Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática

Linguagens Formais e Autómatos / Compiladores

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

NM

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

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

[2,0] (a) Faça a derivação à esquerda da palavra `a b c d e f`.

[2,0] (b) Determine o conjunto `first`(A B C).
Apresente os passos intermédios e/ou o raciocínio adequados para suportar a sua resposta.

[2,0] (c) Determine o conjunto `follow`(B).
Apresente os passos intermédios e/ou o raciocínio adequados para suportar a sua resposta.

[2,0] (d) Um símbolo não terminal é útil se for simultaneamente acessível e produtivo; é inútil caso contrário. Mostre que E é útil e D é inútil. Apresente os passos intermédios e/ou o raciocínio adequados para suportar a sua resposta.

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.

---

2. Sobre o alfabeto $A = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ considere a linguagem L_2 tal que:

   - em qualquer das suas palavras, o número de ocorrências da letra `a` mais o número de ocorrências da letra `b` é igual ao número ocorrências da letra `c`;
   - em qualquer prefixo de uma palavra de L_2, o número de ocorrências da letra `a` mais o número de ocorrências da letra `b` é maior ou igual ao número ocorrências da letra `c`.

[2,0] (.) Construa uma gramática independente do contexto que represente a linguagem L_2.

---

3. Sobre o alfabeto $T_3 = \{\texttt{NUM}, \texttt{POLY}, \texttt{'('}, \texttt{')'}\}$, considere a gramática G_3 dada a seguir assim como os conjuntos `predict` das suas produções.

   draw → *seq* ;
   seq → ε | *poly seq* ;
   poly → POLY *point point xpoints* ;
   point → '(' NUM NUM ')' ;
   xpoints → ε | *point xpoints* ;

Produção	predict
draw → *seq* $	{ POLY , $ }
seq → ε	{ $ }
seq → *poly seq*	{ POLY }
poly → POLY *point point xpoints*	{ POLY }
point → '(' NUM NUM ')'	{ '(' }
xpoints → ε	{ POLY , $ }
xpoints → *point xpoints*	{ '(' }

[2,0] (a) Construa a tabela de decisão para um reconhecedor (*parser*) **descendente** com *lookahead* de 1 da gramática G_3.

(b) A construção de um reconhecedor (*parser*) **ascendente** para uma gramática baseia-se na colecção de conjuntos de itens. O elemento inicial dessa colecção para a gramática G_3 está parcialmente descrito a seguir.

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

Complete-o e determine mais 5 elementos desse conjunto, incluindo os diretamente alcançáveis a partir de Z_0.

---

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 (*polylines*). O símbolo terminal NUM tem um atributo associado, designado v, que representa um número (coordenada). O símbolo não terminal point representa as coordenadas X e Y de um ponto. Finalmente, considere que dispõe da função `drawLine(x1, y1, x2, y2)` que desenha uma linha entre os pontos `(x1,y1)` e `(x2,y2)`.

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

POLY '(' NUM NUM ')' '(' NUM NUM ')' POLY '(' NUM NUM ')' '(' NUM 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.

(b) Construa uma gramática de atributos que permita invocar a função `drawLine` de forma adequada para cada linha poligonal incluída num desenho.

CÁBULA OFICIAL

---

first:

st(α) {
if (α ∈ T_ε) then /* one terminal or empty */
return {α}
else if (α ∈ N) then /* one non terminal */
return $\bigcup_{(\alpha \rightarrow \beta) \in P}$ first(β)
lse /* more than one symbol */
h = head(α) /* the first symbol */
β = tail(α) /* the remaining symbols */
if $\varepsilon \notin$ first(h) then
return first(h)
else
return (first(h) − {ε}) ∪ first(β)