :
   a) Leaf is either a terminal or null string.
   b) Each internal node has two well defined children. One of these might be null but both can not be null string.
   c) The set of identifiers of the tree is same as the set of terminals of the language.
   d) The set of identifier of parent is based on its children.
5. Specification of operator : Three types of nodes are assumed that is
   Node type:
   Alternation(OR)/Concatenation(CON)/Repetation/ (STAR).

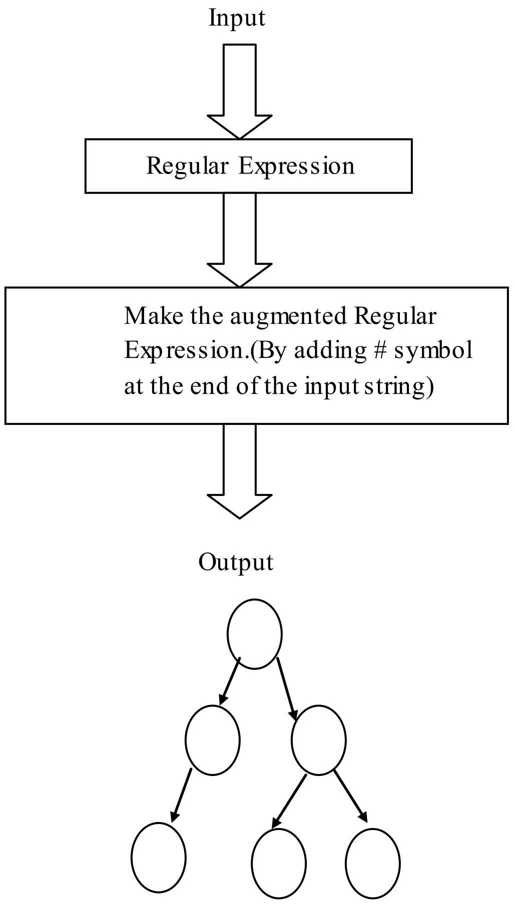
Input
Regular Expression
Make the augmented Regular Expression.(By adding # symbol at the end of the input string)
Output

Figure 2 : Step for syntax tree construction

5. CONCLUSIONS AND FUTURE WORK

There is no need to verify all the phases of compiler. If we just verify the lexical analyzer then the chances of bugs and error in the targeted output is obviously very less. So for this syntax tree will be constructed for given Regular Expression and these verification is done using the formal specification language VDM. As compare to other formal specification language VDM provide concurrency control that is a greatest advantage. One more advantage of using VDM is that software requirement specification can be automatically converted into Computer source code. So here syntax tree is constructed successfully for the verification of lexical analyzer using VDM.

REFERENCES

[1] C. J. Burgess, "The Role of Formal Methods in Software Engineering Education and Industry," Technical Report, University of Bristol, UK, 1995.

[2] K. A. Buragga, and N. A. Zafar, "Formal Parsing Analysis of Context-Free Grammar using Left Most Derivations, ICSEA, 2011.

[3] Nazir Ahmad zafar"Automatic construction of Formal Syntax tree Based On Regular Expression" Proceedings of the World Congress on Engineering 2012 Vol II WCE 2012, July 4 - 6, 2012, London, U.K.

[4] Fawaz Alsaade,,Nazir Ahmad Zafar" Syntax-Tree Regular Expression Based DFA Formal Construction" July-2012 /*Intelligent Information Managementv*.

[5] Michael Bebenita, Florian Brandner, Manuel ahndrich "SPUR: A Trace-Based JIT Compiler for CIL" 2010 MSR-TR

[6] Gr. Thurmair, V. Aleksić, Chr. Schwarz "Large-scale lexical analysis " 2009/ PANACEA.

[7] Danny Dube and Anass Kadiri "Automatic construction of parse trees for lexemes" 2006/ National Science and Engineering Research Council of Canada.

[8] H. Beek, A. Fantechi, S. Gnesi, and F. Mazzanti, "State/Event-Based Software Model Checking," Integrated Formal Methods, Springer, vol. 2999, pp. 128-147, 2004.

[9] J. A. Anderson, "Automata Theory with Modern Applications," Cambridge University Press, 2006.

[10] M. v. d. Brand, A. Sellink, and C. Verhoef, "Generation of Components for Software Renovation Factories from Context-Free Grammars," CRE, pp. 144-153, 2001.

[11] R. Pressman, "Software Engineering- A Practitioner's Approach", McGraw Hill, 5th edition. 2000.

[12] D. Andrews. Report From The BSI Panel For The Stadardisatio Of VDM (ist/5/50). In VDM '88 VDM | The Way Ahead. Springer Berlin/Heidelberg, 1988.

[13] J.M. Spivey, "The Z Notation,Reference Manual", 2nd edition, Prentice Hall International, 1992.

[14] Dr.(Mrs.) Arvinder Kaur, Samridhi Gulati, Sarita Singh" Analysis of Three Formal Methods-Z, B and VDM" International Journal of Engineering Research & Technology (IJERT) Vol. 1 Issue 4, June - 2012 ISSN: 2278-0181

[15] Pascal Felber1 and Michael K. Reiter "Advanced concurrency control in Java" 2002 John Wiley & Sons, Ltd.