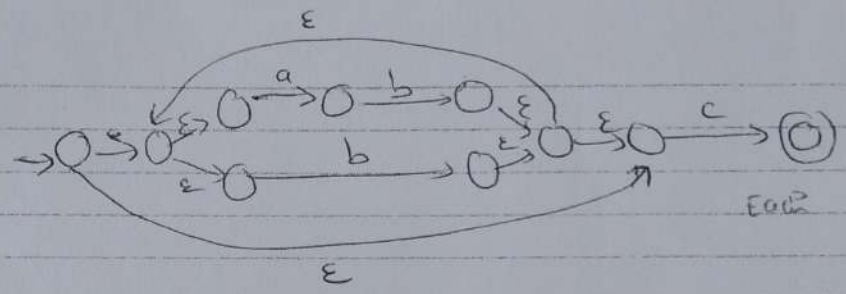
Exam

Duration : 1h and 30 min		
First Name :	Last Name :	Group :

Exercise 1 : **(3pts)**

Tranform the following regular expressions into Finite Automata using the templates seen in the class.

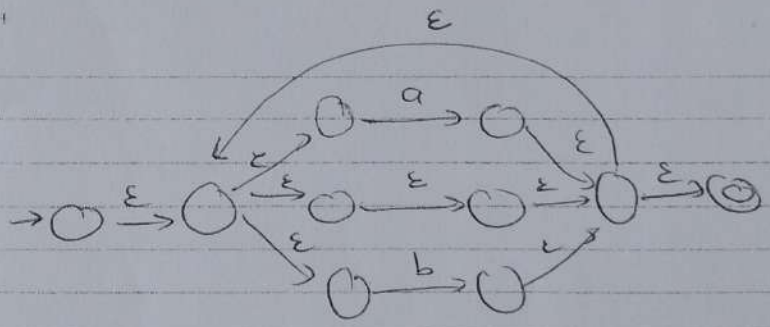
1. (ab|b)* c



1,5

Each error mini 0,25

2. (a| ε | b)*



1,5

Each error is a pattern -0,25

Exercise 2 : **(12 pts)**

Let us consider the grammar G whose production rules are the following :

R={ S→ E; S|ε

E→ E+T|E-T| T

T→ T*F| T/F| F

F→ const| (E)}

The terminals of this grammar are $V_t = \{ ; , +, -, *, /, \text{const}, (,) \}$

1. Is this grammar left-recursive? If it is, eliminate the existing recursion.

* This grammar is left-recursive 0,25

Elimination

$S \rightarrow E;S \mid \varepsilon$
$E \rightarrow TE'$
$E' \rightarrow +TE' \mid -TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid /FT' \mid \varepsilon$
$F \rightarrow \text{Const} \mid (E)$

2,25 pts

2. To write a Top-Down parser, what grammar should we use?

To write a Top-Down parser, we should use the left-recursive free version of the grammar 0,5

0,5 pts

3. Compute the FIRST and the FOLLOW sets.

	FIRST	FOLLOW
S	Const (ε	#
E	Const (	;)
E'	+ - ε	;)
T	Const (	+ - ;)
T'	* / ε	+ - ;)
F	Const (	* / + - ;)

3 pts

0,5 for each line

4. Build the Top-Down parsing table for this grammar.

	;	+	-	*	/	Const	(	)	#
S						E;S	E;S		ε
E						TE'	TE'		
E'	ε	+TE'	-TE'					ε	
T						FT'	FT'		
T'	ε	ε	ε	*FT'	/FT'			ε	
F						Const	(E)		

3,5 pts
each error - 0,25

5. Is this grammar LL(1)? Why?

0,75 pts

This grammar is LL(1). 0,5 Because all the cells of the parsing table contains at most one rule. 0,5

6. Using a stack and a buffer, simulate the syntactic analysis of expression 3*(1+

Input Buffer	stack	action	rule
3*(1+#	#S	production	S→E,°S
3*(1+#	#S,°E	production	E→TE'
3*(1+#	#S,°E'T	production	T→FT'
3*(1+#	#S,°E'T'F	production	F→Const
3*(1+#	#S,°E'T'const	Matching	–
*(1+#	#S,°E'T'	production	T'→*FT
*(1+#	#S,°E'T'F*	Matching	–
(1+#	#S,°E'T'F	production	F→(E)
(1+#	#S,°E'T')E(	Matching	–
1+#	#S,°E'T')E	production	E→TE'
1+#	#S,°E'T')E'T	production	T→FT'
1+#	#S,°E'T')E'T'F	production	F→Const
1+#	#S,°E'T')E'T'const	Matching	–
+#	#S,°E'T')E'T'	production	T'→ε
+#	#S,°E'T')E'	production	E'→+T
+#	#S,°E'T')E'T+	Matching	–
#	#S,°E'T')E'T	ERROR	

0,75 0,75 0,25 0,5

Exos 3

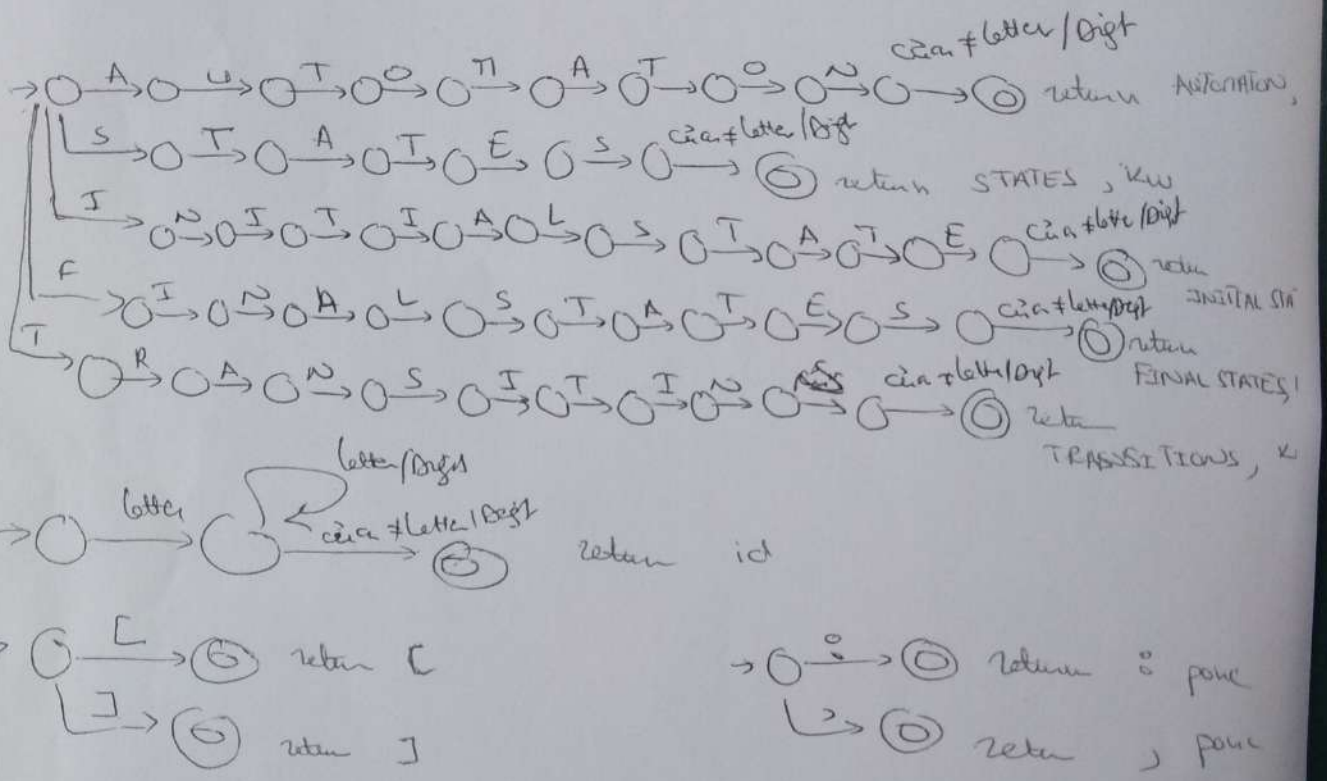
① Keywords → AUTOMATON | STATES | INITIA STATE | FINAL STATES | TRANSITIO

identifier → Letter (Letter | Digit)*

brackets → [|]

punctuation → : | ,

②



③ r_1 <automaton> → AUTOMATON identifier STATES : <states> TRANSITIONS <transitions>

r_2 <states> → [<seq_states>] INITIAL STATE : identifier FINAL STATES : [<seq_states>

r_3 <seq_states> → identifier , <seq_states> | identifier

r_4 <transitions> → [identifier , identifier , identifier] <transitions> | r_6 ε

④ The grammar is not left-recursive.

The grammar is not left-factorise because of rule.

<seq_states> → identifier , <seq_states> | identifier

Elimination

r_1 , r_2 , r_5 , r_6

r_{31} <seq_states> → identifier <seq_states''>

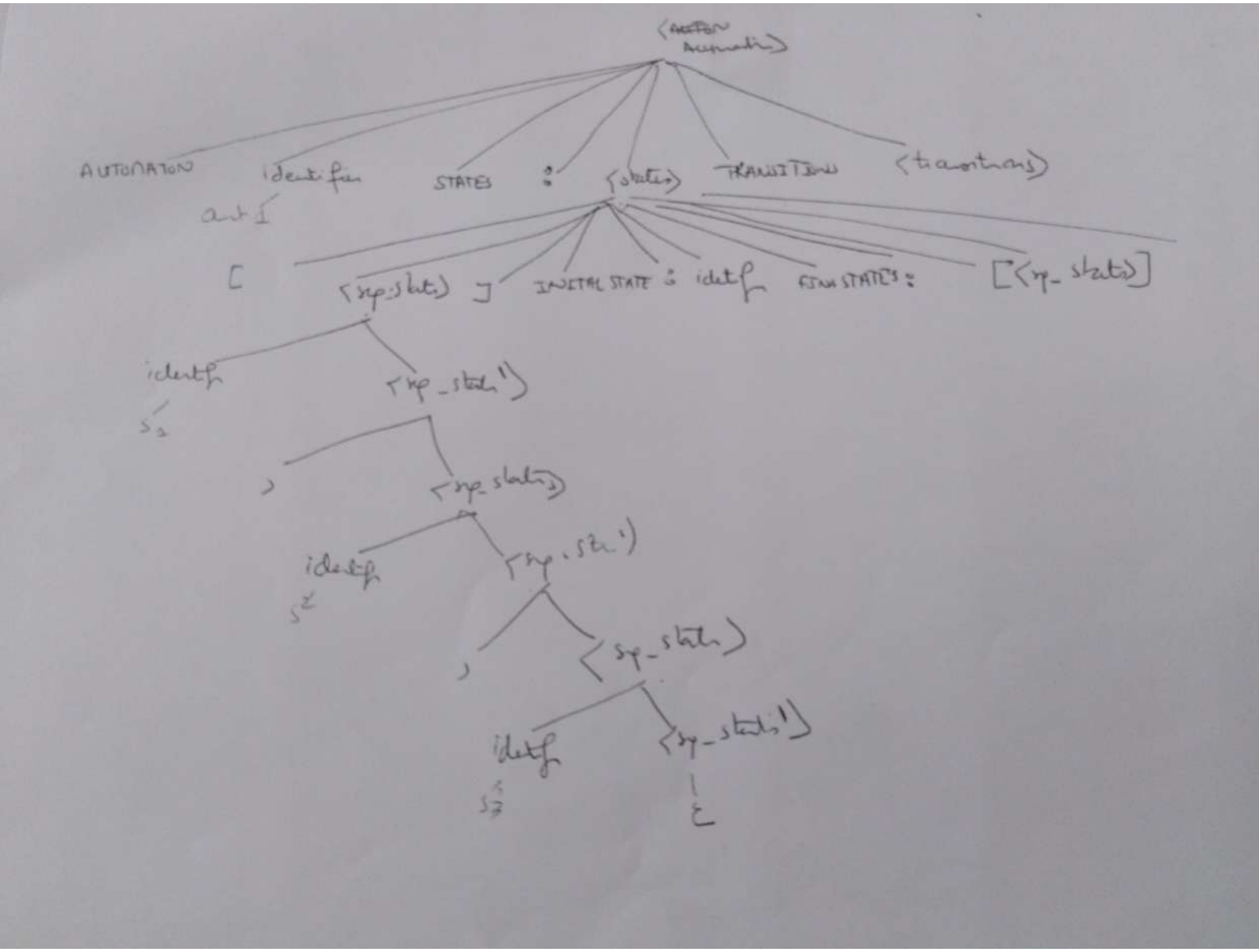
r_{32} <seq_states> → , <seq_states> | ε r_{34}

FIRST | Follow

	FIRST	Follow
<automaton>	AUTOMATON	#
<states>	[	TRANSITIONS
<sp- states>	identifier	]
<sp- states'>	, ε	]
<transitions>	[ε	#

Predictive Table

	AUTOMATON	FINAL STATES	INITIAL STATE	TRANSITIONS	STATES	identf	[	]	,	:	#
<automaton>	r_1										
<states>							r_2				
<- states>						r_{31}					
<p. states'>								r_{33} r_{32}			
<transitions>							r_5			r_6	


AUTOMATON
identifier
STATES
:
TRANSITIONS
[
]
INITIAL STATE
:
FINAL STATES
:
]