CS302-2023-24: Quiz #1

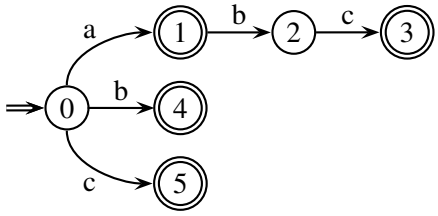
Duration : 60 minutes | Max Marks : 50

- No explanations will be provided. In case of a doubt, make suitable assumptions and justify.
- There are no partial marks other than those indicated in the paper.
- Please write only the final answer in your answer sheet.
- Answers must be in ink. Answers written using pencil will not be evaluated.

---

(1) Consider the following lex script, the associated DFA, and the input string string abdc to be scanned. **(5+6+5=16)**

```
%%
abc         { printf ("Found ABC\n"); }
a           { printf ("Found A\n");   }
b           { printf ("Found B\n");   }
c           { printf ("Found C\n");   }
```



(a) What is the output printed by the scanner for the given input string?

(b) For each lexeme found, provide the following details. Do not give the complete trace. Just give the details when a lexeme is found.

Lexeme found	Input string matched	Input Buffer	Last final state	Current State

Assume that if a character is discarded, it is not echoed on the output. Also, recall that

- a lexeme found is always a prefix of the input string matched which must be a prefix of the input buffer,
- the last final state may be same as the current state, and
- the lexeme found is removed from the input buffer to find the next lexeme.

(c) How many times are the following transitions made in the DFA?

$0 \xrightarrow{a} 1$, $1 \xrightarrow{b} 2$, $2 \xrightarrow{c} 3$, $0 \xrightarrow{b} 4$, and $0 \xrightarrow{c} 5$.

Answer:

(a) The output of the scanner is

```
Found A
Found B
Found C
```

Accepted variations:

```
Found A
Found B
dFound C
```

The description an unmatched character not being ECHOed on output was supposed to be common to all sub-questions but an editing mistake made it ambiguous, hence we will accept this answer too.

(b) The required details are

Lexeme found	Input string matched	Input Buffer	Last final state	Current State
a	ab	abdc	1	2
b	b	bdc	4	4
c	c	c	5	5

The request for allowing

Lexeme found	Input string matched	Input Buffer	Last final state	Current State
a	ab	abdc	1	1

is not accepted because a lexeme is found when the characters are copied to yytext and the associated action is carried out.

(c) The following transitions are made once: $0 \xrightarrow{a} 1$, $1 \xrightarrow{b} 2$, $0 \xrightarrow{b} 4$, and $0 \xrightarrow{c} 5$.
The following transition is never made: $2 \xrightarrow{c} 3$.

---

(2) Since grammar $E \rightarrow E + E \mid E * E \mid id$ is ambiguous, a student tried to create an unambiguous version of the grammar by using the following two options: **(4+4+4=12)**

- Grammar $G_1 : E \rightarrow E + id \mid E * id \mid id$.
- Grammar $G_2 : E \rightarrow id + E \mid id * E \mid id$.

(a) Give all possible rightmost derivations for the input $id + id * id$ for both G_1 and G_2.

(b) Give all possible rightmost derivations for the input $id * id + id$ for both G_1 and G_2.

(c) Assuming that id is replaced by numbers in G_1 and G_2, and the parse tree is evaluated to create a calculator, what are values of the expressions $10 + 20 * 30$ and $10 * 20 + 30$ for the two grammars.

Answer:

(a) Both grammars have a single derivation for $id + id * id$

(i) Grammar $G_1 : E \Rightarrow E * id \Rightarrow E + id * id \Rightarrow id + id * id$

(ii) Grammar $G_2 : E \Rightarrow id + E \Rightarrow id + id * E \Rightarrow id + id * id$

(b) Both grammars have a single derivation for $id * id + id$

(i) Grammar $G_1 : E \Rightarrow E + id \Rightarrow E * id + id \Rightarrow id * id + id$

(ii) Grammar $G_2 : E \Rightarrow id * E \Rightarrow id * id + E \Rightarrow id * id + id$

(c) The result of evaluation of the expressions is

(i) For Grammar G_1: $10 + 20 * 30 = 900$, and $10 * 20 + 30 = 230$.

(ii) For Grammar G_2: $10 + 20 * 30 = 610$, and $10 * 20 + 30 = 500$.

---

(3) Consider Grammar G_1 of Question 2, the corresponding SLR(1) parsing table, and the given input string (the subscripts distinguish between the different occurrences of the terminal id). **(8+6+8=22)**

1 $E \rightarrow E + id$
2 $E \rightarrow E * id$
3 $E \rightarrow id$

	id	$+$	$*$	\$	E
0	$s2$				$c1$
1		$s3$	$s4$	Accept	
2		$r3$	$r3$	$r3$	
3	$s5$				
4	$s6$				
5		$r1$	$r1$	$r1$	
6		$r2$	$r2$	$r2$	

Input

$id_1 + id_2 * id_3\ id_4$ \$

(a) Describe the stack configurations in the following situations. Use the format shown in the class (the stack grows in the direction of the arrow) | Stack → | Remaining Input | Next Action |.

(i) Just before shifting + on the stack.

(ii) Just after shifting id_2 on the stack.

(iii) Just before shifting $*$ on the stack.

(iv) Just after shifting id_3 on the stack.

(b) List all viable prefixes for the grammar.

(c) Give the sets of items for the following states: 3, 4, 5, and 6.

Answer:

(a) The stack configurations are:

(i) Just before shifting + on the stack | \$ 0 E 1 | $+id_2 * id_3\ id_4$ \$ | Shift and go to 3 |

(ii) Just after shifting id_2 on the stack | \$ 0 E 1 + 3 id_2 5 | $*id_3\ id_4$ \$ | Reduce by 1 |

(iii) Just before shifting * on the stack | \$ 0 E1 | $*id_3\ id_4$ \$ | Shift and go to state 4 |

(iv) Just after shifting id_3 on the stack | \$ 0 E1 $*$ 4 id_3 6 | id_4 \$ | Error |

Accepted variation: It does not matter if you use subscripts to *id*'s or not.

(b) The set of all viable prefixes is $\{\varepsilon,\ id,\ E,\ E+,\ E+id,\ E*,\ E*id\}$

Accepted variation. We will accept an answer if it exclude ε because slide 72/94 in parsing-slides-sanyal-part4.pdf suggests that the DFA of LE(0) items consider all states except I_0 as final states. I do not see any reason for this exception and ε should be a considered a viable prefix.

(c) States 3, 4, 5, and 6 all have a single item (and hence only kernel items, no closure items).

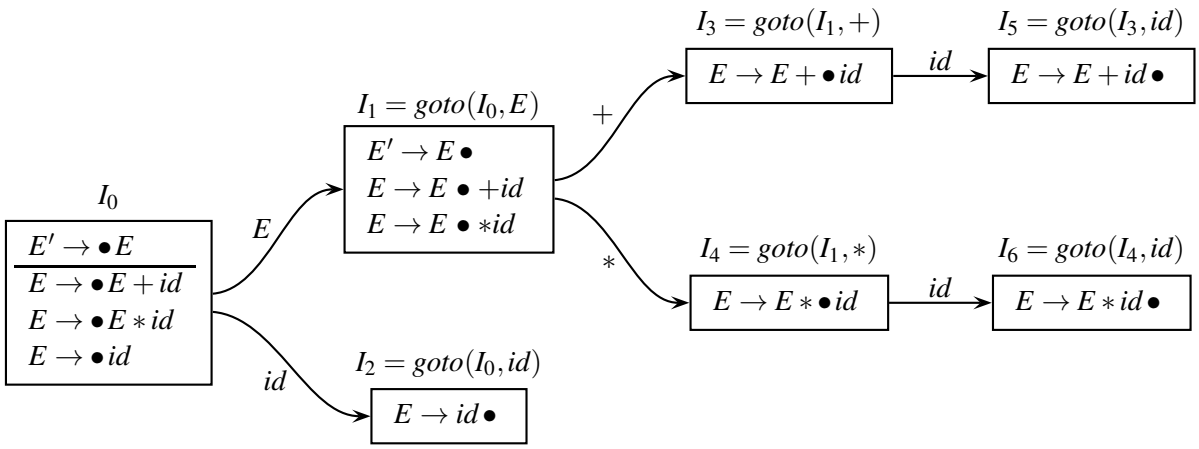
i. State 3: $E \rightarrow E + \bullet id$

ii. State 4: $E \rightarrow E * \bullet id$

iii. State 5: $E \rightarrow E + id\bullet$

iv. State 6: $E \rightarrow E * id\bullet$

The DFA of LR(0) items for the grammar is as shown below (not required in the answer)


I_0
$E' \to \bullet E$
$E \to \bullet E + id$
$E \to \bullet E * id$
$E \to \bullet id$
E
id
$I_1 = goto(I_0, E)$
$E' \to E \bullet$
$E \to E \bullet + id$
$E \to E \bullet * id$
$+$
$*$
$I_2 = goto(I_0, id)$
$E \to id \bullet$
$I_3 = goto(I_1, +)$
$E \to E + \bullet id$
id
$I_5 = goto(I_3, id)$
$E \to E + id \bullet$
$I_4 = goto(I_1, *)$
$E \to E * \bullet id$
id
$I_6 = goto(I_4, id)$
$E \to E * id \bullet$

Best Luck!