

Compiladores

Prof. Marc Antonio Vieira de Queiroz

Ciência da Computação - UNIFIL

LAB 7

marc.queiroz@unifil.br

21/05/2013

Roteiro I

- 1 Eliminação da recursão à esquerda
- 2 Algoritmo para eliminação de recursão à esquerda em uma gramática

Eliminação da recursão à esquerda I

Uma gramática possui recursão à esquerda se ela tiver um não-terminal A tal que exista uma derivação $A \xRightarrow{+} A\alpha$ para uma cadeia α .

Os métodos de análise descendentes não podem tratar gramáticas com recursão à esquerda, de modo que uma transformação é necessária para eliminar a recursão.

$A \rightarrow A\alpha|\beta$ substituído por:

Eliminação da recursão à esquerda II

$$\begin{array}{l} A \rightarrow \beta A' \\ A' \rightarrow \alpha A' \mid \epsilon \end{array}$$

Figura: 1

Exemplo 4.17 I

A gramática de expressão não recursiva à esquerda:

$$E \rightarrow T E'$$

$$E' \rightarrow + T E'$$

$$T \rightarrow F T'$$

$$T' \rightarrow * F T'$$

$$F \rightarrow (E) \mid \mathbf{id}$$

Exemplo 4.17

$$\begin{array}{lcl} E & \rightarrow & E + T \mid T \\ T & \rightarrow & T * F \mid F \\ F & \rightarrow & (E) \mid \mathbf{id} \end{array}$$

Recursão à esquerda imediata

A recursão à esquerda imediata pode ser eliminada pela técnica a seguir:

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

Resultando em:

$$\begin{array}{l} A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{array}$$

O algoritmo 4.19, elimina sistematicamente a recursão à esquerda de uma gramática.

Exceções são ciclos ou produções ϵ .

ALGORITMO 4.19: Eliminando a recursão à esquerda.

ENTRADA: Gramática G sem ciclos ou produções- ϵ .

SAÍDA: Uma gramática equivalente a G sem recursão à esquerda.

MÉTODO: Aplique o algoritmo da Figura 4.11 a G . Observe que a gramática resultante sem recursão à esquerda pode ter produções- ϵ .

- 1) arrume os não-terminais em uma ordem crescente qualquer A_1, A_2, \dots, A_n .
- 2) **for** (cada i de 1 até n) {
- 3) **for** (cada j de 1 até $i-1$) {
- 4) substitua cada produção da forma $A_i \rightarrow A_j \gamma$ pelas
 produções $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_k \gamma$, onde
 $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$ são produções- A_j
- 5) }
- 6) elimine as recursões esquerdas imediatas nas produções- A_i
- 7) }

FIGURA 4.11 Algoritmo para eliminar a recursão à esquerda de uma gramática.

Exemplo I

Vamos aplicar o algoritmo na gramática abaixo:

$$\begin{array}{l} S \rightarrow A a \mid b \\ A \rightarrow A c \mid S d \mid \epsilon \end{array}$$

Ordenamos os não-terminais S, A. Como não há recursão à esquerda imediata entre as produções-S, nada acontece durante o loop externo para $i=1$. Para $i=2$, substituímos o S em $A \rightarrow Sd$ para obter as produções A.

$$A \rightarrow A c \mid A a d \mid b d \mid \epsilon$$

A eliminação da recursão à esquerda imediata entre essas produções A gera a gramática a seguir:

$$\begin{array}{l} S \rightarrow A a \mid b \\ A \rightarrow b d A' \mid A' \\ A' \rightarrow c A' \mid a d A' \mid \epsilon \end{array}$$

Fatoração à Esquerda

A fatoração à esquerda é uma transformação útil na produção de gramáticas adequadas para um reconhecedor gramático preditivo, ou descendente.

Quando a escolha entre duas ou mais alternativas das produções-A não é clara, ou seja, elas começam com a mesma forma sentencial, podemos reescrever essas produções para adiar a decisão até que tenhamos lido uma cadeia de entrada longa o suficiente para tomarmos a decisão correta.

$stmt \rightarrow$	if <i>expr</i> then <i>stmt</i> else <i>stmt</i>
	if <i>expr</i> then <i>stmt</i>

Em geral, se $A \rightarrow \alpha\beta_1 | \alpha\beta_2$ forem duas produções A, e a entrada começar com uma cadeia não vazia derivada de α , não saberemos para qual cadeia expandir, $\alpha\beta_1$ ou $\alpha\beta_2$.

Fatoração à Esquerda

Contudo, podemos adiar a decisão expandindo A para $\alpha A'$ e após ler a entrada derivada de α , expandimos A' para β_1 ou para β_2 .

$$\begin{array}{l} A \rightarrow \alpha A' \\ A' \rightarrow \beta_1 \mid \beta_2 \end{array}$$

Algoritmo para Fatoração à Esquerda de uma gramática

ALGORITMO 4.21: Fatoração à esquerda de uma gramática.

ENTRADA: Gramática G .

SAÍDA: Uma gramática equivalente a G , fatorada à esquerda.

MÉTODO: Para cada não-terminal A , encontre o prefixo α mais longo, comum a duas ou mais de suas alternativas. Se $\alpha \neq \epsilon$ — ou seja, existe um prefixo comum não-trivial —, substitua todas as produções- A , $A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n | \gamma$, onde γ representa todas as alternativas que não começam com um α , por

$$A \rightarrow \alpha A' | \gamma$$

$$A' \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$$

Exemplo 4.22

A gramática a seguir abstrai o problema do "else vazio"

$$\begin{array}{l} S \rightarrow iEtS \mid iEtSeS \mid a \\ E \rightarrow b \end{array}$$

Neste exemplo, i , t e e representam **if**, **then** e **else**; E e S representam respectivamente "expressão condicional" e "comando".

Exemplo 4.22

$$\begin{aligned} S &\rightarrow i E t S S' \mid a \\ S' &\rightarrow e S \mid \epsilon \\ E &\rightarrow b \end{aligned}$$

Construções de linguagens não-livres de contexto I

Algumas construções sintáticas encontradas em linguagens de programação típicas não podem ser especificadas usando apenas gramáticas livres de contexto.

Exemplo 4.25: A linguagem neste exemplo é uma abstração do problema de verificar se os identificadores foram declarados antes de serem usados em um programa. A linguagem consiste em cadeias da forma wcw , onde o primeiro w representa a declaração de um identificador w , c representa um trecho de programa, e o segundo w representa o uso do identificador.

A linguagem abstrata é $L_1 = \{wcw \mid w \text{ está em } (a|b)^*\}$. L_1 consiste em todas as palavras compostas de uma cadeia de a s e b s separados por c , como em $aabcaab$. Nesse caso a dependência de contexto de L_1 implica diretamente a dependência de contexto de linguagens de programação como C e Java, que exigem a declaração dos

Construções de linguagens não-livres de contexto II

identificadores antes de seus usos e permitam identificadores de qualquer tamanho.

Por esse motivo, uma gramática para C ou Java não distingue entre identificadores que possuem cadeias de caracteres diferentes. Em vez disso, todos os identificadores são representados na gramática por um token como `id`. Em um compilador para tais linguagens, a fase de análise semântica verifica se os identificadores são declarados antes de serem utilizados.