ALGORITMO do predict:

$$\texttt{predict}(A \to \alpha) = \begin{cases} \texttt{first}(\alpha) & \text{se } \varepsilon \notin \texttt{first}(\alpha) \\ (\texttt{first}(\alpha) - \{\varepsilon\}) \cup \texttt{follow}(A) & \text{se } \varepsilon \in \texttt{first}(\alpha) \end{cases}$$

ALGORITMO do first:

```
first(α) {
    if (α = ε) then
        return (ε)
    h = head (α)          # com |h| = 1
    ω = tail (α)          # tal que α = hω
    if (h ∈ T) then
        return (h)
    else
        return ⋃_{(h→β,ω)∈P} first(β, ω)
}
```

ALGORITMO do follow:

1. $\$ \in \texttt{follow}(S)$
2. if $(A \to \alpha B \in P)$ then
   $\texttt{follow}(B) \supseteq \texttt{follow}(A)$
3. if $(A \to \alpha B \beta \in P) \wedge (\varepsilon \notin \texttt{first}(\beta))$ then
   $\texttt{follow}(B) \supseteq \texttt{first}(\beta)$
4. if $(A \to \alpha B \beta \in P) \wedge (\varepsilon \in \texttt{first}(\beta))$ then
   $\texttt{follow}(B) \supseteq ((\texttt{first}(\beta) - \{\varepsilon\}) \cup \texttt{follow}(A))$

1. Sobre o alfabeto $T_1 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\}$ considere a gramática G_1 dada a seguir e seja L_1 a linguagem por ela descrita.

   S → A B S | A
   A → ε | a A
   B → b C f | C e
   C → A | c C d D
   D → D d | e | d E
   E → e E | f E

   (a) Faça a derivação à esquerda da palavra abcdef. Tem de apresentar todos os passos (derivações diretas).

   (b) Mostre que $\{\mathtt{b}, \mathtt{c}\} \subset \texttt{predict}(S \to A\ B\ S)$. Apresente os passos intermédios e/ou o raciocínio adequados para suportar a sua resposta.

   (c) Determine o conjunto follow(D). Apresente os passos intermédios e/ou o raciocínio adequados para suportar a sua resposta. A simples apresentação da resposta final **não** será cotada.

   (d) Um símbolo não terminal é útil se for simultaneamente acessível e produtivo; é inútil caso contrário. Analisando e apresentando a acessibilidade e a produtividade dos símbolos não terminais D e E, mostre que, na gramatica G_1, D é útil e E é inútil. Apresente os passos intermédios e/ou o raciocínio adequados para suportar a sua resposta.

   (e) A gramática G_1 é inadequada à implementação de um reconhecedor descendente com *lookahead* de 1. Diga porquê e altere-a de forma a obter uma equivalente que o permita. Basta transcrever as partes alteradas.

Sobre o alfabeto $T_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}\}$ considere a linguagem L_2 tal que:

$$L_2 = \{\ (\mathtt{ab})^n (\mathtt{c}|\mathtt{d})^k (\mathtt{e})^{n+k}\ :\ n > 0 \wedge k \geq 0\ \}$$

(a) Apresente 3 palavras pertencentes a L_2 com o menor número de letras possível.

(b) Construa uma gramática livre de contexto que represente a linguagem L_2.

continua na página seguinte

3. Sobre o alfabeto $T_3 = \{$ RDR, LINE, DASHED, DOTTED, $\}$, considere a gramática G_3 dada a seguir, assim como os conjuntos `predict` das suas produções.

P1	*draw* → *seq*	
P2	*seq* → *line seq*	
P3	| ε	
P4	*line* → *type* LINE *pt pt xpts*	
P5	*type* → DASHED	
P6	| DOTTED	
P7	| ε	
P8	*pt* → RDR RDR	
P9	*xpts* → *pt xpts*	
P10	| ε	

Produção	predict
draw → *seq*	{ DASHED, DOTTED, LINE, $ }
seq → *line seq*	{ DASHED, DOTTED, LINE }
seq → ε	{ $ }
line → *type* LINE *pt pt xpts*	{ DASHED, DOTTED, LINE }
type → DASHED	{ DASHED }
type → DOTTED	{ DOTTED }
type → ε	{ LINE }
pt → RDR RDR	{ RDR }
xpts → *pt xpts*	{ RDR }
xpts → ε	{ DASHED, DOTTED, LINE, $ }

(a) Preencha a tabela de decisão para um reconhecedor preditivo descendente (top-down) com lookahead de 1 da gramática G_3. Na sua resposta, pode usar as referências das produções (P1 a P10).

	RDR	LINE	DASHED	DOTTED	(	)	$
draw							
seq							
line							
type							
pt							
xpts							

b) A construção de um reconhecedor (*parser*) **ascendente** para uma gramática livre de contexto baseia-se num conjunto de estados, sendo cada estado caraterizado por um conjunto de itens. O elemento (estado) inicial desse conjunto para a gramática G_3 está parcialmente descrito a seguir.

$$Z_0 = \{\ \text{draw} \rightarrow \bullet\ \text{seq}\ \$\ \} \cup \cdots$$

Complete o elemento (estado) Z_0 e determine os elementos (estados) diretamente alcançáveis a partir de Z_0. Pode responder em baixo.

3. Sobre o alfabeto $T_3 = \{$ NUM, LINE, DASHED, DOTTED, $\}$, considere a gramática G_3 dada a seguir, assim como os conjuntos predict das suas produções.

draw → *seq*	
seq → *line seq*	
| ε	
line → *type* LINE *pt pt xpts*	
type → DASHED	
| DOTTED	
| ε	
pt → NUM NUM	
xpts → *pt xpts*	
| ε	

Produção	predict
draw → *seq*	{ DASHED , DOTTED , LINE , $ }
seq → *line seq*	{ DASHED , DOTTED , LINE }
seq → ε	{ $ }
line → *type* LINE *pt pt xpts*	{ DASHED , DOTTED , LINE }
type → DASHED	{ DASHED }
type → DOTTED	{ DOTTED }
type → ε	{ LINE }
pt → NUM NUM	{ NUM }
xpts → *pt xpts*	{ NUM }
xpts → ε	{ DASHED , DOTTED , LINE , $ }

(a) Preencha a tabela de decisão para um reconhecedor preditivo descendente (*top-down*) com *lookahead* de 1 da gramática G_3. Na sua resposta, pode usar as referências das produções (P1 a P10).

	NUM	LINE	DASHED	DOTTED	(	)	$
draw	—	P1, P2	P1, P2	P1, P2	P1, P2	P1, P3	P1, P3
seq	—	P2, P4	P2, P4	P2, P4	P2, P4	P3	P3
line	P8, P9	P4, P7, P11	P4, P5, P10	P4, P6, P11	P4	P10	P10
type	—	P7	P5	P6	—	—	—
pt	P8	—	—	—	—	—	—
xpts	P9	P10	P10	P10	P9	P10	P10

(b) A construção de um reconhecedor (*parser*) **ascendente** para uma gramática livre de contexto base num conjunto de estados, sendo cada estado caraterizado por um conjunto de itens. O elemento (est inicial desse conjunto para a gramática G_3 está parcialmente descrito a seguir.

$$Z_0 = \{\ draw \rightarrow \bullet\ seq\ \$\ \} \cup \cdots$$

Complete o elemento (estado) Z_0 e determine os elementos (estados) diretamente alcançáveis a de Z_0. Pode responder em baixo.

4. Considere a gramática G_3 dada no exercício anterior. Uma palavra na linguagem dada por G_3 descreve um desenho definido por um conjunto de linhas poligonais, onde:

- O símbolo terminal `NUM` tem um atributo (intrínseco) associado, designado `v`, que representa um número.
- O símbolo não terminal *pt* representa as coordenadas X e Y de um ponto.
- O símbolo não terminal *type* é um modificador que define se a linha poligonal é desenhada a tracejado ou ponteado. Se não estiver presente a linha é desenhada a contínuo (sólido).
- Finalmente, dispõe-se da função `drawLine(type, x1, y1, x2, y2)` que desenha um segmento de reta entre os pontos `(x1,y1)` e `(x2,y2)`, a sólido, tracejado ou ponteado, dependendo do valor do argumento `type` (*solid*, *dashed*, *dotted*).

1,5] (a) Trace a árvore de derivação da palavra

`LINE NUM NUM NUM NUM DOTTED LINE NUM NUM NUM NUM NUM NUM NUM NUM`

Se quiser, ao traçar a árvore, pode abreviar a designação dos símbolos não terminais, usando *s*, *t*, *l*, *p* e *x* em vez de *seq*, *type*, *line*, *pt* e *xpts*.

,5] (b) Sem alterar a gramática e sem introduzir variáveis globais, complete a gramática de atributos abaixo tal que ela adequadamente invoque a função `drawLine`, nos sítios onde está, de modo a desenhar as linhas poligonais. Note que uma linha poligonal com n pontos representa $n-1$ segmentos de reta.

Produção	Regra semântica
$draw \rightarrow seq$	
$seq_0 \rightarrow line\ seq_1$	
$seq \rightarrow \varepsilon$	
$line \rightarrow type$ `LINE` $pt_1\ pt_2\ xpts$	drawLine(
$type \rightarrow$ `DASHED`	
$type \rightarrow$ `DOTTED`	
$type \rightarrow \varepsilon$	
$pt \rightarrow$ `NUM`$_1$ `NUM`$_2$	
$xpts \rightarrow \varepsilon$	
$xpts \rightarrow pt\ xpts$	drawLine(