# Lexical Analysis Implementation by Using Deterministic Finite Automata (DFA)

## تنفيذ التحليل المعجمي باستخدام الأوتوماتا المحدودة الحتمية (DFA)

**MA:** Hassan .k. Mohamed. Assistant Lecturer, Computer Department. College of Arts and Sciences Sluk. Benghazi University.
**EmaillL**: Hassn.efakhari@uob.edu.ly
**MA**: Fathia . A. A. Albadri. Lecturer at the Faculty of Arts and Sciences. Benghazi University.
**Email**:Fathia.Elbadre@uob.edu.ly
**MA**: Raja . A. Mohamed. Lecturer at the Faculty of Information Technology. Benghazi University.
**Email**: Raja.moftah@uob.edu.ly

أ. حسن خليفة محمد. محاضر مساعد بقسم الحاسوب. كلية الآداب والعلوم سلوق. جامعة بنغازي.
أ. فتحية عبدالله الفضيل البدري. محاضر بكلية الآداب والعلوم الأبيار. جامعة بنغازي.
أ. رجاء عبدالعاطي محمد. محاضر بكلية تقنية المعلومات . جامعة بنغازي.

| تاريخ استلام البحث | تاريخ قبول البحث | تاريخ نشر البحث |
|---|---|---|
| 2021 /9 /22 | 2021 / 10 / 10 | 2021 / 11 / 7 |

**Abstract:** A compiler is a computer program (or set of programs) that transforms source code written in a programming language into another computer language known as the target language, often having a binary form identified as object code. The compiler has some phases and the main concern of this report is about the implementation of Lexical Analysis using the Deterministic Finite Automata (DFA) that is a finite state machine that accepts or rejects finite strings of symbols and only produces a unique computation of the automaton for each input string.

Through building a Finite Automata using a model known as JFLAP and utilizing it to execute some patterns, which were integer number, real number, operators, and some keywords. This model approved that DFA is accurate to produce a unique computation for each input string. In addition, using JFLAP approved that the time-consuming.

**Keywords**: Automata, Lexical, Deterministic Finite Automata, Token, Lexical Analyzer.

**الملخص:** المترجم هو برنامج كمبيوتر (أو مجموعة من البرامج) يقوم بتحويل الشفرة المصدر المكتوبة بلغة برمجة إلى لغة كمبيوتر أخرى تُعرف باسم اللغة الهدف، وغالبًا ما يكون لها نموذج ثنائي يتم تحديده على أنه شفرة الهدف. يحتوي المترجم على بعض المراحل ويكون الشاغل الرئيسي لهذا التقرير حول تنفيذ التحليل المعجمي باستخدام الأوتوماتيكية المحدودة المحددة (DFA)وهي عبارة عن آلة ذات حالة محدودة تقبل أو ترفض سلاسل محدودة من الرموز وتنتج فقط حسابًا فريدًا للأتمتة لكل سلسلة إدخال.

من خلال بناء آلية محدودة باستخدام نموذج يعرف باسم JFLAP واستخدامه لتنفيذ بعض الأنماط، والتي كانت عبارة عن عدد صحيح، وعدد حقيقي، وعوامل تشغيل، وبعض الكلمات الرئيسية. وافق هذا النموذج على أن DFA دقيق لإنتاج حساب فريد لكل سلسلة إدخال. بالإضافة إلى ذلك، باستخدام JFLAP وافق على أن يستغرق وقتا طويلا.

**الكلمات داله:** الأوتوماتا، معجمي، الأوتوماتا المحددة المحدودة، الرمز المعجمي، محلل معجمي.

### 1. Introduction

The computer has some stages, which have to be done to convert the program to the machine code, which is understandable, by the computer. The machine code is faster than the other programming languages, but it could be

challenging to write a program using it; the compiler does this job to change the program from any programming language to machine code. The stages involved in this action are Lexical Analysis, Syntax Analyzer, Semantic Analyzer, Intermediate Code Generator, Code Optimizer, Code Generator, and Out Target Program. The output of each phase is the input of the next stage. The first stage is Lexical Analysis, which breaks up a program into tokens, and then sends these tokens to the next stage. The stage can be modeled using a Finite State Machine. This study will use a model program to generate and simulate automata because using pen and paper could be difficult, error-prone, and consuming of time. In more detail, it will be discussed in the following sections [1].

## 2. Lexical Analysis

Lexical Analysis takes a stream of characters and generates a stream of tokens (names, keywords, punctuation, etc.). This is a key task is to remove all the white spaces and comments. This will make parsing much easier. A lexical token is a sequence of characters that can be treated as a single unit. Some tokens have a value, this is called a lexeme. A scanner is the piece of code that performs the lexical analysis. If any incorrect input is provided by the programmer, Lexical Analysis correlates the error with the source file and line number. A key issue for scanners is speed. Scanners can be automatically generated using a combination of regular expression and finite automata [2][3].

### 2.1 Terminology used in Lexical Analysis

1. Token:
   A set of input strings that are related through a similar pattern.
2. Lexeme:
   The actual input string, which represents the Token.
3. Pattern
   The rule, which a Lexical Analysis follow to create a Token.

### 2.2 Types of Token

Each keyword is a token
Each punctuation symbol is a token
Each operator is a token
An identifier is a token (and we are interested in its value)
An integer literal is a token (and we are interested in its value)
Spaces, new lines, tabs and comments are ignored.

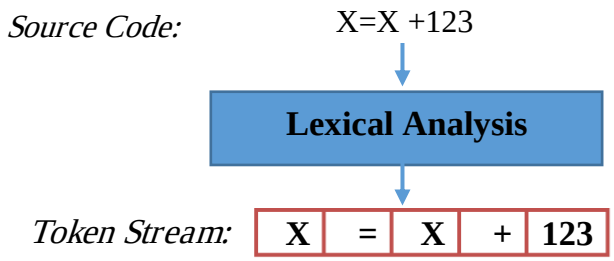



*Fig1: Example of Lexical Analysis*

### 2.3 Lexical Analysis flow of events

Syntax Analyzer in the compiler serve as the master program, so it first send a request to get a valid token from the lexical analyzer.
Scanner (Lexical Analyzer) do its pattern matching to create a valid token if possible and sends back to the Syntax Analyzer and get suspended.
Syntax Analyzer do the grammar check over the Token and asks for next Token.
This process continues recursively till input string is consumed

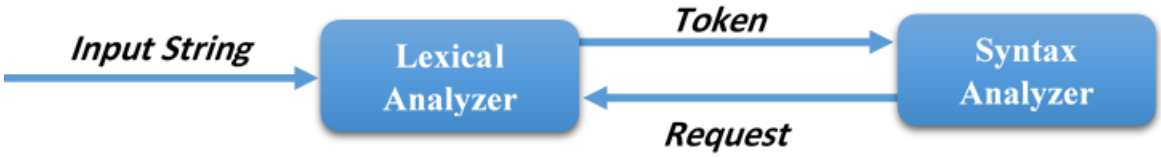



*Fig2: Work of Lexical Analysis*

### 2.4 Building a scanner (Lexical Analyzer)

Specify the valid tokens of a given language in terms of Regular Excretion.
Create a NFA (None Deterministic Finite Automata) from the previous Regular Expression.
Convert from the previous NFA to a DFA (Deterministic Finite Automata) and try to minimize it if possible.
Translate a DFA to program with any of programming languages.

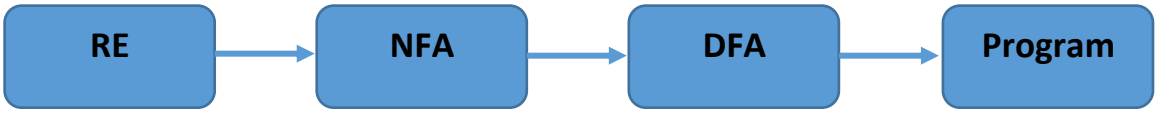



*Fig3: Stages of building Lexical Analysis*

These steps could be time consuming, so there are some software, which convert the Regular Expiration automatically to program in different languages like C or Java. Flex Scanner Generator is one of these software

The input to the flex program (known as flex compiler) is a set of patterns or specification of the tokens of the source language. Actions corresponding to different matches can also be specified.
The output from the flex software compiles the regular expression specification to a DFA and implements it as a C program with the action codes properly embedded in it [4].

## 3. Model of Some Patterns

This section is about some pattern that will be used to determine the Token by the Lexical Analysis, this model will be use a DFA since it could recognize a specific rule over input string. The pattern first construct as a Regular Expiration, so let define the basic RE pattern.

### 3.1 The Following Abbreviations are generally used:

[axby] means (a|x|b|y)
[a − e] means [abcde]
M? means (M | ε)
M+ means (MM*)
. Means any single character except a newline character
”a+*” is a quotation and the string in quotes literally stands for itself.

### 3.2 Disambiguation Rules for Scanners

Is do99 and identifier or a keyword (do) followed by a number (99)? Most modern lexical-analyzer generators follow 2 rules to disambiguate situations like above.
• ***Longest match:*** The longest initial substring that can match any regular expression is taken as the next token.
• ***Rule priority:*** In the case where the longest initial substring is matched by multiple regular expressions, the first regular expression that matches determines the token type. So do99 is an identifier. See table1 below.

| ***Regular Expiration*** | ***Represent*** |
|---|---|
| **If** | if(Key Word) |
| **While** | while(Key Word) |

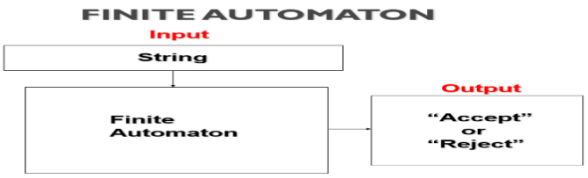
| | |
|---|---|
| + | Operator (Addition) |
| / | Operator (Subtraction) |
| **[a − zA − Z][a − zA − Z0 − 9]*** | Identifier |
| **[0 − 9]+** | Integer number |
| **([0 − 9] +"." [0 − 9]*) \| ([0 − 9] *"."[0 − 9]+)** | Real number |
| **"//"[a − zA − Z0 − 9]* "\n" \| (""\|"\n"\|"\t") +** | comment or white space |

*Table 1. Regular expression abbreviations*

### 3.3 Finite Automata

Regular expressions are good for specifying lexical tokens. Finite automata are good for recognizing regular expressions. A finite automata consists of a set of nodes and edges. Edges go from one node to another node and are labelled with a symbol. Nodes represent states. One of the nodes represents the start node and some of the nodes are final states. A deterministic finite automaton (DFA) is a finite automaton in which no pairs of edges leading away from a node are labelled with the same symbol. A nondeterministic finite automaton (NFA) is a finite automaton in which two or more edges leading away from a node are labelled with the same symbol. Consequently, DFA is a single state after reading any sequence of inputs. The number of states of the DFA can be exponential in the number of states of the NFA. Where NFA has the facility to be in several states at once. As mentioned in section 2. Each state of DFA has always one exiting transition arrow for every symbol in the alphabet. In other words, Labels on the transition arrows are symbols from the alphabet. DFA works could be summarized in Fig1. Below. When input string and finite automata model are applied, the output has accepted or rejected.

Fig 1. Show Finite Automation process.




### 3.4 DFA Model

The patterns, which are used as an Integer number, Real number, Operators, and some keywords.

**DFA D=** ***(Q, ∑, δ, q0, F)***

*Q = {q0, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, q13, q14, q15, q16, q17, q18, q19, q20, q21, q22, q23}*

*∑ = {[0-9], [a-z], [A-Z], +, -, /, *, =, ==, w ,h ,i ,l ,e ,n ,t ,f ,m ,a , .}*

$\delta = Q \times \Sigma \rightarrow Q$

$q0 \in Q = \{q0\}$

$F \subseteq Q = \{q1, q3, q4, q9, q11, q13, q14, q15, q16, q17, q18, q19, q23\}$

Where JFLAP program create and simulate automata. Additionally, it is easy along with it supports creation of DFA and NFA, Regular Expressions, PDA, Turing Machines, Grammars and more. See Fig2. That is clear transition diagram. Along with table2. Transition Table.

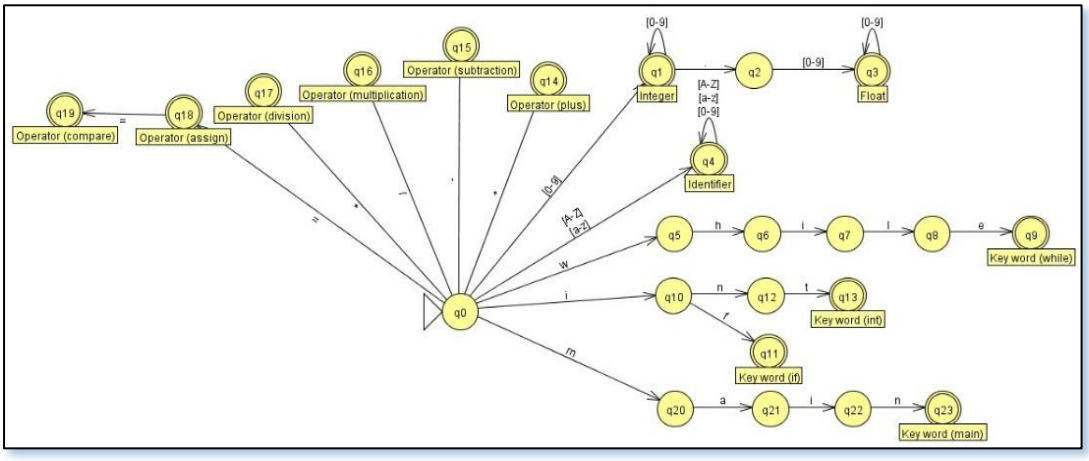



Fig2: Transition Diagram

| | [0-9] | [a-z] | [A-Z] | + | - | / | * | = | == | w | h | i | l | e | n | t | f | m | a | . |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| >q0 | q1 | q4 | q4 | q14 | q15 | q16 | q17 | q18 | q19 | q5 | - | q10 | - | - | - | - | - | q20 | - | - |
| q1* | q1 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | q2 |
| q2 | q3 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q3* | q3 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q4* | q4 | q4 | q4 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q5 | - | - | - | - | - | - | - | - | - | - | q6 | - | - | - | - | - | - | - | - | - |
| q6 | - | - | - | - | - | - | - | - | - | - | - | q7 | - | - | - | - | - | - | - | - |
| q7 | - | - | - | - | - | - | - | - | - | - | - | - | q8 | - | - | - | - | - | - | - |
| q8 | - | - | - | - | - | - | - | - | - | - | - | - | - | q9 | - | - | - | - | - | - |
| q9* | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q10 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | q12 | - | q11 | - | - | - |
| q11* | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q12 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | q13 | - | - | - | - |
| q13* | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q14* | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q15* | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q16* | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q17* | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q18* | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q19* | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| q20 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | q21 | - |
| q21 | - | - | - | - | - | - | - | - | - | - | - | q22 | - | - | - | - | - | - | - | - |
| q22 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | q23 | - | - | - | - | - |
| q23* | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

Table 2: Transition Table

Fig3. Clarifying simulation using JFLAP

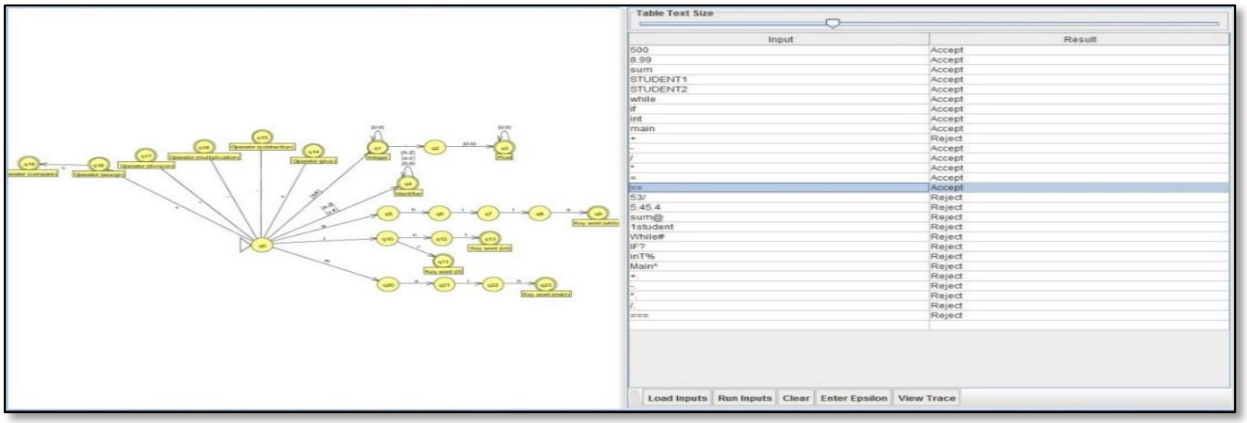



Fig5: Simulation Using JFLAP

### 3.5. Some Accepting and Rejecting Scenarios

#### 3.5.1. Accepting Strings

a- ==. b- 23.5. c- Sum1. d- While.

Here through using the JFLAP program after input string of expression the program checks it. Systematically via each symbol when examines all expressions related to language consequently the results explain it accepts. For example, the program checking section a. == operation in Fig6. Start from the initial state. Along with moving through transition to check, the next symbol and so on until examining all symbols and typing its acceptance. Fig 7. Showed acceptance expression and given the name of the string. Where b- 23.5 shows if float. c- Sum1 here shows its identifier. d- While where are keywords.

1 2

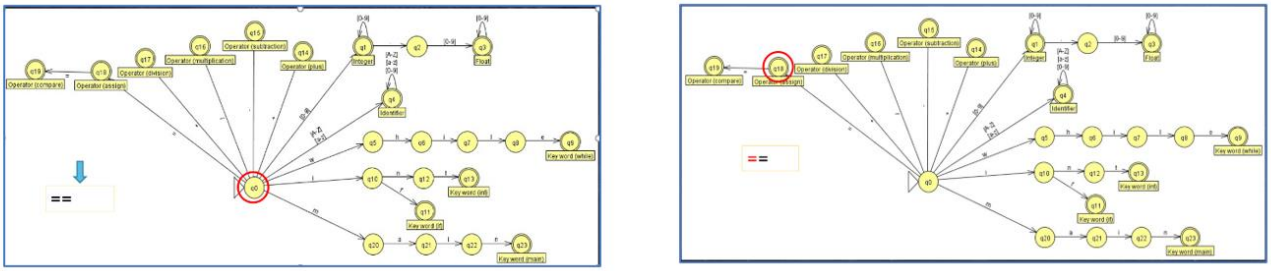


Fig6. Start from initial state

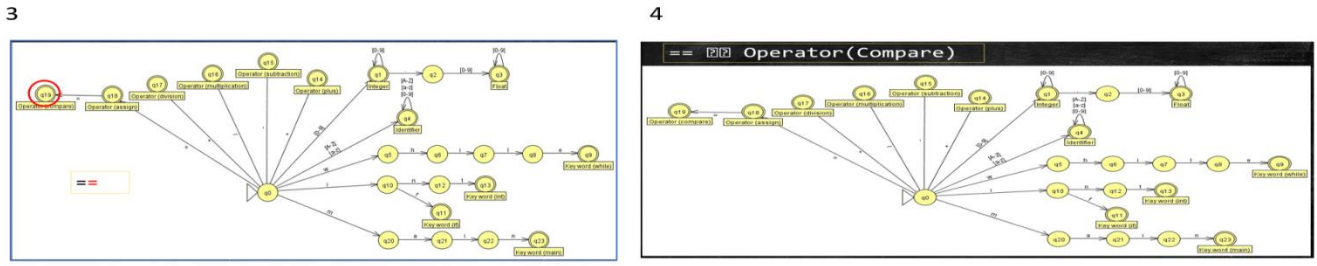



Fig 7. Showed acceptance expression and given the name of string

### 3.5.2. Unacceptable String

By using JFLAP program after input string of expression the program check it. Gradually for each symbol when examine all and shows that expression are incorrect and do not related to language. Example the program check 1sum. Here in Fig8. Shows that initial state check an integer symbol. After accept look to another symbol, which is S. Fig.9. Shows that rejected the expression.

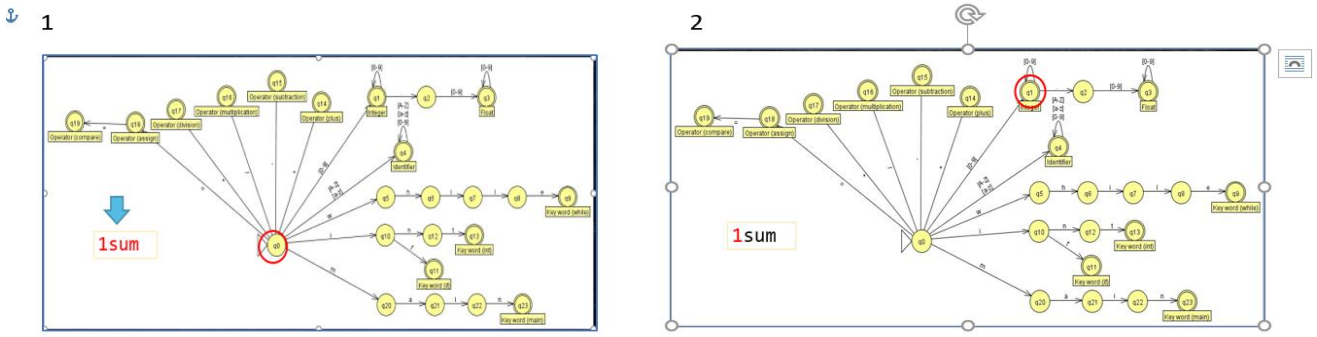



Fig8. Shows that initial state check integer symbol

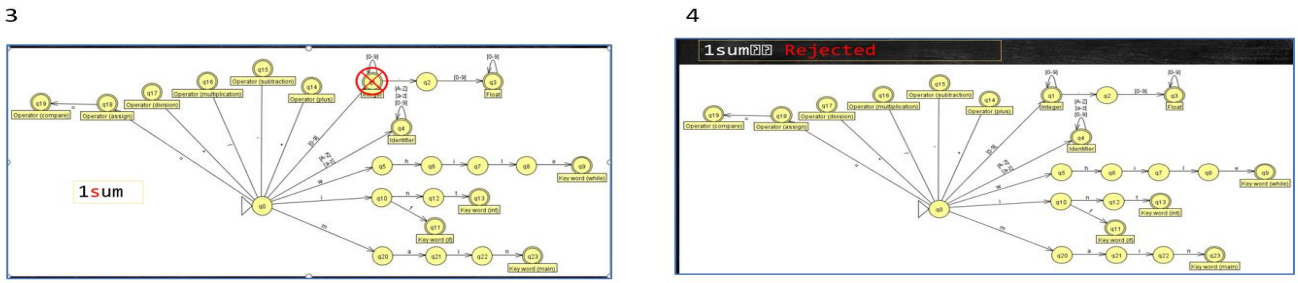



Fig.9. shows that rejected the expression.

## 4. Conclusion

Automata models are being used in different types of software development phases and its benefit are that it can be reused as a general model for some software development phases and it could reduce the cost of using the real system first and diagnoses for reliability for example. Searching to use of automata theory in complier found many usages of it and the main concern was about using DFA in Lexical Analysis phase in complier.

## 5. References

http://en.wikipedia.org/wiki/Automata_theory.
Jacquemard, F., Klay, F., & Vacher, C. (2009). Rigid tree automata. In *Language and Automata Theory and Applications* (pp. 446-457). Springer Berlin Heidelberg.
Ezhilarasu, P., & Krishnaraj, N. (2015). Applications of Finite Automata in Lexical Analysis and as a Ticket Vending Machine–A Review. *Int. J. Comput. Sci. Eng. Technol*, *6*(05), 267-270.
Abdulnabi, N. L., & Ahmad, H. B. (2019). Data type Modeling with DFA and NFA as a Lexical Analysis Generator. *Academic Journal of Nawroz University*, *8*(4), 415-420.
Ipate, F. (2012). Learning finite cover automata from queries. *Journal of Computer and System Sciences*, *78*(1), 221-244.
Fraser, C. W., & Hanson, D. R. (1995). *A retargetable C compiler: design and implementation*. Addison-Wesley Longman Publishing Co., Inc..
Steven S. Muchnick. (1997). *Advanced compiler design implementation*. Morgan Kaufmann.
Lesk, M. E., & Schmidt, E. (1975). Lex: A lexical analyzer generator.