Universidade de Aveiro

Departamento de Electrónica, Telecomunicações e Informática

Linguagens Formais e Autómatos + Compiladores

Exame teórico-prático, parte 2 (Ano Letivo de 2018/2019) modelo

NºMec: **Nome:**

1. Sobre o alfabeto $T_1 = \{\texttt{t b z w a o v n}\}$ considere a gramática G_1 dada a seguir e seja L_1 a linguagem por ela descrita.

$$
\begin{array}{lcl}
P & \rightarrow & \varepsilon \mid X\ I\ \texttt{t}\ P \mid X\ \texttt{b}\ P\ \texttt{z}\ P \\
X & \rightarrow & \varepsilon \mid \texttt{w}\ C \\
I & \rightarrow & \varepsilon \mid \texttt{a} \\
C & \rightarrow & T \mid C\ \texttt{o}\ T \\
T & \rightarrow & \texttt{v} \mid \texttt{n}\ T
\end{array}
$$

[1,5] (a) Mostre que a palavra $\texttt{a t w n v b z}$ pertence a L_1.

Optei por fazer uma derivação à esquerda.

P => X I t P => I t P => a t P => a t X b P z P => a t w C b P z P
=> a t w T b P z P => a t w n T b P z P => a t w n v b P z P
=> a t w n v b z P => a t w n v b z

Mas poderia ser com outro tipo de derivação ou com uma árvore de derivação.
A menos que seja pedido de uma forma específica.

[1,5] (b) Considere o conjunto $F = \texttt{first}(X\ I\ \texttt{t}\ P)$.
Das seguintes afirmações, assinale as que são verdadeiras.

[x] $\texttt{t} \in F$	[ ] $\texttt{b} \in F$
[x] $\texttt{w} \in F$	[x] $\texttt{a} \in F$

first(X I t P) = ?
Aplicando o algoritmo: h = X ; w = I t P
Como h é não terminal e há 2 produções começadas por X
first(X I t P) = first(I t P) U first(w C I t P)
first(I t P) = ?
Aplicando o algoritmo: h = I ; w = t P
Como h é não terminal e há 2 produções começadas por I
first(I t P) = first(t P) U first(a t P)
O first de uma expressão começada por um terminal é o conjunto singular com esse terminal. Logo
first(w C I t P) = {w}
first(t P) = {t}
first(a t P) = {a}
Donde:
first(I t P) = {a t}
first(X I t P) = {a t w}

[1,5] (c) Considere o conjunto $G = \texttt{follow}(T)$.
Das seguintes afirmações, assinale as que são verdadeiras.

[x] $\texttt{t} \in G$	[x] $\texttt{b} \in G$
[ ] $\texttt{w} \in G$	[x] $\texttt{a} \in G$

Por definição $ \in follow(S)
follow(T) = ?
- pela produção C -> T, follow(T) contém o follow(C)
follow(C) = ?
- pela produção X -> w C, follow(C) contém o follow(X)
- pela produção C -> C o T, follow(C) contém first(o T) = {o}
follow(X) = ?
- pela produção S -> X I t P, follow(X) contem first(I t P) = {a t}
- pela produção S -> X b P z P, follow(X) contém first(b P z P) = {b}
Logo:
follow(X) = {a b t}, follow(C) = {a b t o}, follow(T) = {a b t o}

[2,0] (d) Considere o conjunto $H = \texttt{predict}(P \rightarrow X\ I\ \texttt{t}\ P)$.
Das seguintes afirmações, assinale as que são verdadeiras.

[x] $\texttt{t} \in H$	[ ] $\texttt{b} \in H$
[x] $\texttt{w} \in H$	[x] $\texttt{a} \in H$

first(X I t P) = {a t w} (ver 1b)
Como \e \not\in first(X I t P)
predict(P -> X I t P) = {a t w}

[2,0] (e) As produções começadas por P e C tornam a gramática G_1 inadequada à implementação de um reconhecedor descendente com *lookahead* de 1. Altere-a de forma a obter uma equivalente que o permita.

não pode ter prefixos comuns
é preciso resolver
P -> X I t P | X b P z P

não pode ter recursividade à esquerda
é preciso resolver
C -> T | C o T

Removendo os prefixos comuns
P -> \e | X Y
Y -> I t P | b P z P

Analisando a recursividade à esquerda
C => C o T => C o T o T => ... => C (o T)* => T (o T)*

Permite eliminá-la
C -> T Z
Z -> \e | o T Z

2. Considere o alfabeto $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ e seja L_2 o conjunto de todas as expressões regulares definíveis sobre o alfabeto A. L_2 é uma linguagem independente do contexto definida sobre o alfabeto $T_2 = A \cup \{ \mathtt{(}, \mathtt{)}, \mathtt{*}, \mathtt{|} \}$, em que `*` é o operador de fecho, `|` o operador de escolha e em que o operador de concatenação é implícito. Em termos de precedência, e da mais alta para a mais baixa, estão as operações de fecho, concatenação e escolha. Os parêntesis podem ser usados para alterar a precedência por defeito.

[3,0] (.) Construa uma gramática independente do contexto que represente a linguagem L_2.

De forma a evitar confusões entre o operador | da linguagem e o usado na gramática, usa-e + no lugar do primeiro
Exemplos de palavras aceites pela linguagem:
a*
a+c
ac
(ac)*
((ac|b)*|aa)**

Definição da precedência: uma expressão é uma soma de T, um T é uma concatenação de F, um F é K fechado 0 ou mais vezes, e K é uma letra ou uma expresão entre parêntesis
E = T + T + T ...
T = F F F ...
F = K * * ...
K = N | (E)

Gramática, onde se usa o + em vez do |:

E -> T | E + T
T -> F | T F
F -> K | F *
K -> N | (E)
N -> a | b | c

3. Sobre o alfabeto $T_3 = \{\texttt{NUM}, \texttt{BOX}, \texttt{CIRCLE}, \texttt{THICKNESS}, \texttt{COLOR}, \texttt{'\{'}, \texttt{'\}'}\}$, considere a gramática G_3 dada a seguir e seja L_3 a linguagem por ela descrita.

$$
\begin{array}{rcl}
\textit{draw} & \rightarrow & \textit{seq} \\
\textit{seq} & \rightarrow & \varepsilon \\
 & | & \textit{seq}\ \ \textit{item} \\
\textit{item} & \rightarrow & \texttt{COLOR}\ \ \texttt{NUM} \\
 & | & \texttt{THICKNESS}\ \ \texttt{NUM} \\
 & | & \texttt{CIRCLE}\ \ \textit{point}\ \ \texttt{NUM} \\
 & | & \texttt{BOX}\ \ \textit{point}\ \ \texttt{'\{'}\ \ \ \textit{seq}\ \ \texttt{'\}'} \\
\textit{point} & \rightarrow & \texttt{NUM}\ \ \texttt{NUM}
\end{array}
$$

Considere ainda a coleção de conjunto de itens usada na contrução de um reconhecedor ascendente parcialmente apresentada a seguir, onde $\delta(Z_i, a)$ representa a função de transição de estado.

$Z_0 = \{\textit{draw} \rightarrow \bullet\, \textit{seq}\ ,\ \textit{seq} \rightarrow \bullet\ ,\ \textit{seq} \rightarrow \bullet\, \textit{seq}\ \ \textit{item}\}$

$Z_1 = \delta(Z_0, \textit{seq}) = \{\textit{draw} \rightarrow \textit{seq}\, \bullet\ ,\ \textit{seq} \rightarrow \textit{seq}\, \bullet\, \textit{item}\ ,\ \textit{item} \rightarrow \bullet\, \texttt{COLOR}\, \texttt{NUM}\ ,\ \textit{item} \rightarrow \bullet\, \texttt{THICKNESS}\, \texttt{NUM}\ ,$
$\quad \textit{item} \rightarrow \bullet\, \texttt{CIRCLE}\, \textit{point}\, \texttt{NUM}\ ,\ \textit{item} \rightarrow \bullet\, \texttt{BOX}\, \textit{point}\ \texttt{'\{'}\ \textit{seq}\ \texttt{'\}'}\}$

$Z_2 = \delta(Z_1, \textit{item}) = \{\textit{seq} \rightarrow \textit{seq}\, \textit{item}\, \bullet\}$

$Z_3 = \delta(Z_1, \texttt{COLOR}) = \{\textit{item} \rightarrow \texttt{COLOR}\, \bullet\, \texttt{NUM}\}$

$Z_4 = \delta(Z_1, \texttt{THICKNESS}) = \{\textit{item} \rightarrow \texttt{THICKNESS}\, \bullet\, \texttt{NUM}\}$

$Z_5 = \delta(Z_1, \texttt{CIRCLE}) = \{\cdots\}$

$Z_6 = \delta(Z_1, \texttt{BOX}) = \{\cdots\}$

$Z_7 = \delta(Z_3, \texttt{NUM}) = \{\textit{item} \rightarrow \texttt{COLOR}\, \texttt{NUM}\, \bullet\}$

$Z_8 = \delta(Z_4, \texttt{NUM}) = \{\textit{item} \rightarrow \texttt{THICKNESS}\, \texttt{NUM}\, \bullet\}$ follow(seq) = { $ CO TH CI BO '}' }

[2,0] (a) Preencha as linhas da tabela de reconhecimento (*parsing*) para um reconhecedor ascendente relativamente aos estados Z_0 a Z_4.

ACTION GOTO

	NUM	BOX	CIRCLE	THICKNESS	COLOR	{	}	$	*seq*	*item*	*point*
Z_0		r2	r2	r2	r2		r2	r2	z1		
Z_1		s, z6	s, z5	s, z4	s, z3			accept		z2	
Z_2		r3	r3	r3	r3		r3	r3			
Z_3	s, z7										
Z_4	s, z8										

[2,0] (b) Determine os conjuntos de itens definidores dos estados Z_5, Z_6 e de mais três, além dos apresentados.

Usa-se CI, BO em vez de CIRCLE, BOX

z5 = \d(z1,CI) = { item -> CI . point NUM } U { point -> . NUM NUM }

z6 = \d(z1,BO) = { item -> BO . point '{' seq '}' } U { point -> . NUM NUM }

z9 = \d(z5,point) = {item -> CI point . NUM }

z10 = \d(z5,NUM) = { point -> NUM . NUM }

z11 = \d(z6, point) = { item -> BO point . '{' seq '}' }

este é um adicional, pedido por um aluno

z12 = \d(z11, '{') = { item -> BO point '{' . seq '}' } U
{ seq -> . , seq -> . seq item }

4. Considere novamente a gramática G_3 dada no exercício anterior. Uma palavra na linguagem dada por G_3 descreve um desenho definido por uma sequência das seguintes operações gráficas (*item*):

- COLOR NUM, que permite mudar a cor da caneta de desenho para a dada por NUM.
- THICKNESS NUM, que permite mudar a espessura da caneta de desenho para a dada por NUM.
- CIRCLE *point* NUM, que desenha um circunferência centrada no ponto dado por *point* e com raio dado por NUM, usando a caneta de desenho ativa.
- BOX *point* '{' *seq* '}', que cria um sub-desenho com um *offset* dado por *point* em relação ao desenho dentro do qual fica. O ponto (0,0) do sub-desenho é o ponto *point* do desenho onde está incluído.

Apenas o símbolo terminal NUM tem um atributo associado, designado v e que representa um número. O símbolo não terminal *point* representa as coordenadas X e Y de um ponto. A configuração inicial do sistema é caraterizada por cor 0, espessura 1 e *offset* (0,0). Finalmente, considere que dispõe da função `drawCircle(x, y, r, c, t)` que desenha uma circunferência centrada no ponto `(x,y)`, com raio `r`, usando uma caneta de desenho com cor `c` e espessura `t`.

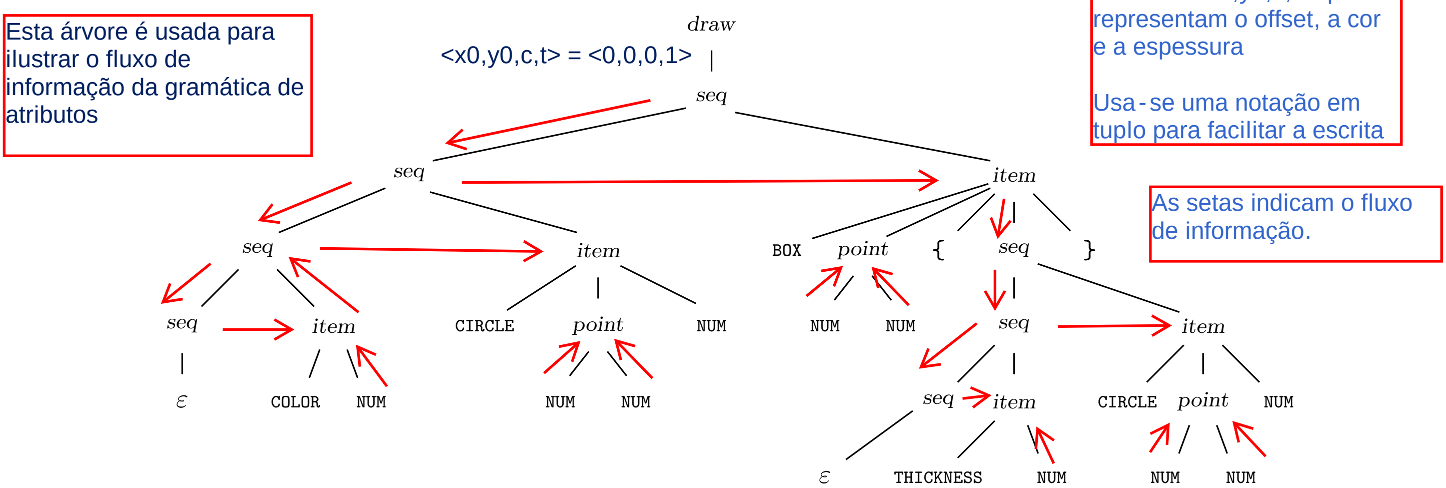
[1,5] (a) Trace a árvore de derivação da palavra

COLOR NUM CIRCLE NUM NUM NUM BOX NUM NUM '{' THICKNESS NUM CIRCLE NUM NUM NUM '}'

Se quiser, ao traçar a árvore, pode abreviar a designação dos símbolos, desde que isso não afete a interpretação da sua resposta.

Associam-se a seq e item 4 atributos <x0,y0,c,t> que representam o offset, a cor e a espessura

Usa-se uma notação em tuplo para facilitar a escrita



[3,0] (b) Construa uma gramática de atributos que permita invocar a função `drawCircle` de forma adequada para cada circunferência incluída num desenho.

Produção	**Regra semântica**
draw → *seq*	seq.<x0,y0,c,t> = <0,0,0,1>
seq → ε	
seq_1 → seq_2 *item*	seq2.<x0,y0,c,t> = seq1.<x0,y0,c,t> item.<x0,y0,c,t> = seq2.<x0,y0,c,t> seq1.<x0,y0,c,t> = item.<x0,y0,c,t>
item → COLOR NUM	item.c = NUM.v
item → THICKNESS NUM	item.t = NUM.v
item → CIRCLE *point* NUM	drawCircle(item.x0+point.x,item.y0+point.y,NUM.v,item.c,item.t);
item → BOX *point* { *seq* }	seq.<x0,y0,c,t> = <item.x0+point.x, item.y0+point.y, item.c, item.t>
point → $\texttt{NUM}_1$ $\texttt{NUM}_2$	point.x = NUM1.v; point.y = NUM2.v

CÁBULA OFICIAL

`first`:

`first`(α) $\{$

  `if` $(\alpha \in T \ \lor\ \alpha = \varepsilon)$ `then`

    `return` $\{\alpha\}$

  `else if` $(\alpha \in N)$ `then`

    `return` $\displaystyle\bigcup_{(\alpha \to \beta) \in P} \texttt{first}(\beta)$

  `else`

    $h = \texttt{head}(\alpha)$ $\#\ com\ |h| = 1$

    $\beta = \texttt{tail}(\alpha)$

    `if` $\varepsilon \notin \texttt{first}(h)$ `then`

      `return` $\texttt{first}(h)$

    `else`

      `return` $(\texttt{first}(h) - \{\varepsilon\}) \cup \texttt{first}(\beta)$

$\}$

`follow`:

`foreach` $X \in N$

  $\texttt{follow}(X) = \emptyset$

$\$ \in \texttt{follow}(S)$

`do`

  `if` $(A \to \alpha B) \in P$ `then`

    $\texttt{follow}(B) \supseteq \texttt{follow}(A)$

  `else` `/*` $(A \to \alpha B \beta) \in P$ `*/`

    `if` $\varepsilon \in \texttt{first}(\beta)$ `then`

      $\texttt{follow}(B) \supseteq \texttt{first}(\beta)$

    `else`

      $\texttt{follow}(B) \supseteq (\texttt{first}(\beta) - \{\varepsilon\}) \cup \texttt{follow}(A)$

`while` *<any set changes>*

`predict`:

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