UFES – CEUNES - Engenharia de Computação e Ciência da Computação
Linguagens Formais e Autômatos – 2018/1
Prof. *Henrique Monteiro Cristovão*

Exercícios

1) Crie uma gramática para gerar números reais.
   Defina a 4-upla de elementos da gramática G = (V, T, P, S)
   Exemplos de strings desta linguagem: { 2.5, +2.5, -2.5, +2, -2, +0, 0, .5, -.5}

2) Faça uma gramática para gerar todos os números romanos.
   Veja alguns exemplos de números romanos e seus respectivos valores em números arábicos:

I = 1	VI = 6	XX = 20	LXX = 70	CCC = 300	DCCC = 800
II = 2	VII = 7	XXX = 30	LXXX = 80	CD = 400	CM = 900
III = 3	VIII = 8	XL = 40	XC = 90	D = 500	M = 1000
IV = 4	IX = 9	L = 50	C = 100	DC = 600	MM = 2000
V = 5	X = 10	LX = 60	CC = 200	DCC = 700	MMM = 3000

   Obs.:
   - o menor número romano é: I (= 1)
   - o maior número romano possível é: MMMCMXCIX (=3999)

   Como proposta inicial para este exercício, faça apenas para os números de I a XCIX (1 a 99).

   Dica: crie uma variável gramatical para a parte das dezenas e outra para a parte das unidades.

3) Refaça a gramática dos números reais (exercício 1), agora como gramática regular.

4) Crie expressões regulares para representar cada uma das linguagens regulares descritas a seguir:
   a. Números romanos de 1 a 99.
   b. Valores monetários em Reais. Possuem exatamente duas casas decimais depois da vírgula, e usam ponto como separador de milhar.
      Exemplos: { R$2,35 ; R$1.546,98 ; R$1,00 ; R$10.000.000.000,00 }
   c. Strings que possuem uma quantidade par de a's sobre o alfabeto Σ = {a,b,c}
   d. Strings que possuem tamanho múltiplo de três, sobre o alfabeto Σ = {a,b,c}
   e. Strings com pelo menos um '1' sobre o alfabeto Σ = {0,1}
   f. Strings terminadas em '110' sobre o alfabeto Σ = {0,1}
   g. Strings contendo '00' sobre o alfabeto Σ = {0,1}
   h. Strings iniciadas e terminadas em '10' sobre o alfabeto Σ = {0,1}
   i. Strings formadas por uma seqüência de a's seguida de uma seqüência de b's e terminada por uma seqüência de c's. É possível que cada uma das três seqüências descritas não possuam caracteres, mas se eles ocorrerem devem estar na ordem: a's, b's e c's. A string vazia não pode ser aceita. Considere o alfabeto Σ = {a,b,c}

5) (ENADE Eng de Comp, 2017)

Um compilador transforma uma linguagem, em geral textual, em outra linguagem. Um dos tipos de linguagens que um compilador pode transformar são as linguagens regulares, que podem ser descritas utilizando-se expressões regulares compostas por símbolos isolados agrupados com operadores * e U e organizadas com auxílio de parênteses.

Nesse contexto, avalie as afirmações a seguir.

I. A palavra 10010100 pertence à linguagem representada por (100*)*.

II. A palavra 10010 pertence à linguagem representada por (1(10)*0)*.

III. Existe somente uma expressão regular para representar uma linguagem regular.

É correto o que se afirma em

A I, apenas.
B II, apenas.
C I e III, apenas.
D II e III, apenas.
E I, II e III.

6) (POSCOMP, 2004).

Seja $\Sigma = \{a, b\}$. Uma expressão regular denotando a linguagem $L = \{w \in \Sigma^*$ tal que toda ocorrência de "a" em w é imediatamente seguida de "b"$\}$ é:

(a) $(a^*b)^*$
(b) $(b + ab)^*$
(c) a^*b
(d) $b + (ab)^*$
(e) $(ab)^*$

7) (POSCOMP, 2004).

As seguintes expressões regulares denotam as linguagens P, Q, L e R, respectivamente: $(1 + 10)^*$, $(0 + 01)^*$, $(0 + 1)^*$, $0(11)^* + 1(00)^*$. Não se pode afirmar que:

(a) $P \cap Q \neq \emptyset$
(b) $P \cup Q \neq L$
(c) $P \cap Q = \{\epsilon\}$
(d) $(1 + 0)^* \setminus P = Q$
(e) $R \subset L \setminus (P \cup Q)$

8) (POSCOMP, 2009)

Seja o alfabeto $\Sigma = \{a, b\}$ e a linguagem regular

$$L = \{ \omega \mid \omega \in \Sigma^* \text{ e o nº de a's em } \omega \text{ é par } \}.$$

Qual das expressões regulares abaixo gera essa linguagem?

A) (a b* a b*)*
B) ((a a)* | b*)*
C) (b* | (a a)* | b*)*
D) (b* a b* a b*)*
E) (a a | b)*

9) (POSCOMP, 2012)

Assinale a alternativa que apresenta, corretamente, uma expressão regular que denota todas as *strings* de a's e b's que têm pelo menos dois b's consecutivos.

a) (a*-bb)(a-ba)*(a+b)*

b) (a+ba)*bb(ba+a)*

c) (a+b)*ba*b(a+b)*

d) (a+bb)*(bb+a)*

e) (a+ba)*bb(a+b)*

10) (ENADE, 2014)

Considere as seguintes expressões regulares cujo alfabeto é {a, b}.

R1 = a(a U b)*

R2 = b(a U b)*

Se L(R) é a linguagem associada a uma expressão regular R, é correto afirmar que

A) L(R1) = L(R2).
B) L(R2) = {w | w termina com b}.
C) existe um autômato finito determinístico cuja linguagem é igual a L(R1) U L(R2).
D) se R3 é uma expressão regular tal que L(R3) = L(R1) ∩ L(R2), então L(R3) é uma linguagem infinita.
E) um autômato finito não determinístico que reconheça L(R1) U L(R2) tem, pelo menos, quatro estados.

11) (ENADE, 2014)

Expressões regulares constituem formas sucintas de descrever linguagens regulares. Uma de suas aplicações é descrever padrões a serem procurados em um texto. As expressões regulares R1, R2, R3 e R4 a seguir utilizam a seguinte convenção: o fecho de Kleene é denotado por * e a união é denotada pelo símbolo |.

- R1 = a*ba*ba*ba*
- R2 = a*(a|b)a(a|b)*
- R3 = a*ab*a(a|b)
- R4 = (a|b)*

Em relação às linguagens definidas pelas expressões regulares apresentadas, conclui-se que a cadeia **abbb** está contida apenas nas linguagens definidas por

A) R1 e R4.
B) R2 e R3.
C) R2 e R4.
D) R1 e R3.
E) R2, R3 e R4.

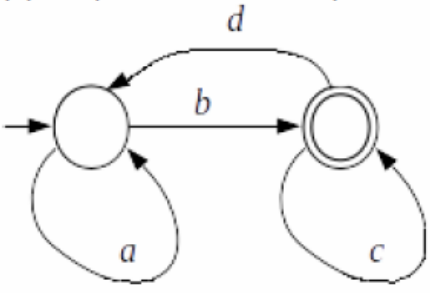
12) (POSCOMP, 2013)

Se o estado inicial for também estado final em um autômato finito, então esse autômato

a) não aceita a cadeia vazia.
b) não tem outros estados finais.
c) é determinístico.
d) aceita a cadeia vazia.
e) é não determinístico.

13) (POSCOMP, 2016)

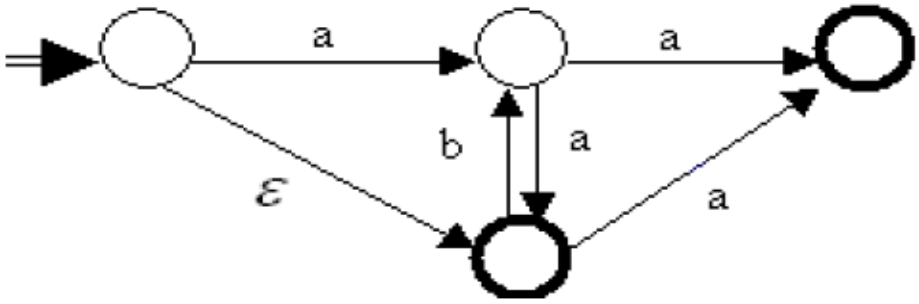
O grafo rotulado G(r), exposto abaixo, representa qual expressão regular?



A) $r = ab^*(da^* + cb)^*$
B) $r = a^*b(d^* + cb)$
C) $r = (bb + d)^*(aa + c)^*$
D) $r = a^*b(c + da^*b)^*$
E) $r = a^*c^*(b + d)^*$

14) (POSCOMP, 2008)

Considere o autômato finito mostrado na figura abaixo (os círculos em negrito representam estados terminais).



A esse respeito, assinale a afirmativa ***FALSA***.

A) A palavra *aaa* é reconhecida pelo autômato.

B) A palavra *ababa* não é reconhecida pelo autômato.

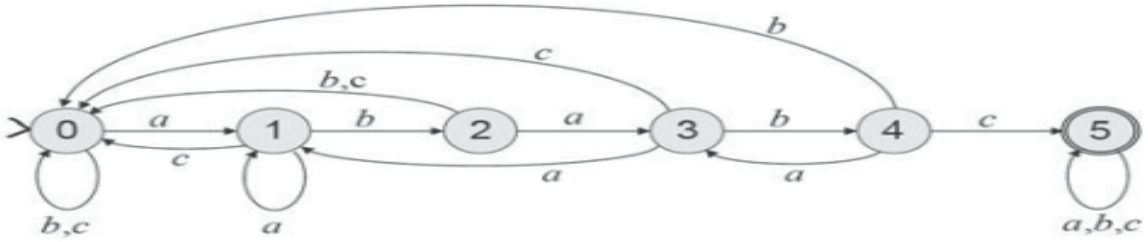
C) A palavra vazia é reconhecida pelo autômato.

D) A palavra *aba* é reconhecida pelo autômato.

E) A palavra *baba* é reconhecida pelo autômato.

15) (ENADE, 2011)

Autômatos finitos possuem diversas aplicações práticas, como na detecção de sequências de caracteres em um texto. A figura abaixo apresenta um autômato que reconhece sequências sobre o alfabeto $\Sigma = \{a, b, c\}$ e uma gramática livre de contexto que gera um subconjunto de Σ^*, em que λ representa o *string* vazio.



$$
\begin{aligned}
S &\rightarrow aS|bS|cS|abA \\
A &\rightarrow abA|abcB \\
B &\rightarrow aB|bB|cB|\lambda
\end{aligned}
$$

Analisando a gramática e o autômato acima, conclui-se que

A a linguagem gerada pela gramática é inerentemente ambígua.

B a gramática é regular e gera uma linguagem livre de contexto.

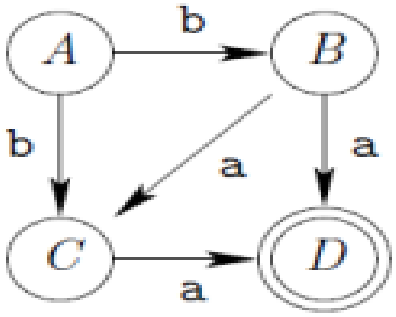
C a linguagem reconhecida pelo autômato é a mesma gerada pela gramática.

D o autômato reconhece a linguagem sobre Σ em que os *strings* possuem o prefixo *ababc*.

E a linguagem reconhecida pelo autômato é a mesma que a representada pela expressão regular $(a + b + c)^*(ab)^*abc(a + b + c)^*$.

16) (POSCOMP, 2009)

Considere o autômato finito não-determinístico a seguir, sendo A o estado inicial e D o único estado de aceitação.



Que autômato finito determinístico com d como sua função de transição de estado aceita a mesma linguagem?

A) Estado Inicial A, estados de aceitação C e D
d (A, b) = B
d (B, a) = C
d (C, a) = D

B) Estado Inicial A, estado de aceitação C
d (A, b) = B
d (B, a) = C
d (C, a) = C

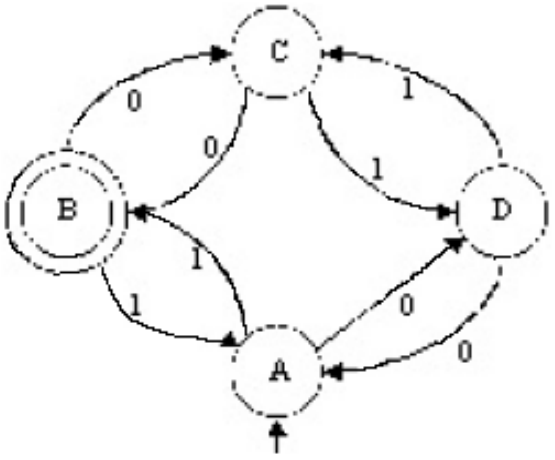
C) Estado Inicial A, estado de aceitação D
d (A, b) = B
d (B, a) = D
d (B, b) = C
d (C, a) = D

D) Todas as respostas acima estão corretas.

E) É impossível converter esse autômato finito não determinístico em um autômato finito determinístico.

17) (ENADE, 2005)

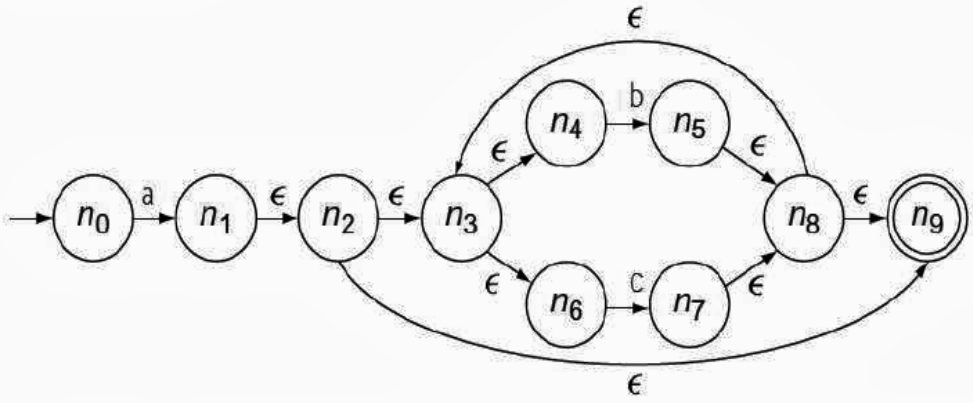
Que cadeia é reconhecida pelo autômato representado pelo diagrama de estados ao lado?



A) 101010
B) 111011000
C) 11111000
D) 10100
E) 00110011

18) (POSCOMP, 2012)

Considere o autômato a seguir.



(COOPER, K.; TORCZON, L. Engineering a Compiler. 2nd Edition. San Francisco: Morgan Kaufmann Publishers, 2012. p.51.)

Assinale a alternativa que apresenta a expressão regular que gera a mesma linguagem reconhecida pelo autômato.

a) $(ab)c^*$
b) $(a|b)c^*$
c) $a(b|c)^*$
d) $a(bc)^*$
e) $a(b)^*c$

19) Faça um AFD para cada item do exercício 4. Desenhe o grafo e a sua definição através da 5-upla de elementos M = (∑, Q, δ, q0, F).

Obs.: todo AFND ou AFλ gerado deve ser reconstruído como um AFD.

20) A partir do AFND fornecido através da quíntupla de elementos, faça o desenho do grafo, converta-o para um AFD representando-o pelo desenho do grafo, reescreva a sua quíntupla de elementos agora como AFD, escreva a ER equivalente e a gramática regular através de sua 4-upla de elementos.

M = (∑, Q, δ, q0, F)

∑ = { a,b,c }

Q = { q0, q1, q2, q3 }

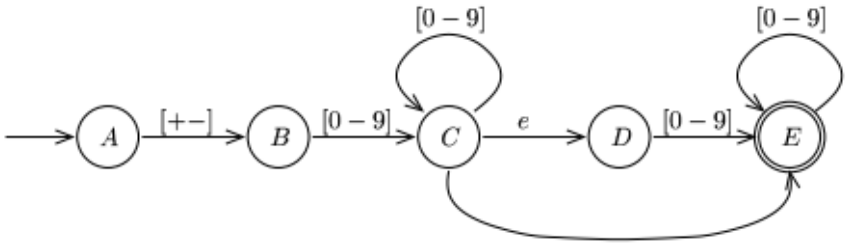
δ – função transição:

δ	a	b	c
q0	{q0,q1}	{q0}	{q0}
q1	{}	{q2}	{}
q2	{}	{}	{q3}
q3	{}	{}	{}

F = {q3}

21) (POSCOMP, 2002)

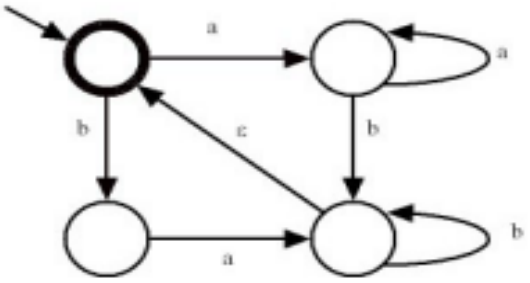
Assinale quantas seqüências de caracteres a seguir são reconhecidas pelo autômato finito abaixo. As quatro seqüências de caracteres (separadas por vírgulas) são: 0, +567, -89.5, -3e3.



(a) 0 (b) 1 (c) 2 (d) 3 (e) 4

22) (POSCOMP, 2008)

Seja o autômato finito mostrado na figura abaixo que opera sobre o alfabeto $\Sigma = \{a,b\}$ (o círculo em negrito indica um estado terminal):



Analise as seguintes afirmativas.

I. O autômato finito mostrado na figura é determinístico.

II. O autômato finito mostrado na figura é não-determinístico.

III. O autômato finito mostrado na figura reconhece a palavra vazia.

A análise permite concluir que

A) todas as afirmativas são falsas.

B) somente a afirmativa I é falsa.

C) somente a afirmativa II é falsa.

D) somente a afirmativa III é falsa.

E) nenhuma das afirmativas é falsa.

.

23) A partir do alfabeto ∑ = { a,b }, faça um AFND e uma ER para reconhecer e denotar, respectivamente a linguagem:

L = { w | o quinto símbolo da direita para esquerda de w é **a** }

24) Para cada uma das linguagens regulares descritas a seguir, faça uma ER (expressão regular), em seguida faça um AFD (autômato finito determinístico), e por último escreva a GR (gramática regular) equivalente a partir do algoritmo de transformação de ADF em GR.

Considere as letras minúsculas {a,b,c} como alfabeto.

a. $\{ a^i b^n a^p b^k \mid i >= 0, n >= 1, p >= 2, k >= 3 \}$

b. $\{ a^i b a b^{2k} \mid i >= 1, k>=0 \}$

c. $\{ a^{2i} b^{2n+1} c^k \mid i,n,k >= 0 \}$

d. $\{ (ab)^i c b^k \mid i,k >= 0\}$

25) (POSCOMP, 2003)

Uma gramática G é definida por:

$$G = (\{x, y, z\},\ \{S, W, X, Y, Z\},\ P,\ S)$$

na qual os membros de P são:

$$\begin{aligned} S &\to WZ \\ W &\to X \mid Y \\ X &\to x \mid xX \\ Y &\to y \mid yY \\ Z &\to z \mid zZ \end{aligned}$$

Qual das expressões regulares abaixo corresponde a esta gramática?

(a) $(xx^* \mid yy^*)zz^*$
(b) $xx^* \mid yy^* \mid zz^*$
(c) $xx^*(yy^* \mid zz^*)$
(d) $(xx \mid yy)^*zz^*$
(e) $xx^*yy^*zz^*$

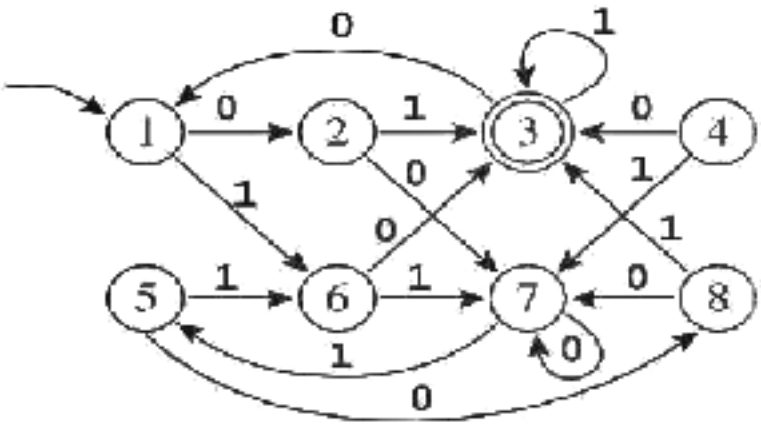
26) A partir de uma sequenca de caracteres formada apenas por dígitos, ou seja, o alfabeto $\sum = \{ 0,1,2,3,4,5,6,7,8,9 \}$, faça um AFD para buscar o número 153.

27) Considerando um texto formado apenas por elementos do alfabeto $\sum = \{ a,b,c \}$, construa uma AFD para buscar as palavras “aba” e “bac”.

28) Considerando um texto formado apenas por elementos do alfabeto $\sum = \{ 0,1 \}$, construa uma AFD para buscar as seguintes três palavras: “10”, “111” e “01”.

29) (POSCOMP, 2013)

Considere o autômato a seguir.



Sobre esse autômato, considere as afirmativas a seguir.

I. Os estados 3 e 7 são equivalentes.
II. Os estados 4 e 6 são equivalentes.
III. Os estados 1 e 5 são equivalentes.
IV. Os estados 2 e 8 são equivalentes.

Assinale a alternativa correta.

a) Somente as afirmativas I e II são corretas.
b) Somente as afirmativas I e IV são corretas.
c) Somente as afirmativas III e IV são corretas.
d) Somente as afirmativas I, II e III são corretas.
e) Somente as afirmativas II, III e IV são corretas.

30) Verifique que o Lema do Bombeamento funciona nas seguintes linguagens regulares.

a. $a^* b c^+$
b. $(a|b)^* ab$
c. $(ab)^+$

31) Descubra se as seguintes linguagens são regulares ou não.

Dica: para provar que uma linguagem é regular basta apresentar uma das três abstrações (AFD, ER ou GR); para provar que ela não é regular use o Lema do Bombeamento provando por absurdo.

Observe ainda que apenas mostrar que uma linguagem satisfaz o lema do bombeamento não é suficiente para provar que ela é regular.

a. $\{ (ab)^n (ab)^n \mid n >= 0\}$

b. $\{ (ab)^n (cd)^n \mid n >= 0\}$

c. { w | w é palavra da ling. com alfabeto {a,b} e contém quantidade maior de símbolos 'b' }

d. { w | w é palavra da ling. com alfabeto {a,b} e contém a mesma quantidade de 'a' e 'b' }

e. { $(ab)^n a^k$ | n >= k }

f. { w | w é palavra da ling. com alfabeto {a,b} e contém quantidade par de 'a' e par de 'b' }

g. { $a^n b^k c^{n+k}$ | n,k >= 0}

32) (POSCOMP, 2013)

Sobre o Lema do Bombeamento (*pumping lemma*) para linguagens regulares, considere as afirmativas a seguir.

I. Se o alfabeto $\sum = \{a, b\}$, então pode-se provar por absurdo, por meio do Bombeamento, que a linguagem $L_1 = \{w \in \sum^* \mid w \text{ termina com } b\}$ não é regular.

II. Se o alfabeto $\sum = \{a, b\}$, então pode-se provar por absurdo, por meio do Bombeamento, que a linguagem $L_2 = \{(a^n)^2 \mid n \geq 1\}$ não é regular.

III. Se o alfabeto $\sum = \{a, b\}$, então pode-se provar por absurdo, por meio do Bombeamento, que as linguagens $L_3 = \{a^{n!} \mid n > 1\}$, $L_4 = \{a^n b a^m b a^{n+m} \mid n, m > 1\}$ e $L_5 = \{a^{m+1} b^{n+1} \mid 2 < n < m < 3n\}$ não são regulares.

IV. Se a linguagem for do tipo 3, então aplica-se o Bombeamento.

Assinale a alternativa correta.

a) Somente as afirmativas I e II são corretas.

b) Somente as afirmativas I e IV são corretas.

c) Somente as afirmativas III e IV são corretas.

d) Somente as afirmativas I, II e III são corretas.

e) Somente as afirmativas II, III e IV são corretas.

33) (POSCOMP, 2011)

Considere a seguinte propriedade sobre uma linguagem formal L: "Existe um número $p \geq 0$, tal que para qualquer palavra $w \subset L$, $|w| \geq p$, existem palavras x, y e z, com $y \neq \varepsilon$ e $|xy| \leq p$, tais que, para qualquer inteiro $i \geq 0$, a palavra $xy^i z \in L$".
Com base no enunciado e nos conhecimentos sobre o tema, atribua V (verdadeiro) ou F (falso) para as afirmativas a seguir.

( ) Se L é aceita por AFND, então L satisfaz a propriedade acima.

( ) A linguagem formada de 1's e 0's com igual quantidade de ocorrências das palavras 01 e 10 satisfaz a propriedade acima.

( ) A propriedade acima é falsa para a linguagem $0^i 1^k 2^j / i, j, k \geq 0$ e se $i = 1$, então $k = j$.

( ) A linguagem $\{a^n b^n c^n / n \geq 0\}$ não satisfaz a propriedade acima.

( ) A linguagem $\{a^n b^m / n, m \geq 0 \text{ e } n \neq m\}$ satisfaz a propriedade acima.

Assinale a alternativa que contém, de cima para baixo, a sequência correta.

a) V, V, V, V, F.

b) V, V, F, V, F.

c) V, F, V, F, F.

d) F, V, V, F, V.

e) F, V, F, V, V.

34) (POSCOMP, 2012)

Seja um Autômato Finito Não Determinístico (AFN) com 6 estados. Aplicando-se o algoritmo de conversão de um AFN para um Autômato Finito Determinístico (AFD), em quantos estados, no máximo, resultaria o AFD considerando-se os estados inúteis?

a) 12
b) 36
c) 64
d) 1024
e) 46656

35) Faça uma Máquina de Moore para classificar operadores lógicos AND, OR e NOT (maiúsculos ou minúsculos), variáveis formadas por letras minúsculas e a palavra reservada if (minúscula).
Defina a sua 7-upla de elementos.

ER para cada saída da máquina:

<OP-LOGICO> `(a|A)(n|N)(d|D) | (o|O)(r|R) | (n|N)(o|O)(t|T)`
<IF> `if`
<VAR> `(a|...|z)+`

Observe atentamente a característica determinística necessária para a máquina.
Exemplos de variáveis possíveis: `a, an, anda, x, ifa`
Observe ainda que a máquina não deve aceitar entradas como: `Ax, Anda`

36) Faça uma Máquina de Mealy sobre o alfabeto {0,1} para:

a. Ligar (1) ou desligar (0) uma máquina. Como o início do funcionamento da máquina é custoso, o comando para ligar (1) deve ser repetido 3 vezes consecutivamente para que funcione. Já o comando de desligar (0) é dado uma única vez.

b. Devolver o bit lido com atraso de uma transição. Para o primeiro bit emita um '0'.

Obs.: em todos os casos defina a sua 6-upla de elementos.

37) Faça uma máquina de estados finitos (decida entre Máquina de Moore e Máquina de Mealy) para representar a venda de café em um quiosque automatizado. O café custa 60 centavos e a máquina aceita somente moedas de 10, 50 centavos e 1 real. O café é feito automaticamente quando a máquina atinge crédito de 60 centavos. Havendo troco, o mesmo entra como crédito para o próximo café.

38) (POSCOMP, 2014)

Sobre as linguagens regulares, considere as afirmativas a seguir.

I. As linguagens regulares podem ser expressas por máquinas de Moore e de Mealy.

II. As linguagens regulares podem ser expressas por um autômato finito.

III. Se A e B são linguagens regulares, então $A \cap B$ também é.

IV. Seja $B = \{ba, na\}$. Pode-se dizer que $B^* = \{\lambda, ba, na, ab, an, baba, bana, naba, anab, nana, aban, bababa, babana, banaba, banana, nababa, nabana, nanaba, nanana, abanba, babababa, ...\}$.

Assinale a alternativa correta.

a) Somente as afirmativas I e II são corretas.

b) Somente as afirmativas I e IV são corretas.

c) **Somente as afirmativas III e IV são corretas.**

d) Somente as afirmativas I, II e III são corretas.

e) Somente as afirmativas II, III e IV são corretas.

39) (POSCOMP, 2004)

Seja a seguinte linguagem, onde ϵ representa a sentença vazia:

S	→	AB	|	CD
A	→	a	|	ϵ
B	→	b	|	f
C	→	c	|	g
D	→	h	|	i

Qual o conjunto de terminais que podem começar sentenças derivadas de S ?

a) {a, c, g}

b) {a, b, f, c, g}

c) {a, b, f, c, g, h, i}

d) {a, c, g, h, i}

e) {a, b, f}

40) (POSCOMP, 2011)

Considere, a seguir, a gramática livre de contexto:

$$S \rightarrow aS|Sb|c$$

Qual expressão regular gera a mesma linguagem que a gramática definida acima?

a) a* c a*

b) a+ b+ c

c) a+ c b+

d) c a* a*

e) c a+ b+

41) (ENADE, 2008)

DISCURSIVA

Qualquer expressão aritmética binária pode ser convertida em uma expressão totalmente parentizada, bastando reescrever cada subexpressão binária a ⊗ b como (a ⊗ b), em que ⊗ denota um operador binário. Expressões nesse formato podem ser definidas por regras de uma gramática livre de contexto, conforme apresentado a seguir. Nessa gramática, os símbolos não-terminais E, S, O e L representam expressões, subexpressões, operadores e literais, respectivamente, e os demais símbolos das regras são terminais.

E → (S O S)

S → L | E

O → + | - | * | /

L → a | b | c | d | e

Tendo como referência as informações acima, faça o que se pede a seguir.

A Mostre que a expressão `(a * (b / c))` pode ser obtida por derivações das regras acima. Para isso, desenhe a árvore de análise sintática correspondente.

B Existem diferentes derivações para a expressão `((a + b) * c) + (d * e))`. É correto, então, afirmar que a gramática acima é ambígua? Justifique sua resposta.

42) Faça uma gramática livre de contexto (GLC) para cada uma das linguagens de nível 2, ou seja, livres de contexto.

a. $\{ a^{2i} b^{2i} c^{2k} \mid i >= 0, k >= 0 \}$

b. $\{ a^{i} b^{i} a^{k} b^{k} \mid i >= 0, k >= 1 \}$

c. $\{ a^{i} b^{k} a^{k} b^{i} \mid i,k >= 0\}$

d. { $(ab)^i$ c b^i | i >= 0}

e. { a^i c^k b^{2i} | i >= 1, k >= 0}

f. { a^i c^k b^n | i=k ou inclusivo k=n}

g. { a^i b^n | i é diferente de n}

h. { a^i b^n | 0 <= i <= n <= 2i}

i. { a^{i+n} b^i c^n | i,n >= 0}

j. palíndromos ímpares sobre {a,b}

k. strings sobre {a,b} com a mesma quantidade de a's e b's.

43) (POSCOMP, 2015)

Considerando as linguagens $L = \{ 0^n1^n2^i \mid n \geq 0 \text{ e } i \geq 0 \}$ e $M = \{ 0^i1^n2^n \mid n \geq 0 \text{ e } i \geq 0 \}$, pode-se afirmar que

(A) a linguagem $L \cup M$ pode ser gerada por uma gramática livre de contexto.

(B) a linguagem M pode ser gerada por uma gramática regular.

(C) a linguagem L pode ser aceita por um autômato finito determinístico.

(D) a linguagem $L \cap M$ pertence à classe das linguagens livres de contexto.

(E) a linguagem M pode ser denotada por uma expressão regular.

44) (POSCOMP, 2010)

Considerando as linguagens $L_1 = \{a^i b^m c^n; i \geq 0, m \geq 0, n \geq 0\}$ e $L_2 = \{a^i b^m c^n; i \geq 0, m \geq 0, n = i + m\}$ sobre o alfabeto $\sum = \{a, b, c\}$, considere as afirmativas a seguir.

I. L_1 **é uma linguagem regular.**

II. L_2 **é uma linguagem regular.**

III. Existe um autômato de pilha determinístico que reconhece L_1**.**

IV. A linguagem L_2 **pode ser gerada pela** $G = (\{X, Y\}, \{a, b, c\}, \{X \to aXb, X \to Y, Y \to cYb, Y \to \lambda\}, X)$**, onde** λ **é a palavra vazia.**

Assinale a alternativa correta.

a) Somente as afirmativas I e II são corretas.

b) Somente as afirmativas II e IV são corretas.

c) Somente as afirmativas III e IV são corretas.

d) Somente as afirmativas I, II e III são corretas.

e) **Somente as afirmativas I, III e IV são corretas.**

45) (POSCOMP, 2002)

Para a gramática a seguir, qual o conjunto de terminais que pode aparecer como primeiro terminal após o não-terminal A, em qualquer forma sentencial gerada pela gramática abaixo (isto é, não necessariamente imediatamente após A), onde ε representa a sentença vazia?

$S \rightarrow ABCDd$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow bC \mid \varepsilon$
$C \rightarrow cD \mid \varepsilon$
$D \rightarrow e$

a) {d}

b) {b}

c) {b,c,e}

d) {b,c,d,e}

e) {e}

46) A partir da gramática fornecida, descreva, com precisão a linguagem gerada. Se a linguagem for regular, use a notação de expressão regular para a representá-la. Considere letras maiúsculas como símbolos não terminais (variáveis) e letras minúsculas como símbolos terminais.

a)
```
S -> aS | bB | λ
B -> cB | λ
```

b)
```
S -> aSb | aAb
A -> bAa | λ
```

c)
```
S -> aSbb | A
A -> cA | c
```

d)
```
S -> AAASb | λ
A -> a | λ
```

e)
```
S -> aaSB | λ
B -> bB | b
```

47) (POSCOMP, 2003).

Considere o seguinte trecho de programa:

```
1. i:= 1;
2. while i <= n do
   begin
3.    sum:= sum + a[i];
4.    i:= i + 1;
   end;
```

Considere que:
- I representa a inicialização da variável `i:= 1` na linha 1;
- T representa o teste da linha 2;
- A representa os comandos da linha 3;
- P representa o incremento na linha 4.

Qual é a expressão regular que representa todas as seqüências de passos possíveis de serem executados por este trecho de programa?

(a) $I(TAP)^+$
(b) $I(TAP)^*$
(c) $IT^+A^*P^*$
(d) $IT(APT)^*$
(e) $IT(APT)^+$

48) Crie uma gramática livre de contexto para gerar uma lista de expressões aritméticas formadas apenas por números inteiros sem sinal de 20 a 99 e pelo operador de adição (as expressões são separadas por ponto e vírgula).

Assuma a existência dos seguintes tokens:
<DÍGITO1> - representa exatamente um dígito de 0 a 9.
<DÍGITO2> - representa exatamente um dígito de 2 a 9.
<SOMA> - representa o caractere '+'.
<PT-VIRG> - representa o caractere ';'.

Observe um exemplo de string que deve ser gerada pela gramática:
`23 + 44 + 99 ; 25 ; 51 + 20`

49) (ENADE, 2008)

Considere a gramática G definida pelas regras de produção ao lado, em que os símbolos não-terminais são `S`, `A` e `B`, e os símbolos terminais são `a` e `b`.

```
S → AB
AB → AAB
A → a
B → b
```

Com relação a essa gramática, é correto afirmar que

A) a gramática G é ambígua.
B) a gramática G é uma gramática livre de contexto.
C) a cadeia `aabbb` é gerada por essa gramática.
D) é possível encontrar uma gramática regular equivalente a G.
E) a gramática G gera a cadeia nula.

50) Elabore uma gramática livre de contexto para gerar expressões com parênteses, colchetes e chaves balanceadas, as quatro operações básicas, números e variáveis. Admita a existência de tokens para representar cada um dos elementos citados.

Observe exemplos de strings da linguagem:

```
[2*(4+3)],  100, {w-[3-5]+{4-(x-((y)))-1}-9}-8,  valor+contador,

({[(2)]}), [2-{x-5}]
```

51) A partir do exercício anterior, faça a seguinte modificação: parênteses, colchetes e chaves são opcionais, mas se eles aparecerem na mesma expressão eles devem obedecer a seguinte ordem: parênteses (mais internos), colchetes e chaves (mais externos).

Exemplos corretos:

```
[2*(4+3)],  100, {w-[3-5]+{4-(x-((y)))-1}-9}-8,  valor+contador
```

Exemplos incorretos:

```
({[(2)]}), [2-{x-5}]
```

52) **Desafio:** no exercício anterior, somente permita a ocorrência de colchetes se houver parênteses internos, e chaves se houver colchetes.

53) Desenvolva uma gramática livre de contexto para gerar expressões aritméticas no estilo da linguagem LISP. Use apenas números inteiros sem sinal. Considere, para cada operação, exatamente dois operandos. Suponha a existência de tokens para os elementos que compõem as expressões.

Exemplos:
```
(+ 2 3)
(- (* 20 30) 4)
(+ (+ 1 4) (+ 3 (+ 3 5)))
```

54) Elabore uma gramática para gerar declarações de classes em Java. Assuma que o corpo da classe já está pronto.

Observações:

- Antes da palavra reservada `class` é possível aparecer um modificador de acesso: `public`, `private` ou `protected`.

É possível a ocorrência da palavra abstract (antes ou depois do modificador de acesso).

- Só pode haver uma classe Pai no caso de herança com `extends`, e vários Pais em herança com `implements` (herança múltipla). As cláusulas `extends` e `implements` devem aparecer nesta ordem.
- Classes interface só podem herdar de classes interface (devem usar a palavra `implements`). Neste caso, não possuem modificador de acesso.

Observe quatro exemplos de strings desta linguagem:

```
public abstract class MinhaClasse { corpo }
public class MinhaClasse extends Pai1 implements Pai2,Pai3 {<corpo>}
abstract private class MinhaClasse extends Pai { <corpo> }
interface MinhaInterface { <corpo> }
```

55) Sobre ambigüidade em gramáticas livres de contexto, resolva as seguintes situações:

a) Crie uma gramática não ambígua para resolver o problema do else flutuante. Considere que um else sempre pertence ao if mais próximo.
Por exemplo, no caso abaixo (com identação errada de forma proposital) o else pertence ao `if(y>8)`

Considere disponíveis todos os tokens necessários.

```
if(x>5)
   if(y>8)
      comando1;
else
      comando2;
```

b) Prove que a gramática a seguir é ambígua.
Proponha uma gramática equivalente não ambígua.

```
exp   ->   exp <OPER-ADIT> exp
         | <ABRE-PAR> exp <FECHA-PAR>
         | <NUM>
```

56) (POSCOMP, 2009)

Considere uma produção pertencente a uma gramática G dada por:

$$L \rightarrow L\ a\ S \mid S$$

Assinale a alternativa abaixo que, substituindo essa produção, elimina a recursividade à esquerda criando uma gramática equivalente:

A) $L \rightarrow R\ S$
   $R \rightarrow a\ S\ R \mid \varepsilon$

B) $L \rightarrow S\ R$
   $R \rightarrow a\ S\ R \mid \varepsilon$

C) $L \rightarrow S\ R$
   $R \rightarrow S\ a\ R \mid \varepsilon$

D) $L \rightarrow S\ a\ R$
   $R \rightarrow S\ a\ R \mid \varepsilon$

E) $L \rightarrow R\ S$
   $R \rightarrow a\ R\ S \mid \varepsilon$

57) Retire a recursividade à esquerda e fatore (quando for possível) os trechos de gramáticas a seguir.

a)
```
lista -> <IDENT> <VIRG> lista | <IDENT>
```

b)
```
comandos -> comandos comando | <FIM>
comando  -> <IDENT> <ATRIB> exp
          | <IDENT> tipo
tipo     -> <REAL> | <INT>
exp      -> exp <OPERADOR> exp
          | <ABRE-PAR> exp <FECHA-PAR>
          | <NUM>
          | <IDENT>
```

c) `S -> aS | bS | cS | aA | aB | aC | bA | bB | bC`

d) `lista -> lista <VIRG> fim | lista <PT-VIRG> fim | fim`

e) `termo -> termo outro | fator <OP> | fator <OPCONT>`

f) `S -> AaB | AaS | a`

g) `S -> abB | aB | abC | abBC | ag | g | h`

h) `A -> Aa | b | c | Ad`

i) `S -> Sab | SA | aA | aS | c`

j)
```
exp   ->   exp <OP> exp | termo
termo ->  <ABRE-PAR> exp <FECHA-PAR> | <NUM> | <IDENT>
```

58) Reescreva as gramáticas na Forma normal de Chomsky e na Forma Normal de Greibach. Considere as letras maiúsculas como símbolos não terminais e as minúsculas como símbolos terminais.

a) `S -> aS | Sb | c`

b) `E -> E o E | a E a | n`

59) A partir da gramática BNF do Pascal fornecida no anexo A desta lista de exercícios, responda os itens solicitados. Não use seu conhecimento sobre a linguagem Pascal. É importante ressaltar que muitos erros que um compilador acusa são provenientes do analisador semântico e que não poderiam ser detectados pela gramática usada na análise sintática.

a) O comando `for` permite expressões com variáveis em seus limites?
Exemplo: `for x := y+1 to a*b do ...`

b) A gramática permite números reais nos valores de limite de um comando `for`?
Exemplo: `for x:= 2.5 to 8.5 do ...`

c) A gramática permite um `for` onde os valores inicial e final estão invertidos?
Exemplo: `for x:= 10 to 5 do ...`

d) A gramática permite uma chamada aninhada nos índices de um vetor?
Exemplo: `Vetor[Valor[Codigo[x]]]`

e) O ponto-e-vírgula antes do `else` é proibido, obrigatório ou facultativo?

f) A gramática aceita divisão por zero em expressões aritméticas?

g) Os operadores `DIV` e `MOD` podem ser usados para números reais?

h) É possível uma construção aninhada de `BEGIN` e `END`?
Se sim, analise como ficam as duas situações abaixo, ou seja, verifique se o ponto-e-vírgula é proibido, facultativo ou obrigatório:

Situação 1:
```
BEGIN
      BEGIN
            x := x + 1;
      END;
END;
```

Situação 2:
```
BEGIN
      BEGIN
            x := x + 1;
      END;
      y := y + 2;
END;
```

i) É possível uma construção aninhada de WHILE?
Por exemplo:
```
while Flag1 do
      while Flag2 do
            while Flag3 do
                  x := x + 1;
```

j) A estrutura `while` permite expressões condicionais contendo atribuições, tal como é permitido na Linguagem C?
Exemplo: `while x:=x+1 do ...`

k) A estrutura `while` permite expressões contendo chamadas de funções?
Exemplo: `while Verifica(x) do ...`

60) Analise a gramática do Java que segue no anexo B. Observe que ela usa um o formato BNF estendido (EBNF), ou seja, o lado direito das regras usa os mesmos metasímbolos que são permitidos nas expressões regulares.

A seta é substituída por um sinal de igual e os terminais são representados entre apóstrofos, além disso a regra completa é terminada por um ponto-e-vírgula.

Responda cada item abaixo a partir da análise sobre esta gramática.

a) Quantas classes podem ser herdadas com extends e com implements?
A linha de código abaixo é válida?
`class Monitor extends Aluno,Pessoa implements Funcionario, Cidadão`
É possível trocar de posição as palavras `extends` e `implements`?

b) No sintaxe do Java atual, o uso das chaves é obrigatório nos comandos de tratamento de erros `catch` e `finally`, mesmo que exista apenas uma linha em seu corpo. Sendo que é possível vários `catch` para um mesmo `try, mas apenas um finally`.
Como a gramática fornecida se comporta diante de tal situação?

Exemplo correto:

```
try {
...
}
catch(Erro1 e) {
   coisa1();
}
catch(Erro2 e) {
   coisa2();
}
finally {
   coisa3();
}
```

Exemplo incorreto:

```
try {
...
      }
catch(Erro1 e)
         coisa1();
      catch(Erro2 e)
   coisa2();
      finally
         coisa3();
```

c) Na declaração de uma função em Pascal é possível usar a escrita de um mesmo tipo para vários argumentos:

```
function FazAlgo(arg1,arg2:Integer; arg3,arg4:Real): UmTipo;
```

Existe possibilidade de fazer algo parecido em Java?

d) Existem várias situações nesta gramática onde as regras poderiam ser simplesmente substituídas por expressões regulares, ou seja, poderiam ser meramente tokens. Cite alguns casos em que isto ocorre, fornecendo em seguida a respectiva expressão regular.

61) Refaça o exercício 53 (gramática livre de contexto para gerar expressões aritméticas no estilo da linguagem LISP) utilizando-se do recurso do formato BNF.

62) (ENADE, 2014)

Uma gramática livre do contexto (GLC) é um modelo computacional geralmente utilizado para definir linguagens de programação e, quando está de acordo com a Forma de Backus-Naur (BNF), permite que alguns operadores sejam utilizados no lado direito de suas produções, como o operador | (*pipe*) que indica seleção, o operador [ ] que indica que o operando em questão é opcional, e o operador * que indica repetição de 0 ou mais vezes.

Projetar um compilador para uma determinada linguagem envolve, entre outras coisas, especificar quais são os símbolos válidos nesta linguagem, bem como quais são as regras sintáticas que a definem.

A linguagem de programação Java é uma linguagem com suporte à orientação a objetos que não permite herança múltipla e que permite que uma classe implemente múltiplas interfaces. A seguir, exibem-se trechos de código sintaticamente válidos na linguagem Java.

Trecho 1:

```
class A extends B {  }
```

Trecho 2:

```
class F implements C {  }
```

Trecho 3:

```
class J extends A implements C, D {  }
```

No trecho 1, cria-se uma classe chamada A que herda de uma classe chamada B. No trecho 2, cria-se uma classe chamada F que implementa uma interface chamada C. No trecho 3, cria-se uma classe chamada J que herda de uma classe chamada A e implementa duas interfaces, chamadas C e D.

63) Refaça o exercício 53 (gramática livre de contexto para gerar expressões aritméticas no estilo da linguagem LISP) utilizando-se do recurso do formato EBNF.

64) Faça uma gramática no formato EBNF para gerar declaração de arrays multidimensionais em Pascal.

Formato geral:
```
lista_de_identificadores : array [li1..ls1, li1..ls1,..., li1..ls1] of tipo;
```

Exemplo de uma string da linguagem:
```
x,y,z : array [0..10,50..500] of Elemento;
```

Escreva duas soluções: uma considerando a existência de tokens já classificados, e outra partindo do 'zero', ou seja usando os símbolos terminais diretamente na gramática.

65) Reescreva cada gramática a seguir no formato EBNF.

a)
```
lista ->  <IDENT> | <IDENT> <VIRG> lista
```

b)
```
exp ->  termo <ADD> exp
      | termo <SUB> exp
      | termo
termo -> <NUM> | <IDENT>
```

c)
```
inicio     -> <BEGIN> corpo <END> <EOF>
corpo      -> declaraVar comandos
declaraVar -> <DECVAR> listaVar
listaVar   -> <IDENT> | <IDENT> listaVar
comandos   -> comando comandos | comando
comando    -> atribuicao | se | enquanto | leia | exibe
```

d)
```
declaraFunção -> <FUNCTION> <IDENT> parâmetro <DOIS-PT>
                 <IDENT> <PT-VIRG> declaraVar corpoFunção
parâmetro -> <ABRE-PAR> listaArgumentos <FECHA-PAR> | λ
listaArgumentos -> argumento <PT-VIRG> ListaArgumentos | argumento
argumento -> referência defVar
referência -> <VAR> | λ
defVar -> listaIdent <DOIS-PT> tipo
tipo -> <IDENT>
listaIdent -> <IDENT> <VIRG> listaIdent | <IDENT>
declaraVar -> λ | <VAR> listaDeclara
listaDeclara -> defVar <PT-VIRG> listaDeclara | defVar
corpoFunçao -> blocoComandos
```

66) A partir da linguagem descrita a seguir, crie todos os tokens necessários através de suas expressões regulares. Crie também a gramática livre de contexto no formato EBNF.

A seguinte linguagem descrita representa a Lógica Proposicional e trabalha com expressões proposicionais. Ela possui 4 comandos:

1. comando de atribuição:

```
variável := expressão
```

2. comando de exibição:

```
out expressão_1, expressão_2, ..., expressão_n
```

3. comando de entrada de dados:

```
in variável_1, variável_2, ..., variável_n
```

4. comando condicional:

```
if(expressão) {
    comandos_1
}
else {
   comandos_2
}
```

Os comandos de atribuição, exibição e de entrada de dados devem ser terminados por ponto-e-vírgula.

Os nomes das variáveis são formadas por uma seqüência de letras minúsculas e/ou dígitos. Mas, sempre devem iniciar por letra.

A linguagem é case-sensitive.

A expressão é formada por parênteses balanceados, valores lógicos, variáveis, operadores lógicos.

Os valores lógicos podem ser: `1, 0, "v", "f", "verdadeiro", "falso" "false", "true"` (observe que em alguns casos exige-se aspas).

Os operadores podem ser: `v, ^, ', ->, <->`

Exemplos de programas aceitos:

```
in a;
out a;
```

```
in a, b,c;
out a ^ b, a v b, a -> c;
```

```
in a,b,c;
d := a ^ (b <-> c);
if(d) {
   out d ^ a;
}
else{
   out d ^ c;
```

```
}
```

```
a := "v";
b := "f";
c := a -> b;
out c,c ^ 1;
```

```
in a,b,c;
d := (b' v a) ^ ((b <-> c) v (c'' ^ a)')';
if(a ^ (b <-> c)) {
   e := 1;
   f := 0;
   out d ^ a;
}
else{
   if(d) {
      e := 0;
   }
   else{
      f := 1;
   }
   out d v a;
}
out e ^ f;
```

```
out "v" ^ "f" -> "true" <-> "false";
```

67) (ENADE, 2014)

Considere que:

- <classdecl> é um não terminal cujo intuito é dar origem a declarações de classes;
- <classbody> é um não terminal cujo intuito é dar origem ao corpo de classes;
- os terminais aparecem entre aspas duplas;
- "id" é um símbolo que representa qualquer identificador válido, como nomes de classes ou variáveis.

A gramática que especifica uma linguagem que não permita herança múltipla e que implemente zero ou mais interfaces é

A <classdecl> "class" "id" ["extends "] "id" <classbody>
B <classdecl> "class" "id" ("extends " "id")* <classbody>
C <classdecl> "class" "id" ["extends "] "id" ["implements" (, | "id")*] <classbody>
D <classdecl> "class" "id" ["extends " "id"] ["implements" "id" ("," "id")*] <classbody>
E <classdecl> "class" "id" ["extends " "id"] "implements" "id" ("," "id")* <classbody>

68) (POSCOMP, 2014)

Sobre o lema do bombeamento (*pumping lemma*) para linguagens regulares, considere as afirmativas a seguir.

I. Seja o alfabeto $\sum = \{a, b\}$. Pode-se provar por absurdo, através do bombeamento, que a linguagem $L_1 = \{w \in \sum^* \mid w \text{ termina com } b\}$ não é regular.

II. Seja o alfabeto $\sum = \{a, b\}$. Pode-se provar por absurdo, através do bombeamento, que a linguagem $L_2 = \{(a^n)^2 \mid n \geq 1\}$ não é regular.

III. Seja o alfabeto $\sum = \{a, b\}$. Pode-se provar por absurdo, através do bombeamento, que as linguagens $L_3 = \{a^{n!} \mid n \geq 1\}$, $L_4 = \{a^n b a^m b a^{n+m} \mid n, m \geq 1\}$ e $L_5 = \{a^{m+1} b^{n+1} \mid 2 \leq n \leq m \leq 3n\}$ não são regulares.

IV. Se a linguagem for do tipo 3, pode-se aplicar o bombeamento.

Assinale a alternativa correta.

a) Somente as afirmativas I e II são corretas.

b) Somente as afirmativas I e IV são corretas.

c) Somente as afirmativas III e IV são corretas.

d) Somente as afirmativas I, II e III são corretas.

e) Somente as afirmativas II, III e IV são corretas.

69) (POSCOMP, 2008)

Analise as seguintes afirmativas.

I. Todo autômato finito não-determinístico pode ser simulado por um autômato finito determinístico.

II. Todo autômato finito determinístico pode ser simulado por um autômato finito não-determinístico.

III. Todo autômato finito não-determinístico pode ser simulado por um autômato de pilha determinístico.

IV. Todo autômato de pilha determinístico pode ser simulado por um autômato finito não-determinístico.

V. Todo autômato finito não-determinístico pode ser simulado por uma máquina de *Turing* determinística.

A análise permite concluir que estão ***CORRETAS***

A) apenas as afirmativas I, II, III e IV.

B) apenas as afirmativas II, III e V.

C) apenas as afirmativas I, II, III e V.

D) apenas as afirmativas II e IV.

E) apenas as afirmatias I, II e IV.

70) De acordo com a Hierarquia de Chomsky, marque (3) para linguagem regular, (2) para linguagem livre de contexto e, (1) para linguagem sensível ao contexto, (0) para linguagem enumerável recursivamente.

a) ( ) Autômato finito com movimentos vazios.

b) ( ) Gramáticas usadas na maioria das linguagens de programação conhecidas.

c) ( ) Autômato finito determinístico.

d) ( ) Autômato finito não determinístico.

e) ( ) Expressão regular.

f) ( ) Máquina de Turing.

g) ( ) $\{ a^i b^i c^i \ ; i > 0 \}$

h) ( ) Linguagens naturais.

i) ( ) $\{ a^i b^n c^i \ ; i,n > 0 \}$

j) ( ) Linguagem tipo 3

k) ( ) Linguagem tipo 2

l) ( ) Linguagem tipo 1

m) ( ) Linguagem tipo 0

71) Complete as frases usando conceitos adequados no contexto da Hieraquia de Chomsky.

a) ..... é um formalismo usado para denotar (representar) linguagens regulares.

b) ..... é um formalismo gerador (ou axiomático) para linguagens regulares.

c) ..... é um formalismo reconhecedor para linguagens regulares.

d) ..... é um formalismo gerador (ou axiomático) para linguagens livres de contexto.

e) ..... é um formalismo reconhecedor para linguagens livres de contexto.

f) ..... é um formalismo gerador (ou axiomático) para linguagens sensíveis ao contexto.

g) ..... é um formalismo reconhecedor para linguagens sensíveis ao contexto.

h) ..... é um formalismo gerador (ou axiomático) para ling. enumeráveis recursivamente.

i) ..... é um formalismo reconhecedor para linguagens enumeráveis recursivamente.

72) (POSCOMP, 2002)

Sobre a Hierarquia de Chomsky, podemos afirmar:

(a) Uma linguagem que é recursivamente enumerável não pode ser uma linguagem regular
(b) As linguagens livres de contexto e as linguagens sensíveis a contexto se excluem
(c) Uma linguagem que não é regular é livre de contexto
(d) As linguagens reconhecidas por autômatos a pilha são as linguagens regulares
(e) Há linguagens que não são nem livres de contexto nem sensíveis a contexto

73) (POSCOMP, 2007)

Seja a linguagem formal $L = \{a^n b^{2n} c, n \geq 0\}$. Analise as seguintes assertivas.

I. L é uma linguagem livre de contexto.

II. A gramática $G = (\{S, X\}, \{a, b, c\}, \{S \rightarrow Xc, X \rightarrow aXbb|\epsilon\}, S)$ gera a linguagem L.

III. L não pode ser reconhecida por um autômato com pilha.

A análise permite concluir que estão **CORRETAS**

(a) apenas as assertivas I e II.

(b) apenas as assertivas I e III.

(c) apenas as assertivas II e III.

(d) todas as assertivas.

(e) nenhuma das assertivas.

74) (ENADE, 2005)

estado	símbolo lido na fita	símbolo gravado na fita	direção	próximo estado
início	●	●	direita	0
0	0	1	direita	0
0	1	0	direita	0
0	⌂	⌂	esquerda	1
1	0	0	esquerda	1
1	1	1	esquerda	1
1	●	●	direita	parada

Na tabela acima, estão descritas as ações correspondentes a cada um dos quatro estados (início, 0, 1, parada) de uma máquina de Turing, que começa a operar no estado "início" processando símbolos do alfabeto {0,1,●, ⌂}, em que '⌂' representa o espaço em branco. Considere que, no estado "início", a fita a ser processada esteja com a cabeça de leitura/gravação na posição 1, conforme ilustrado a seguir.

1	2	3	4	5	6	7	8	9	10	11	...
●	0	1	1	0	1	⌂	⌂	⌂	⌂	⌂	...

Considerando essa situação, assinale a opção que indica corretamente a posição da cabeça de leitura/gravação e o conteúdo da fita após o término da operação, ou seja, após a máquina atingir o estado "parada".

(A)

1	2	3	4	5	6	7	8	9	10	11	...
●	0	0	1	1	1	1	0	0	1	1	...

(B)

1	2	3	4	5	6	7	8	9	10	11	...
●	0	1	1	0	1	⌂	⌂	⌂	⌂	⌂	...

(C)

1	2	3	4	5	6	7	8	9	10	11	...
●	0	1	1	0	1	0	1	0	0	1	...

(D)

1	2	3	4	5	6	7	8	9	10	11	...
●	⌂	⌂	⌂	⌂	⌂	1	⌂	⌂	⌂	⌂	...

(E)

1	2	3	4	5	6	7	8	9	10	11	...
●	1	0	0	1	0	⌂	⌂	⌂	⌂	⌂	...

75) Enquadre, dentro da Hierarquia de Chomky, cada uma das linguagens a seguir como livre de contexto (L) ou (S) sensível ao contexto.
Após a linguagem ter sido contextualizada no nível apropriado, construa o formalismo gerador mais adequado (GLC - gramática livre de contexto, ou GSC - gramática sensível ao contexto), e desenhe a máquina abstrata referente ao formalismo reconhecedor mais adequado (AP - autômato com pilha, ou MT - máquina de Turing).

a) ( ) $\{ c\ a^n\ c\ b^n\ c \mid n \geq 0 \}$

b) ( ) $\{ a^n\ b^n \mid n \geq 2 \}$

c) ( ) palíndromos ímpares sobre {a,b}

d) ( ) $\{ a^i\ b^n\ c^k \mid k = i+n \}$

e) ( ) $\{ a^i\ b^n\ c^k \mid n = i+k \}$

f) ( ) $\{ a^{2i}\ b^{2i}\ c^{2k} \mid i \geq 0, k \geq 0 \}$

g) ( ) $\{ a^n\ b^n\ c^n\ d^n \mid n \geq 1 \}$

h) ( ) $\{ a^{2n}\ b\ a^n\ a^{2n} \mid n \geq 0 \}$

i) ( ) $\{ a^i\ b^i\ a^k\ b^k \mid i \geq 0, k \geq 1 \}$

j) ( ) $\{ a^n\ b^k\ a^n\ b^k \mid i \geq 0, k \geq 0 \}$

k) ( ) $\{ a^i\ b^k\ a^k\ b^i \mid i,k \geq 0\}$

l) ( ) $\{ (ab)^i\ c\ b^i \mid i \geq 0\}$

m) ( ) strings sobre {a,b} com a mesma quantidade de a's e b's.

n) ( ) $\{ a^i\ b^n \mid i$ é diferente de $n\}$

o) ( ) $\{ bb\ a^i\ bb\ a^i\ bb\ a^i\ bb \mid i \geq 1 \}$

p) ( ) $\{ a^{2i}\ b^i\ c^{2i} \mid i \geq 0 \}$

q) ( ) { w c w | w é uma string qualquer sobre {a,b} }

r) ( ) { w c w c w | w é uma string qualquer sobre {a,b} }

76) (POSCOMP, 2015)

A gramática G = ({S, A, B}, {0, 1}, P, S), onde P é dado pelas regras de produção

S → 0AB | 1BA
A → 0AS | 1A | ε
B → 0B | 1BS | ε

gera uma linguagem que

(A) pertence à classe Regular.

(B) contém a cadeia vazia ε.

(C) pode ser aceita por um autômato com pilha.

(D) pode ser denotada por uma expressão regular.

(E) é igual ao conjunto de cadeias $\{ x \in \{0, 1\}^* \mid x$ tem quantidade igual de zero (0) e de um (1) $\}$

77) (POSCOMP, 2012)

Sobre gramáticas e linguagens, considere as afirmativas a seguir.

I. Uma gramática na Forma Normal de Chomsky pode ser ambígua.

II. Uma gramática ambígua pode gerar uma linguagem inerentemente não ambígua.

III. Uma gramática na Forma Normal de Greibach pode ser convertida para a Forma Normal de Chomsky.

IV. O algoritmo de conversão de Gramática Livre de Contexto para Gramática na Forma Normal de Chomsky pode ser diretamamente aplicado a uma gramática que não seja λ-livre.

Assinale a alternativa correta.

a) Somente as afirmativas I e II são corretas.
b) Somente as afirmativas I e IV são corretas.
c) Somente as afirmativas III e IV são corretas.
d) Somente as afirmativas I, II e III são corretas.
e) Somente as afirmativas II, III e IV são corretas.

78) (POSCOMP, 2014)

Observe a gramática a seguir.

S → aAbba
aAb → aabbbA | ab
bAb → bbA
bAa → Bbaa
bB → Bb
aB → aA

Sobre essa gramática, assinale a alternativa correta.

a) É irrestrita e aceita a linguagem $\{a^n b^{2n+1} a^n \mid n \geq 1\}$.
b) É irrestrita e aceita a linguagem $\{a^n b^{2n} a^n \mid n \geq 1\}$.
c) É sensível ao contexto e aceita a linguagem $\{a^n b^{2n+1} a^n \mid n \geq 1\}$.
d) É sensível ao contexto e aceita a linguagem $\{a^n b^{2n} a^n \mid n \geq 1\}$.
e) É livre de contexto e aceita a linguagem $\{a^n b^{2n+1} a^n \mid n \geq 1\}$.

79) (POSCOMP, 2016)

A linguagem $L = \{a^n b^m \mid n \leq m + 3\}$, para $n \geq 0$ e $m \geq 0$, é:

A) Regular e gerada pela gramática $S \to aA, A \to baA \mid \varepsilon$.

B) Sensível ao contexto e gerada pela gramática $S \to aSBC, S \to aBC, CB \to BC, aB \to ab, bB \to bb, bC \to bc, cC \to cc$.

C) Recursivamente enumerável e gerada por uma gramática sem restrições nas regras de produção.

D) Estrutura de frase e gerada por uma gramática sem restrições nas regras de produção.

E) Livre de contexto e gerada pela gramática $S \to aaaA, A \to aAb \mid B, B \to Bb \mid \varepsilon$.

80) (ENADE Cienc Comp, 2017)

Considere o seguinte alfabeto:

$\Sigma = \{(,), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -\}$.

Considere, ainda, uma linguagem L definida sobre esse alfabeto.

$L = \{w \mid w \in \Sigma^*$, para cada ocorrência de '(' em w, existe uma ocorrência de ')'$\}$

Por exemplo, a cadeia $x = (2 + (3 - 4))$ pertence a L, mas a cadeia $y = (2 + (3 - 4)$ não pertence a L.

Com relação à linguagem L, avalie as asserções a seguir e a relação proposta entre elas.

I. A linguagem L não pode ser considerada regular.

PORQUE

II. Autômatos finitos não possuem mecanismos que permitam contar infinitamente o número de ocorrências de determinado símbolo em uma cadeia.

A respeito dessas asserções, assinale a opção correta.

A As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.

B As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.

C A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.

D A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.

E As asserções I e II são proposições falsas.

ANEXO A

SUDKAMP, Thomas A. **Languages and machines**: an introduction to the theory of computer science. 2. ed. Massachusets: Addison-Wesley Publishing Company, Inc., 1997. Appendix III.

Backus-Naur Definition of Pascal

The programming language Pascal was developed by Niklaus Wirth in the late 1960s. The language was defined using the notation known as the Backus-Naur form (BNF). The metasymbol $\{u\}$ denotes zero or more repetitions of the string inside the brackets. Thus the BNF rule $A \rightarrow \{u\}$ is represented in a context-free grammar by the rules $A \rightarrow uA \mid \lambda$. The variables in the BNF definition are enclosed in $\langle\ \rangle$. The null string is represented by $\langle empty \rangle$. The BNF rule and its context-free counterpart are given to illustrate the conversion from one notation to the other.

BNF definition:

$$\langle unsigned\ integer \rangle \rightarrow \langle digit \rangle\ \{\langle digit \rangle\}$$

Context-free equivalent:

$$\langle unsigned\ integer \rangle \rightarrow \langle digit \rangle \mid \langle digits \rangle$$

$$\langle digits \rangle \rightarrow \langle digit \rangle\ \langle digits \rangle \mid \lambda$$

The context-free rules can be simplified to

$$\langle unsigned\ integer \rangle \rightarrow \langle digit \rangle\ \langle unsigned\ integer \rangle \mid \langle digit \rangle$$

The start symbol of the BNF definition of Pascal is the variable $\langle program \rangle$. A syntactically correct Pascal program is a string that can be obtained by a derivation initiated with the rule $\langle program \rangle \rightarrow \langle program\ heading \rangle; \langle program\ block \rangle$.

Reprinted from Kathleen Jensen and Niklaus Wirth, *Pascal: User Manual and Report,* 2d ed., Springer Verlag, New York, 1974.

⟨*program*⟩ → ⟨*program heading*⟩ ; ⟨*program block*⟩

⟨*program block*⟩ → ⟨*block*⟩

⟨*program heading*⟩ → **program** ⟨*identifier*⟩ (⟨*file identifier*⟩ {, ⟨*file identifier*⟩})

⟨*file identifier*⟩ → ⟨*identifier*⟩

⟨*identifier*⟩ → ⟨*letter*⟩ {⟨*letter or digit*⟩}

⟨*letter or digit*⟩ → ⟨*letter*⟩ | ⟨*digit*⟩

⟨*letter*⟩ → a | b | . . . | z

⟨*digit*⟩ → 0 | 1 | . . . | 9

⟨*block*⟩ → ⟨*label declaration part*⟩ ⟨*constant definition part*⟩ ⟨*type definition part*⟩
⟨*variable declaration part*⟩ ⟨*procedure and function declaration part*⟩
⟨*statement part*⟩

⟨*label declaration part*⟩ → ⟨*empty*⟩ | **label** ⟨*label*⟩ {, ⟨*label*⟩} ;

⟨*label*⟩ → ⟨*unsigned integer*⟩

⟨*constant definition part*⟩ → ⟨*empty*⟩ |
const ⟨*constant definition*⟩ {; ⟨*constant definition*⟩} ;

⟨*constant definition*⟩ → ⟨*identifier*⟩ = ⟨*constant*⟩

⟨*constant*⟩ → ⟨*unsigned number*⟩ | ⟨*sign*⟩ ⟨*unsigned number*⟩ | ⟨*constant identifier*⟩ |
⟨*sign*⟩ ⟨*constant identifier*⟩ | ⟨*string*⟩

⟨*unsigned number*⟩ → ⟨*unsigned integer*⟩ | ⟨*unsigned real*⟩

⟨*unsigned integer*⟩ → ⟨*digit*⟩ {⟨*digit*⟩}

⟨*unsigned real*⟩ → ⟨*unsigned integer*⟩ . ⟨*digit*⟩ {⟨*digit*⟩} |
⟨*unsigned integer*⟩ . ⟨*digit*⟩ {⟨*digit*⟩} E ⟨*scale factor*⟩ |
⟨*unsigned integer*⟩ E ⟨*scale factor*⟩

⟨*scale factor*⟩ → ⟨*unsigned integer*⟩ | ⟨*sign*⟩ ⟨*unsigned integer*⟩

⟨*sign*⟩ → + | −

⟨*constant identifier*⟩ → ⟨*identifier*⟩

⟨*string*⟩ → '⟨*character*⟩ {⟨*character*⟩}'

⟨*type definition part*⟩ → ⟨*empty*⟩ | **type** ⟨*type definition*⟩ {; ⟨*type definition*⟩} ;

⟨*type definition*⟩ → ⟨*identifier*⟩ = ⟨*type*⟩

⟨*type*⟩ → ⟨*simple type*⟩ | ⟨*structured type*⟩ | ⟨*pointer type*⟩

⟨*simple type*⟩ → ⟨*scalar type*⟩ | ⟨*subrange type*⟩ | ⟨*type identifier*⟩

⟨*scalar type*⟩ → ⟨*identifier*⟩ {, ⟨*identifier*⟩}

⟨*subrange type*⟩ → ⟨*constant*⟩ . . ⟨*constant*⟩

⟨*type identifier*⟩ → ⟨*identifier*⟩

⟨*structured type*⟩ → ⟨*unpacked structured type*⟩ | **packed** ⟨*unpacked structured type*⟩

⟨*unpacked structured type*⟩ → ⟨*array type*⟩ | ⟨*record type*⟩ | ⟨*set type*⟩ | ⟨*file type*⟩

⟨*array type*⟩ → **array** [⟨*index type*⟩ {, ⟨*index type*⟩}] **of** ⟨*component type*⟩

⟨*index type*⟩ → ⟨*simple type*⟩

⟨*component type*⟩ → ⟨*type*⟩

⟨*record type*⟩ → **record** ⟨*field list*⟩ **end**

⟨*field list*⟩ → ⟨*fixed part*⟩ | ⟨*fixed part*⟩ ; ⟨*variant part*⟩ | ⟨*variant part*⟩

⟨*fixed part*⟩ → ⟨*record section*⟩ {; ⟨*record section*⟩}

⟨*record section*⟩ → ⟨*field identifier*⟩ {, ⟨*field identifier*⟩} : ⟨*type*⟩ | ⟨*empty*⟩

⟨*variant part*⟩ → **case** ⟨*tag field*⟩ ⟨*type identifier*⟩ **of** ⟨*variant*⟩ {; ⟨*variant*⟩}

⟨*tag field*⟩ → ⟨*field identifier*⟩ : | ⟨*empty*⟩

⟨*variant*⟩ → ⟨*case label list*⟩ : (⟨*field list*⟩) | ⟨*empty*⟩

⟨*case label list*⟩ → ⟨*case label*⟩ {, ⟨*case label*⟩}

⟨*case label*⟩ → ⟨*constant*⟩

⟨*set type*⟩ → **set of** ⟨*base type*⟩

⟨*base type*⟩ → ⟨*simple type*⟩

⟨*file type*⟩ → **file of** ⟨*type*⟩

⟨*pointer type*⟩ → ↑ ⟨*type identifier*⟩

⟨*variable declaration part*⟩ → ⟨*empty*⟩ |
var ⟨*variable declaration*⟩ {; ⟨*variable declaration*⟩} ;

⟨*variable declaration*⟩ → ⟨*identifier*⟩ {, ⟨*identifier*⟩} : ⟨*type*⟩

⟨*procedure and function declaration part*⟩ → {⟨*procedure or function declaration*⟩ ;}

⟨*procedure or function declaration*⟩ → ⟨*procedure declaration*⟩ |
⟨*function declaration*⟩

⟨*procedure declaration*⟩ → ⟨*procedure heading*⟩ ⟨*block*⟩

⟨*procedure heading*⟩ → **procedure** ⟨*identifier*⟩ ; |
procedure ⟨*identifier*⟩ (⟨*formal parameter section*⟩
{; ⟨*formal parameter section*⟩}) ;

⟨*formal parameter section*⟩ → ⟨*parameter group*⟩ | **var** ⟨*parameter group*⟩ |
function ⟨*parameter group*⟩ |
procedure ⟨*identifier*⟩ {, ⟨*identifier*⟩}

⟨*parameter group*⟩ → ⟨*identifier*⟩ {, ⟨*identifier*⟩} : ⟨*type identifier*⟩

⟨*function declaration*⟩ → ⟨*function heading*⟩ ⟨*block*⟩

⟨*function heading*⟩ → **function** ⟨*identifier*⟩ : ⟨*result type*⟩ ; |
function ⟨*identifier*⟩ (⟨*formal parameter section*⟩
{; ⟨*formal parameter section*⟩}) : ⟨*result type*⟩ ;

⟨*result type*⟩ → ⟨*type identifier*⟩

⟨*statement part*⟩ → ⟨*compound statement*⟩

⟨*statement*⟩ → ⟨*unlabeled statement*⟩ | ⟨*label*⟩ : ⟨*unlabeled statement*⟩

⟨*unlabeled statement*⟩ → ⟨*simple statement*⟩ | ⟨*structured statement*⟩

⟨*simple statement*⟩ → ⟨*assignment statement*⟩ | ⟨*procedure statement*⟩ |
⟨*go to statement*⟩ | ⟨*empty statement*⟩

⟨*assignment statement*⟩ → ⟨*variable*⟩ := ⟨*expression*⟩ |
⟨*function identifier*⟩ := ⟨*expression*⟩

⟨*variable*⟩ → ⟨*entire variable*⟩ | ⟨*component variable*⟩ | ⟨*referenced variable*⟩

⟨*entire variable*⟩ → ⟨*variable identifier*⟩

⟨*variable identifier*⟩ → ⟨*identifier*⟩

⟨*component variable*⟩ → ⟨*indexed variable*⟩ | ⟨*field designator*⟩ | ⟨*file buffer*⟩

⟨*indexed variable*⟩ → ⟨*array variable*⟩ [⟨*expression*⟩ {, ⟨*expression*⟩}]

⟨*array variable*⟩ → ⟨*variable*⟩

⟨*field designator*⟩ → ⟨*record variable*⟩ . ⟨*field identifier*⟩

⟨*record variable*⟩ → ⟨*variable*⟩

⟨*field identifier*⟩ → ⟨*identifier*⟩

⟨*file buffer*⟩ → ⟨*file variable*⟩ ↑

⟨*file variable*⟩ → ⟨*variable*⟩

⟨*referenced variable*⟩ → ⟨*pointer variable*⟩ ↑

⟨*pointer variable*⟩ → ⟨*variable*⟩

⟨*expression*⟩ → ⟨*simple expression*⟩ | ⟨*simple expression*⟩ ⟨*relational operator*⟩
⟨*simple expression*⟩

⟨*relational operator*⟩ → = | <> | < | <= | >= | > | **in**

⟨*simple expression*⟩ → ⟨*term*⟩ | ⟨*sign*⟩ ⟨*term*⟩ |
⟨*simple expression*⟩ ⟨*adding operator*⟩ ⟨*term*⟩

⟨*adding operator*⟩ → + | − | **or**

⟨*term*⟩ → ⟨*factor*⟩ | ⟨*term*⟩ ⟨*multiplying operator*⟩ ⟨*factor*⟩

⟨*multiplying operator*⟩ → * | / | **div** | **mod** | **and**

⟨*factor*⟩ → ⟨*variable*⟩ | ⟨*unsigned constant*⟩ | (⟨*expression*⟩) | ⟨*function designator*⟩ |
⟨*set*⟩ | **not** ⟨*factor*⟩

⟨*unsigned constant*⟩ → ⟨*unsigned number*⟩ | ⟨*string*⟩ | ⟨*constant identifier*⟩ | **nil**

⟨*function designator*⟩ → ⟨*function identifier*⟩ |
⟨*function identifier*⟩ (⟨*actual parameter*⟩ {, ⟨*actual parameter*⟩})

⟨*function identifier*⟩ → ⟨*identifier*⟩

⟨*set*⟩ → [⟨*element list*⟩]

⟨*element list*⟩ → ⟨*element*⟩ {, ⟨*element*⟩} | ⟨*empty*⟩

⟨*element*⟩ → ⟨*expression*⟩ | ⟨*expression*⟩ . . ⟨*expression*⟩

⟨*procedure statement*⟩ → ⟨*procedure identifier*⟩ |
⟨*procedure identifier*⟩ (⟨*actual parameter*⟩
{, ⟨*actual parameter*⟩})

⟨*procedure identifier*⟩ → ⟨*identifier*⟩

⟨*actual parameter*⟩ → ⟨*expression*⟩ | ⟨*variable*⟩ |
⟨*procedure identifier*⟩ | ⟨*functional identifier*⟩

⟨*go to statement*⟩ → **goto** ⟨*label*⟩

⟨*empty statement*⟩ → ⟨*empty*⟩

⟨*empty*⟩ → λ

⟨*structured statement*⟩ → ⟨*compound statement*⟩ | ⟨*conditional statement*⟩ |
⟨*repetitive statement*⟩ | ⟨*with statement*⟩

⟨*compound statement*⟩ → **begin** ⟨*statement*⟩ {; ⟨*statement*⟩} **end**

⟨*conditional statement*⟩ → ⟨*if statement*⟩ | ⟨*case statement*⟩

⟨*if statement*⟩ → **if** ⟨*expression*⟩ **then** ⟨*statement*⟩ |
if ⟨*expression*⟩ **then** ⟨*statement*⟩ **else** ⟨*statement*⟩

⟨*case statement*⟩ → **case** ⟨*expression*⟩ **of** ⟨*case list element*⟩ {; ⟨*case list element*⟩} **end**

⟨*case list element*⟩ → ⟨*case label list*⟩ : ⟨*statement*⟩ | ⟨*empty*⟩

⟨*case label list*⟩ → ⟨*case label*⟩ {, ⟨*case label*⟩}

⟨*repetitive statement*⟩ → ⟨*while statement*⟩ | ⟨*repeat statement*⟩ | ⟨*for statement*⟩

⟨*while statement*⟩ → **while** ⟨*expression*⟩ **do** ⟨*statement*⟩

⟨*repeat statement*⟩ → **repeat** ⟨*statement*⟩ {; ⟨*statement*⟩} **until** ⟨*expression*⟩

⟨*for statement*⟩ → **for** ⟨*control variable*⟩ := ⟨*for list*⟩ **do** ⟨*statement*⟩

⟨*for list*⟩ → ⟨*initial value*⟩ **to** ⟨*final value*⟩ | ⟨*initial value*⟩ **downto** ⟨*final value*⟩

⟨*control variable*⟩ → ⟨*identifier*⟩

⟨*initial value*⟩ → ⟨*expression*⟩

⟨*final value*⟩ → ⟨*expression*⟩

⟨*with statement*⟩ → **with** ⟨*record variable list*⟩ **do** ⟨*statement*⟩

⟨*record variable list*⟩ → ⟨*record variable*⟩ {, ⟨*record variable*⟩}

⟨character⟩ → ⟨letter⟩ | ⟨digit⟩ | ! | @ | # | ···

ANEXO B

TITTEL, Ed; GAITHER, Mark. **60 minutos para aprender Java**. São Paulo:Berkeley, 1996. Apêndice B.

Apêndice B: Gramática da Linguagem Java

Esta é uma breve gramática para uma unidade de compilação do Java. Como os programas do Java são formados por uma ou mais unidades de compilação, você pode generalizar desde o nível de compilação até o nível de programa, tanto com programas auxiliares quanto com aplicativos.

A gramática possui símbolos terminais não-definidos de DocComment, Identifier, String e Character. Os textos entre aspas indicam terminais literais (por exemplo, '(' indica a presença obrigatória de um parêntese esquerdo.)

Cada regra tem o seguinte formato:

```
não-terminal = meta-expressão;
```

Para esclarecimento, cada regra termina com um ponto-e-vírgula em sua própria linha e é separada da regra seguinte por uma linha em branco.

Outras metanotações incluem:

- :| para alternação (either-or, one of many etc.)
- (...) para agrupamentos.
- Sufixo ? para 0 ou 1 ocorrências obrigatórias.
- Sufixo + para 1 ou mais ocorrências obrigatórias.
- Sufixo * para 0 ou mais ocorrências obrigatórias.

Trata-se de uma gramática no estilo BNF, mas que não segue explicitamente as regras de produção de Wirth. Para análise automática, alguns metasímbolos e algumas pontuações precisam ser reformatados (BNF significa Backus-Naur Form, uma notação específica para linguagens de computadores desenvolvida nos anos 70 e geralmente utilizada para descrever as linguagens de maneira formal e completa; se você já aprendeu Pascal ou Modula 2, é muito provável que já tenha visto uma gramática BNF).

```
CompilationUnit =
  PackageStatement? ImportStatement* TypeDeclaration*
;
PackageStatement =
  'package' PackageName ';'
;
ImportStatement =
  'import' PackageName '.' '*' ';'
  | 'import' ( ClassName | InterfaceName )';'
;
TypeDeclaration =
  DocComment? ClassDeclaration
  | DocComment? InterfaceDeclaration
  | ';'
;
ClassDeclaration =
  Modifier* 'class' Identifier
  ('extends' ClassName)?
  ('implements' InterfaceName (',' InterfaceName)*)?
  '{' FieldDeclaration* '}'
;
InterfaceDeclaration =
  Modifier* 'interface' Identifier
  ('extends' InterfaceName (',' InterfaceName)*)?
  '{' FieldDeclaration* '}'
;
FieldDeclaration =
  DocComment? MethodDeclaration
  | DocComment? ConstructorDeclaration
  | DocComment? VariableDeclaration
  | StaticInitializer
  | ';'
;
MethodDeclaration =
  Modifier* Type Identifier '(' Parameterlist? ')' ( '[' ']' )*
  ( '{' Statement* '}' | ';' )
;
ConstructorDeclaration =
  Modifier* Identifier '(' Parameterlist? ')'
  '{' Statement* '}'
;
VariableDeclaration =
  Modifier* Type VariableDeclarator (',' VariableDeclarator)* ';'
;
```

```
VariableDeclarator =
  Identifier ('[' ']')* ('=' VariableInitializer)? .
;
VariableInitializer =
  Expression
| '{'   ( VariableInitializer  ( ',' VariableInitializer )* ','? )? '}'
;
StaticInitializer =
  'static' '{' Statement* '}'
;
ParameterList =
  Parameter (',' Parameter)*
;
Parameter =
  Type Identifier ('[' ']')*
;
Statement =
  VariableDeclaration
| Expression ';'
| '{' Statement* '}'
| 'if' '(' Expression ')' Statement ('else' Statement)?
| 'while' '(' Expression ')' Statement
| 'do' Statement 'while' '(' Expression ')' ';'
| 'for' '(' ( VariableDeclaration |Expression ';' | ';')
      Expression ? ';' Expression ?')' Statement
| 'try' Statement ('catch'  '(' Parameter ')' Statement)*
    ('finally' Statement)?
| 'switch' '(' Expression ')' '{'
    ( 'case' Expression ':' | 'default' ':' | Statement )*
  '}'
| 'synchronized'  ''' Expression ')' Statement
| 'return' Expression? ';'
| 'throw' Expression ';'
| Identifier ':' Statement
| 'break' Identifier? ';'
| 'continue' Identifier? ';'
| ';'
;
Expression =
  Expression '+' Expression
| Expression '-' Expression
| Expression '*' Expression
| Expression '/' Expression
| Expression '%' Expression
| Expression '^' Expression
| Expression '&' Expression
| Expression '|' Expression
| Expression '&&' Expression
| Expression '||' Expression
| Expression '<<' Expression
| Expression '>>' Expression
| Expression '>>>' Expression
| Expression '=' Expression
| Expression '+='  Expression
| Expression '-=' Expression
| Expression '*=' Expression
| Expression '/=' Expression
| Expression '%=' Expression
| Expression '^=' Expression
| Expression '&=' Expression
| Expression '|=' Expression
| Expression '<<=' Expression
| Expression '>>=' Expression
| Expression '>>>=' Expression
| Expression '<' Expression
| Expression '>' Expression
| Expression '<=' Expression
| Expression '>=' Expression
| Expression '==' Expression
| Expression '!=' Expression
| Expression '.' Expression
| Expression ',' Expression
| Expression 'instanceof' ( ClassName | InterfaceName )
| Expression '?' Expression ':' Expression
| Expression '[' Expression ']'
| '++' Expression
| '--' Expression
| Expression '++'
| Expression '--'
| '-' Expression
| '!' Expression
| '~' Expression
| '(' Expression  ')'
| '(' Type ')' Expression
| Expression  '(' ArgList? ')'
| 'new' ClassName  '(' ArgList?')'
| 'new' TypeSpecifier  ( '[' Expression ']' )+ ('['  ']')*
| 'new' '(' Expression  ')'
| 'true'
| 'false'
| 'null'
| 'super'
| 'this'
| Identifier
| IntegerLiteral
| FloatLiteral
| String
| Character
;
```

```
ArgList =
  Expression (',' Expression )*
;
Type  =
  TypeSpecifier ('[' ']')*
;
TypeSpecifier =
  'boolean'
| 'byte'
| 'char'
| 'short'
| 'int'
| 'float'
| 'long'
| 'double'
| ClassName
| InterfaceName
;
Modifier =
  'public'
| 'private'
| 'protected'
| 'static'
| 'final'
| 'native'
| 'synchronized'
| 'abstract'
| 'threadsafe'
| 'transient'
;
PackageName =
  Identifier
|  PackageName '.' Identifier
;
ClassName =
  Identifier
|  PackageName '.' Identifier
;
InterfaceName =
  Identifier
|  PackageName '.' Identifier
;
IntegerLiteral =
  ('1'|...|'9')('0'|...|'9')*('L'|'l')?
| '0'('1'|...|'7')*('L'|'l')?
| '0'('x'|'X')('0'|...|'9'|'a'|...|'f'|'A'|...|'F')+('L'|'l')?
```

```
FloatLiteral =
  DecimalDigits '.' DecimalDigits? ExponentPart? FloatTypeSuffix?
| '.'DecimalDigits ExponentPart? FloatTypeSuffix?
| DecimalDigits ExponentPart FloatTypeSuffix?
;
DecimalDigits =
  ('0'|...|'9')+
;
ExponentPart =
  ( 'E' | 'e' ) ( '+' | '-' )? DecimalDigits
;
FloatTypeSuffix =
  'F' | 'f' | 'D' | 'd'
;
```

Gabarito das questões objetivas do ENADE e POSCOMP

Nº	Alternativa correta
5	A
6	B
7	D
8	D (contra exemplo = 'b')
9	E
10	C
11	A
12	D
13	D
14	D
15	C
16	A
17	B
17	C
21	D
22	B
25	A
29	E
32	E
33	B (o correto seria VFVVF)
34	C
38	D
39	B
40	A
43	A
44	E
45	D
47	D
49	D

56 B

62 D

67 D

68 E

69 C

72 E

73 A

74 E

76 C

77 D

78 B

79 anulada

80 A