Construção de Compiladores
Período Especial
Aula 17: Análise Sintática Descendente

Bruno Müller Junior

Departamento de Informática
UFPR

2020

Análise Sintática Descenente

- Esquema básico (com retrocessos);
- Melhoramentos:
  - Eliminação de retrocessos;
  - Fatoração;
  - Eliminação da recursão à esquerda;
- Geração de Programa

Análise Sintática Descenente

- Existe mais de uma forma de eliminar retrocessos:
  - Converter a gramática para a Forma normal de Greibach;
  - Analisar o primeiro símbolo que uma variável pode derivar;

Análise Sintática Descenente

- A FNG é caracterizada quando todas as produções de uma gramática estão no seguinte formato:

$$\{A \rightarrow a\}$$
$$\{A \rightarrow X_1\alpha|X_2\alpha|\ldots\}$$

- Com uma gramática neste formato não há dúvidas sobre qual derivação usar quando tiver um token.
- Infelizmente, a transformação gera um grande número de produções, e não é linear.
- Por esta razão, não é adotado.

Análise Sintática Descenente

- Considere as produções abaixo:

$$\begin{aligned} \{A &\rightarrow BA|a\} \\ \{B &\rightarrow b|C|A\} \\ \{C &\rightarrow c\} \end{aligned}$$

- Não é difícil de ver que os terminais válidos para aplicar uma derivação são:
  - A: a, b, c
  - B: a, b, c
  - C: c
- Faça a análise sintática da entrada $\alpha =$ "$bcaa$"

Análise Sintática Descenente

- O primeiro símbolo terminal (ou só "Primeiro") de cada variável pode ser obtido com o seguinte algoritmo:
  1. Desenhe uma tabela com quatro colunas;
  2. Preencha a primeira coluna de cada linha com uma variável;
  3. Preencha a segunda coluna de cada linha com os terminais ou variáveis que podem ser obtidos em uma derivação.
  4. Preencha a terceira coluna com o fecho transitivo da segunda;
  5. Copie os terminais da terceira coluna para a quarta coluna.
- A quarta coluna contém os terminais válidos de cada variável.

Análise Sintática Descenente

$$
\begin{aligned}
S &\rightarrow AS|BA \\
A &\rightarrow aB|C \\
B &\rightarrow bA|d \\
C &\rightarrow c
\end{aligned}
$$

- Passos 1 e 2: Criar a tabela, preencher primeira coluna

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB		
A	aC		
B	bd		
C	c		

Análise Sintática Descenente

$$
\begin{aligned}
S &\rightarrow AS|BA \\
A &\rightarrow aB|C \\
B &\rightarrow bA|d \\
C &\rightarrow c
\end{aligned}
$$

- Passo 3: Fecho transitivo. Onde encontrar uma variavel, incluir a variável e todos os Ψ_p daquela variável

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB	ABaCdbc	
A	aC	aCc	
B	bd	bc	
C	c	c	

Análise Sintática Descenente

$$
\begin{aligned}
S &\rightarrow AS|BA \\
A &\rightarrow aB|C \\
B &\rightarrow bA|d \\
C &\rightarrow c
\end{aligned}
$$

- Passo 4: Descartar as variáveis

	Ψ_p	Ψ_{p^*}	Primeiro
S	AB	ABaCdbc	adbc
A	aC	aCc	ac
B	bd	bc	bd

Análise Sintática Descenente

- Primeiro (S) = adbc
- Primeiro (A) = ac
- Primeiro (B) = db
- Primeiro (C) = c
- Agora já é possível fazer a análise sintática com complexidade $O(n)$. Verifique com a entrada $\alpha =$ "*abcdad*"

$$
\begin{aligned}
S &\rightarrow AS|BA \\
A &\rightarrow aB|C \\
B &\rightarrow bA|d \\
C &\rightarrow c
\end{aligned}
$$

Fatoração

- Um outro problema ocorre quando temos uma gramática G_1 que precisa verificar mais de um token à frente. Exemplo:

$$\begin{aligned} G_1 = \{S &\rightarrow aAS|aSA \\ A &\rightarrow b|c\} \end{aligned}$$

- Em alguns casos, é possível fatorar G_1 gerando uma gramática G_2 (onde $L(G_1) = L(G_2)$) que só precisa do primeiro token.

$$\begin{aligned} G_2 = \{S &\rightarrow a(AS|SA) \\ A &\rightarrow b|c\} \end{aligned}$$

$$\begin{aligned} G_3 = \{S &\rightarrow aX \\ X &\rightarrow AS|SA \\ A &\rightarrow b|c\} \end{aligned}$$

Fatoração - Exemplo

- Fatore a gramática G_1 abaixo. precisa verificar mais de um token à frente. Exemplo:

$$G_1 = \{E \rightarrow T + E | T$$
$$T \rightarrow F * T | F$$
$$F \rightarrow a | (E)\}$$

$$G_2 = \{E \rightarrow T[+E|\epsilon]$$
$$T \rightarrow F[*T|\epsilon]$$
$$F \rightarrow a|(E)\}$$

$$G_3 = \{E \rightarrow TE_1$$
$$E_1 \rightarrow +E|\epsilon$$
$$T \rightarrow FT_1$$
$$T_1 \rightarrow *T|\epsilon$$
$$F \rightarrow a|(E)\}$$

Fatoração - Observações

- O símbolo ϵ (epsilon) indica a cadeia vazia.
  Não é um símbolo de entrada.
- Tanto a notação de G_2 e de G_3 são válidas, porém iremos adotar G_2 por ser mais sintética.
- Observe que em G_2 foi inserido os símbolo "[" e "]", que delimitam a parte fatorada. Não precisa ser este, mas não pode ser um token válido da linguagem (como por exemplo "(" e ")" de G_1.

Eliminação da recursão à esquerda

- Um outro problema ocorre quando a gramática apresenta produções com recursão à esquerda:

$$A \rightarrow A\alpha|\beta$$

- Há uma forma de converter estas produções para outras equivalentes, onde não há recursão à esquerda:

$$A \rightarrow \beta\{\alpha\}$$

- Aqui, o que está entre os símbolos "{" e "}" são repetidos de zero a n vezes.

Argumentação da validade da transformação

- A transformação aplicada para eliminar a recursão à esquerda pode ser demonstrada formalmente, porém iremos só argumentar:
  - No caso transformado ($A \rightarrow \beta\{\alpha\}$), é fácil de ver que as palavras válidas são β, $\beta\alpha$, $\beta\alpha\alpha$, $\beta\alpha\alpha\cdots\alpha$.
  - Já a produção $A \rightarrow A\alpha|\beta$ mostramos que também gera a mesma linguagem montando a árvore de derivação.

Exercício

- Aplique a transformação na gramática abaixo:

$$
\begin{aligned}
E &\rightarrow E + T | T \\
T &\rightarrow T * F | F \\
E &\rightarrow a | (E)
\end{aligned}
$$

Analisador Sintático

- As transformações acima permitem transformar uma gramática de G_1 em uma gramática G_2 apropriada para ASDR.
- A partir de G_2 existem duas formas básicas de construir o analisador sintático: tabelas e programa equivalente.
- Como o método de tabelas será abordado na análise sintática ascendente (LR), vamos descrever a geração de programas.

Geração de Programas

- Seja G_d uma gramática. Gere uma gramática G'_d aplicando os seguintes passos:
- Aumente a gramática ($S' \rightarrow S\#$).
- Fatore a gramática;
- Elimine recursão à esquerda;
- Determine o Próximo para cada variável.
- A partir de G'_d, escreva um programa onde:
  1. Cada variável é mapeada como uma subrotina;
  2. Em cada produção, faça:
     - ao encontrar uma variável, implemente como uma chamada da subrotina correspondenet:
     - ao encontrar um token, consuma o token e procure pelo próximo token da entrada (analisador Léxico)

Exemplo

- aumentar a gramática:

$$\begin{aligned} S &\rightarrow AS|BA \\ A &\rightarrow aB|C \\ B &\rightarrow bA|d \\ C &\rightarrow c \end{aligned}$$

$$\begin{aligned} S' &\rightarrow S\# \\ S &\rightarrow AS|BA \\ A &\rightarrow aB|C \\ B &\rightarrow bA|d \\ C &\rightarrow c \end{aligned}$$

Exemplo

- 2) Fatorar - Ok.
- 3) Eliminar Recursão Esquerda - Ok.
- 4) Determinar primeiro (Já fizemos).
  - Primeiro (S) = adbc
  - Primeiro (A) = ac
  - Primeiro (B) = db
  - Primeiro (C) = c

Exemplo

- Gerar uma subrotina por variável:

$$S' \rightarrow S\#$$

$$S \rightarrow AS|BA$$

```
S' () {
   PROXIMO(token, simbolo);
   S();
   Se (token != #)
      erroFatal (Esperado FDA);
}
```

```
S () {
  caso simbolo for
     {a, c} : A(); S(); break;
     {b, d} : B(); A(); break;
     outros : erroFatal(Simb inv);
  fimCaso;
}
```

Exemplo

$$A \rightarrow aB|C$$

```
A () {
  caso simbolo for
     {a}     :  PROXIMO(token,
                        simbolo);
                B(); break;
     {c}     :  C(); break;
     outros  : erroFatal(Simb inv);
  fimCaso;
}
```

$$B \rightarrow bA|d$$

```
B () {
  caso simbolo for
     {b} : PROXIMO(token, simbolo);
           A(); break;
     {d} : PROXIMO(token, simbolo);
           break;
     outros : erroFatal(Simb inv);
  fimCaso;
}
```

Exemplo

$$C \rightarrow c$$

```
C () {
    caso simbolo for
        {c}     :  PROXIMO(token, simbolo)
                   break;
        outros  :  erroFatal(Simb inv);
    fimCaso;
}
```

Exercício

$$
\begin{aligned}
E &\rightarrow E + T \mid E - T \mid +T \mid -T \\
T &\rightarrow T * F \mid T/F \mid F \\
F &\rightarrow a|b|(E)
\end{aligned}
$$

Exercício

- Página para anotações

Licença

- Slides desenvolvidos somente com software livre:
  - LaTeX usando beamer;
  - Inkscape.
- Licença:
  - Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License. `http://creativecommons.org/licenses/by-nc-nd/2.5/br/`