Análise sintática

O papel do analisador sintático

Prof. Edson Alves

Faculdade UnB Gama

Sumário

Expressões regulares vs. gramáticas livres de contexto

- Qualquer construção que pode ser descrita por uma expressão regular pode ser descrita por uma gramática

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

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

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

1: **for** cada estado i do AFN **do**
2: crie um símbolo não-terminal A_i da gramática
3: **if** o estado i possui um transição para o estado j com rótulo a **then**
4: introduza a produção $A_i \to aA_j$ na gramática
5: **else if** o estado i possui um transição para o estado j com rótulo ϵ **then**
6: introduza a produção $A_i \to A_j$ na gramática
7: **if** o estado i é um estado de aceitação **then**
8: introduza a produção $A_i \to \epsilon$ na gramática
9: **else if** o estado i é o estado de partida **then**
10: 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

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

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

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:

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 $L(G)$

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

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

$$S \to (S)S \mid \epsilon$$

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

$$S \to (S)S \mid \epsilon$$

- Esta gramática gera todas as cadeias de parêntesis balanceadas

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

$$S \to (S)S \mid \epsilon$$

- 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 balanceada

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

$$S \to (S)S \mid \epsilon$$

- 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 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

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

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 derivações com que n passos

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

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

  onde x e y são derivações com que n passos
- Pela hipótese de indução, x e y são 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

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 $S \to \epsilon$

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 $2n$

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

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 $S \rightarrow \epsilon$
- 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

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 $2n$

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

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

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

Eliminando a ambiguidade

- Uma gramática pode ser reescrita para eliminar possíveis ambiguidades

Eliminando a ambiguidade

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

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

Eliminando a ambiguidade

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

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

- Na gramática, **outro** significa qualquer outro enunciado

Eliminando a ambiguidade

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

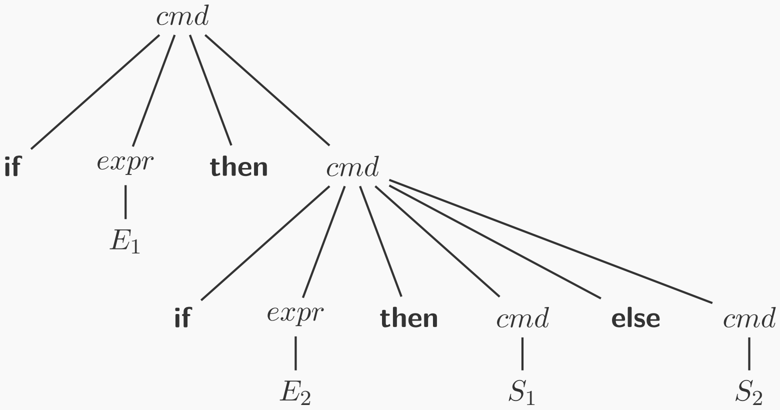
$$\begin{array}{rcl} cmd & \to & \textbf{if } expr \textbf{ then } cmd \\ & | & \textbf{if } expr \textbf{ then else } cmd \\ & | & \textbf{outro} \end{array}$$

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

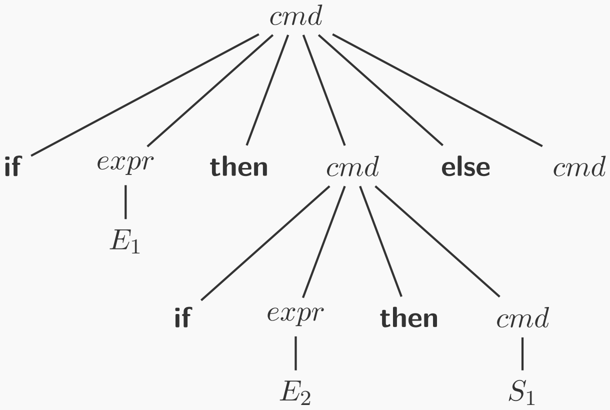
$$\textbf{if } E_1 \textbf{ then if } E_2 \textbf{ then } S_1 \textbf{ else } S_2$$

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

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

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**

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**

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

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 alguma cadeia α

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 alguma cadeia α
- Métodos *top-down* não podem processar gramáticas recursivas à esquerda, demandando uma reescrita da gramática que elimine a recursão à esquerda

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 alguma cadeia 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 $A \rightarrow A\alpha$

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 alguma cadeia 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 uma produção da forma $A \to A\alpha \mid \beta$ 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

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

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

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

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 $i = 1, 2, \ldots, m$

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

1: Liste, em alguma ordem, os não-terminais $A_1, A_2, \ldots, A_n$
2: **for** $i \leftarrow 1, n$ **do**
3:     **for** $j \leftarrow 1, i - 1$ **do**
4:         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

5:     elimine todas as recursões simples à esquerda nas produções-A_i