BITS Pilani K K Birla Goa Campus

Mid Semester Examination

Course: CS F363	**Marks: 60**	Date: 07/03/2020
Compiler Construction	**Total Weightage:** 30% **Mode:** Closed Book	Time: 2.00 PM to 3.30 PM

Note: CFG means context free grammar .

1. (a) Consider the following regular expression: $(0|1)^*.(0|1).(011)^*$ Draw the DFA for the language represented by the above regular expression using the First & Follow Method. $\boxed{4}$

   (b) Provide the correct regular expression for formulating tokens corresponding to variable names, where any valid variable name has to obey the following rules:

   - A variable name can only contain alpha-numeric characters (A-Z, a-z, 0-9) and underscores.
   - A variable name cannot start or end with an uppercase letter.
   - A variable name cannot start or end with a numeric character.
   - A variable name cannot end with the underscore character.

   N.B. - You are allowed to use an augmented form of regular expressions, where you can write $[A-Z]$ as shorthand for $(A|B|\dots|Z)$, and similarly $[0-9]$ for $(0|1|\dots|9)$. $\boxed{3}$

   (c) Draw the DFA for accepting valid variable names from the regular expression obtained in part (b) using the First & Follow Method. $\boxed{3}$

2. Show that the following context-free grammar (CFG) is ambiguous. $\boxed{5}$

$$S \to ScS \,|\, d$$

3. Fill in the blanks appropriately to complete the given pseudo-codes for computing FIRST and FOLLOW sets for X.

   (a) Algorithm for computing FIRST(X) – $\boxed{2}$

   1. If X is terminal, then FIRST(X) IS $\{X\}$.
   2. If $X \to \epsilon$ is a production, then add ϵ to FIRST(X).
   3. If X is non-terminal and $X \to Y_1, Y_2, \dots, Y_k$ is a production, then place a in FIRST(X) if for some i, a is in FIRST(___I____), and ϵ is in all of FIRST(Y_1), …, FIRST(___II____).

   (b) Algorithm for computing FOLLOW(X) – $\boxed{3}$

   1. Place \$ in FOLLOW(S),where S is the start symbol and \$ is the input end marker.
   2. If there is a production $A \to \alpha B \beta$, then everything in FIRST(___III___) except for ϵ is placed in FOLLOW(B).
   3. If there is a production $A \to \alpha B$, or a production $A \to \alpha B \beta$ where FIRST(β) contains ϵ, then everything in FOLLOW(___IV__) is in FOLLOW(___V___).

4. Consider the following CFG, where the set of terminals is $\Sigma = \{a, b, \#, \%, !\}$ and the set of variables is $V = \{S, T, U\}$ with S being the start symbol, which has the following production rules.

$$S \to \%aT \,|\, U!$$

$$T \to aS \,|\, baT \,|\, \epsilon$$

$$U \to \#aTU \,|\, \epsilon$$

   (a) Construct the FIRST sets for each of the non-terminals. $\boxed{3}$

   (b) Construct the FOLLOW sets for each of the non-terminals. $\boxed{3}$

P.T.O. . .

(c) Construct the LL(1) parsing table for the CFG. **4**

(d) Show the sequence of stack, input and action configurations that happen during an LL(1) parse of the string *#abaa%aba!*. At the beginning of the parse, the stack should contain only $. At each step, describe the stack configuration by using a single string whose characters from left to right represent the contents of the stack from top to bottom. For the input configuration at each step, you should drop all the terminals matched already in previous steps from the LHS of the original string. Finally, note that there can only be two different types of action possible at each step: 1) *output <production rule>*, and 2) *match <terminal>*. **5**

5. Consider the following CFG, where the set of terminals is $\Sigma = \{id, @, .\}$ and the set of variables is $V = \{Addr, Name\}$ with *Addr* being the start symbol, which has the following production rules.

$$Addr \to Name \, @ \, Name \, . \, id$$

$$Name \to id \,|\, id \, . \, Name$$

The above grammar can be used to generate valid email addresses, such as:

id@id.id
id.id@id.id.id.id
id.id.id@id.id

(a) Rewrite the grammar to eliminate all LL(1) conflicts. **3**

(b) Construct the FIRST and FOLLOW sets for all non-terminals in your revised grammar. **2**

(c) Using your FIRST and FOLLOW sets from part (b), construct the LL(1) parse table for your revised grammar. **5**

6. Consider the following CFG, where the set of terminals is $\Sigma = \{i, n, (,), \, , \}$ and the set of variables is $V = \{E, A, L, S\}$ with E being the start symbol, which has the following production rules.

$$E \to A \,|\, L$$

$$A \to n \,|\, i$$

$$L \to (\, S \,)$$

$$S \to E \,|\, E \, , \, S$$

(a) Left factor the given CFG, and provide the equivalent CFG after left factoring. **3**

(b) Construct a recursive descent parser for the equivalent CFG obtained above. The parser should only read a string and tell whether it is in this language, where each token is a character. Presume a lexical analyzer is available, and that statement *match*(*c*) will check the current lookahead token to see whether it is character *c*. If so, it will put the next token into variable lookahead. If not, it will print "NO" and stop the program. Statement INIT_LEXER initializes the lexer, setting lookahead to the first token. **12**

END