To convert a regular expression to a context-free grammar (CFG), you can follow a set of standard conversion rules.

Here are the rules for converting a regular expression to a CFG:

1. Terminal Symbols:

Each character or symbol in the regular expression becomes a terminal symbol in the CFG.

2. Start Symbol:

Create a new start symbol for the CFG.

3. Concatenation:

For every concatenation (ab) in the regular expression, add a new production rule in the CFG. The left-hand side of the rule should be a non-terminal symbol representing the concatenation, and the right-hand side should be the concatenation of the non-terminal symbols representing the individual characters/symbols.

4. Union:

For every union (a + b) in the regular expression, add a new production rule in the CFG. The left-hand side of the rule should be a non-terminal symbol representing the union, and the right-hand side should have two alternatives, one for each character/symbol in the union.

5. Kleene Star:

For every Kleene star (a*) in the regular expression, add a new production rule in the CFG.

The left-hand side of the rule should be a non-terminal symbol representing the Kleene star, and the right-hand side should have two alternatives. One alternative should be the non-terminal symbol representing the character/symbol, followed by the non-terminal symbol representing the Kleene star itself. The other alternative should be ε (epsilon), indicating an empty string.

6. Parentheses:

If the regular expression contains parentheses, treat the contents within the parentheses as a separate expression and apply the conversion rules recursively.

By following these rules, you can convert a regular expression into an equivalent context-free grammar (CFG). Note that the resulting CFG may contain additional non-terminal symbols and production rules compared to the original regular expression.

Practice problem on Regular expresion to CFG

Q1. Write given CFG for R.E (011 + 1)**(01)** (RGPV 2020).

Here’s the context-free grammar (CFG) for the given regular expression (011 + 1)(01):

Non-terminal symbols:
S: Start symbol
A: Represents the regular expression (011 + 1)*
B: Represents the regular expression (01)*

Terminal symbols:
0, 1: Binary digits

Production rules:

1. S -> AB
2. A -> 011A
3. A -> 1A
4. A -> ε (epsilon)
5. B -> 01B
6. B -> ε (epsilon)

Explanation of the CFG:

The CFG consists of two non-terminal symbols, A and B, which represent the two parts of the regular expression. Non-terminal symbol S is the start symbol, which represents the entire regular expression (011 + 1)(01).

The production rules define the valid derivations for each non-terminal symbol. Rule 1 states that the start symbol S can be derived as AB, meaning it can be split into parts A and B. Rule 2 defines that A can be derived as 011A, indicating that the regular expression (011 + 1) can repeat zero or more times. Rule 3 states that A can be derived as 1A, allowing the regular expression 1 to repeat zero or more times. Rule 4 allows A to be derived as ε (epsilon), indicating that it can be empty.

Similarly, Rule 5 defines that B can be derived as 01B, indicating that the regular expression 01 can repeat zero or more times. Rule 6 allows B to be derived as ε (epsilon), indicating that it can be empty.

By applying these production rules, you can generate strings that match the given regular expression.

Q2. Regular Expression: (a + b)*

A. CFG:

Start Symbol: S
Terminal Symbols: a, b
Production Rules:
S -> Sa
S -> Sb
S -> ε (epsilon)

Q3. Regular Expression: ab

A. CFG:

Start Symbol: S
Terminal Symbols: a, b
Production Rules:
S -> aS
S -> bS
S -> ε (epsilon)

Q4. Regular Expression: (aa + b)*c

A. CFG:

Start Symbol: S
Terminal Symbols: a, b, c

Production Rules:

S -> SS

S -> a

S -> b

S -> c

Q5. Regular Expression: (a + b)c*

A. CFG:

Start Symbol: S

Terminal Symbols: a, b, c

Production Rules:

S -> aS

S -> bS

S -> cS

S -> ε (epsilon)

Q6. Regular Expression: (abc*)+

A. CFG:

Start Symbol: S

Terminal Symbols: a, b, c

Production Rules:

S -> T+

T -> aT

T -> bT

T -> cT

T -> ε (epsilon)

Related Posts:

1. Regular expression to Regular grammar
2. RGPV TOC What do you understand by DFA how to represent it
3. What is Regular Expression
4. RGPV short note on automata
5. RGPV TOC properties of transition functions
6. RGPV TOC What is Trap state
7. CFL are not closed under intersection
8. NFA to DFA | RGPV TOC
9. Moore to Mealy | RGPV TOC PYQ
10. DFA accept even 0 and even 1 |RGPV TOC PYQ
11. Short note on automata | RGPV TOC PYQ
12. DFA ending with 00 start with 0 no epsilon | RGPV TOC PYQ
13. DFA ending with 101 | RGPV TOC PYQ
14. Construct DFA for a power n, n>=0 || RGPV TOC
15. Construct FA divisible by 3 | RGPV TOC PYQ
16. Construct DFA equivalent to NFA | RGPV TOC PYQ
17. RGPV Define Mealy and Moore Machine
18. RGPV TOC Short note on equivalent of DFA and NFA
19. RGPV notes Write short note on NDFA
20. CNF from S–>aAD;A->aB/bAB;B->b,D->d.
21. NDFA accepting two consecutive a’s or two consecutive b’s.
22. Grammar is ambiguous. S → aSbS|bSaS|∈
23. leftmost and rightmost derivations

24. Construct Moore machine for Mealy machine
25. Design a NFA that accepts the language over the alphabet, Σ= {0, 1,2} where the decimal equivalent of the language is divisible by 3.
26. Definition of Deterministic Finite Automata
27. Notations for DFA
28. How do a DFA Process Strings?
29. DFA solved examples
30. Definition Non Deterministic Finite Automata
31. Moore machine
32. Mealy Machine
33. Regular Expression Examples
34. Regular expression
35. Arden’s Law
36. NFA with ∈-Moves
37. NFA with ∈ to DFA Indirect Method
38. Define Mealy and Moore Machine
39. What is Trap state ?
40. Equivalent of DFA and NFA
41. Properties of transition functions
42. Mealy to Moore Machine
43. Moore to Mealy machine
44. Diiference between Mealy and Moore machine
45. Pushdown Automata
46. Remove ∈ transitions from NFA
47. TOC 1
48. Diiference between Mealy and Moore machine
49. What is Regular Set in TOC

50. DFA which accept 00 and 11 at the end of a string
51. DFA end with 1 contain 00 | RGPV TOC draw
52. RGPV TOC design finite automata problems
53. Minimization of DFA
54. Construct NFA without ∈
55. RGPV TOC PYQs
56. Introduction to Automata Theory