

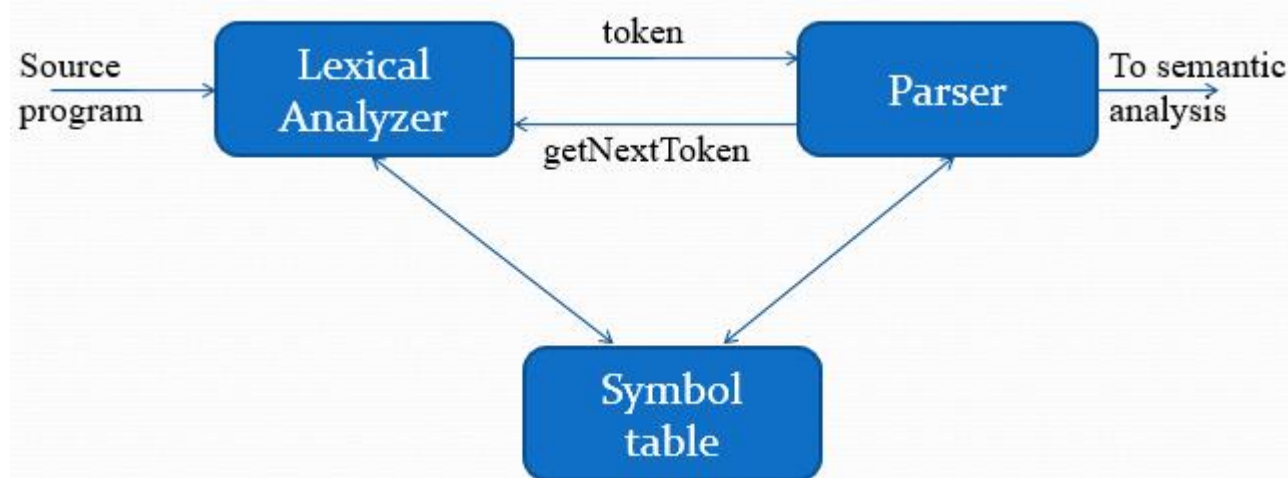


Content of this lecture

- Regular expression, Finite state machine/Finite Automata and their application in Lexical Analyzer

Role of Lexical Analyzer

- Read the source program character by character to produce tokens.
- Remove white space
- Delete comments
- Keep track of line number
- Report error to error handler
- Pass information about the token to symbol table



What are Tokens

- Token is a group of characters described by a pattern.
- Keywords, identifiers, constants and operators are examples of tokens.
- Token has two parts (i) Token type/code (ii) Token Value/Attribute. Some tokens have only type and some have both.
- Token type stores the information about the type of token while token value stores the pointer where value associated with that token is stored.
- A **lexeme** is a sequence of characters in the source program that matches the pattern for a token.

What are Tokens

$E = M * C ** 2$

- <id, pointer to symbol table entry for E>
- <assign-op>
- <id, pointer to symbol table entry for M>
- <mult-op>
- <id, pointer to symbol table entry for C>
- <exp-op>
- <number, integer value 2>

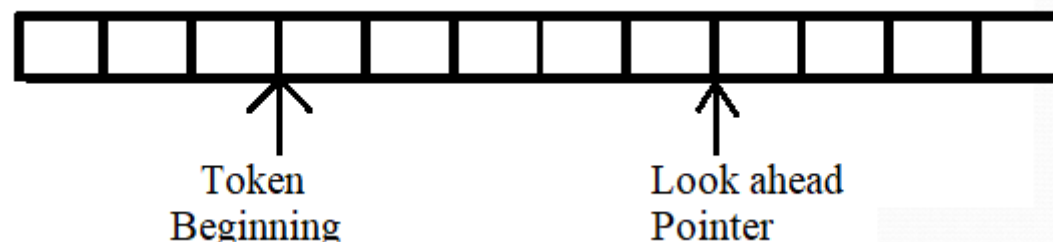
Token	Informal description	Sample lexemes
if	Characters i, f	if
else	Characters e, l, s, e	else
comparison	< or > or <= or >= or == or !=	<=, !=
id	Letter followed by letter and digits	pi, score, D2
number	Any numeric constant	3.14159, 0, 6.02e23
literal	Anything but “ sorrounded by “	“core dumped”

Issue of Design & Implementation of Lexical Analyzer

- There are three aspects of this problem:
 - We need some method to describe the possible tokens which can appear in the input stream. Regular expression is used to achieve this.
 - After deciding what the tokens are, we need some methods to recognize those tokens in the input stream. Finite state machine/Finite automata is used for this purpose.
 - After recognizing tokens we need some mechanism to perform various actions as tokens are recognized.

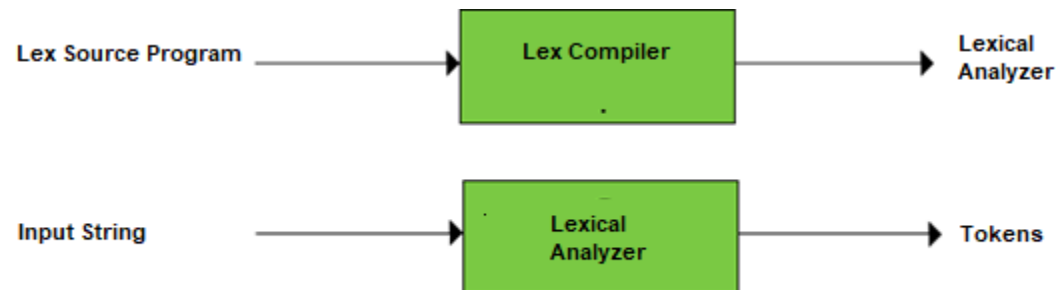
Input Buffering

- To discover token, Lexical Analyzer scans the character of source program one at time.
- Sometimes lexical analyzer may need to look ahead some symbols to decide about the next token.
- So it is desirable for lexical analyzer to read its input from an input buffer. We need to introduce a two buffer scheme to handle large look ahead safely.
- We use a buffer and two pointers. Buffer is divided into two halves. One pointer marks the beginning of token being discovered and second pointer (look ahead pointer) scans ahead of the beginning pointer until the token is discovered.



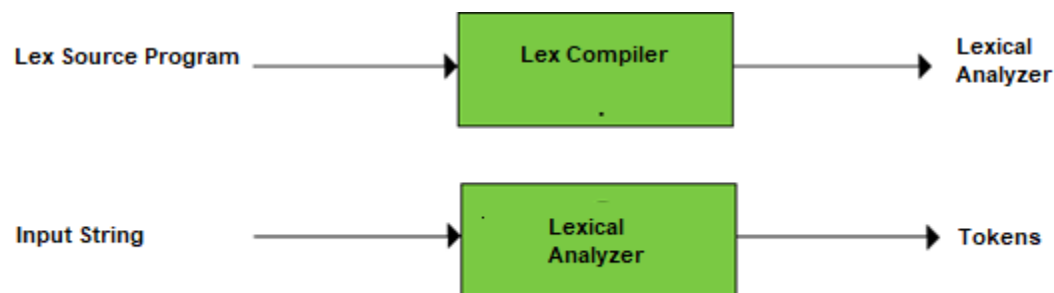
Lex Tool

- Lex is an acronym that stands for "lexical analyzer generator". It is a tool used for automatic generation of lexical analyzer.
- Input to this tool is a program, which is describing the set of regular expression and corresponding actions to be taken on recognition of token.
- Output of Lex is transition table and program to simulate an arbitrary automata expressed as a transition table. This output acts as lexical analyser which will generate tokens.



Lex Tool

- Lex source program consists of 2 parts:
 - Sequence of Auxiliary Definitions: It denotes the statement of the form: $D_i = R_i$, where D_i is shortcut name of regular expression R_i .
 - Sequence of Translation rules: It is a set of rules or actions, which tells lexical analyser what it has to do on finding a token. Rules define the statement of form $p_1 \{action_1\} p_2 \{action_2\} \dots p_n \{action_n\}$. Where p_i describes the regular expression and $action_i$ describes the actions that lexical analyser should takes when pattern p_i matches a lexeme.



Implementation of lexical Analyser

- Using Lex tool, generate lexical analyzer. Input to the Lex tool will be Lex program which is collection of regular expression and the corresponding action to be taken on recognition of token and output will be transition table and program to simulate that NDFA.
- Introducing a new start state and connect with the starting state of all the NDFA's through epsilon.
- Convert the NDFA with epsilon into the minimum state DFA.
- If none of the state of DFA includes any final state of NDFA then control returns to an error condition.
- If final state of the DFA includes more than one final states of NDFA, then final state for the pattern coming first in the translation rules has the priority.

Implementation of lexical Analyser - Numerical

state, for the pattern coming first in the list.

Example 19: Convert the following LEX program into Lexical Analyzer. (Finite automata).

AUXILIARY DEFINITIONS

—
—
—

TRANSLATION RULES

$a \quad \{ \}$
 $abb \quad \{ \}$
 $a^* b^+ \quad \{ \}$

Hint : $\{b^+ \text{ means } bb^+\}$

Ans. 1. Convert the patterns into NFA's

2. Make a Combined NFA

3. Convert NFA to DFA

$A = \epsilon\text{-closure}(0) = \{0, 1, 3, 7\}$

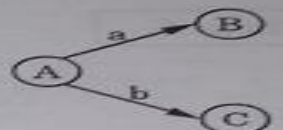
Implementation of lexical Analyser - Numerical

For State A

$T_a = \{$	$-$	2	4	7	$\} = \{2, 4, 7\}$
	$a \uparrow$	$a \uparrow$	$a \uparrow$	$a \uparrow$	
$A = \{$	0	1	3	7	$\}$
	$b \downarrow$	$b \downarrow$	$b \downarrow$	$b \downarrow$	
$T_b = \{$	$-$	$-$	$-$	8	$\} = \{8\}$

$\therefore \epsilon\text{-closure}(T_a)$
 $= \epsilon\text{-closure}(\{2, 4, 7\})$
 $= \{2, 4, 7\} = B$

$\therefore \epsilon\text{-closure}(T_b)$
 $= \epsilon\text{-closure}(\{8\})$
 $= \{8\} = C$



```

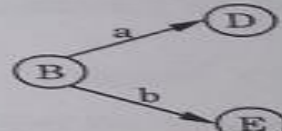
graph LR
    A((A)) -- a --> B((B))
    A -- b --> C((C))
  
```

For State B

$T_a = \{$	$-$	$-$	7	$\} = \{7\}$
	$a \uparrow$	$a \uparrow$	$a \uparrow$	
$B = \{$	2	4	7	$\}$
	$b \downarrow$	$b \downarrow$	$b \downarrow$	
$T_b = \{$	$-$	5	8	$\} = \{5, 8\}$

$\therefore \epsilon\text{-closure}(7) = \{7\} = D$

$\epsilon\text{-closure}(\{5, 8\})$
 $= \{5, 8\} = E$



```


graph LR
    B((B)) -- a --> D((D))
    B -- b --> E((E))
  
```

For State C

$T_a = \{$	$-$	$\} = \phi$
	$a \uparrow$	
$C = \{$	8	$\}$
	$b \downarrow$	
$T_b = \{$	8	$\}$

$\therefore \epsilon\text{-closure}(\phi) = \phi$

$\epsilon\text{-closure}(8) = \{8\} = C$



```

graph LR
    C((C)) -- b --> C
    C -- a --> F((( )))
  
```


Implementation of lexical Analyser - Numerical

For State D

$$T_a = \{ 7 \} = \phi$$

$$D = \{ 7 \}$$

$$T_b = \{ 8 \}$$

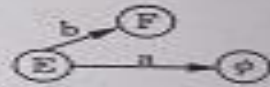
$$\therefore \epsilon\text{-closure}(7) = \{ 7 \} = D \mid \epsilon\text{-closure}(8) = \{ 8 \} = C$$


For State E

$$T_a = \{ \quad \quad \quad \} = \phi$$

$$E = \{ \quad \quad \quad \}$$

$$T_b = \{ \quad \quad \quad \} = \{ 6, 8 \}$$

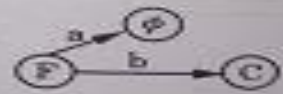
$$\therefore \epsilon\text{-closure}(\phi) = \phi \mid \epsilon\text{-closure}(\{ 6, 8 \}) = \{ 6, 8 \} = F$$


For State F

$$T_a = \{ \quad \quad \quad \} = \phi$$

$$F = \{ \quad \quad \quad \}$$

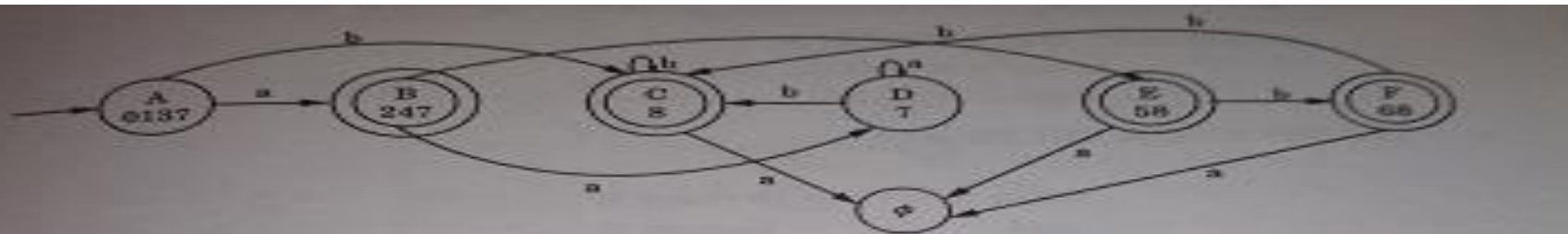
$$T_b = \{ \quad \quad \quad \} = \{ 8 \}$$

$$\therefore \epsilon\text{-closure}(\phi) = \phi \mid \epsilon\text{-closure}(8) = \{ 8 \} = C$$


Combining all transition Diagrams, we get complete DFA. Since state 2, 6, 8 are final states in NFA.

States in NFA having there states i.e. 2, 4, 7, 8, 5, 3, 6, 8, 1, 2, 4, 3

Implementation of lexical Analyser - Numerical



State	a	b	Tokens Recognize
0137	247	8	none
247	7	58	a
8	ϕ	8	a^*b^+
7	7	8	none
58	ϕ	68	a^*b^+
68	ϕ	8	abb
ϕ	ϕ	ϕ	none

Tokens Recognized :

- 0137 → No state in {0,1,3,7} is Final state. Therefore , no token will be Recognized by this state.
- 247 → State 2 in these states is final state \therefore state 2 accepts a in combined NFA. Therefore, 247 will accept a.
- 8 → \therefore 8 is Final State in combined NFA. It accepts a^*b^+ in combined NFA.
- 7 → \therefore 7 is not final state & therefore it accepts nothing.
- 58 → \therefore 8 is Final state but 5 is non-Final state. State 8 accepts a^*b^+ in combined NFA. Therefore 58 will accept a^*b^+
- 68 → Both states 6 & 8 are final states. But 6 accepts abb and 8 accepts a^*b^+ in combined NFA. But abb comes before a^*b^+ in Translation rules given in Question. Therefore state 68 will accept token abb.

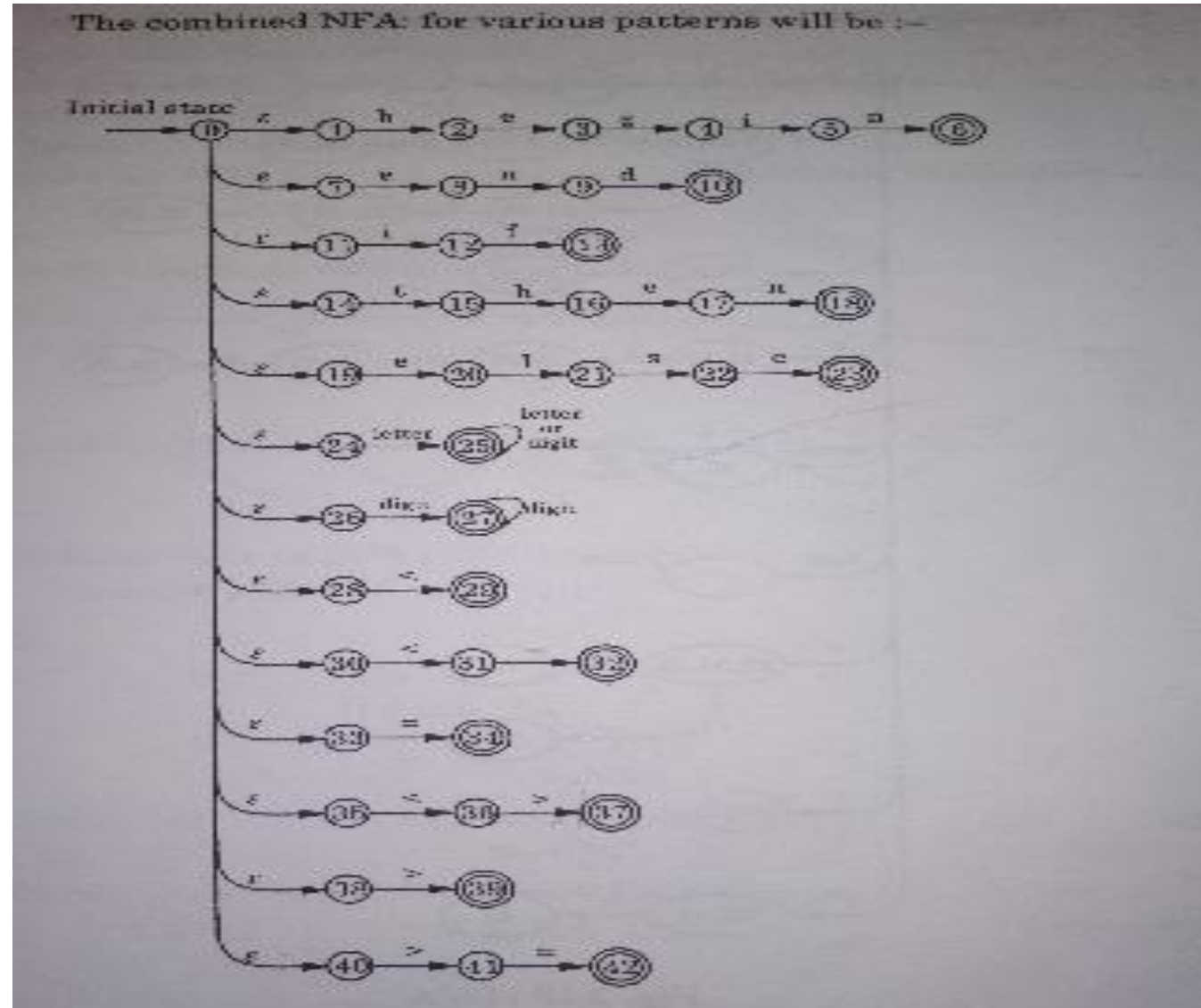
Implementation of lexical Analyser - Numerical

Example 20: Design Lexical Analyzer for following LEX Program.

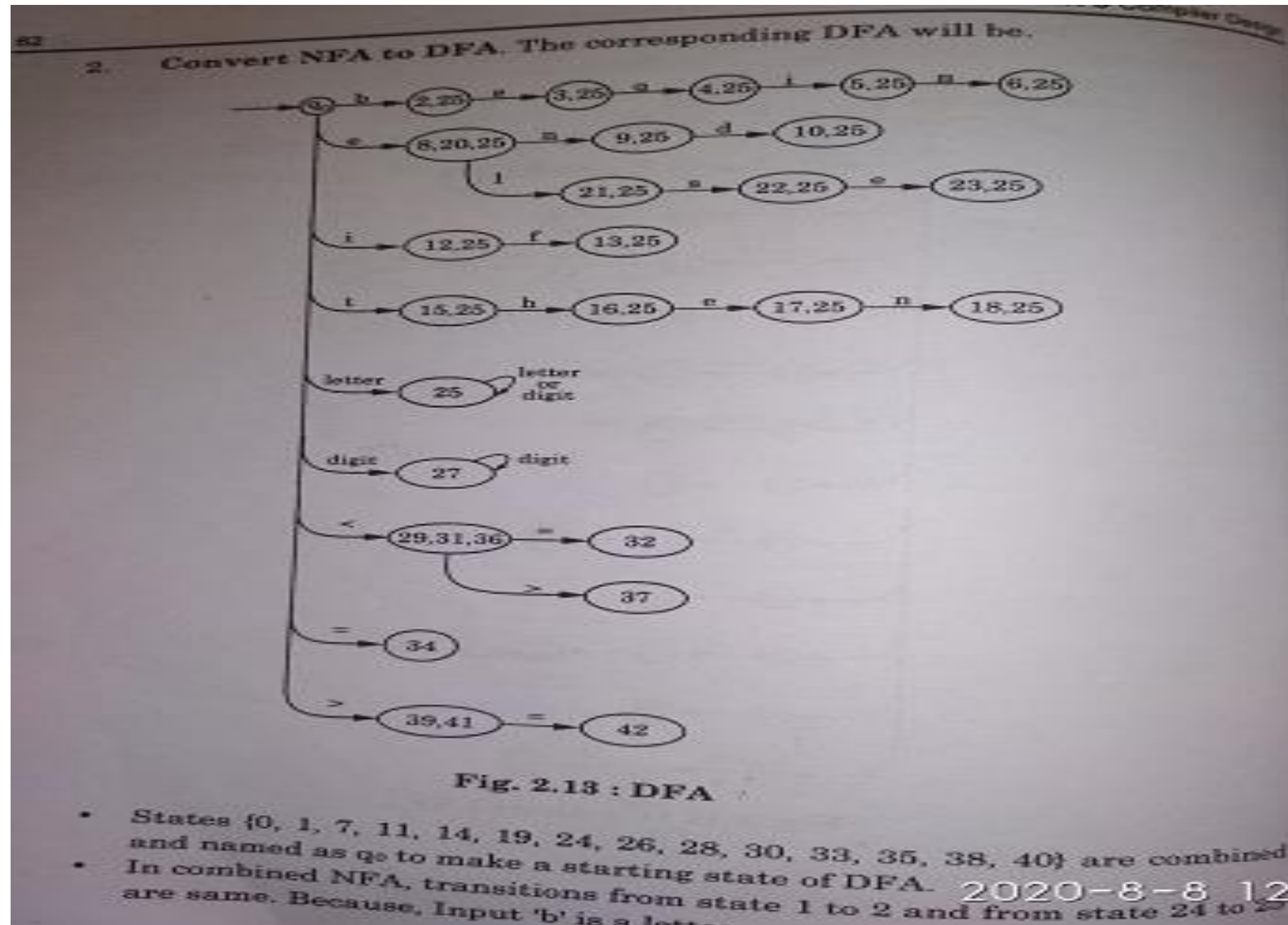
AUXILIARY DEFINITION S	
letter	= A B C Z
digit	= 0 1 2 9

TRANSLATION RULES	
begin	{return 1}
end	{return 2}
If	{return 3}
then	{return 4}
else	{return 5}
letter (letter + digit)*	{value = Install () ;} {return 6}
digit+	{value = Install () ;} {return 7}
<	{value = 1 ;} {return 8}
<=	{value = 2 ;} {return 8}
=	{value = 3 ;} {return 8}
<>	{value = 4 ;} {return 8}
>	{value = 5 ;} {return 8}
>=	{value = 6 ;} {return 8}

Implementation of lexical Analyser - Numerical



Implementation of lexical Analyser - Numerical



THANK YOU



Dr Anurag Jain

Assistant Professor (SG), Virtualization Department, School of Computer Science,
University of Petroleum & Energy Studies, Dehradun - 248 007 (Uttarakhand)

Email: anurag.jain@ddn.upes.ac.in , dr.anuragjain14@gmail.com

Mobile: (+91) -9729371188