First IA IInd Sem. M.Tech., [CSE] 14/2/2020
CSE 5251 Advanced System Software
Answer Scheme

1.A) Suppose a source program contains the assignment statement **position = initial + rate * 60.** Write the Translation of an assignment of the statement, while considering the all the phases of compiler. 2M

1 B) i) The following is a grammar for regular expressions over symbols a and b only, using in place of 1 for union, to avoid conflict with the use of vertical bar as a metasymbol in grammars:

rexpr -> rexpr + rterm | rterm
rterm -> rterm rfactor | rfactor
rfactor -> rfactor * | rprimary
rprimary -> a | b

a) Left factor this grammar.
b) Does left factoring make the grammar suitable for top-down parsing?
c) In addition to left factoring, eliminate left recursion from the original grammar.
d) Is the resulting grammar suitable for top-down parsing? 1.5M

ii) Suppose a situation arises in which the lexical analyzer is unable to proceed because none of the patterns for tokens matches any prefix of the remaining input. Mention the error recovery strategy to handle the aforesaid situation. Also, write the other possible error recovery. 1.5M

2 A) Devise predictive parsers and show the parsing tables. You may left-factor and/or eliminate left-recursion from grammar first.

S -> S + S | S S | (S) | S * | a 2M

2.B) i) The following grammar is proposed to remove the "dangling-else ambiguity".

stmt -> if expr then stmt
| matchedStmt
matchedStmt -> if expr then matchedStmt else stmt
| other

Show that this grammar is still ambiguous. 1.5M

ii) If Σ is an alphabet of basic symbols, then a regular definition is a sequence of definitions. Write the regular definitons for Unsigned numbers (integer or floating point) are strings such as 5280, 0.01234, 6.336E4, or 1.89E-4. 1.5M

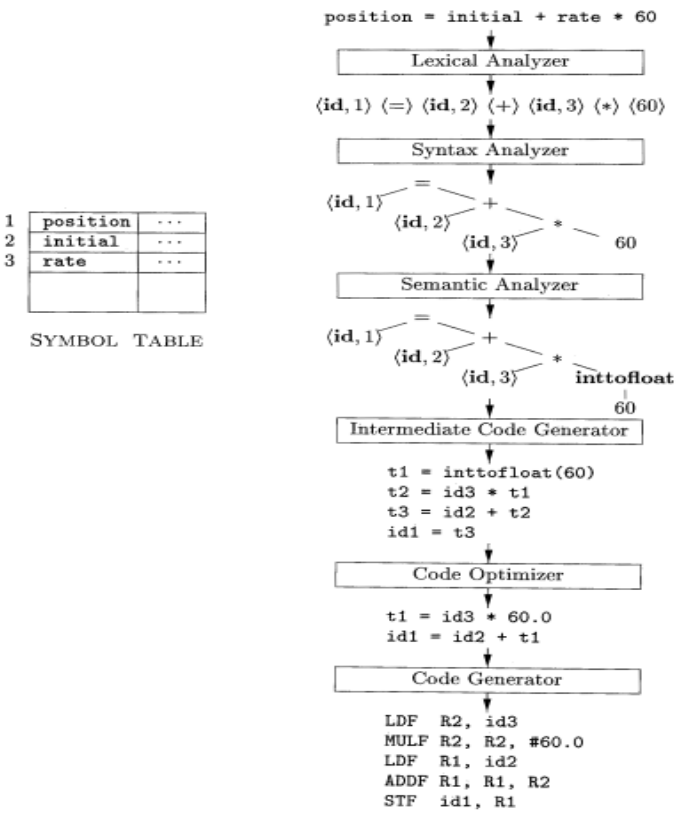
3.A) Construct the combined NFA's for a, abb, and a*b+, then convert it into DFA. Write the Sequence of sets of states entered in NFA, when processing input *aaba. 2M*

3.B) Construct the SLR sets of items for the following(augmented) grammar.

0) S' -> S
1) S -> aB
2) B -> aBAB
3) B -> ε
4) A -> +
5) A -> *

Compute the GOTO function for these sets of items. Show the parsing table for this grammar. Is the grammar SLR? 3M

1.A)



1.B) i)

a) No common factor

b) Not suitable top-down parsing.

c) rexpr -> rterm A
A -> + rterm A | ε
rterm -> rfactor B
B -> rfactor B | ε
rfactor -> rprimary C
C -> * C | ε
rprimary -> a | b

d) Suitable for top-down parsing.

1.B) ii) The simplest recovery strategy is "panic mode" recovery. We delete successive characters from the remaining input, until the lexical analyzer can find a well-formed token at the beginning of what input is left. This recovery technique may confuse the parser, but in an interactive computing environment it may be quite adequate.

Other possible error-recovery actions are:

1. Delete one character from the remaining input.
2. Insert a missing character into the remaining input.
3. Replace a character by another character.
4. Transpose two adjacent characters.

Transformations like these may be tried in an attempt to repair the input. The simplest such strategy is to see whether a prefix of the remaining input can be transformed into a valid lexeme by a single transformation. This strategy makes sense, since in practice most lexical errors involve a single character. A more general correction strategy is to find the smallest number of transformations needed to convert the source program into one that consists only of valid lexemes, but this approach is considered too expensive in practice to be worth the effort.

2.A) step1.Extracting the left common factor

S -> SA | (S) | a

A -> +S | S | *
Further extraction of terminator
S -> SA | T
A -> +S | S | *
T -> (S) | a

Step 2. Eliminate left recursion
i = 1
S -> TB
B -> AB | ε
i = 2
j = 1
A -> +S | TB | *
i = 3
j = 1
No processing required
j = 2
No processing required
Get the final production
S -> TB
B -> AB | ε
A -> +S | TB | *
T -> (S) | a

step3. first && follow
first(T) = [(, a]
first(A) = [+, *] + first(T) =[+, *, (, a]
first(B) = [ε] + first(A) = [ε, +, *, (, a]
first(S) = first(T) = [(, a]

follow(T) = [$, +, *, (, a]
follow(A) = [$, +, *, (,), a]
follow(B) = [$]
follow(S) = [$, +, *, (,), a]

	(	)	+	*	a	$
S	S -> TB				S -> TB	
B	B -> AB		B -> AB	B -> AB	B -> AB	B -> ε
A	A -> TB		A -> +S	A -> *	A -> TB	
T	T -> (S)				T -> a	

2.B) i)Looking at a sample code, we represent the hierarchy of code parsing by indenting
if expr
then
if expr
then matchedStmt
else
if expr
then matchedStmt
else stmt

This code can also be parsed into
if expr
then
    if expr
    then matchedStmt
    else
        if expr
        then matchedStmt
        else stmt

So this is still an ambiguous grammar. the reason is
matchedStmt -> if expr then matchedStmt else stmt
Last of stmt, If it contains else The words can be considered to belong to this stmt.
Can also be considered as belonging to this matchedStmt of the sentence.
ii)

$$
\begin{array}{rcl}
\textit{digit} & \rightarrow & 0 \mid 1 \mid \cdots \mid 9 \\
\textit{digits} & \rightarrow & \textit{digit digit}^{*} \\
\textit{optionalFraction} & \rightarrow & . \ \textit{digits} \mid \epsilon \\
\textit{optionalExponent} & \rightarrow & (\ \mathtt{E} \ (\ + \mid - \mid \epsilon \ ) \ \textit{digits} \ ) \mid \epsilon \\
\textit{number} & \rightarrow & \textit{digits optionalFraction optionalExponent}
\end{array}
$$

3A)

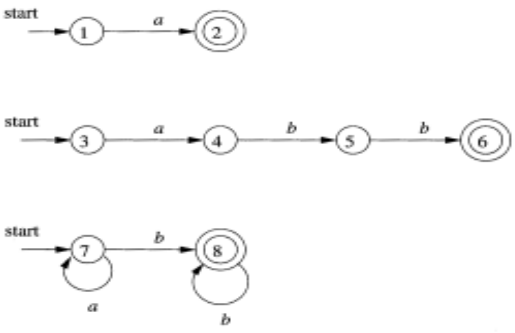


Figure 3.51: NFA's for **a**, **abb**, and **a*b+**

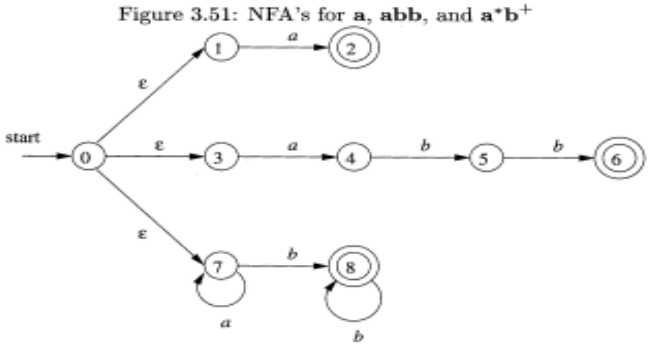


Figure 3.52: Combined NFA

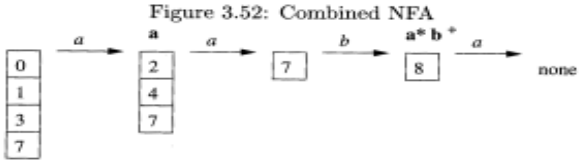


Figure 3.53: Sequence of sets of states entered when processing input *aaba*

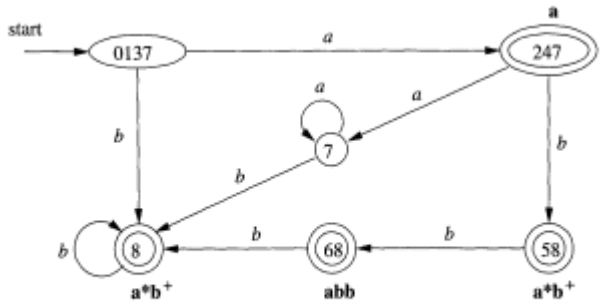
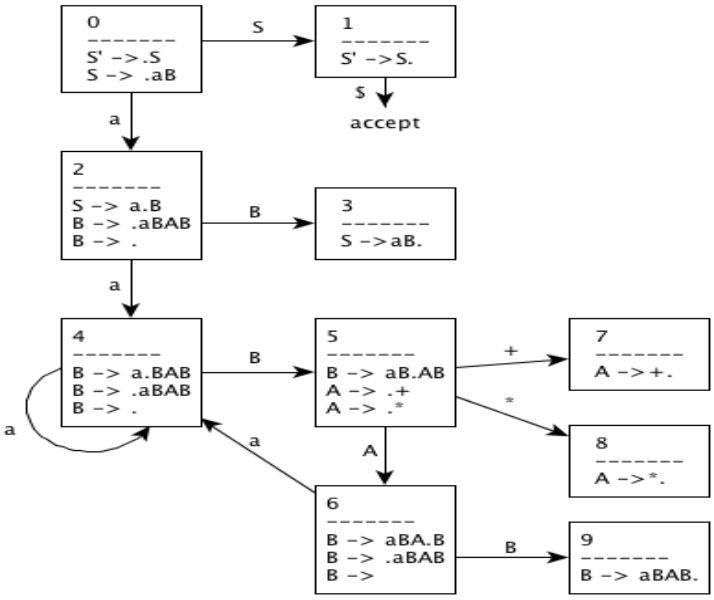


Figure 3.54: Transition graph for DFA handling the patterns **a**, **abb**, and **a*b+**

3B)




```
FOLLOW(S) = [$]
FOLLOW(A) = [a, $]
FOLLOW(B) = [+, * ,$]
```

状态	ACTION				GOTO		
	a	+	*	$	S	A	B
0	s2				s1		
1				acc			
2	s4	r3	r3	r3			s3
3				r1			
4	s4	r3	r3	r3			s5
5		s7	s8			s6	
6	s4	r3	r3	r3			s9
7	r4			r4			
8	r5			r5			
9		r2	r2	r2			

No conflicts, this is obviously an SLR grammar