17 Feb 2019 **CS335: MID SEMESTER EXAM** **Total Marks: 85**

NOTE:

- Presenting your answers properly is your responsibility. You lose credit if you can not present your ideas clearly, and in proper form. Please DO NOT come back for re-evaluation saying, "What I actually meant was ....".
- Be precise and write clearly. Remember that somebody has to read it to evaluate!
- If required, make suitable assumptions and write them along with your answer.
- There are **5 questions** in this paper on **4 pages**.
- **This will be a silent exam**. No assistance will be provided during the exam. If you find any typo or error, mention it in your answer book. Make suitable and reasonable corrections and proceed with the solution.

---

1. For each of the subpart below, you have to write a regular expression. The mark will be given on the basis of *compactness* of the regular expression..... **(Marks: 10)**

   (a) Write a regular expression to match REAL numbers having **exactly one decimal point** and the second digit after decimal point (if it exists) greater than 4.
   Some Valid Matches: 1.5, 2.05, 00.2, 1259.762, 000.0900
   Some Invalid Matches: 12, 12.35.54, 12.123, 12.

   (b) Write a regular expression to match POSITIVE INTEGERS that are **multiples of FOUR** and **do not have superfluous ZEROes.** Here superfluous ZERO is a ZERO that can be deleted from a number without changing its value.
   Some Valid Matches: 4, 124, 40
   Some Invalid Matches: 00, 122, 2, 04

2. ..... **(Marks: 20)**
   Consider the following grammar G, with non-terminals = {stmt} and terminals = {TRY, CATCH, OTHER}:

stmt	→	TRY stmt
	|	TRY stmt CATCH stmt
	|	OTHER

   (a) Is this grammar ambiguous? Justify your answer.
   (b) Mr Ambi Guous thinks G is ambiguous. He writes the following grammar, G' having an additional non-terminal "matched", that is supposed to be an unambiguous rewrite of G:

stmt	→	TRY stmt
	|	matched
matched	→	TRY matched CATCH stmt
	|	OTHER

   Prove whether G' is ambiguous or unambiguous.
   (c) Is the language of G inherently ambiguous? Justify your answer.

3. ..... (Marks: 10)

Consider the following grammar:

T	$\rightarrow$	T $\Rightarrow$ T
T	$\rightarrow$	T $\oplus$ T
T	$\rightarrow$	t

where T is a non-terminal, and $\Rightarrow$, $\oplus$ and t are terminals. The grammar is obviously ambiguous. Rewrite the grammar to make it unabiguous using the following rules:

- $\oplus$ has higher precedence than $\Rightarrow$.
- $\oplus$ is left-associative.
- $\Rightarrow$ is right-associative.

4. ..... (Marks: 15)

Consider the grammar with nonterminals = {S, op, A}, terminals = {num, +, -} as below:

S	$\rightarrow$	A op
op	$\rightarrow$	+ | -
A	$\rightarrow$	num, A | num

(a) You have to write a syntax directed **translation scheme** that calculates an attribute called *val* of S. The op + denotes the maximum of the list of numbers (num) seen, and – denotes the minimum of the numbers seen. For example, if S derives `1,2,3 +` then S.val = max(1, 2, 3) = 3 and if S derives 1,2,3 –, S.val is min(1,2,3) = 1.

Assume that the terminal num has a synthesized attribute called *val* which is supplied by the lexical analyzer.

You may modify the grammar if you want. However, the language generated by the modified grammar should be the same as the original grammar. Further, you can introduce more attributes as you desire. However, the attributes can not have dynamic size (i.e., you can not use lists, dynamic arrays, sets or similar dynamic data structures as attributes to hold values).

(b) For each attribute of each grammar symbol, list whether it is Synthesized or Inherited.

5. ..... (Marks: 30)

Consider the following grammar (with rule numbers). The augmentation is already taken care of by introducing the rule (0) S → L.

(0)	S	→	L
(1)	L	→	L B
(2)	L	→	B
(3)	B	→	B – F
(4)	B	→	F
(5)	F	→	[L]
(6)	F	→	t

where L, B, F are non-terminals; and –, [,], and t are terminals. The set of LR(1) itemsets, without closure[1], for the grammar (not in any particular order) are as follows:

Set#	Contents	Set#	Contents
0	S → •L, $	1	S → L•, $ L → L•B, $ [t
2	L → B•, $ [t B → B•– F, $ [t –	3	B → F•, $ [t –
4	F → [•L], $ [t –	5	F → t•, $ [t –
6	L → L B•, $ [t B → B•– F, $ [t –	7	B → B – •F, $ [t –
8	F → [L•], $ [t – L → L•B,] [t	9	L → B•,] [t B → B•– F,] [t –
10	B → F•,] [t –	11	F → [•L],] [t –
12	F → t•,] [t –	13	B → B – F•, $ [t –
14	F → [L]•, $ [t –	15	L → L B•,] [t – B → B•– F,] [t –
16	B → B –•F,] [t –	17	F → [L•],] [t – L → L•B,] [t
18	B → B – F•,] [t –	19	F → [L]•,] [t –

Note that multiple lookahead symbols are given as a list (for example, $ [t).

(a) Complete the closure for each itemset.

[1]These are called the *kernel* items

(b) Complete the CLR parse table for the grammar. Use Set#s above as states. Use the rule numbers as given above for reduce entries. Finally, use the structure of the parse table (rows and columns) as shown next.

State	ACTION					GOTO		
	–	[	]	t	$	L	B	F