Aluno: Matheus Lopes de Freitas

1. Dado o autômato finito abaixo escreva a expressão regular e a gramática regular correspondente. Este autômato é um AFD ou AFN? Explique.

R: O autômato é um AFD pois ao analisarmos a função de transição, vemos que não há nenhum símbolo do alfabeto em que, no mesmo estado, haja transição para outro estado com o mesmo símbolo.

2. Defina a expressão regular para o complemento da linguagem reconhecida pela AF apresentado na questão 1

R:

A expressão $a(a+b)*c(c*)a(a+b)*c(c*)$ representa palavras que:

Começam com um a

Seguem com qualquer número de a's e b's (inclusive zero)

Depois aparece um c obrigatório

E depois vem qualquer número de c's (inclusive zero)

Logo, o complemento será o seguinte:

$(b+c)(a+b+c)* \cup a(a+b)* \cup a(a+b)*c(c*(a+b)(a+b+c)*)$

3. Classifique as linguagens abaixo segundo a hierarquia de Chomsky. Que tipo de reconhecedor

(no mínimo) seria necessário para reconhecer cada uma delas.

a. L1={a^nba^m | n>= 1 e m >= 2}

Linguagem Livre de Contexto. Autômato com pilha

b. L2={anbman | n >= 0 e m >= 1}

Linguagem Livre de Contexto. Autômato com pilha

c. L3={anbnan | n >= 1 }

Linguagem Sensível ao Contexto. Máquina de Turing

d. L4={anbnan | n >= 1 e n < 100}

Linguagem regular (pois é limitada). Autômato Finito

4.

a)

S -> AB

A -> a|aA

B→bbC| bBa

b)

S -> ABA

A -> ε | aA

B -> b | bB

c)

S -> aSa | T

T -> ab | aTb

5. Prove que as linguagens regulares são fechadas sobre a união, interseção e complemento.

União:

Dado dois AFDs, podemos construir um novo autômato que reconhece a união por meio de um produto cartesiano Q1 x Q2. Assim, ele aceita uma palavra se ao menos um dos dois autômatos aceitar.

Intersecção:

Usamos a mesma construção de cima, mas mudamos os estados finais para que só sejam aceitas palavras que são aceitas pelos dois autômatos.

Complemento:

Basta inverter os estados finais, os que eram finais passam a não ser e os que não eram passam a ser. Dessa forma, continuamos com um AFD.

6. Defina uma gramática que gere a linguagem:

L = {0^n1^n |n >= 1} U {0^n1^2n  | n>= 1}

G = (V, T, P, S), ondee:

V = {S, A, B}

T = {0, 1}

P = {

S -> A | B

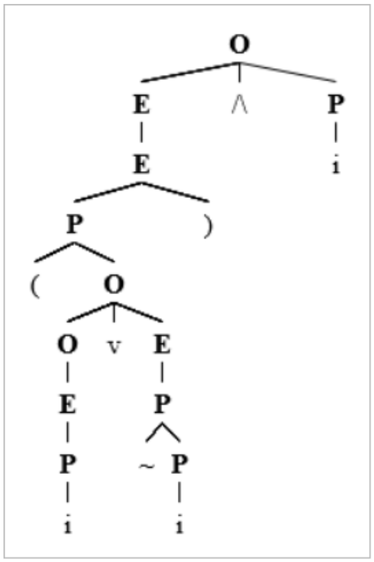
A -> 0A1 | 01

B -> 0B11 | 011

}

7. Dada a gramática G = {{L, O, E, P}, {i, ^, v, ~}, O, Φ}. Onde Φ é: O → O v E O → E E → E ^ P E → P P → (O) P → ~P P → i Verifique se as cadeias abaixo podem ser geradas pela gramática. Utilize derivação canônica mais à esquerda e construa a árvore sintática.

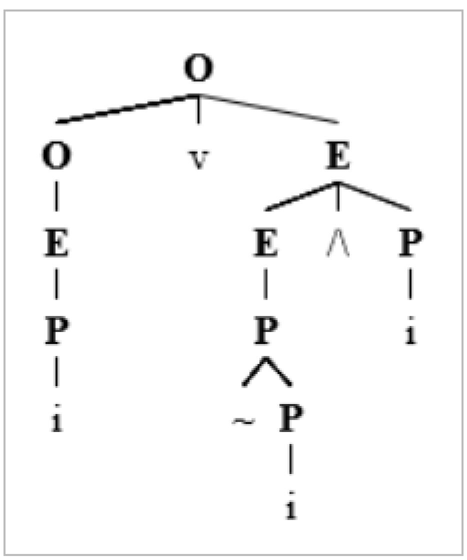
A) ( i v ~ i ) ^ i




B)  i ~ i v i

Não é gerada pela gramática.

C) i v ~ i ^ i




D)  ( i v ~ i ^ ) i

Não é gerada pela gramática.

8. Considerando a gramática dada na questão anterior, sendo que os terminais {^, v} representam operadores binários, {~} representa um operador unário e {i} os operandos, qual é a precedência e associatividade definida para os operadores por essa gramática?

R: A precedência é dada pela profundidade do terminal, enquanto a associatividade está ligada a recursividade da gramática.

1. Precedência:

   ~ > ^ > v

2. Associatividade é feita à esquerda.

9. Dada a gramática G = {{S}, {a}, S, {S → SS+, S→SS*, S→ a}}, mostre se as cadeias aa+a* e a+a podem ser geradas por esta gramática.

R: aa+a* pode ser gerada, enquanto a+a não.

A gramática é uma implementação da notação pós-fixa.

10. O que são gramáticas ambíguas?
R: São gramáticas em que uma palavra pode ser gerada a partir de árvores de derivação diferentes.

11. Defina uma GLC para linguagem {akbncm | n > m + k }.

S → A X C Y

A → aA | ε

C → cC | ε

X → bX | ε

Y → bY | b

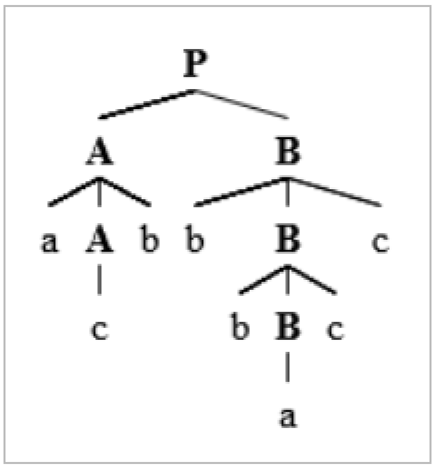
12. Dada a gramática:

P → AB

A → aAb | c

B → bBc | a

a) Mostre a derivação mais à esquerda para: acbbbacc.




b) Defina L(G) utilizando notação de conjuntos.

Dada a GLC, rescreva a gramática eliminando a ambiguidade e gerando a mesma linguagem:

<Cmd> → if < Expr> then <Cmd>

<Cmd> → if < Exprl> then <Cmd> else <Cmd>

<Cmd> → c

< Exprl> → b

sendo <Cmd> e <Exprl> variáveis não terminais e if, then, else, r, b terminais da linguagem.

R:

<Cmd> → if < Expr> then <Cmd’>

<Cmd’> →  c else <Cmd> | c

< Exprl> → b