Gramáticas Livres de Contexto

Marcelo Johann

Conteúdo da aula

- Trabalho - primeira etapa
  - Definição, código e estrutura
  - Mais elementos e detalhes de lex
- GLCs
  - Gramática, produção, derivações, árvores
  - Ambíguas, sem ciclos, ε-livres, fatoradas à esquerda, recursivas à esquerda, simplificadas
- Transformações
  - Eliminação de produções vazias, de recursividade à esquerda (direta, indireta), fatoração

Hierarquia de Linguagens de Chomsky

Tipo 0	Sem restrição recur. enum.	Qualquer $\underline{u} \rightarrow \underline{v}$ desde que $\underline{u}$ seja não vazio
Tipo 1	Sensível ao contexto	$\underline{w}A\underline{z} \rightarrow \underline{w}v\underline{z}$ ($\underline{w},\underline{z}$ são o contexto)
Tipo 2	Livre do Contexto	$A \rightarrow \underline{v}$ (1 símbolo a esquerda)
Tipo 3	regular	$A \rightarrow a \mid aB \mid \varepsilon$ (derivações a direita)

Definições

- Gramáticas
- Produção
- Derivações
- Árvores de Derivação

Usando Gramáticas para definir a sintaxe

- Muitas construções de linguagens de programação são recursivas
  - Regexp não podem reconhecer tais construções.
- Exemplo de definição recursiva:
  - se S_1 e S_2 são enunciados e E é uma expressão, então "if E then S_1 else S_2" é um enunciado
- Exemplo de definição com uma gramática:

  cmd → if expr then cmd else cmd
  expr → (bool) | expr && expr | expr || expr
  bool → true | false

Gramáticas formais: Notações e Definições

- Uma Gramática Formal é um conjunto de 4 elementos:
  - G := {S, **P**, **N**, **T**}
- **S:** o Símbolo Inicial (S ∈ **N**)
- **P:** Conjunto de regras de produções do tipo

  $\underline{u} \rightarrow \underline{v}$
- **T:** conjunto de símbolos terminais
  - Palavras ou tokens da linguagem
- **N:** conjunto de símbolos não terminais
  - Símbolos que podem ser substituídos pelas produções
- **Vocabulário e notações:**
  - **V:** alfabeto
    - **V = N ∪ T** (**N ∩ T = Ø**)
  - **u:** string pertencente a V

Derivações

- Seja:
  - um não-terminal A ∈ **N**,
  - uma regra de produção p = { A → v } ∈ **P**, e
  - w, z dois strings quaisquer.

  A transformação:

  $$wAz \Rightarrow wvz$$

  É chamada "**derivação** em um passo usando p"
- Se $w_1 \Rightarrow w_2 \Rightarrow w_3 \ldots \Rightarrow w_n$ então podemos dizer

  $$w_1 \Rightarrow^* w_n$$

  - "derivação em múltiplos (0 ou mais) passos"; também $\Rightarrow^+$ (um ou mais)

Definição da Linguagem Gerada

- A linguagem gerada G (notada L(G)) é:
  - O conjunto formado por **todos os strings de** símbolos **terminais deriváveis a partir** do símbolo inicial **S**
  - L(G) = {s | s é um elemento de T* e S ⇒+ s}
- Equivalência:
  - G_1 e G_2 são equivalentes se $L(G_1) = L(G_2)$
    - ⇒ Todos strings gerados estão em L
    - ⇐ Todos strings w ∈ L podem ser gerados
- Convenções
  - Símbolos que representam terminais em minúsculos:
    - **u, v, x, y**, ...
  - Símbolos que representam não-terminais em maiúsculos:
    - **X, Y, TERM, S**,...
  - Símbolos que representam formas sentenciais (seqüências de terminais e não-terminais): letras gregas ou sublinhadas:
    - **α, β, ω, w, z**

Caso particular: Gramática Regular

- Gramática regular:
  - produções exclusivamente da forma:
    - A → wB
    - A → w,
    - onde w ∈ T* e A, B ∈ N (gramática linear à direita)
  - A linguagem gerada por uma gramática regular é regular.
  - Pode-se gerar os mesmos *strings* através de uma expressão regular.

Exemplo G={N,T,P,S}

N = {S, B, C, D}

T = {0, 1}

S -> 0B | 1C
B -> 1S
C -> 0C | 0D | 1
D -> 0

S

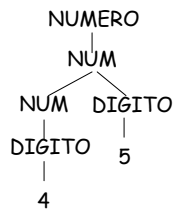
Exercício: fornecer expressão regular equivalente à gramática acima.

Árvore de derivação (**Parse Tree**)

- Árvore de derivação:
  - representação gráfica da derivação de uma sentença (*string*)
  - estrutura hierárquica que originou a sentença
    - A raiz da árvore representa o símbolo inicial
    - Os vértices interiores são não-terminais
    - Os símbolos terminais e a palavra vazia são folhas

Árvore de derivação – exemplo 1

- Gramática G = (
  {NUMERO, NUM, DIGITO},
  {0,1,2,...,9},
  P,
  NUMERO)
- P (regras de produção) =
  NUMERO → NUM
  NUM → NUM DIGITO | DIGITO
  DIGITO → 0|1|...|9



Árvore de derivação para a sentença 45

Derivação mais à esquerda e mais à direita

- Derivação mais à esquerda de uma sentença:
  - seqüência de formas sentenciais que se obtém derivando sempre o símbolo não-terminal mais à esquerda.
- Derivação mais à direita de uma sentença:
  - seqüência de formas sentenciais que se obtém derivando sempre o símbolo não-terminal mais à direita.

Árvore de derivação – exemplo 2

Gramática G = ({E}, {+, -, *, /, (,), x}, P, E) sendo

P = {E → E + E | E –E| E*E | E/E | (E) | x}

A mesma árvore pode representar mais de uma derivação para uma mesma sentença

Possíveis derivações para a árvore:

$E \Rightarrow E + E \Rightarrow x + E \Rightarrow x + E * E \Rightarrow x + x * E \Rightarrow x + x * x$

$E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + E * x \Rightarrow E + x * x \Rightarrow x + x * x$

$E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow x + E * E \Rightarrow x + x * E \Rightarrow x + x * x$

Isso não é problema....

MAS....

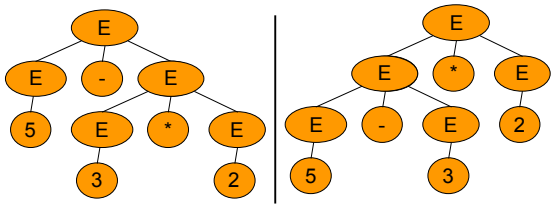
Gramática Ambígua

E → E OP E | "(" E ")" | x

OP → "*" | "/" | "+" | "-"

- Considere 5 – 3 * 2

Quando há mais de uma árvore de derivação para uma mesma sentença



Tipos ou Características

- Gramáticas Ambíguas
- Gramáticas sem ciclos
- Gramáticas ε-livres
- Gramáticas fatoradas à esquerda
- Gramáticas recursivas à esquerda
- Gramáticas simplificadas

Outras Classificações de Gramáticas

- Gramática sem ciclos:
  - Uma gramática sem ciclos é uma GLC que não possui derivações da forma

    $A \Rightarrow^+ A$ para algum $A \in N$
- Gramática ε-livre :
  - GLC que não possui produções vazias do tipo

    $A \rightarrow \varepsilon$

    exceto a produção $S \rightarrow \varepsilon$ (S é o símbolo inicial).

Outras Classificações de Gramáticas

- Gramática fatorada à esquerda:
  - GLC que **não** possui produções do tipo $A \rightarrow \alpha\beta_1 | \alpha\beta_2$ para alguma forma sentencial α.
- Gramática recursiva à esquerda:
  - GLC que permite a derivação

    $A \Rightarrow^+ A\alpha$ para algum $A \in N$

    O não terminal A deriva ele mesmo, de forma direta ou indireta, como

OBS: RECONHECEDOR TOP-DOWN não aceita gramáticas recursivas à esquerda

Transformações de GLCs

- (A) Eliminação de produções vazias
- (B) Eliminação de recursividade à esquerda:
  - recursão direta
  - recursão indireta
- (C) Fatoração de uma gramática

Eliminação de Produções Vazias

- Objetivo:
  - eliminar produções da forma $A \rightarrow \varepsilon$.
- Algoritmo: seja G = (N,T,P,S) uma GLC
  - Etapa 1:
    - construir N_ε , o conjunto de não-terminais que geram a palavra vazia:

    $N_\varepsilon = \{A \mid A \rightarrow \varepsilon\}$; // é um conjunto de símbolos

    Repita

    $N_\varepsilon = N_\varepsilon \cup \{X \mid X \rightarrow X_1...X_n \in P \text{ tq } X_1,...,X_n \in N_\varepsilon\}$

    Até que o cardinal de N_ε não aumente.

Eliminação de Produções Vazias

- Etapa 2:
  - construir o conjunto de produções sem produções vazias:

  gera $G_1 = (N,T,P_1,S)$, onde P_1 é construído como segue:

  $P_1 = \{A \rightarrow \alpha \mid \alpha \neq \varepsilon\}$;

  Repita

  Para toda produção $A \rightarrow \alpha \in P_1$ e $X \in N_\varepsilon$ tal que $\alpha = \alpha_1 X \alpha_2$ e $\alpha_1\alpha_2 \neq \varepsilon$

  Faça $P_1 = P_1 \cup \{A \rightarrow \alpha_1\alpha_2\}$

  Até que o cardinal de P_1 não aumente

Eliminação de Produções Vazias

- Etapa 3:
  - incluir a geração da palavra vazia, se necessário:
  - Se a palavra vazia pertence à linguagem, então a gramática resultante é

  $G_2 = (N,T,P_2,S)$, onde $P_2 = P_1 \cup \{S \rightarrow \varepsilon\}$

(B) Eliminação de recursividade à esquerda

- Exemplo de GLC recursiva à esquerda:

  $A \rightarrow Aa \mid b$

- Gramáticas transformadas equivalentes:

Com a palavra vazia	Sem a palavra vazia
$A \rightarrow bX$	$A \rightarrow b \mid bX$
$X \rightarrow aX \mid \varepsilon$	$X \rightarrow a \mid aX$

Obs: pode ainda haver recursão indireta!

(B) Outro exemplo de recursividade

- E -> E+T | T
- T -> T*F | F
- F -> (E)| **Id**

A regra E -> E+T | T se torna:

E -> TE'

E' -> +TE'| ε

A regra T- > T*F | F se torna:

T -> FT'

T'-> *FT' | ε

(C) Fatoração de uma gramática

- Elimina a indecisão de qual produção aplicar quando duas ou mais produções iniciam com a mesma forma sentencial

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$

Se torna:

$A \rightarrow \alpha X$

$X \rightarrow \beta_1 \mid \beta_2$

(C) Exemplo de Fatoração a Esquerda

Cmd → if Expr then Cmd else Cmd
Cmd → if Expr then Cmd
Cmd → Outro

- Fatorando a esquerda:

Cmd → if Expr then Cmd ElseOpc
Cmd → Outro
ElseOpc → else Cmd | ε

Leituras e Tarefas

Fazer o Trabalho!