Análise sintática

Escrevendo uma gramática

Prof. Edson Alves

Faculdade UnB Gama

Expressões regulares vs. gramáticas livres de contexto

- Qualquer construção que pode ser descrita por uma expressão regular pode ser descrita por uma gramática
- A recíproca nem sempre é verdadeira
- Por exemplo, a expressão regular $(a \mid b)^*abb$ e a gramática

$$
\begin{aligned}
A_0 &\to aA_0 \mid bA_0 \mid aA_1 \\
A_1 &\to bA_2 \\
A_2 &\to bA_3 \\
A_3 &\to \epsilon
\end{aligned}
$$

  descrevem a mesma linguagem
- É possível converter automaticamente um autômatico finito não-determinístico em uma gramática que gere a mesma linguagem do AFN

Algoritmo de conversão de um AFN para uma gramática livre de contexto

Input: um AFN
Output: uma gramática livre de contexto

for cada estado i do AFN **do**
    crie um símbolo não-terminal A_i da gramática
    if o estado i possui um transição para o estado j com rótulo a **then**
        introduza a produção $A_i \to aA_j$ na gramática
    else if o estado i possui um transição para o estado j com rótulo ϵ **then**
        introduza a produção $A_i \to A_j$ na gramática
    if o estado i é um estado de aceitação **then**
        introduza a produção $A_i \to \epsilon$ na gramática
    else if o estado i é o estado de partida **then**
        torne o estado A_i o símbolo de partida da gramática

Razões para o uso de expressões regulares para definir a estrutura léxica

1. As regras léxicas de uma linguagem geralmente são simples, sendo as expressões regulares suficientes para descrevê-las
2. As expressões regulares, em geral, descrevem os tokens da linguagem de forma mais concisa e clara do que as gramáticas livres de contexto
3. É possível gerar analisadores léxicos mais eficientes a partir de expressões regulares do que a partir de gramáticas arbitrárias
4. A separação da estrutura léxica da estrutura sintática permite a modularização da interface de vanguarda

Verificando a linguagem gerada por uma gramática

- A prova que uma gramática G gera uma linguagem $L(G)$ é feita em duas etapas:
  1. mostrar que cada cadeia gerada por G está em $L(G)$
  2. mostrar que cada cadeia em $L(G)$ pode ser gerada por G
- Por exemplo, considere a gramática

$$S \rightarrow (S)S \mid \epsilon$$

- Esta gramática gera todas as cadeias de parêntesis balanceadas
- Para provar esta afirmação, primeiro é preciso provar que qualquer cada sentença derivável de S é uma cadeia de parêntesis balanceada
- Esta prova é feita por indução no número de passos da derivação

Verificando a linguagem gerada por uma gramática

- Em apenas um passo de derivação, a única cadeia gerada é a cadeia vazia ϵ, a qual é trivialmente balanceada
- Suponha que qualquer derivação com menos do que n passos gere uma cadeia balanceada
- Uma derivação com exatamente n passos tem a forma

$$S \Rightarrow (S)S \overset{*}{\Rightarrow} (x)S \overset{*}{\Rightarrow} (x)y$$

  onde x e y são cadeias obtidas por meios de derivações que totalizam exatamente $n - 1$ passos
- Pela hipótese de indução, x e y são cadeias de parêntesis balanceadas e, portanto, a derivação S com exatamente n passos também é balanceada

Verificando a linguagem gerada por uma gramática

- A prova que qualquer cadeia balanceada é derivável a partir de S é feita por meio de indução no comprimento da cadeia
- A menor cadeia balanceada é a cadeia vazia, que é derivável a partir de S por meio da produção $S \to \epsilon$
- Suponha que todas as cadeias balanceadas com comprimento menor do que $2n$ sejam deriváveis a partir de S e que w seja uma cadeia balanceada de tamanho $2n$
- Certamente w inicia com um parêntesis à esquerda
- Seja (x) o menor prefixo de w com o mesmo número de parêntesis à esquerda e à direita
- Assim, $w = (x)y$, onde x e y são cadeias balanceadas com comprimento menor do que $2n$
- Pela hipótese de indução, x e y são deriváveis a partir de S
- Assim, w é derivável a partir de S por meio da derivação

$$S \Rightarrow (S)S \stackrel{*}{\Rightarrow} (x)S \stackrel{*}{\Rightarrow} (x)y$$

Eliminando a ambiguidade

- Uma gramática pode ser reescrita para eliminar possíveis ambiguidades
- Por exemplo, considere a gramática abaixo, que torna o `else` opcional:

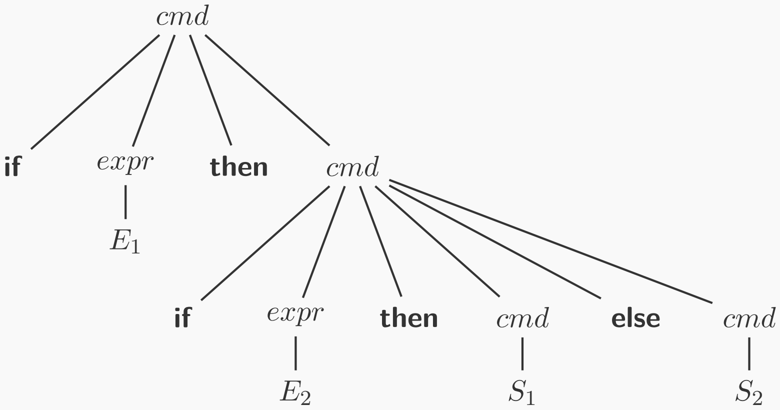
$$
\begin{array}{rcl}
cmd & \to & \textbf{if}\ expr\ \textbf{then}\ cmd \\
 & | & \textbf{if}\ expr\ \textbf{then}\ cmd\ \textbf{else}\ cmd \\
 & | & \textbf{outro}
\end{array}
$$

- Na gramática, **outro** significa qualquer outro enunciado
- Esta gramática é ambígua: a cadeia

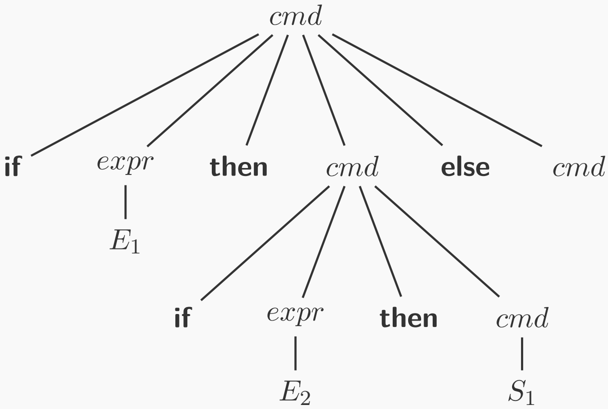
$$\textbf{if}\ E_1\ \textbf{then}\ \textbf{if}\ E_2\ \textbf{then}\ S_1\ \textbf{else}\ S_2$$

possui duas árvores gramaticais distintas

Primeira árvore gramatical para a expressão ‘if E_1 then if E_2 then S_1 else S_2’



Segunda árvore gramatical para a expressão ‘**if** E_1 **then if** E_2 **then** S_1 **else** S_2’



Reescrita para a eliminação da ambiguidade

- Na maioria das linguagens, a primeira das duas árvores seria a esperada
- A regra geral é associar cada **else** ao **then** anterior mais próximo ainda não associado
- Para reescrita, a ideia é que um enunciado entre um **then** e um **else** precisa estar associado, isto é, não pode terminar em um **then** não associado a um **else**

$$
\begin{array}{rcl}
cmd & \rightarrow & cmd\_associado \\
 & | & cmd\_nao\_associado \\
cmd\_associado & \rightarrow & \textbf{if}\ expr\ \textbf{then}\ cmd\_associado\ \textbf{else}\ cmd\_associado \\
 & | & \textbf{outro} \\
cmd\_nao\_associado & \rightarrow & \textbf{if}\ expr\ \textbf{then}\ cmd \\
 & | & \textbf{if}\ expr\ \textbf{then}\ cmd\_associado\ \textbf{else}\ cmd\_nao\_associado
\end{array}
$$

Gramáticas recursivas à esquerda

- Uma gramática é recursiva à esquerda se possui um não-terminal A tal que existe um derivação $A \stackrel{+}{\Rightarrow} A\alpha$ para algum α
- Métodos *top-down* não podem processar gramáticas recursivas à esquerda, demandando uma reescrita da gramática que elimine a recursão à esquerda
- Uma recursão simples à esquerda acontece se existe um produção $A \to A\alpha$
- A recursão simples à esquerda de $A \to A\alpha \mid \beta$ pode ser eliminada ao substituí-la pelas produções

$$
\begin{aligned}
A &\to \beta A' \\
A' &\to \alpha A' \mid \epsilon
\end{aligned}
$$

Exemplo de eliminação de recursão simples à esquerda

$$
\begin{aligned}
E &\to E + T \mid T \\
T &\to T \times F \mid F \\
F &\to (E) \mid \textbf{id}
\end{aligned}
$$

$$
\begin{aligned}
E &\to TE' \\
E' &\to + TE' \mid \epsilon \\
T &\to FT' \\
T' &\to \times FT' \mid \epsilon \\
F &\to (E) \mid \textbf{id}
\end{aligned}
$$

Eliminação de recursão à esquerda

- No caso geral, é possível eliminar todas as recursões simples à esquerda nas produções-A de uma só vez
- Primeiramente, organize todas as produções-A na forma

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

  onde nenhum β_j começa com um A
- Em seguida, substitua estas produções-A pelas produções

$$\begin{aligned} A &\to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A' \\ A' &\to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \epsilon \end{aligned}$$

- Esta substituição elimina todas as recursões simples à esquerda de uma só vez, desde que $\alpha_i \neq \epsilon$ para todo $i = 1, 2, \ldots, m$
- Esta técnica, porém, não elimina recursões à esquerda envolvendo derivações com dois ou mais passos

Algoritmo para eliminação de recursão à esquerda

Input: Uma gramática G sem ciclos (isto é, produções $A \overset{+}{\Rightarrow} A$) e sem produções-$\epsilon$ (do tipo $A \to \epsilon$)

Output: Uma gramática equivalente a G sem recursão à esquerda

Liste, em alguma ordem, os não-terminais $A_1, A_2, \ldots, A_n$

for $i \leftarrow 1, n$ **do**

  for $j \leftarrow 1, i-1$ **do**

    substitua cada produção $A_i \to A_j\gamma$ pelas produções

$$A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \ \ldots \ \mid \delta_k\gamma,$$

onde $A_j \to \delta_1 \mid \delta_2 \mid \ \ldots \ \mid \delta_k$ são todas as produções-A_j atuais

  elimine todas as recursões simples à esquerda nas produções-A_i

Exemplo de eliminação de recursão à esquerda

- Considere a gramática
$$
\begin{aligned}
S &\to Aa \mid b \\
A &\to Ac \mid Sd \mid \epsilon
\end{aligned}
$$
- Observe que o não-terminal S é recursivo à esquerda, pois
$$
S \Rightarrow Aa \Rightarrow Sda
$$
- A aplicação do algoritmo de eliminação de recursão poderia não funcionar, por conta da produção-ϵ do não-terminal A, mas neste caso em particular o algoritmo de fato elimina a recursão

Exemplo de eliminação de recursão à esquerda

- Usando a ordenação S, A, a primeiro iteração do algoritmo não altera a gramática, uma vez que S não tem recursão simples à esquerda
- Na segunda iteração, as produções-A que envolvem S devem ser substituídas, obtendo

$$A \to Ac \mid Aad \mid bd \mid \epsilon$$

- A eliminação da recursão simples à esquerda nas produções-A resulta na gramática livre de recursão à esquerda

$$\begin{aligned} S &\to Aa \mid b \\ A &\to bdA' \mid A' \\ A' &\to cA' \mid adA' \mid \epsilon \end{aligned}$$

Fatoração à esquerda

- A fatoração à esquerda é uma técnica útil para a criação de gramáticas que beneficiam a análise preditiva
- A ideia central da fatoração à esquerda é evitar ambiguidades, quando duas ou mais produções tenham prefixos comuns
- Por exemplo, considere as produções $A \to \alpha\beta_1 \mid \alpha\beta_2$
- Se a entrada contém uma cadeia não-vazia derivada a partir de α, não é possível decidir, de antemão, qual das duas produções usar
- A fatoração à esquerda propõe a reescrita das produções da seguinte forma, que elimina a ambiguidade

$$
\begin{aligned}
A &\to \alpha A' \\
A' &\to \beta_1 \mid \beta_2
\end{aligned}
$$

Exemplo de fatoração à esquerda

- Muitas linguagens permitem que o comando **if-then-else** tenha um **else** vazio:

$$\begin{aligned} cmd &\to \textbf{if}\ expr\ \textbf{then}\ cmd\ \textbf{else}\ cmd \\ &\mid\ \textbf{if}\ expr\ \textbf{then}\ cmd \end{aligned}$$

- Esta gramática pode ser abstraída da seguinte forma

$$\begin{aligned} &S \to iEtS \mid iEtSeS \mid a \\ &E \to b \end{aligned}$$

- A fatoração à esquerda esta gramática resulta em

$$\begin{aligned} &S \to iEtSS' \mid a \\ &S' \to eS \mid \epsilon \\ &E \to b \end{aligned}$$

Exemplos de linguagens não livres de contexto

- Considere a linguagem abstrata

$$L_1 = \{wcw \mid w \in (a \mid b)^*\}$$

- As sentenças de L_1 são formada por uma cadeia w de a's e b's, seguida de um c e repetida em seguida (por exemplo, $abaabcabaab$)
- É possível demostrar que a linguagem L_1 não é livre de contexto
- Tal linguagem abstrai o problema de se verificar se um identificador (w) foi declarado antes de seu uso (c é o que separa a declaração do primeiro uso)
- Não sendo possível definir tal regra sintaticamente, esta verificação fica postergada para a análise semântica

Exemplos de linguagens não livres de contexto

- Considere a linguagem abstrata

$$L_2 = \{a^n b^m c^n d^m \mid n \geq 1 \text{ e } m \geq 1\}$$

- As senteças de L_2 são cadeias onde o número de a's coincide com o número de c's, e o número de b's coincide com o número de d's, em ordem lexicográfica
- L_2 também não é livre de contexto
- Ela abstrai o problema de ser verificar se o número de parâmetros na declaração de uma função é igual ao número de parâmetros na chamada (a^n e b^m seriam as listas de parâmetros de dois procedimentos e c^n e d^n o número de parâmetros na chamada destes procedimentos)

Exemplos de linguagens não livres de contexto

- A linguagem

$$L_3 = \{a^n b^n c^n \mid n \geq 1\}$$

  também não é livre de contexto
- A diferença entre uma linguagem livre de uma não-livre pode ser sutil
- Por exemplo, a linguagem

$$L_1' = \{wcw^R \mid w \in (a \mid b)^*\},$$

  onde w^R é o reverso de w, é livre de contexto
- Ela pode ser gerada pela gramática $S \to aSa \mid bSb \mid c$

Exemplos de linguagens não livres de contexto

- A linguagem
$$L_2' = \{a^n b^m c^m d^n \mid n \geq 1 \text{ e } m \geq 1\}$$
também é livre de contexto
- A gramática abaixo gerada L_2':
$$\begin{aligned} S &\to aSd \mid aAd \\ A &\to bAc \mid bc \end{aligned}$$
- A linguagem
$$L_2'' = \{a^n b^n c^m d^m \mid n \geq 1 \text{ e } m \geq 1\}$$
também é livre de contexto

Exemplos de linguagens não livres de contexto

- Ela pode ser gerada pela gramática

$$
\begin{aligned}
S &\to AB \\
A &\to aAb \mid ab \\
B &\to cBd \mid cd
\end{aligned}
$$

- Por fim, a linguagem

$$L_3' = \{a^n b^n \mid n \geq 1\}$$

  é livre de contexto, gerada pela gramática $S \to aSb \mid ab$
- Informalmente, pode-se afirmar que um autômato finito não pode realizar contagens e que uma gramática pode manter uma contagem de dois itens, mas não de três

Referências

1. **AHO**, Alfred V, **SETHI**, Ravi, **ULLMAN**, Jeffrey D. *Compiladores: Princípios, Técnicas e Ferramentas*, LTC Editora, 1995.