Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 12, Number 3 (2016), pp. 2085–2111

http://www.ripublication.com/gjpam.htm

Great Evaluator: An Automated Assessment System for Evaluating Regular Grammars in Automata Theory

Madhumitha Ramamurthy,
Assistant Professor, Department of CSE,
Sri Krishna College of Engineering and Technology.

Dr. Ilango Krishnamurthi,
Dean, Department of CSE,
Sri Krishna College of Engineering and Technology.

Stanly Mammen
Research Scholar,
Department of CSE
Sri Krishna College of Engineering and Technology.

Abstract

Answer script evaluation is an important activity in educational domain. Many researchers have made various attempts to automate this process. In an answer script, there are different types of answers like textual answers, mathematical answers and diagrams to be evaluated. In view of automated evaluation, each answer type requires different techniques for assessment. For instance, the approaches for evaluating textual answers may not be suitable evaluating mathematical answers or diagrammatic answers. Moreover, we might encounter some interesting cases where there are multiple correct answers exist for a given question. Here we consider such a case and we propose our approach for design of novel tool for automated evaluation of regular grammars constructed for a given regular expression in automata theory domain. We also propose our approach for automated generation of regular grammars for comparing the student grammar incase of missing/incorrect answer key. We propose two of our approaches namely 'Reduced Expression Graph' and 'Parse Tree based top-down and bottom-up' for generation of answer key. Our experimental results prove that our approach works well for any number of grammars constructed for the same regular expression.

Keywords: Assessment, Automata Theory, Regular Grammar, Regular Expression.

1. Introduction

Assessment is an important activity in educational domain which evaluates learners' knowledge of the concepts learnt (GA Brown, J Bull, M Pendlebury, 2013) [1]. Automated assessment of answers in an answer script makes use of Computer Assisted Assessment (CAA) Technology. The main advantages of this type of assessment are time management, reduced human labor, increased effectiveness and efficiency. This type of automatic evaluation will be more authentic since humans are not involved in evaluating task. A fairer and more efficient assessment can be achieved through automated assessment since it is free of judgments myths and value biases. It can also provide better and correct feedback to students.

Assessment can be classified as formative and summative type assessment. Formative assessment is an informal assessment which does not give score to the students'; instead it helps to improve students' learning. Summative assessment is a type of formal assessment whereby student learning is assessed by conducting exams and scoring their answers. The summative assessment is classified into objective and subjective assessment. The objective form of assessment has only one correct answer to evaluate and it is the most common form which comprises of multiple choice questions, true/false questions; fill in the blank questions and matching questions. The subjective form of assessment of answers may encounter multiple answers to evaluate which may be of different kind like textual, mathematical, diagrammatic, programming answers, etc. The proposed problem of evaluating the regular grammar in automata theory is a form of subjective answer.

A crucial issue associated with automated evaluation of subjective form of answer is that it has multiple equivalent correct answers for a given question. Obviously, automated evaluation of such type of answers is a difficult task. Such a problem can appear in text, graphical or mathematical form. In text, there can be multiple equivalent definitions for a single terminology. Similarly, in graphical or diagrammatic answers, there can be multiple correct answers like as in ER-diagram, or multiple equivalent diagrams as in finite state automata. In automata theory, such a case appears in evaluation of multiple equivalent regular grammars for regular expression.

The proposed system takes two set of grammars, a student written grammar for evaluation and the grammar written by the teacher as answer key. The grammar given by the student will be compared with the grammar specified by the teacher for the purpose of evaluation. The given student grammar cannot be evaluated correctly, if the answer key is incorrect/missing. Automated systems for answer key generation may help in this regard. Here we also propose our approach for automated generation of answer keys.

The rest of the paper is organized as follows. Section 2 outlines the related work. Section 3 describes the definitions and terminologies required. Section 4 describes the proposed approach used in this paper. Section 5 shows the experimental results; Section 6 shows the discussion of the proposed approach and Section 7 gives the conclusion.

2. Related Work

Few works have been done on assessment of automata theory exercises. Some of the works are listed below.

A Cetinkaya, 2007 [2] introduces an approach for automated generation of Regular Expression using Grammatical Evolution. Regular Expression is constructed by applying recursive rules. The generated regular expression is tested for its correctness by using genetic algorithm. We propose an approach for automated generation of regular grammars from regular expressions.

L Salmela, J Tarhio, 2004 [4] proposed an automated assessment system for evaluating home assignments of compiler design course. The topics considered for assessment is Finite State Automata (FSA), LL Parsing and LR Parsing. The system uses automated compiler exercises (ACE) Clients and Verifiers in the system, where the verifier component verifies the answers given by the ACE Client. The proposed system is also an automated assessment system which evaluates correctness of regular grammar constructed for a given regular expression against the system generated answer key.

J Zhang, Z Qian, 2013 [5] introduces algorithms for showing the equivalence between regular grammar and finite automata in accepting languages. Our system uses the approach given in (J Zhang, Z Qian, 2013) for converting regular language to finite automata.

V Tscherter, 2004 [6] is a tool for tutoring and grading structured exercises in theory of computation. It is an automated tutor which helps to teach basic concepts of theory of computation. This tool generates exercises, checks and comments every step of the student solution to the given exercise. Finally feedback is given to the student as either correct or incorrect answer. The system considers exercises like converting deterministic finite automata (DFA) to minimized DFA (using state table), string reverse problem, checking equivalence between finite automata and regular expression (RE) and vice-versa. Our system, even though not a tutoring system, is an assessment system for evaluation of regular grammars constructed for a regular expression.

S.V. Vucasinovic, P.S. Stanimirovic, 2006 [7] describes an algorithm for automatic generation of regular language from regular grammar. The algorithm generates all the regular expression from the regular grammar and makes the union of all the regular expressions. This makes the regular language for the corresponding regular grammar. The approach first removes or eliminates all irregular grammars and the regular grammar is passed to a recursive function to produce the regular language. We propose an approach for automated generation of regular grammar from regular expression.

S. Bhargava, G.N. Purohit, 2011 [8] proposed a method to construct a minimal-DFA (MDFA). This approach is based on set of grammar rules which is in the form of a parsing algorithm. This method combines many graph's DFA's to obtain one graph DFA. Thus dependency between conversion of RE to non-deterministic automata (NFA), NFA to DFA and DFA to MDFA is removed and thereby it utilizes less memory and time. Amit, VK Salar, 2012 [11] proposed an approach to construct NFA from regular expression without ϵ transitions. Generally using Thompson construction K. Thompson, 1968 [12], NFA's are constructed with ϵ-transitions, but Amit, VK Salar, 2012 [11] proposed

a method to remove the ϵ-transitions in NFA by connecting the NFA's based on the behavior of ending states. Similarly, A.M. Afat, 2014 [10] describes the procedure for conversion of NFA to DFA by using subset construction algorithms. The input to the system is 5 tuples of FA which identifies whether it is a NFA or DFA. If identified as DFA, the program automatically reads the input string to check whether it belongs to the given language. If the approach identifies it as NFA, then NFA will be converted to DFA by subset construction algorithm (JE Hopcroft, R Motwani, JD Ullman, 2010) [13] and then checks whether the given string belongs to the language. We propose a reduced expression graph and parse tree based approach to construct automata from regular expressions. In our parse tree based approach, we propose two alternate methods, top-down and bottom-up approach to construct automata and automated generation of grammar.

R. Sinhna, A. Dewangan, 2012 [9] describes about the algorithms and methods which takes regular expression as input and converts it into the finite automata and then generates java code for the finite automata. Similarly, S. Memeti, 2012 [4] proposes a tool for generating java source code for a given regular expression or finite automata. The system converts regular expression to NFA and then to DFA and finally, java source code for DFA is generated. Our system propose methods to convert regular expressions to automata and also constructs automata for the regular grammar given by the student answer and finally verifies equivalence between the two.

There is no proposed automated approach for evaluating regular grammar specified by the regular expression in automata theory. This paper proposes an automated approach for evaluating regular grammars.

3. Definitions and Terminologies

3.1. Regular Expressions

A regular expression (D. Goswami and K. V. Krishna, 2010) [16] is defined over an alphabet $\sum$ recursively as follows.

1. ϕ, ξ, and a, for each a $\in \sum$, are regular expressions representing the languages ϕ, $\left\{\sum\right\}$, $\{a\}$, respectively.

2. If r and s are regular expressions representing the languages R and S, respectively, then so are:

   (a) $(r + s)$ representing the language $R \cup S$,

   (b) (rs) representing the language RS, and

   (c) $(r*)$ representing the language $R*$.

3.2. Regular Grammar

A grammar (John Martin, 2010) [15] is regular if $G = \left(V, \sum, S, P\right)$ where V is non-terminal, $\sum$ is terminal, P is set of production rules of the form $A \rightarrow \sigma B$ or $A \rightarrow \lambda$, where $A, B \in V$ and $\sigma \in \sum$ and S is the start symbol.

3.3. Finite Automata

A finite automata (John Martin, 2010) [15] is represented as $\left(Q, \sum, \delta, q0, A\right)$ where Q is a finite set of states, $\sum$ is a finite set of inputs called the input alphabet, δ is a function which maps $Q \times \sum$ into Q, $q_0 \in Q$ is the initial state and A is the subset of Q is the set of final states.

3.4. Deterministic Finite Automata

A Deterministic Finite Automaton (DFA) (John Martin, 2010) [15] is represented as $M = \left(Q, \sum, \delta, q_0, A\right)$ consisting of a finite set of states (Q), finite set of input symbols $\left(\sum\right)$, a transition function represented by $\delta : Q \times \sum \rightarrow Q$, a start state, $q_0 \in Q$ and a set of accepting states $A \subseteq Q$.

3.5. Non-Deterministic Finite Automata

A Nondeterministic Finite Automaton (NFA) (John Martin, 2010) [15] is represented as $M = \left(Q, \sum, \delta, q_0, A\right)$ consisting of a finite set of states Q, a finite set of input symbols $\sum$, a transition function $\delta : Q \times \left(\sum \cup \right) \rightarrow 2^Q$, a start state $q_0 \in Q$ and a set of accepting states $A \subseteq Q$.

4. Proposed Work

This paper proposes an approach named GREAT Evaluator (Grammar REgular type AuTomated EVALuator) to automate the evaluation of regular grammars constructed for regular expression written by the students. There are two important challenges; firstly there are multiple correct answers for the given problem, i.e., each student can construct different correct grammars for the same regular expression. Secondly, the grammar should produce only the language denoted by the regular expression. In practice, it is possible to write grammar which produces other strings other than L(G). Here we propose a procedure based approach for solving this problem.

Generally, an answer key will be provided for answer script evaluation. In the problem addressed, since there are multiple equivalent solutions to the problem, a direct solution does not exist. There may be cases where answer key is missing or incorrect. Hence

we propose our approach for automated answer key generation and answer evaluation using the same. Assume for a given a language $L(r)$, denoted by the regular expression r, there exists a grammar G and its corresponding language $L(G)$. Here, for the given regular expression r, the student constructs the grammar G_S and system generates the grammar G_A. An equivalent grammar G_K is given as answer key. It is sufficient to prove that both the grammars correspond to the same language $L(G)$.

In this paper, we propose an approach to automate the evaluation of regular grammar and also we propose two approaches namely, reduced expression graph and parse tree based solution to automate the generation of answer $key(G_A)$ for automated evaluation of regular grammar if the answer key is incorrect/missing or if there are many equivalent grammars to be evaluated.

4.1. Automated Evaluation of regular grammar

This section proposes our approach for design of a tool to evaluate the regular grammar written by the student. Given a regular expression, student has written a grammar G_S. The task of "GREAT Evaluator" is to assess or evaluate whether the grammar written by the student G_S specifies the language given by the regular expression. Hence, the answer key/regular grammar specified by the teacher G_K is used to evaluate the grammar written by the student G_S.

Our proposed architecture in Fig.1 takes two set of grammars for evaluation G_S and G_K. The system evaluates the grammar written by the student by comparing it with G_K, for which corresponding automata's for G_S and G_K is constructed and verified for equivalence of the automata's to find the correctness of the grammar written by the student.

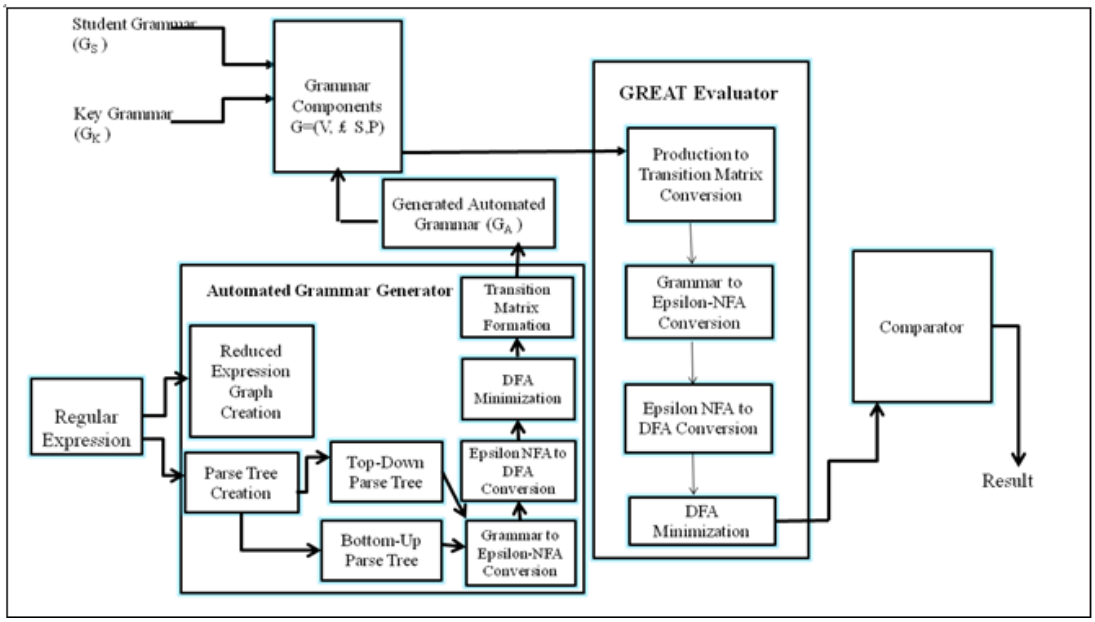

Figure 1: System Architecture for GREAT Evaluator

$S \rightarrow aS_1 \mid bS_1 \mid S_1$
$S_1 \rightarrow aS_2$
$S_2 \rightarrow bS_2 \mid S_3$
$S_3 \rightarrow aS_3 \mid bS_3 \mid S_1 \mid \epsilon$

Figure 2: Key Grammar-G_K

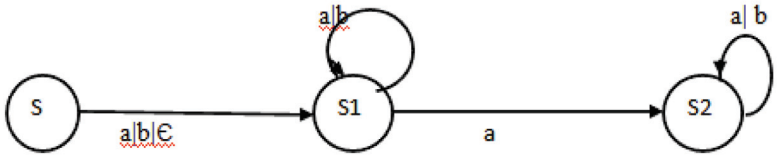

Figure 3: Finite Automata for the Key Grammar-FA_K

4.1.1 Automata Generation from key grammar G_K

The regular grammar specified by the teacher (G_K) as answer key is used to evaluate the grammar written by the student (G_S) in the answer script. For illustration, a grammar given by the teacher for a regular expression $(a/b)*ab*(a/b)*$ is considered as answer key as given in Fig.2 and its corresponding automata (FA_K) is given in Fig. 3. The ϵ-NFA is converted to DFA using the subset construction algorithm (JE Hopcroft, R Motwani, JD Ullman, 2010) [13] and is given in Fig. 4 and Fig. 5. The resulting DFA is given in Fig. 6.

Є- Closure(S) ={S,S1}
Є- Closure(S1) ={S1}
Є- Closure(S2) ={S2}

Figure 4: ϵ-Closure for the NFA states

Minimizing DFA The DFA thus generated in Fig. 7 is the renaming of the states, ie., the state [S,S1] as A, the state [S1,S2] as B, the state S1 as C of the Fig.6. This generated DFA given in Fig. 7 is minimized using the standard procedure (JE Hopcroft, R Motwani, JD Ullman, 2010) [13] as illustrated in Fig.8 and the resulting minimized DFA (FA_K) is given in Fig. 9.

4.1.2 Automata Generation from student Grammar G_S

The grammar written by the student in the answer script (G_S) is compared with the grammar specified by the teacher as answer key (G_K) or with the automated generated grammar by the system (G_A). If the grammar written by the student is equivalent with

δ([S,S1],a)	= Є- Closure(δ((S,a) U δ(S1,a))
	= Є- Closure({S1U S2})
	= [S1,S2]
δ([S,S1],b)	= Є- Closure(δ((S,b) U δ(S1,b))
	= Є- Closure({S1})
	= [S1]
δ([S1,S2],a)	= Є- Closure(δ((S1,a) U δ(S2,a))
	= Є- Closure({S1,S2})
	= [S1,S2]
δ([S1,S2],b)	= Є- Closure(δ((S1,b) U δ(S2,b))
	= Є- Closure({S1,S2})
	= [S1,S2]
δ([S1],a)	= Є- Closure(δ((S1,a)))
	= Є- Closure({S1,S2})
	= [S1,S2]
δ([S1],b)	= Є- Closure(δ((S1,b))
	= Є- Closure({S1})
	= [S1]

Figure 5: Subset construction –States of DFA

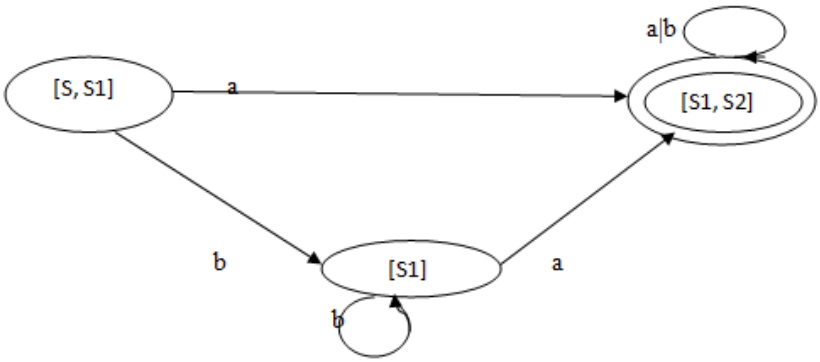

Figure 6: Generated DFA

any one of the answers ie., G_K or G_A, then the grammar written by the student G_S is considered to be valid.

For comparison, first the student answer is converted to DFA and then the DFA is minimized. The grammar written by the student G_S is given in Fig. 10 and it is converted into DFA. as given in Fig. 11. The finite automata thus generated in Fig. 11 is a DFA and is minimized for checking the equivalence between two automata's.

Minimizing DFA The DFA thus generated in Fig. 11 is minimized using the standard procedure (JE Hopcroft, R Motwani, JD Ullman, 2010) [13] as illustrated in Fig. 12 and the resulting minimized DFA (FA_S) is given in Fig. 13.

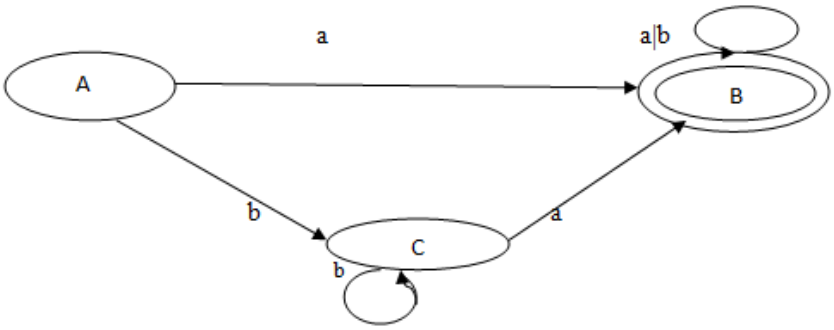

Figure 7: Diagrammatic representation of DFA with renamed states

δ(A,a)	= **B**
δ(A,b)	= C
δ(B,a)	= **B**
δ(B,b)	= **B**
δ(C,a)	= **B**
δ(C,b)	= C

Figure 8: Transition of DFA with renamed states

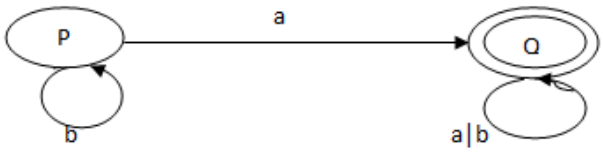

Figure 9: Minimized DFA-FA_K

S→aA|bB
B→aA|bB
A→aA|bA|∈

Figure 10: Student Grammar-G_S

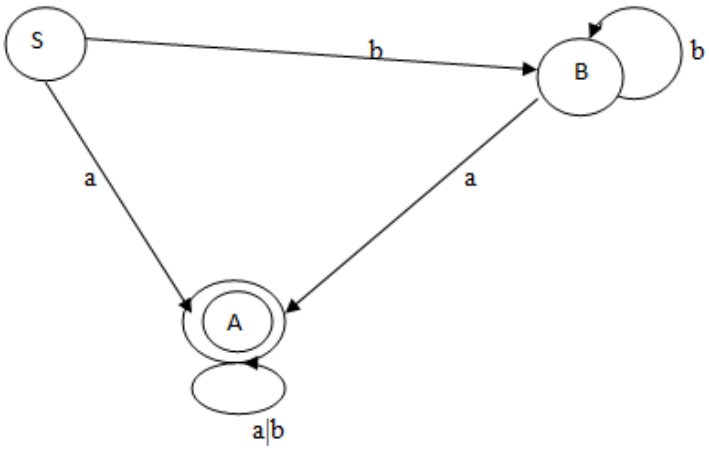

Figure 11: Generated DFA

δ(S,a)	= **A**
δ(S,b)	= B
δ(B,a)	= **A**
δ(B,b)	= B
δ(A,a)	= **A**
δ(A,b)	= **A**

Figure 12: Transition of DFA with renamed states

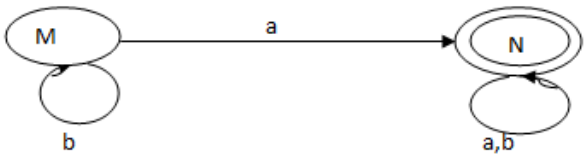

Figure 13: Minimized DFA-FA_S

4.1.3 Comparison of Automata's

The equivalence between the three automata's given in Fig. 9, Fig. 13 and Fig. 46 is verified for equivalence using the standard procedure (JE Hopcroft, R Motwani, JD Ullman, 2010) [13] and the results are given in Table 1 and Table 2.

The equivalence of two DFA finds whether the grammar written by the student is equivalent with key grammar and automated generated grammar. Table 1 shows the equivalence between of FA_A and FA_K grammars. Table 2 shows the equivalence between of FA_A and FA_S grammars.

Table 1: Equivalence between FA_A and FA_K.

Parent State	Input- a	Input-b
P,X	Q,Y	P,X
Q,Y	Q,Y	Q,Y

Table 2: Equivalence between FA_A and FA_S.

Parent State	Input- a	Input-b
M,X	N,Y	M,X
N,Y	N,Y	N,Y

4.2. Automated Answer Key Generation for automated evaluation of regular grammar

Generally, an answer key will be provided for answer script evaluation. In the problem addressed, since there are multiple equivalent solutions to the problem, a direct solution does not exist. There may be cases where answer key is missing or incorrect. Hence we propose our approach for automated answer key generation for automated answer evaluation using the same if the answer key is missing or incorrect or if there are multiple equivalent grammars for a regular expression.

We propose two approaches namely, reduced expression graph based solution and parse tree based solution to the problem. In the first approach, initially we construct a reduced expression graph representing the regular expression using the algorithm given in Fig. 14. This algorithm also constructs the NFA corresponding to the regular expression r. In the second approach, a parse tree is constructed from the regular expression initially and NFA is constructed corresponding to the regular expression. In both the approaches, the ϵ-transitions, if any, are eliminated forming the finite automata. The finite automata thus generated is minimized by eliminating unreachable states and by combining equivalent states and an automated grammar is generated from the minimized

Construction of ReducedExpressionGraph(RegExp)

1. Consider the priorities for the operators ( ), /,+,* in decreasing order.
2. Process the *RegExp* with left to right associativity.
3. For each sub-expression e in *RegExp*
4. Create the rule NON-TERMINAL→e
5. Replace e with a new NON-TERMINAL
6. Next e
7. Replace TERMINALS if any with new NON-TERMINAL
8. Create the rule new NON-TERMINAL→TERMINAL
9. Connect between list of NON-TERMINALS

Figure 14: Construction of ReducedExpressionGraph(RegExp)

automata. This automated grammar G_A is used as the key grammar G_S in automated evaluation of grammar written by the student G_S.

4.2.1 Automated Answer key Generation using Reduced Expression graph

For the generation of automated answer key, we propose our approach which we call as 'Reduced Expression Graph' approach. In this context, we introduce our terminology called an 'Expression Graph' which represents a regular expression as a set of states transiting from one state to another. Each state in expression graph denotes a non-terminal symbol in the equivalent grammar and a transition denotes rule set as given in Figure 16.

Automata construction from Reduced Expression Graph A regular expression can be expressed as a union of two or more sub-expressions and each sub-expression can be represented as a state. By applying this rule, we convert each sub-expression into an equivalent state and generate a corresponding substitution rule. In this process, the innermost sub-expression is reduced first and then proceeds to the uppermost, producing a substitution rule in each step. Finally, the resulting expression will consist of only concatenation operation, which is equivalent to union operation. Each sub-expression can be represented by a state, forming the equivalent NFA as shown in Fig. 16. A complete automata can be constructed by applying the substitution rule iteratively using the algorithm given in Fig. 14. We illustrate this with an example.

Assume a regular expression given by $(a/b) * ab * (a/b)*$. The priority is given to the simple parenthesis, i.e., each enclosed sub-expression and is substituted by a non terminal symbol. This reduction is represented by the corresponding substitution rule. Each of this non-terminal represent a state in the corresponding NFA. The NFA produced is a DFA with ϵ-transitions.

$$(a/b) * ab * (a/b) \tag{1}$$

In the equation (1), the innermost sub-expression is identified with left to right associativity and is substituted by a non-terminal. The corresponding substitution is repre-

sented by a substitution rule. The next priority is assumed for OR ('/ 'or '+') operation.

$$A * \bullet a \bullet b * \bullet B * \qquad \text{By Rule 1: } A \rightarrow a/b, \quad \text{Rule2: } B \rightarrow a/b \tag{2}$$

The next priority is given to Kleen Closure operation and the reduction is made by applying rule 3, 4 and 5.

$$C \bullet a \bullet D \bullet E \qquad \text{By Rule 3: } C \rightarrow A*, \quad \text{Rule 4: } D \rightarrow b*, \quad \text{Rule 5: } E \rightarrow B* \tag{3}$$

Finally, terminals if any, is substituted by non-terminals, producing rule 6.

$$C \bullet F \bullet D \bullet E \qquad \text{By Rule 6: } F \rightarrow a \tag{4}$$

Now, to construct ϵ-NFA, we apply the substitution rules iteratively to the reduced graph structure. This reduced graph structure denotes the regular expression as union of constituent regular expression. The procedure is given in Fig. 15 and is illustrated here.

```
Algorithm ExpansionofReducedExpressionGraph
1.  For each node n in ExpressionGraph
2.  while not expandable(n)
3.      rule = SubstitutionRule(n)
4.      for each token t in rule
5.          if t= NON-TERMINAL then
6.              n=t
7.          elseif t=TERMINAL then
8.              INPUTSYMBOL=t
9.              addINPUTSYMBOL(n, INPUTSYMBOL, t.NEXT)
10.         elseif t='*' then
11.             INPUTSYMBOL= Є
12.             addRECURSION(t.NEXT,t.PREVIOUS)
13.         elseif t='/' then
14.             INPUTSYMBOL=/
15.             addINPUTSYMBOL(n, INPUTSYMBOL, t.NEXT)
16.         endif
17.     Next t
18.  Next
19. Next n
20. End
```

Figure 15: Algorithm for construction of Reduced Expression Graph

Initially, the reduced graph consists of four nodes representing four states. The graph is expanded by using the substitution rules from left to right recursively and the result of applying the substitution rules is given in Fig. 17, Fig. 18, Fig. 19, Fig. 20, Fig. 21 and Fig. 22 shows the final automata constructed.

4.2.2 Automated Answer key Generation with parse tree

A parse tree, generally, is an ordered, rooted tree which represents order of evaluation of an expression. Here, the expression under consideration is a regular expression denoting

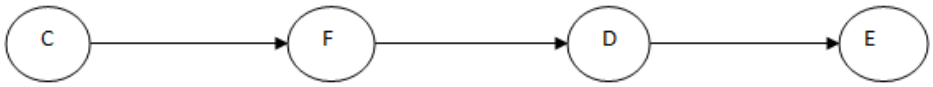

Figure 16: Reduced Graph - C.F.D.E

Now, applying the substitution Rule 3: $C \rightarrow A*$ we get Fig. 17,

Figure 17: Substitution for $C \rightarrow A*$

Now, applying Rule 1: $A \rightarrow a/b$ we get Fig. 18,

Figure 18: Substitution for $A \rightarrow a/b$

Now, applying Rule 6: $F \rightarrow a$ we get Fig. 19,

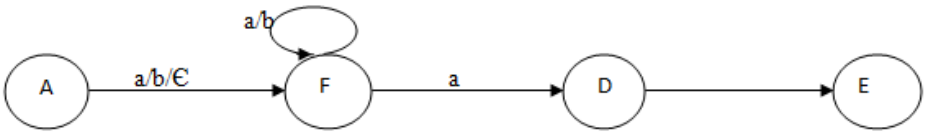

Figure 19: Substitution for $F \rightarrow a$

Now, applying Rule 4: $D \rightarrow b*$ we get Fig. 20,

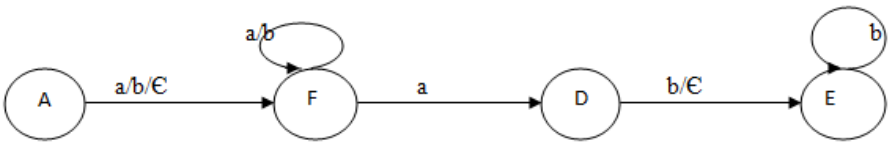

Figure 20: Substitution for $D \rightarrow b*$

Now, applying Rule 5: $E \rightarrow B*$ we get Fig. 21,

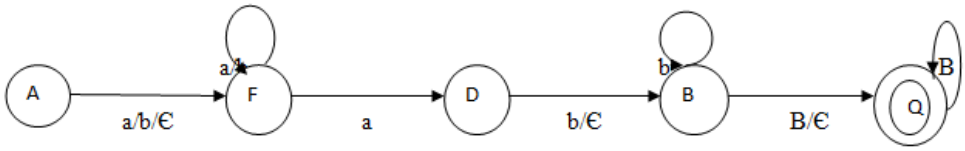

Figure 21: Substitution for $E \rightarrow B*$

Finally, applying Rule 2: $B \rightarrow a/b$ we get Fig. 22,

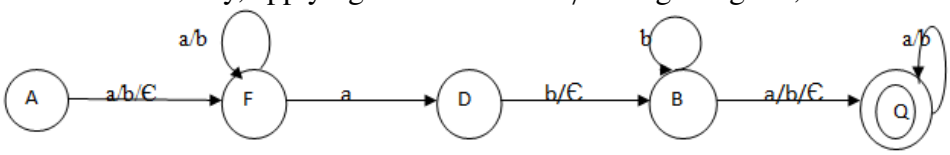

Figure 22: Substitution for $B \rightarrow a/b$

language L(G). In this section we demonstrate our approach for generating the grammar G for the language L(G) given by the expression r. In our approach, we first assign priorities to the operators in the regular expression. The operator kleen closure (*) is alone considered to be unary. The operators ( ), /,*, • are evaluated in decreasing order of precedence and with left to right associativity. Considering the operator precedence and associativity, the sub-expressions is retrieved and added to the parse tree using the algorithm given Fig. 23 Fig. 24 show the constructed parse tree using the algorithm given in Fig. 23.

The root node of the parse tree represents the start symbol of the grammar and each node represents a non-terminal (NT) and the leaf nodes represents terminal (T) symbols. For construction of production rules, we traverse the parse tree and the production rules are produced by assuming parent node as LHS and its children node as RHS as given in the Fig. 25 and the algorithm Fig. 26. Fig. 27 shows the generated grammar using the algorithm given in Fig. 26 and the generated grammar is a context free grammar.

This context free grammar which is generated in Fig. 27 has to be converted to regular grammar for evaluation of the grammar written by the student and hence we propose our 'Top-Down and Bottom-Up' parsing approaches for construction of automata from parse tree. The automata thus generated is used to generate the regular grammar for evaluating the grammar written by the student.

Top-down approach of parse tree to construct automata The top-down approach parses the tree from the root node of the parse tree and Fig. 28 shows the rules used in the proposed top-down approach for construction of automata. We have proposed 8 rules namely single replacement rule, single transition rule, double replacement rule, double transition rule, backward insertion rule, backward transition rule, recursion rule 1 and recursion rule 2.

```
Algorithm ConstructParseTree(RegExp)
Priority('( )') →3
Priority ('/')  →2
Priority ('*')  →1
Priority ('•')  →0
ParseTree parseTree→NULL
Exp=RegExp
i=0
 While Exp.length>1
    Exp=retrieveNextSubExpression(RegExp)
    Node parentNode=createNode('A'+i, operator(Exp))
    For each operand opr in Exp
        Node childNode=createNode(operand)
        parentNode.child=childNode
    Next opr
 AddToParseTree(parseTree,parentNode)
ReduceRegularExpression(exp,RegExp,parentNode.name)
Next
End
```

Figure 23: Parse Tree Construction Algorithm

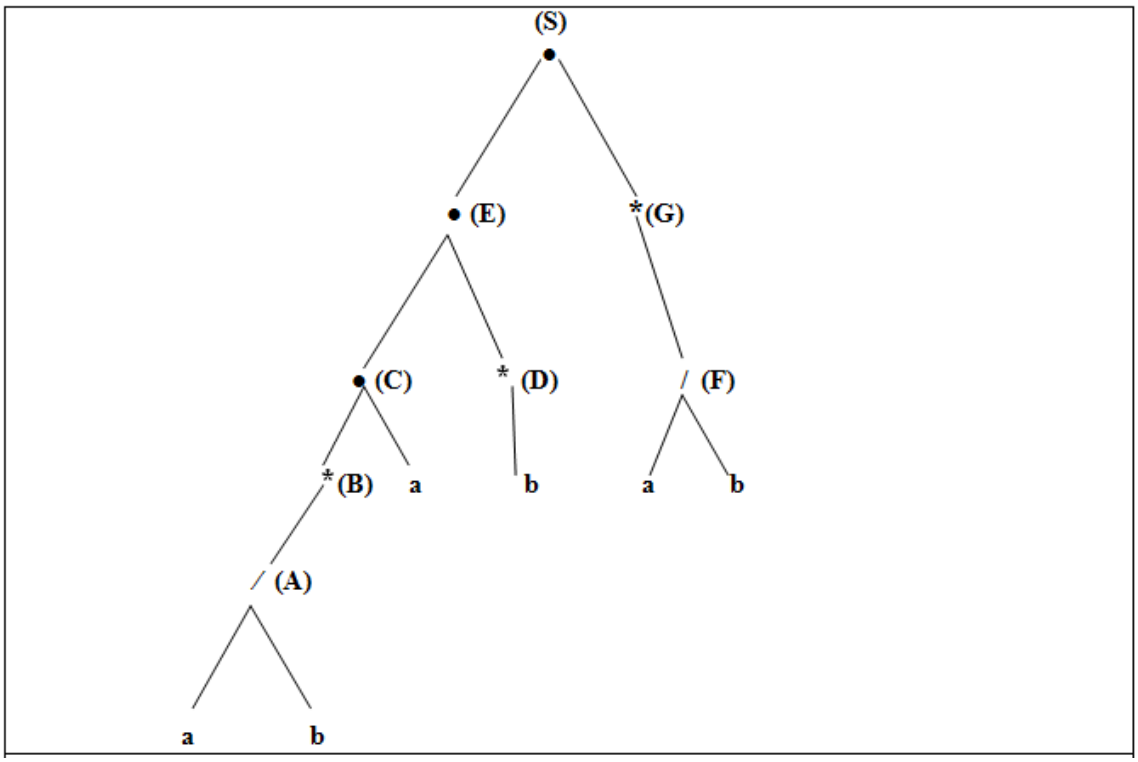

Figure 24: Parse tree Construction(RegExp)

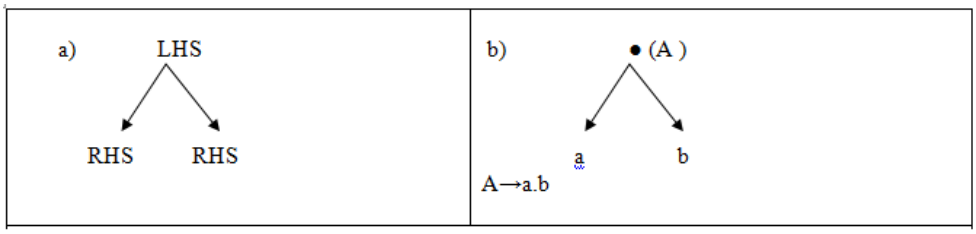

Figure 25: Diagrammatic Representation for generation of production rules

```
Algorithm ParseTreeToGrammar(parsetree)
For height=0 to parsetree.height
  Node siblingNodes[ ]= parsetree.retrieveAllNodesOfHeight(height)
    For each node in siblingNodes
      LHS=node.Name
      For each child=childNode(node)
        RHS=RHS+ child.Name
      Next child
    Next node
Next height
End
```

Figure 26: Algorithm for Grammar Generation from Parse Tree

```
A→a/b
B→A*     i.e. B→AB |Є
C→B.a
E→C.D
D→b*     i.e. D→bD |Є
S→E.G
G→F*     i.e. G→FG |Є
F→a/b
```

Figure 27: Automated Generated Grammar G_S using Parse Tree

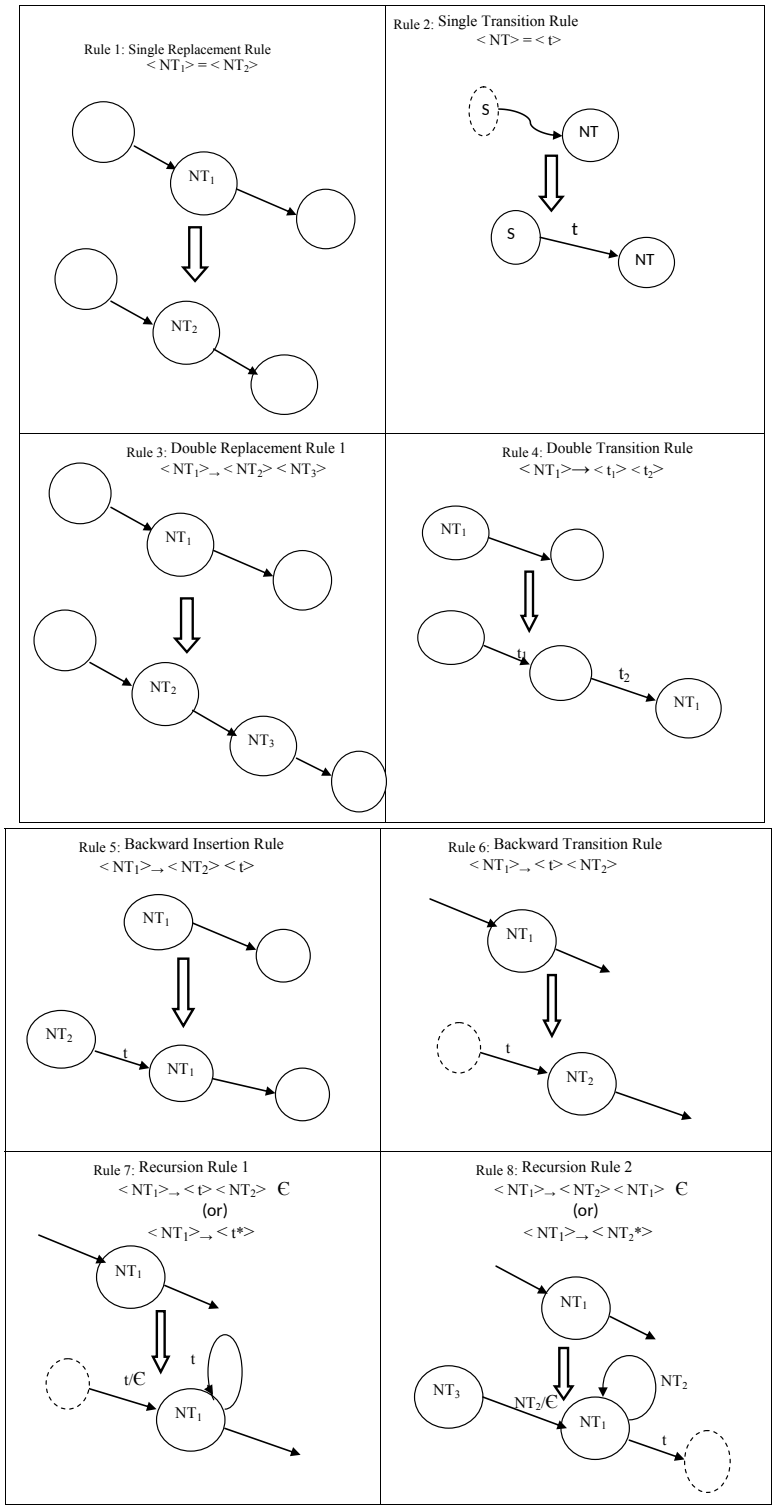

Figure 28: Rules for Top Down Approach

Figure 29: Start State

Now, by applying Rule 3, $S \rightarrow E.G$, we get

Figure 30: Substitution for $S \rightarrow E.G$

Now, by applying Rule 3, $E \rightarrow C.D$, we get

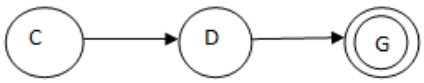

Figure 31: Substitution for $E \rightarrow C.D$

The automata is constructed by expanding the rules in Fig. 28. First, we begin the construction of automata from the root node of the parse tree as given in Fig. 29 and the rules in Fig. 28 are applied to in order of constructed states, as demonstrated from Fig. 30 to Fig. 37. Finally, the automata is constructed as given in Fig. 37, using our top-down approach. This automata constructed by using our top-down approach is the same as the one constructed using the reduced expression graph approach in Fig. 22. The only difference between Fig. 22 and Fig. 37 is in the naming of the states. The automata in Fig. 22 and Fig. 37 is ϵ-NFA and it is converted to DFA and finally minimized to generate the regular grammar.

Bottom-up Approach of the parse tree to construct automata Our proposed bottom-up approach parses the tree from the leaf node of the parse tree and Fig. 38 shows the steps used in bottom-up approach for construction of automata. The automata constructed by following the steps given in Fig. 38 using our bottom-up approach will be similar to the automata constructed using top-down approach given in Fig. 37.

Conversion of ϵ-NFA to DFA The automata given in Fig. 22 and Fig. 37 is ϵ-NFA. This ϵ-NFA is converted to DFA using the standard subset construction algorithm (JE Hopcroft, R Motwani, JD Ullman, 2010) [13] as illustrated in Fig. 39, Fig. 40, Fig. 41, Fig. 42 and Fig. 43.

Now, by applying Rule 5, $C \rightarrow B.a$, we get

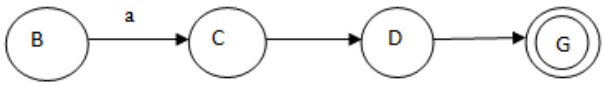

Figure 32: Substitution for $C \rightarrow B.a$

Now, by applying Rule 8, $B \rightarrow A*$ i.e. $B \rightarrow AB/\epsilon$, we get

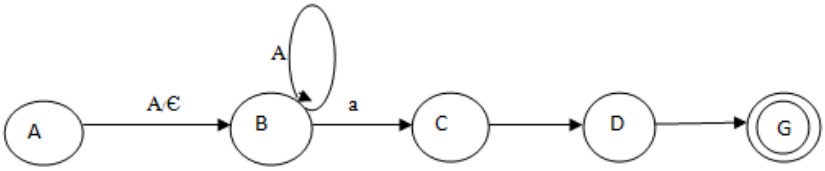

Figure 33: Substitution for $B \rightarrow A*$ i.e. $B \rightarrow AB/\epsilon$

Now, by applying Rule 2, $A \rightarrow a/b$, we get

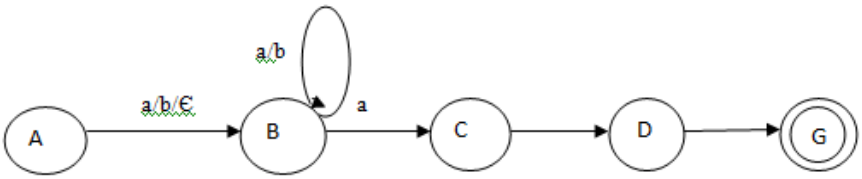

Figure 34: Substitution for $A \rightarrow a/b$

Now, by applying Rule 8, $D \rightarrow b*$ i,e., $D \rightarrow bD/\epsilon$, we get

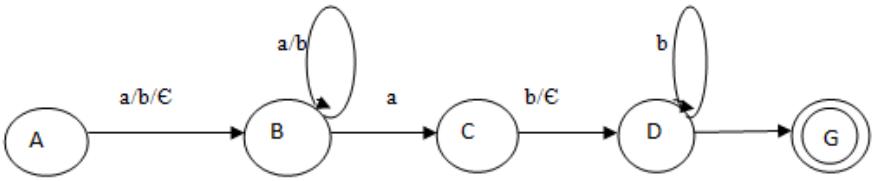

Figure 35: Substitution for $D \rightarrow b*$ i,e., $D \rightarrow bD/\epsilon$

Now, by applying Rule 8, $G \rightarrow F*$ i,e., $G \rightarrow FG/\epsilon$, we get

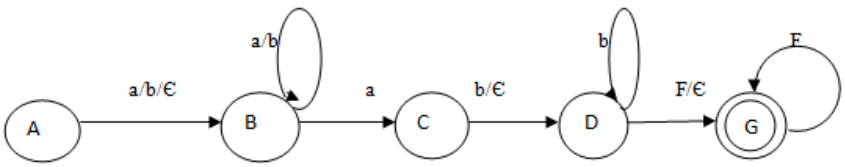

Figure 36: Substitution for $G \rightarrow F*$ i,e., $G \rightarrow FG/\epsilon$

Finally, by applying Rule 2, $F \rightarrow a/b$, we get

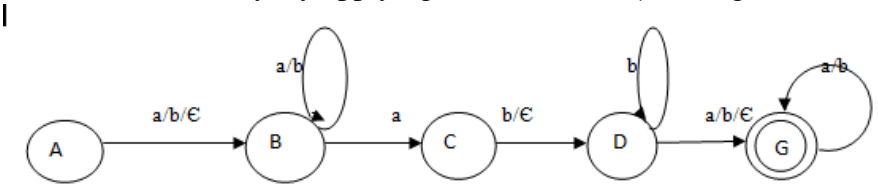

Figure 37: Substitution for $F \rightarrow a/b$

1. Start with the initial state S.
2. Read from leaf to root.
3. a) <NT> → T_1/T_2, from previous state transit to state NT with T_1/T_2. b)If previous state does not exist create one.
4. a) <NT> → $T_1.T_2$, from previous state transit to new state NT on T_1. <current-state> =NT. b) Create new state NT' and apply transition current state to NT' on T_2.
5. a) <NT_1> → NT*, T=reduceRecursively(NT_2), apply recursion on NT_1, with input T. b) Apply transition Є/T on current state to NT.
6. a) <NT>→ T*, create state NT if doesn't exist. b) Apply recursion on NT state on input T. c) Apply transition Є/T on current state to NT.
7. a) <NT_1> →<NT_2><NT_3>, replace NT_1 by NT_2. b) Expand(NT_2), Expand(NT_3).
8. a) <NT_1> →<NT_2><T>, replace NT_1 by NT_2. b) Transit from current state to NT_2 with T.
9. a) <NT_1> →<T><NT_2>, transit NT_1 to NT_2 on T. b) Current state= NT_2. If NT_1 previous state does not exist create one.

Figure 38: Steps for Bottom-up approach

	{a}	{b}	{Є}
A	F	F	F
F	D	-	-
D	-	B	B
B	Q	Q	Q
Q	-	-	-

Figure 39: State Transition Table for the Fig. 37

Q ={A,F,D,B,Q}

Q1={A,F,D,B}
Q2={Q}

Figure 40: Set of final and non-final states

Є- Closure(A) ={A,F}
Є- Closure(F) ={F}
Є- Closure(D) ={D,B,Q}
Є- Closure(B) ={B,Q}
Є- Closure(Q) ={Q}

Figure 41: ϵ-Closure for the states

δ([A,F],a)	= Є- Closure(δ([A,F],a)) = Є- Closure({F,D}) = [F,D,B,Q]
δ([A,F],b)	= Є- Closure(δ([A,F],b)) = Є- Closure({F}) = [F]
δ([F],a)	= Є- Closure(δ([F],a)) = Є- Closure({F,D}) = [F,D,B,Q]
δ([F],b)	= Є- Closure(δ([F],b)) = Є- Closure({F}) = [F]
δ([F,D,B,Q],a)	= Є- Closure(δ([F,D,B,Q],a)) = Є- Closure({F,D,Q}) = [F,D,B,Q]
δ([F,D,B,Q],b)	= Є- Closure(δ([F,D,B,Q],b)) = Є- Closure({F,B,Q}) = [F,B,Q]
δ([F,B,Q],a)	= Є- Closure(δ([F,B,Q],a)) = Є- Closure({F,D,Q}) = [F,D, B,Q]
δ([F,B,Q],b)	= Є- Closure(δ([F,B,Q],b)) = Є- Closure({F,B,Q }) = [F,B,Q]

Figure 42: Subset Construction-States of DFA

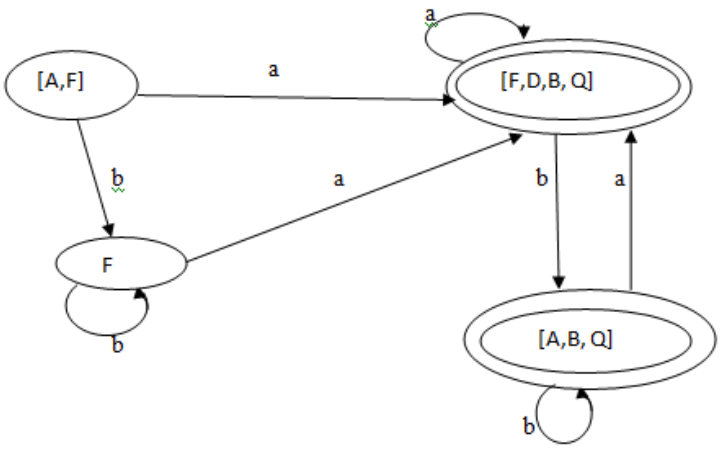

Figure 43: Diagrammatic representation of DFA

Minimization of DFA The DFA given in Fig. 44 is minimized to generate the corresponding regular grammar. Minimization of DFA generates an equivalent DFA with minimum number of states.

The Fig. 44 shows the same DFA with renaming of states generated in Fig. 43. Now consider the state [A,F] as A, the state [F,D,B,Q] as B, the state A as C and the state [A,B,Q] as D. Fig. 44 shows the finite automata with new states.

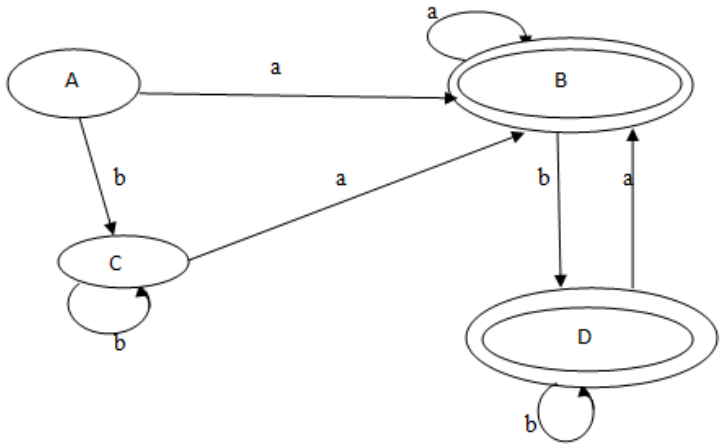

Figure 44: Diagrammatic representation of DFA with renamed states

δ(A,a)	= **B**
δ(A,b)	= C
δ(B,a)	= **B**
δ(B,b)	= D
δ(C,a)	= **B**
δ(C,b)	= C
δ(D,a)	= **B**
δ(D,b)	= D

Figure 45: Transition table of DFA

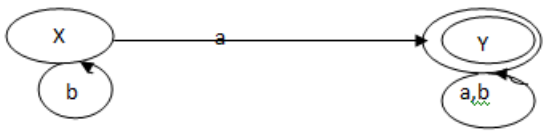

Figure 46: Minimized DFA - FA_A

```
Algorithm from TransMatrix to G_A

        for i=0 to n-1
          for j=0 to n-1
            if Q(i,j) ≠ NULL then
               write i → Q(i,j)•j
              if j = = END_STATE
                write I → Q(i,j)
              end if
            end if
          next;
        next;
  End
```

Figure 47: Transition Matrix

```
X→bX| aY
Y→aY| bY|Є
```

Figure 48: System generated regular grammar (G_A)

The DFA given in Fig. 44 is minimized using the procedure given in (JE Hopcroft, R Motwani, JD Ullman, 2010) [13]. The transition table is given in Fig. 45 and the minimized DFA (FA_A) is given in Fig. 46. The algorithm to generate grammar from transition matrix is given in Fig. 47. The automated grammar G_A thus generated is given in the Fig. 48.

4.3. Automated Scoring for Regular Grammar

Our automated scoring approach follows boolean evaluation strategy like mostly found in mathematical evaluation. Depending upon equivalence between the reference grammar and student written grammar, the student answer is either classified as correct or incorrect. In case of correct answer, full score is assigned else a score value of zero is assigned.

5. Experimental Results

We have tested the proposed approach with the set of regular expression and the corresponding regular grammar given in [14]. Fig. 49 lists the contents along with the automated regular grammar generated by our proposed approaches reduced expression graph and parse tree based approach. It was observed that our approach generated equivalent grammars in comparison with the grammar definition given in [14]. The accuracy was found to be 100 percentage while evaluating the regular grammar written by the student, when the student answer is found to be correct/in-correct. The proposed approach of generation of automated grammar is useful when the key grammar is not given for evaluation or if the key grammar is incorrect. The automated generated grammar can

Regular Expression	Regular Grammar	Automated grammar generated by our approach
a*	Non –Terminals : {S} Terminals : {a} Productions : S - £ | aS Start Symbols : S	A - £ | aA
a*b	Non –Terminals : {S} Terminals : {a,b} Productions : S - b | aS Start Symbols : S	A - aA|bQ
ba*	Non –Terminals : {S,A} Terminals : {a,b} Productions : S - bA A - £ | Aa Start Symbols : S	A – bQ Q - £|aQ
(ab)*	Non –Terminals : {S} Terminals : {a,b} Productions : S - £| abS Start Symbols : S	A -£| abA
a*bc*	Non –Terminals : {S,C} Terminals : {a,b.c} Productions : S - a S | b C C - £ | c C Start Symbols : S	A - aA|bQ Q - cQ|£
a*b*	Non –Terminals : {S,B} Terminals : {a,b} Productions : S - £ | aS | B B - b | Bb Start Symbols : S	A -aA|bQ|£ Q- bQ|£
a*bb*	Non –Terminals : {S,B} Terminals : {a,b} Productions : S - aS | B B - b | bB Start Symbols : S	A -aA|bQ Q- bQ|£

Figure 49: Sample Dataset of regular grammar definition and automatically generated regular grammar

be used in evaluation of any equivalent regular grammar written by the student for the given regular expression.

6. Discussion

We have proposed the system GREAT Evaluator (Grammar REgular type AuTomated EVALuator), which is an automated system to evaluate the regular grammars in the domain of automata theory. This system can be used to evaluate multiple equivalent regular grammars written by the students in automata theory exams. Generally, in manual evaluation, an answer key for the questions may be given for the human evaluators. But, at times given answer key given may be incorrect or missing, which may lead to deviated or in-correct evaluation. This can be avoided by following our approach which generates automated answer key for automated evaluation.

We propose two approaches for automated generation of answer key i.e., the automated regular grammar for a given regular expression. The first approach is the reduced

expression graph based approach and the second approach is the parse tree based approach to generate the automated grammar.

Our first approach named as reduced expression graph produces the automata which is ϵ-NFA. This ϵ-NFA is then converted to DFA which is then minimized to produce the automated grammar using the algorithm given in Fig. 15.

Our second approach named as parse tree based approach, generates context free grammar using the algorithm given in Fig. 27. From the generated context-free grammar, automata is constructed using the procedure given in 4.2.2.1 and 4.2.2.2. The ϵ-NFA generated by the above mentioned approaches are the same.

Our system uses the automated generated answer key grammar to evaluate the grammar written by the student. In addition to the automated grammar, the system is also provided with the key grammar written by the teacher for evaluation.

For automated evaluation of the grammar written by the student, the automata is generated from all the grammars and checked for equivalence between the automata's. Thus the grammar written by the student should have equivalence between automated answer key grammar or the answer key written by the teacher according to the procedure given in the previous section. The accuracy in evaluation of regular grammar is 100 percentage when the student answer is correct or in-correct.

7. Conclusion

We have proposed our approach for automated evaluation of regular grammars constructed for a regular expression. We have proposed two new approaches namely, Reduced Expression Graph approach and Parse Tree based approach for automation of the answer key generation for a given regular expression. In the Parse Tree based approach we have proposed two novel methods of constructing automata from which regular grammar will be generated. After generation of regular grammar, the key grammar and grammar written by the student is converted to minimized DFA. These minimized automata's are compared for equivalence to evaluate the regular grammar written by the student. Our experimental results prove that the system gives 100 percentage accuracy in automated evaluation of regular grammar.

References

[1] GA Brown, J Bull, M Pendlebury, Regular Expression Generation through Grammatical Evolution, (Routledge, London), 2013.

[2] A Cetinkaya, Assessing student learning in higher education, GECCO '07 Proceedings of the 9th annual conference companion on Genetic and evolutionary computation, Pages 2643–2646, 2007.

[3] L Salmela, J Tarhio, ACE: Automated compiler exercises, Proceedings of the Fourth Finnish/Baltic Sea Conference of Computer Science, 2004.

[4] S Memeti, Automatic Java Code Generator for Regular Expressions and Finite Automata, Student thesis, 2012.

[5] J Zhang, Z Qian, The Equivalent Conversion between Regular Grammar and Finite Automata, Journal of Software Engineering and Applications, Volume 6, Pages:33–37, 2013.

[6] V Tscherter, Exorciser: Automatic Generation and Interactive Grading of Structured Exercises in the Theory of Computation, Student Thesis, 2004.

[7] SV Vucasinovic, PS Stanimirovic, Automatic Generation of regular languages from regular grammars, 8th International Mathematica Symposium Avignon, 2006.

[8] S Bhargava, GN Purohit, Construction of a minimal deterministic finite automaton from a regular expression, International Journal of Computer Applications, Volume 15, 2011.

[9] R Sinha, A Dewangan, Transmutation of Regular Expression to Source Code Using Code Generators, International Journal of Computer Trends and Technology, Volume 31, 2012.

[10] AM Afat, The Different Ways to Describe Regular Languages by Using Finite Automata and the Changing Algorithm Implementation, World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, Volume 8, 2014.

[11] Amit, VK Salar, A Handy Technique for Construction of NFA without Epsilon-transitions from a Regular Expression, The International Journal of Computer Science and Applications, Volume 1, 2012.

[12] K Thompson, Regular expression search algorithm, Communications of the ACM, Volume 11, Pages:419–422, 1968.

[13] JE Hopcroft, R Motwani, JD Ullman, Introduction to Automata Theory, Languages and Computations, 3rd Edition, Pearson Education, 2010.

[14] code http://web.cecs.pdx.edu/ black/CS311/Hein20Section2011.4.1.pdf.

[15] John Martin, Introduction to Languages and the Theory of Computation, 4rd Edition, McGraw-Hill Publisher, 2010.

[16] D. Goswami and K. V. Krishna, Formal Languages and Automata Theory, 2010.