INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR	
	Stamp / Signature of the Invigilator

EXAMINATION (Mid Semester)	SEMESTER (Autumn)

Roll Number										Section		Name	
Subject Number	C	S	3	1	0	0	3	Subject Name	*Compilers*				
Department / Center of the Student								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 or 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 2024–2025

Mid-Semester Test

18–September–2024 | 09:00am–11:00am | Maximum marks: 80

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions.*]

Do not write anything on this page.

1. [LL(1) grammars]

In this exercise, we deal with the language $L = \{a^m b^{m+n} \mid m,n \geqslant 0\}$ over the alphabet $\{a,b\}$.

(a) A grammar for L is given below. Here, A is the start symbol, and B is another nonterminal symbol.

$$\begin{array}{lcl} A & \to & aAb \mid B \\ B & \to & bB \mid \varepsilon \end{array}$$

Determine the FIRST and FOLLOW values for the nonterminals. Supply justifications. **(4)**

	Justification
FIRST(A) = $\{a, b, \varepsilon\}$	FIRST(A) = FIRST$(aAb) \cup$ FIRST$(B) = \{a\} \cup \{b, \varepsilon\}$
FIRST(B) = $\{b, \varepsilon\}$	FIRST(B) = FIRST$(bB) \cup$ FIRST$(\varepsilon) = \{b\} \cup \{\varepsilon\}$
FOLLOW(A) = $\{\$, b\}$	Since A is the start symbol, \$ is in FOLLOW(A). Moreover, b is in FOLLOW(A) because of the production $A \to aAb$.
FOLLOW(B) = $\{\$, b\}$	No rule has any symbol after B. The production $A \to B$ implies that everything in FOLLOW(A) is in FOLLOW(B).

Prepare the LL(1) parsing table for the grammar given above. **(4)**

Non-terminal	Input symbol a	b	\$
A	$A \to aAb$	$A \to B$	$A \to B$
B		$B \to bB$ $B \to \varepsilon$	$B \to \varepsilon$

From the parsing table, conclude that the grammar given above is not LL(1). **(1)**

Solution The entry $M[B,b]$ contains multiple productions.

(b) Propose an LL(1) grammar for the language L. Write below only your grammar and its start symbol (and nothing else). There is no need to explain how you came up with the grammar. **(4)**

Solution Introduce a new start symbol S. Then, use the following productions.

$$\begin{array}{lcl} S & \to & AB \\ A & \to & aAb \mid \varepsilon \\ B & \to & bB \mid \varepsilon \end{array}$$

By constructing the FIRST, FOLLOW, and LL(1) parsing tables, prove that your grammar is indeed LL(1). No credit for any other proof method. **(7)**

Solution We have

$$\begin{aligned}
\mathrm{FIRST}(S) &= \mathrm{FIRST}(AB) = \mathrm{FIRST}(A) = \{a\} \\
\mathrm{FIRST}(A) &= \mathrm{FIRST}(aAb) \cup \mathrm{FIRST}(\varepsilon) = \{a, \varepsilon\} \\
\mathrm{FIRST}(B) &= \mathrm{FIRST}(bB) \cup \mathrm{FIRST}(\varepsilon) = \{b, \varepsilon\}
\end{aligned}$$

$$\begin{aligned}
\mathrm{FOLLOW}(S) &= \{\$\} \\
\mathrm{FOLLOW}(A) &= \{b\} \cup \Big(\mathrm{FIRST}(B) - \{\varepsilon\}\Big) \cup \mathrm{FOLLOW}(S) = \{b, \$\} \\
\mathrm{FOLLOW}(B) &= \mathrm{FOLLOW}(S) = \{\$\}
\end{aligned}$$

The LL(1) parsing table is given below.

	a	b	$\$$
S	$S \to AB$	$S \to AB$	$S \to AB$
A	$A \to aAb$	$A \to \varepsilon$	$A \to \varepsilon$
B		$B \to bB$	$B \to \varepsilon$

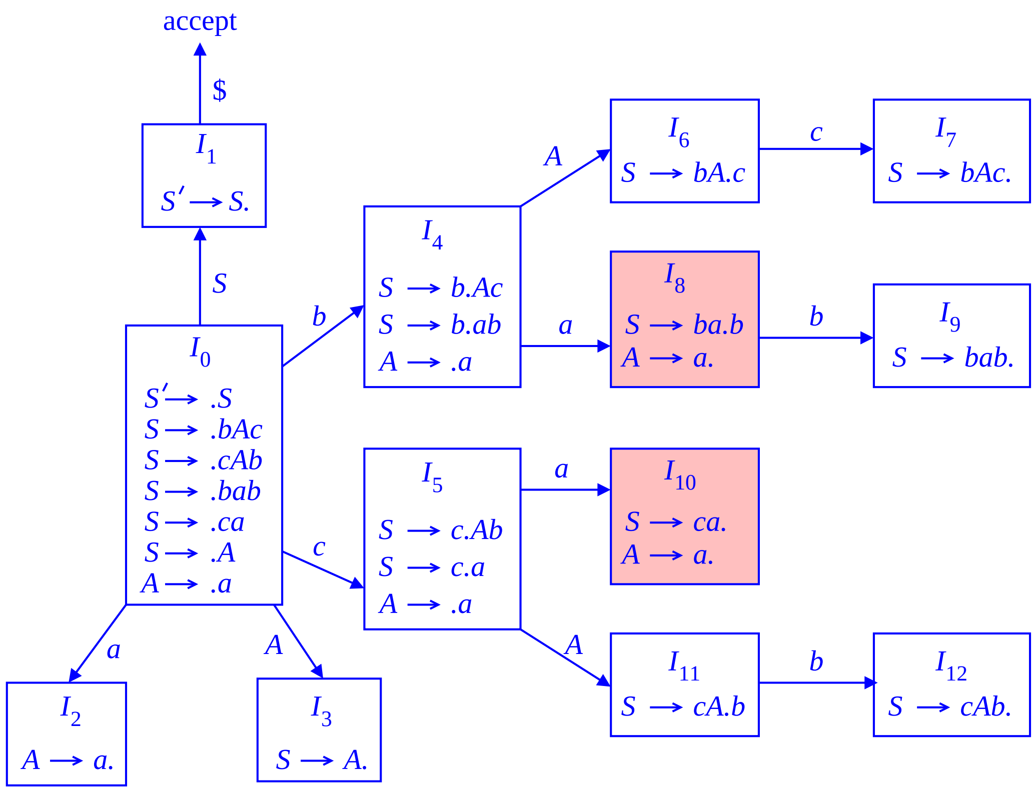
Since no entry in the parsing table has multiple productions, the grammar in concern is LL(1).

2. [Grammars for bottom-up parsing]

Consider the following grammar with terminal symbols a, b, c, with nonterminal symbols S, A, and with the start symbol S.

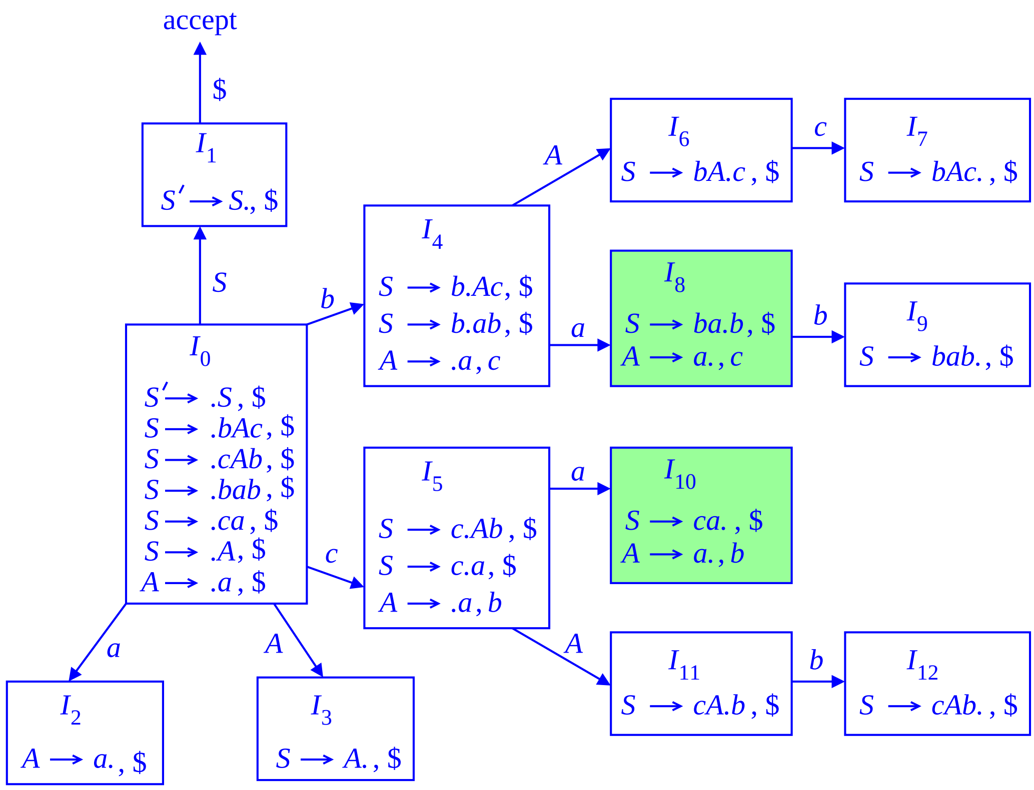
$$
\begin{aligned}
S &\rightarrow bAc \mid cAb \mid bab \mid ca \mid A \\
A &\rightarrow a
\end{aligned}
$$

(a) Draw the complete LR(0) automaton for the grammar. Name the states as $I_0, I_1, I_2, \ldots$. **(6)**



(b) Draw the complete LR(1) automaton for the grammar. Again name the states as $I_0, I_1, I_2, \ldots$. For your convenience, the grammar is given once more below. **(7)**

$$
\begin{aligned}
S &\rightarrow bAc \mid cAb \mid bab \mid ca \mid A \\
A &\rightarrow a
\end{aligned}
$$



(c) Consider the SLR(1) parser based on the LR(0) automaton of Part (a). From your automaton, identify all the states with conflicts. In each case, mention the type of the conflict, and also the input symbol(s) that lead(s) to that conflict. Write in the space below, not in the picture drawn for Part (a). **(4)**

Solution We have FOLLOW$(S) = \{\$\}$, and FOLLOW$(A) = \{b, c, \$\}$. The states with conflicts are now explained.

I_8 [shift-reduce conflict]: Since $b \in$ FOLLOW(A), the reduction $A \to a$ is applicable on input b. This gives a conflict with the shift possibility in the item $S \to ba.b$.

I_{10} [reduce-reduce conflict]: In this state, two reductions can be made. Since \$ is common to both FOLLOW(S) and FOLLOW(A), both reductions are allowed on input \$.

(d) Now, consider the (canonical) LR(1) parser based on the LR(1) automaton of Part (b). Explain how all the conflicts of the SLR(1) parser of Part (c) are resolved by the LR(1) parser. **(2)**

Solution I_8: The reduction $A \to a$ is valid for input symbol c alone. Therefore having a b on the input no longer causes a conflict.

I_{10}: The reduction $A \to a$ is valid for input symbol b alone. That is, on input \$, only the reduction $S \to ca$ is applicable.

(e) Justify whether the given grammar is LALR(1). **(1)**

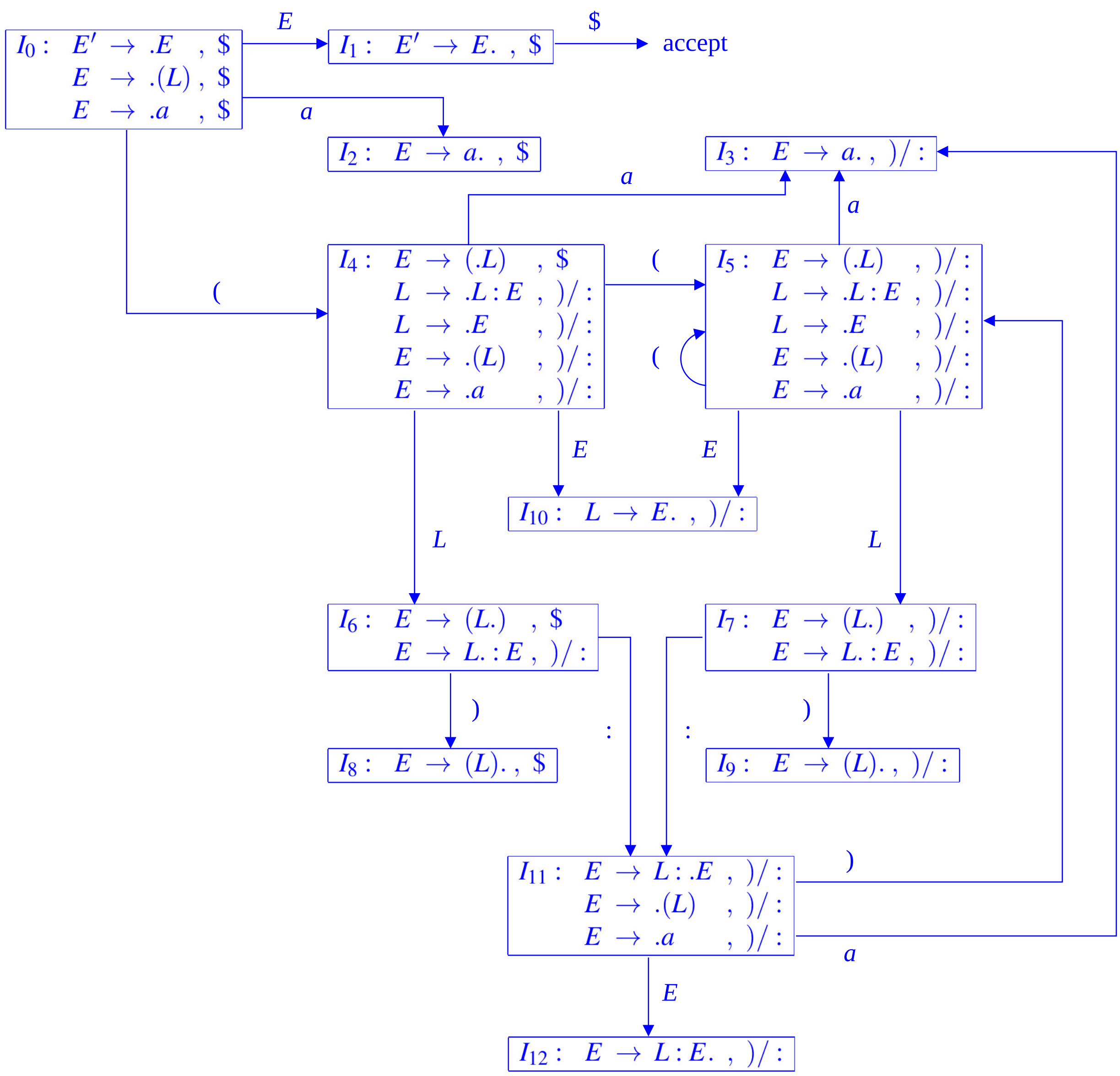
Solution *Yes*. There was no splitting of states in the LR(1) automaton (with respect to the LR(0) automaton). Therefore the LR(1) and the LALR(1) parsing tables will be identical.

3. [LALR(1) grammars]

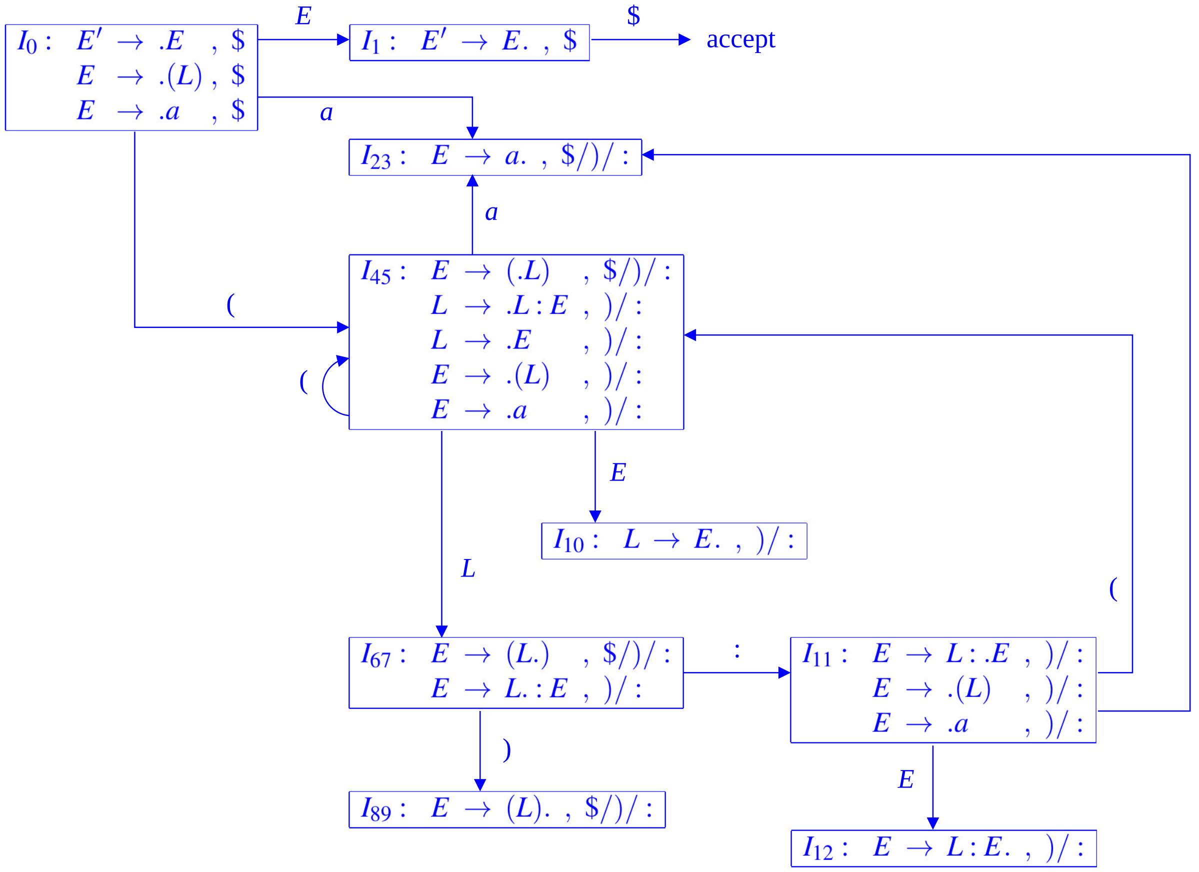
In this exercise, we consider the following grammar. Here, E is the start symbol, L is another nonterminal symbol, and the four terminal symbols are a, left parenthesis (, right parenthesis), and semicolon :.

$$\begin{array}{lcl} E & \rightarrow & (L) \mid a \\ L & \rightarrow & L : E \mid E \end{array}$$

(a) Draw the complete LR(1) automaton for the given grammar. Number the start state as 0 (or I_0). **(5)**



(b) Convert the LR(1) automaton of Part (a) to an automaton for LALR(1) parsing. Draw this complete LALR(1) automaton. Again, number the start state as 0 (or I_0). **(5)**



(c) From the automaton of Part (b), construct the LALR(1) parsing table for the given grammar. **(5)**

Number the rules as follows:

(1)	$E \to (L)$
(2)	$E \to a$
(3)	$L \to L : E$
(4)	$L \to E$

State	Action					Go to	
	a	(	)	:	\$	E	L
0	s23	s45				1	
1					accept		
23			r2	r2	r2		
45	s23	s45				10	67
67			s89	s11			
89			r1	r1	r1		
10			r4	r4			
11	s23	s45				12	
12			r3	r3			

(d) Work out how the string $((a) : a : (a : a))$ can be parsed using the LALR(1) parsing table of Part (c). Show all the steps (shift and reduce) of parsing, in the following format. Assume that the parse stack stores only the states. **(5)**

Parse stack	Remaining input	Action
0	$((a):a:(a:a))\$$	Shift (
0 45	$(a):a:(a:a))\$$	Shift (
0 45 45	$a):a:(a:a))\$$	Shift a
0 45 45 23	$):a:(a:a))\$$	Reduce $E \to a$
0 45 45 10	$):a:(a:a))\$$	Reduce $L \to E$
0 45 45 67	$):a:(a:a))\$$	Shift)
0 45 45 67 89	$:a:(a:a))\$$	Reduce $E \to (L)$
0 45 10	$:a:(a:a))\$$	Reduce $L \to E$
0 45 67	$:a:(a:a))\$$	Shift :
0 45 67 11	$a:(a:a))\$$	Shift a
0 45 67 11 23	$:(a:a))\$$	Reduce $E \to a$
0 45 67 11 12	$:(a:a))\$$	Reduce $L \to L:E$
0 45 67	$:(a:a))\$$	Shift :
0 45 67 11	$(a:a))\$$	Shift (
0 45 67 11 45	$a:a))\$$	Shift a
0 45 67 11 45 23	$:a))\$$	Reduce $E \to a$
0 45 67 11 45 10	$:a))\$$	Reduce $L \to E$
0 45 67 11 45 67	$:a))\$$	Shift :
0 45 67 11 45 67 11	$a))\$$	Shift a
0 45 67 11 45 67 11 23	$))\$$	Reduce $E \to a$
0 45 67 11 45 67 11 12	$))\$$	Reduce $L \to L:E$
0 45 67 11 45 67	$))\$$	Shift)
0 45 67 11 45 67 89	$)\$$	Reduce $E \to (L)$
0 45 67 11 12	$)\$$	Reduce $L \to L:E$
0 45 67	$)\$$	Shift)
0 45 67 89	$\$$	Reduce $E \to (L)$
0 1	$\$$	Accept

4. [Syntax-directed translation]

Consider the expression grammar of a programming language, involving only the division operator **/**.

$$
\begin{array}{lcl}
S & \rightarrow & E \\
E & \rightarrow & E\,\mathbf{/}\,T \mid T \\
T & \rightarrow & \mathbf{num} \mid \mathbf{num.num}
\end{array}
$$

The start symbol is S. This grammar deals with two kinds of operands: (i) integers consisting of sequences of digits, which are indicated by the token **num**, and (ii) floating-point numbers which are indicated by the token **num.num**. The semantics of the grammar directs that the division operation must be interpreted differently, depending on whether it is an integer division or a floating-point division. For example, integer division gives $5/4 = 1$, whereas floating-point division gives $5.0/4.0 = 1.25$.

The programming language allows expressions with both integers and floating-point numbers. The language semantics direct that expressions with mixed operand types should be promoted to floating-point expressions *throughout*. For example, the expression **5/2/2.0** (assuming left associativity for division) evaluates to **1.25**. Here, both the divisions are to be carried out as floating-point divisions, although the first division **5/2** has integer operands only. On the other hand, the integer-only expression **5/2/2** evaluates to **1**.

(a) In what follows, you write an SDD to implement the aforementioned semantics in the grammar for evaluating expressions (involving divisions only). Use three attributes in the SDD as explained below.

(i) A synthesized boolean attribute *isF* (with two possible values *true* and *false*) which indicates if any part of an expression has a floating-point operand.

(ii) An inherited attribute *etype* (with two possible values *int* and *float*) which stores the type of the subexpression. Note that the *etype* of a subexpression depends on the *isF* values of all operands.

(iii) A synthesized attribute *val* which stores the computed value of each subexpression. Note that the computation of *val* for a subexpression depends on the *etype* of that subexpression. Assume that integer-by-integer division is implemented by **idiv**, and float-by-float division by **fdiv**. It is not possible to carry out integer-by-float or float-by-integer divisions. However, there exists a function $ItoF(x)$ that converts an integer x to a floating-point number, and returns that floating-point number.

Fill in the blanks in the following table to implement these semantic rules. Here, annotating the parse tree needs three passes. The first pass computes the *isF* values by a bottom-up traversal of the parse tree. The second pass computes the *etype* values in a top-down fashion. The third pass computes *val* bottom up. **(12)**

Production	Semantic rule
$S \rightarrow E$	$E.etype =$ if ($E.isF$) then *float*, else *int* $S.val = E.val$
$E \rightarrow E_1\ /\ T$	$E.isF =$ if ($E_1.isF$ or $T.isF$) then *true*, else *false* $E_1.etype =$ $E.etype$ $T.etype$ = $E.etype$ $E.val =$ if ($E.etype$ is *int*) then $E_1.val$ **idiv** $T.val$, else $E_1.val$ **fdiv** $T.val$
$E \rightarrow T$	$E.isF =$ $T.isF$ $T.etype$ = $E.etype$ $E.val =$ $T.val$

Continued to the next page

$T \rightarrow$ **num**	$T.isF = false$ $T.val =$ if ($T.etype$ is int) then **num**$.val$, else $ItoF($**num**$.val)$
$T \rightarrow$ **num.num**	$T.isF =$ $true$ $T.val =$ **num.num**$.val$

(b) Given the input expression **7/2/2.0**, construct and draw the parse tree and the dependency graph, and annotate the parse tree, in the context of the SDD of Part (a). **(8)**

Solution In the following figure, the edges of the parse tree are shown as dashed lines. The edges of the dependency graph are shown as solid directed lines. The attributes with values are shown near the parse-tree nodes.