PCD Question Bank

1. How can a lexical analyzer be constructed using Lex tool?
2. Write short notes on LEX.
3. Draw the syntax tree for the augmented regular expression (ε | ab)\*(εa)bba#
4. Draw the syntax tree for the augmented expression (ab)ab\*( ε bb)#
5. Write a context free grammar that generates the set of all strings of a’s and b’s which is a palindrome.
6. Mention the basic issues in parsing.
7. How will you eliminate left factor in a grammar?
8. What is the role of the parser in a compiler model?
9. Draw transition diagrams for predictive parsers for the grammar

E 🡪E + T | T

T 🡪T \* F | F

F 🡪 (E) | id

1. Consider the grammar g=(V,T,P,S). Here V={S,N,Np,ADJ} and T={and, eggs, ham, pencilgreen, cold, tasty}. The set contains the following rules:

Np🡪Np and Np N 🡪eggs | ham | pencil|

Np🡪ADJ Np ADJ 🡪green |cold |tasty|

Np🡪N

1. Show that the grammar is ambiguous by constructing two different parse trees for the sentence “green eggs and ham”.

Consider the grammar g=(V,T,P,S). Here V={S,N,Np,ADJ} and T={and, eggs, ham, pencil green, cold, tasty}. The set contains the following rules:

Np🡪Np and Np N 🡪eggs | bread| pencil|

Np🡪ADJ Np ADJ 🡪boiled |cold |tasty|

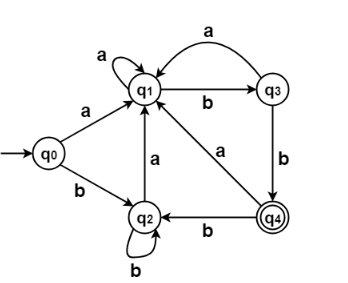
Np🡪N

Is this grammar ambiguous for the sentence “boiled eggs and bread”.

1. Eliminate Left recursion for the following grammar.



1. Compute nullable(n), FIRSTPOS, LASTPOS and FOLLOWPOS for the regular expression (ε | a)\*(ab)+
2. Write the rules for computing the FIRST and FOLLOW functions for constructing predictive parsing table.
3. What is the purpose of minimizing the states of DFA? Minimize the given DFA.



1. Illustrate the heuristic techniques for error-recovery in predictive parsing.
2. Distinguish between context free grammar and regular expressions.
3. Write an algorithm for eliminating left recursion from a grammar. Using the algorithm eliminate left recursion for the following grammar.

S → Aa/b, A→Ac/Sd/e

1. Show the stack implementation of non recursive predictive parsing actions for the input id+id\*id using the grammar given below.

E🡪 TE’

E’ 🡪 +TE’/ ɛ

T🡪 FT’

T’🡪 \*FT’ / ɛ

F🡪 (E) / id

1. Describe the language accepted by the following grammar:

S🡪 AS | B

A🡪aAc | Aa | ɛ

B🡪bBb | ɛ

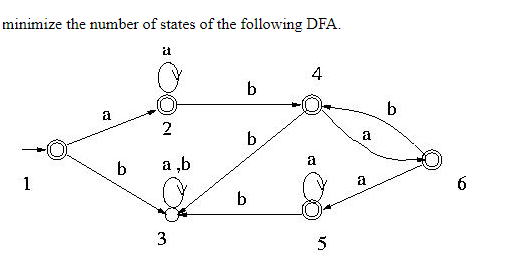
Is this grammar ambiguous? Provide proof for ambiguity.

1. Rewrite the following grammars so they can be parsed by a predictive parser by eliminating left recursion and applying left factoring where necessary.

S 🡪S + a | S + b | c

1. Consider the CFG G = {NT = {E,T,F}, T = {a,b,+,\*}, P, E } with the set of productions as follows:
   1. E → E + T
   2. E → T
   3. T → T \* F
   4. T → F
   5. F → F \*
   6. F → a
   7. F → b

Compute FIRST and FOLLOW functions.



1. To verify the syntax of programming language constructs, compilers use CFG.Why not Regular expression?. Justify with examples.
2. Is it possible to directly obtain a DFA from a regular expression? How is it different from obtaining DFA from an intermediate NFA? Obtain DFA directly for the regular expression (a/b)+c(c/d).
3. How do we eliminate ambiguity from the dangling else grammar?
4. Explain the structure of Lex program. Write Lex specifications for a Desk calculator application.
5. Describe the different strategies that a parser can employ to recover from a syntactic error.
6. Consider the following CFG G = (N={S, A, B, C, D}, T={a,b,c,d}, P, S) where the set of productions P is given below:

S → A

A → BC | DBC

B → Bb | ε

C → c | ε

D → a | d

Is this grammar suitable to be parsed using the recursive descendent parsing method? Justify and modify the grammar if needed.

1. Consider the grammar

bexpr -> bexpr or bterm | bterm   
bterm -> bterm and bfactor | bfactor   
bfactor -> not bfactor | ( bexpr ) | true | false

a)Construct a parse tree for the input string **not (true or false)**

b)Show that this grammar generates all boolean expressions.

1. Consider the grammar



1. Construct FIRST and FOLLOW sets

ii)Construct the predictive parsing table

iii)Show the moves of the predictive parser for input (a,(a,a))

1. Left factoring is a technique to remove non-determinism. How left factoring does avoid backtracking in grammars? Apply left factoring in the following grammar to make it deterministic.

A→aBcC | aBb | aB | a

B →ε

C →ε