



INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Stamp / Signature of the Invigilator

EXAMINATION (Mid Semester)

SEMESTER (Autumn)

Roll Number	9	9	F	B	1	4	6	4	1	Section		Name	Compilers
Subject Number	C	S	3	1	0	0	3	Subject Name					Foolan Barik
Department / Center of the student							Foobarbautic Science and Engineering					Additional Sheets	

Important Instructions and Guidelines for Students

1. You must occupy your seat as per the Examination Schedule/Sitting Plan.
2. Do not keep mobile phones or any similar electronic gadgets with you even in the switched off mode.
3. Loose papers, class notes, books or any such materials must not be in your possession, even if they are irrelevant to the subject you are taking examination.
4. Data book, codes, graph papers, relevant standard tables/charts or any other materials are allowed only when instructed by the paper-setter.
5. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, exchange of these items of any other papers (including question papers) is not permitted.
6. Write on both sides of the answer script and do not tear off any page. **Use last page(s) of the answer script for rough work.** Report to the Invigilator if the answer script has torn or distorted page(s).
7. It is your responsibility to ensure that you have signed the Attendance Sheet. Keep your Admit Card/Identity Card on the desk for checking by the invigilator.
8. You may leave the examination hall for wash room or for drinking water for a very short period. Record your absence from the Examination Hall in the register provided. Smoking and the consumption of any kind of beverages are strictly prohibited inside the Examination Hall.
9. Do not leave the Examination Hall without submitting your answer script to the invigilator. **In any case, you are not allowed to take away the answer script with you.** After the completion of the examination, do not leave the seat until the invigilators collect all the answer scripts.
10. During the examination, either inside or outside the Examination Hall, gathering information from any kind of sources or exchanging information with others or any such attempt will be treated as **'unfair means'**. Do not adopt unfair means and do not indulge in unseemly behavior.

Violation of any of the above instructions may lead to severe punishment.

Signature of the Student

To be filled in by the examiner

Question Number	1	2	3	4	5	6	7	8	9	10	Total
Marks obtained											
Marks Obtained (in words)			Signature of the Examiner					Signature of the Scrutineer			

CS31003 COMPILERS
AUTUMN 2026 – 2027
MID-SEMESTER EXAMINATION
25-SEPTEMBER-2026, 3:00PM – 5:00PM
MAXIMUM MARKS: 100

Instructions to students

- Write your answers in the question paper itself.
- Answer **all** questions.
- Write only in the blank spaces provided in the question paper. Use the empty pages at the end for rough work. Please avoid taking or using supplementary sheets. If you continue some answer to the Rough-Work pages, please supply an appropriate pointer (with the rough-page number) in the space of the corresponding question.
- Do not write anything on this page. Questions start from the next page (Page 3).
- If you need to make any assumptions in some questions, write them clearly in your respective answers.

1. [Lexical Analysis]

(a) Typically, an IP address consists of 4 integers separated by three dots (.) in between, where each integer is in the range [0–255] (for example, 123.45.6.78). Leading 0's are not allowed in the integer components. Spaces too are not allowed anywhere in an IP address. Write below a regular expression in the LEX format for a valid IP address. [4]

```
num          [0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]
ipaddr       {num}\.{num}\.{num}\.{num}
```

(b) For your lab assignments, we restrict the IP addresses so that the assignments can be accessed only from the lab machines. There is a "restriction-access" field in Moodle for setting the allowed IP addresses. An IP address can be specified as in Part (a). A range of IP addresses can be specified using BEGIN/END in the last component. For example, 11.22.33.4/55 indicates the range of IP addresses 11.22.33.4 through 11.22.33.55. Multiple individual IP addresses and/or ranges are allowed, and are separated by commas. The empty list is allowed too (indicating no restriction). However, no spaces are allowed anywhere. For example, the list 210.98.76.5,11.22.33.4/55,123.45.6.78 allows access only to the individual IP addresses 210.98.76.5 and 123.45.6.78, and the range of IP addresses 11.22.33.4 through 11.22.33.55. Write below a regular expression in the LEX format for this access field. Do not redefine, but use the pattern for IP addresses as in your solution of Part (a). [4]

```
ipallowed    {ipaddr}(\/{num})?
accesslist   ({ipallowed}(\/{num})?)*
```

(c) Files can be copied across machines by the Linux utility scp. A remote file or directory name is specified in the format <username>@<IPaddress>:<path>. For example, the shell command

```
scp compiler.txt foobar@10.5.20.15:/users/foobar/test_dump/
```

copies the file compiler.txt from the current directory of the local machine to the folder indicated by the absolute path /users/foobar/test_dump/ of the user called foobar located at the machine with IP address 10.5.20.15. The last slash is optional. Assume the following.

- <username> follows the same naming convention as C variables.
- <IPaddress> is defined as in Part (a).
- <path> always refers to a directory name.
- <path> may contain nested directory names separated by forward slashes (/), where each directory follows the same naming convention as C variables. The string may or may not start with a slash. If it does not start with a slash, then the path is relative to the home directory of the user. This means that an empty string is allowed as <path>. However, there must not be any space anywhere in the string <path>.

Write below a regular expression in the LEX format for a remote address of the form <username>@<IPaddress>:<path>. Use the pattern for IP addresses as in your solution of Part (a) (do not redefine the pattern). [4]

```
name         [a-zA-Z_][a-zA-Z0-9_]*
path         \/?({name}\/)*{name}?
remoteaddr   {name}@{ipaddr}:{path}
```

2. [LL(1) and SLR(1) Parsing]

Let us revisit the language of Class Test 1 (CT1). The grammar given in CT1 was neither LL(1) nor SLR(1). An alternative grammar for the same language is given below. S is the start symbol, and the terminals are a , b , and c .

$$\begin{aligned} S &\rightarrow aS \mid bS \mid T \\ T &\rightarrow aTa \mid bTb \mid cU \\ U &\rightarrow cU \mid \varepsilon \end{aligned}$$

(a) In the left column of the following table, write the FIRST and FOLLOW sets for the non-terminals of this grammar. Supply briefly justifications for each in the right column of the table. [6]

Set	Brief justification
$\text{FIRST}(S) = \{a, b, c\}$	Same as $\{a\} \cup \{b\} \cup \text{FIRST}(T)$
$\text{FIRST}(T) = \{a, b, c\}$	The three T -productions begin with a , b , and c , respectively
$\text{FIRST}(U) = \{c, \varepsilon\}$	Evident from the two U -productions
$\text{FOLLOW}(S) = \{\$ \}$	Start symbol, no production body has S followed by anything
$\text{FOLLOW}(T) = \{a, b, \$ \}$	Production bodies aTa and bTb , and T inherits $\text{FOLLOW}(S)$
$\text{FOLLOW}(U) = \{a, b, \$ \}$	U inherits $\text{FOLLOW}(T)$

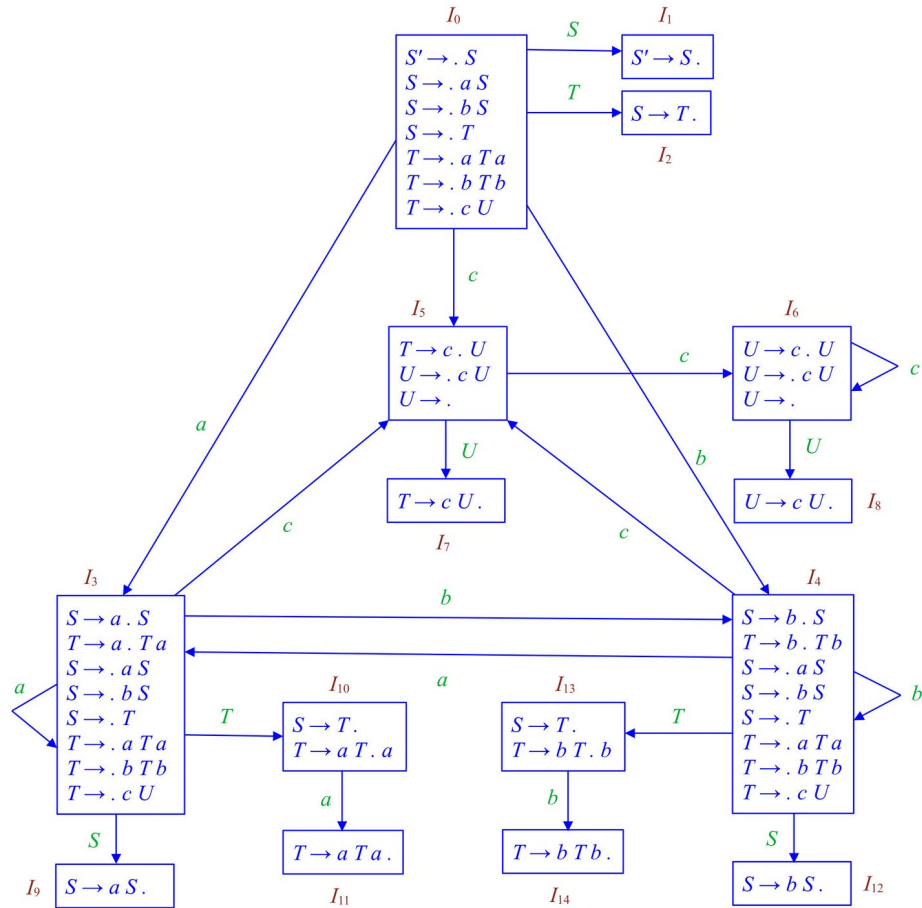
(b) From the above table, justify below that the grammar is not LL(1). Use only the definition of LL(1) grammars to identify all the conflicts, and mention what types of conflicts these are. Do **not** prepare the LL(1) parsing table. [2]

The grammar has two FIRST-FIRST conflicts.

$\text{FIRST}(aS)$ and $\text{FIRST}(T)$ both contain a (so the productions $S \rightarrow aS$ and $S \rightarrow T$ are conflicting).
 $\text{FIRST}(bS)$ and $\text{FIRST}(T)$ both contain b (so the productions $S \rightarrow bS$ and $S \rightarrow T$ are conflicting).

(c) Draw below the LR(0) automaton of the given grammar.

[10]



(d) Justify below that the grammar is SLR(1) but not LR(0). To show it is not LR(0), identify all shift-reduce conflicts in the automaton of Part (c). Then explain how the FOLLOW sets resolve these conflicts, and make it SLR(1). [2]

The states that have shift-reduce conflicts are given below. In all the cases, using the FOLLOW sets resolves the conflicts.

States I_5 and I_6 : $U \rightarrow \cdot c U$ and $U \rightarrow \cdot$ [resolved because c is not in $\text{FOLLOW}(U)$]

States I_{10} and I_{13} : $S \rightarrow T \cdot$ and $a \rightarrow a T \cdot a$ (or $b \rightarrow b T \cdot b$) [resolved because neither a nor b is in $\text{FOLLOW}(S)$]

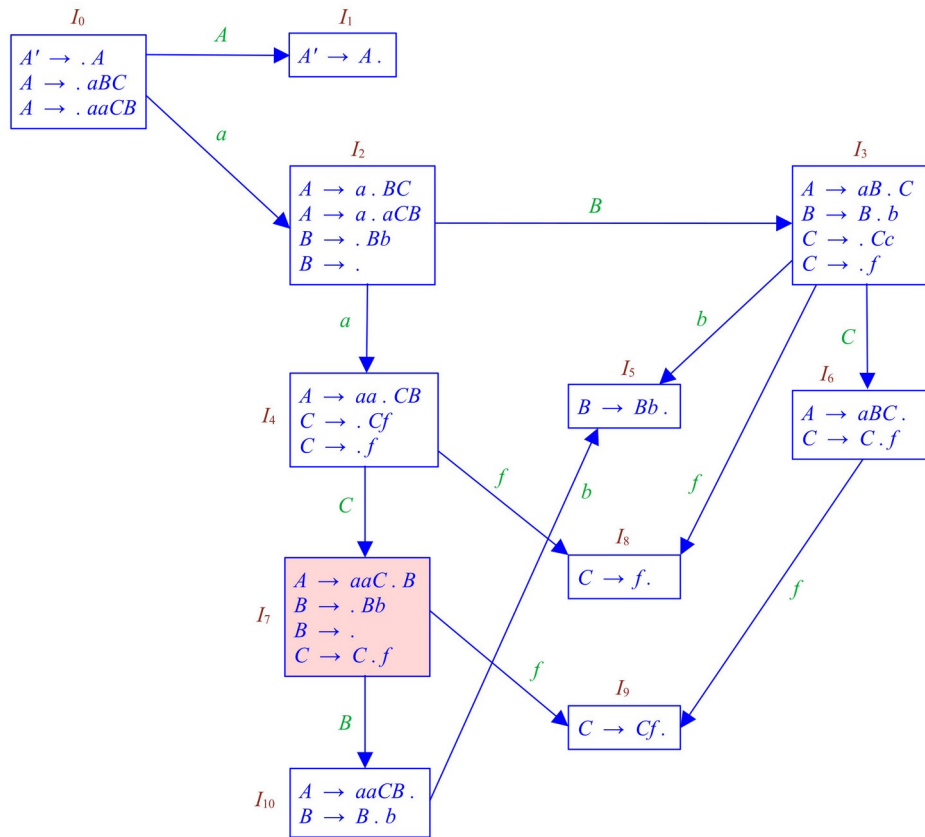
3. [SLR(1), LR(1), and LALR(1) parsing]

A foobarnautic program contains a block B (may be empty) of barfulation steps b , and a block C (must be non-empty) of foobrilation steps f . If the block B appears before the block C , there is a single articulation step a at the beginning. However, if C appears before B , there are two articulation steps at the beginning. In the following grammar, the individual steps a, b, f are treated as terminals. The non-terminals are A (anatomy of a foobarnautic program; the start symbol), and the symbols for the blocks B and C .

$$\begin{aligned} A &\rightarrow aBC \mid aaCB \\ B &\rightarrow Bb \mid \varepsilon \\ C &\rightarrow Cf \mid f \end{aligned}$$

(a) Draw below the LR(0) automaton for the grammar.

[10]



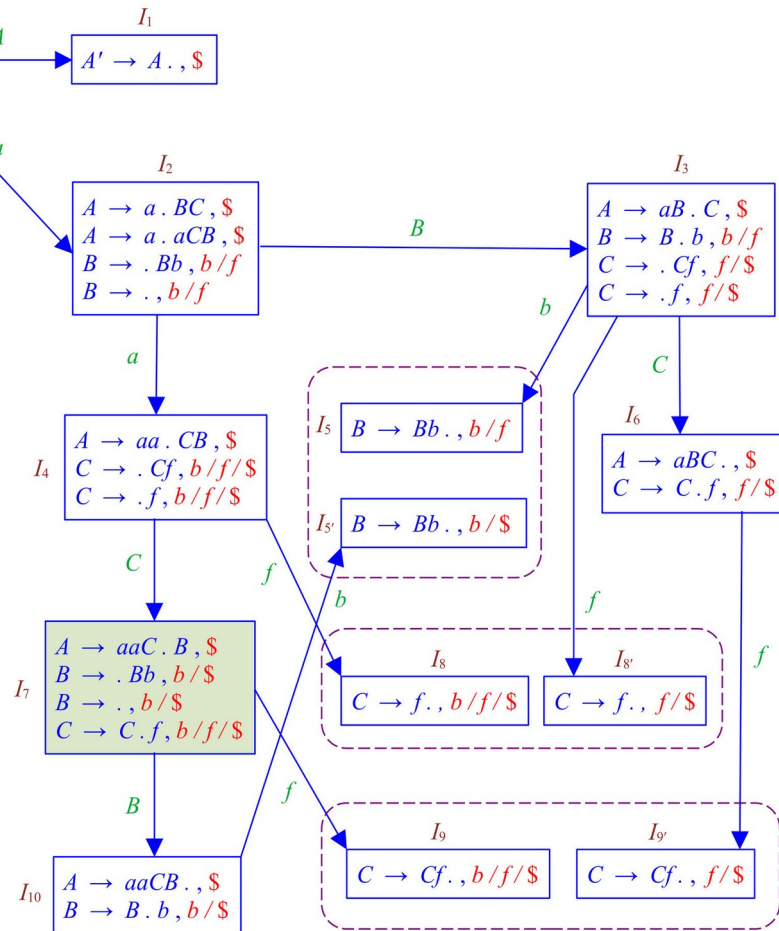
(b) Draw below the LR(1) automaton of the grammar (reproduced here for your convenience).

[12]

$$A \rightarrow aBC \mid aaCB$$

$$B \rightarrow Bb \mid \varepsilon$$

$$C \rightarrow Cf \mid f$$



In the respective spaces below the statements of the following three parts, supply brief justifications. Do **not** prepare any parsing table. You will get no credit(s) if your construction(s) in Part (a) and/or Part (b) is/are faulty.

(c) From the LR(0) automaton of Part (a), deduce that the given grammar is not SLR(1). Identify all the states of the automaton, that have conflicts. Mention what types of conflict these are. [2]

The LR(0) state I_7 has a shift-reduce conflict $B \rightarrow \cdot$ and $C \rightarrow C \cdot f$ on input f . Since $A \rightarrow aBC$, every terminal in $\text{FIRST}(C)$ is in $\text{FOLLOW}(B)$. In particular, since $C \rightarrow f$ is a production, f is in $\text{FIRST}(C)$ (and so in $\text{FOLLOW}(B)$).

Note: The reduction $B \rightarrow \cdot$ does not have a conflict with the shift in the item $A \rightarrow a \cdot aCB$ in State I_2 . This is because a is not in $\text{FOLLOW}(A) = \{ \$ \}$.

(d) From the LR(1) automaton of Part (b), deduce that the given grammar is (canonical) LR(1). In particular, explain how the conflicts identified in Part (c) are resolved by lookahead symbols. [2]

The conflict in the state I_7 is resolved in the LR(1) automaton. This is because the reduction $B \rightarrow \cdot$ is allowed only at the end of input (that is, if the next input symbol is $\$$) or if the next input symbol is b . In particular, the reduction $B \rightarrow \cdot$ is no longer allowed if the next input symbol is f . In that case, the shift $C \rightarrow C \cdot f$ is the only allowed action.

(e) Explain which states of the LR(1) automaton of Part (b) can be merged to form the LALR(1) automaton of the given grammar. Justify whether the given grammar is LALR(1). Do **not** draw the LALR(1) automaton. [2]

The following pairs of states can be merged: $(I_5 \text{ and } I_{5'})$, $(I_8 \text{ and } I_{8'})$, and $(I_9 \text{ and } I_{9'})$. Each of these merged states has only one item (implying a reduction action). That is, no reduce-reduce conflicts are introduced in the LALR(1) automaton, so the given grammar is LALR(1) too.

4. [Syntax-directed definition (SDD)]

A foobar expression (denoted by E) consists of foobar objects (the operands (terminals), denoted by **obj**), and two binary operators foo and bar . These two operators have the following properties.

- The foo operator has the symbol $\diamond$, and is left-to-right associative.
- The bar operator has the symbol $\boxtimes$, and is right-to-left associative.
- The foo operator $\diamond$ has higher precedence than the bar operator $\boxtimes$.
- The default associativity and precedence rules can be altered by using parentheses.

For example, let a, b, c, d, e, f be foobar objects. The unparenthesized expression $a \diamond b \boxtimes c \boxtimes d \diamond e \diamond f$ is to be interpreted as $((a \diamond b) \boxtimes (c \boxtimes ((d \diamond e) \diamond f)))$, whereas the parenthesized expression $a \diamond (b \boxtimes c \boxtimes d) \diamond (e \diamond f)$ is to be interpreted as $(a \diamond (b \boxtimes (c \boxtimes d))) \diamond (e \diamond f)$.

(a) Since $\diamond$ has higher precedence than $\boxtimes$, each term T in E is a foo of foctors F , and E itself is a bar of terms. This part uses the following left-recursive grammar for foobar expressions. Your task is to write an SDD to evaluate foobar expressions. Each non-terminal in the grammar has an attribute val to store its final synthesized value (a foobar object). Fill in the right column of the following table with the semantic rules of your SDD. Use the symbols $\diamond$ and $\boxtimes$ on the right side of the semantic rules. For **obj**'s, use the attribute $lexval$. Use two inherited attributes in you SDD: $topmost$ (a flag), and inh (a foobar object). Since $\boxtimes$ is right-to-left associative but our grammar is left-recursive, a dummy start symbol S is introduced. The $topmost$ flag is set only for every topmost E in the parse tree. Setting this for the outermost E is the only job of S . Do **not** use any attribute other than $lexval$, val , $topmost$, and inh . No lists are permitted. [8]

Production	Semantic Rules
$S \rightarrow E$	$E.topmost = 1$; $S.val = E.val$;
$E \rightarrow E_1 \boxtimes T$	$E_1.topmost = 0$; if $(E.topmost)$ $E_1.inh = T.val$; else $E_1.inh = T.val \boxtimes E.inh$; $E.val = E_1.val$;
$E \rightarrow T$	if $(E.topmost)$ $E.val = T.val$; else $E.val = T.val \boxtimes E.inh$;
$T \rightarrow T_1 \diamond F$	$T.val = T_1.val \diamond F.val$;
$T \rightarrow F$	$T.val = F.val$;
$F \rightarrow \mathbf{obj}$	$F.val = \mathbf{obj.lexval}$;
$F \rightarrow (E)$	$E.topmost = 1$; $F.val = E.val$;

(b) Justify below which type of SDD is designed by you in Part (a): S-attributed, L-attributed, or neither (no credits if your SDD is faulty). [2]

The SDD is neither S-attributed nor L-attributed. The attribute $topmost$ is inherited (from the parent). In the semantic rule for the production $E \rightarrow E_1 \boxtimes T$, the computation of $E_1.inh$ depends on the attribute $T.val$ from the right sibling.

(d) Rewrite in the table below (left column) the grammar of Part (a) using the standard procedure for the elimination of left recursion. Also rewrite the semantic rules for the new productions (right column). The dummy start symbol S is no longer required, because E itself can now be the start symbol. Use only one inherited attribute inh (its value is a foobar object). Use the synthesized attributes $lexval$ and val as in Part (a). [10]

Production	Semantic rules
$E \rightarrow T E'$	$E'.inh = T.val ; \quad E.val = E'.val ;$
$E' \rightarrow \bowtie T E_1'$	$E_1'.inh = T.val ; \quad E'.val = E'.inh \bowtie E_1'.val ;$
$E' \rightarrow \varepsilon$	$E'.val = E'.inh ;$
$T \rightarrow F T'$	$T'.inh = F.val ; \quad T.val = T'.val ;$
$T' \rightarrow \diamond F T_1'$	$T_1'.inh = T'.inh \diamond F.val ; \quad T'.val = T_1'.val ;$
$T' \rightarrow \varepsilon$	$T'.val = T'.inh ;$
$F \rightarrow \mathbf{obj}$	$F.val = \mathbf{obj.lexval} ;$
$F \rightarrow (E)$	$F.val = E.val ;$

(e) Justify below which type of SDD is designed by you in Part (d): S-attributed, L-attributed, or neither (no credits if your SDD is faulty). [2]

This SDD is L-attributed. The productions for E' needs an inherited attribute from its parent. However, T' needs an inherited attribute, the computation of which requires the value of the left sibling only.

Note: We can write the productions for E in the following simpler manner. This eliminates the inherited attributes in the productions of E' .

$E \rightarrow T \bowtie E_1$	$E.val = T.val \bowtie E_1.val ;$
$E \rightarrow T$	$E.val = T.val ;$

