Compiladores

Análise Sintática Descendente

Análise Sintática Descendente

- Contrói a Árvore de derivação de cima para baixo em pré-ordem.
- Constrói uma derivação mais à esquerda
- O funcionamento da Análise Sintática Descendente consiste em a cada passo :
  - Determinar a produção a ser aplicada (principal), e
  - "casar" os símbolos no corpo da produção escolhida com os tokens na cadeia de entrada.

Análise Sintática Descendente Recursiva

A ideia básica neste método é **criar um procedimento para cada não-terminal** que é ativado quando pertecer à produção escolhida para ser aplicada na derivação.

Análise Sintática Descendente Recursiva

Suponha um não-terminal A, um esboço do procedimento para A é dado abaixo, supondo que o token na cadeia de entrada é *a*.

```
void A( ) {
    escolha uma produção A → X1X2...Xk
    para i = 1 até k
        se ( Xi é não-terminal ) então
            ative procedimento Xi( )
        senão
            se (Xi igual ao token a)
                Avance para o próximo token
            Senão
                Erro ( )
}
```

Análise Sintática Descendente Recursiva

O método geral de análise descendente recursivo pode exigir retrocesso.(voltar atrás no reconhe cimento fazendo repetidas leituras sobre a entrada) Por isso o esboço acima precisa sofrer pelo menos 3 modificações:

1- o procedimento deve permitir a escolha de mais do que uma produção de A;

2- o erro( ) só acontece quando todas as produções de A já foram escolhidas; e

3- sempre que uma nova produção for escolhida um ponteiro deve marcar a posição em que se encontra na cadeia de entrada

Funções *First(α)* e *Follow(A)*

Para evitar o retrocesso métodos baseados em tabela são utilizados. Para auxiliar na construção das tabelas sintáticas usamos as funções *first(α)* e *follow(A)*.

First(α)

Defina ***First(α)***, onde α é qualquer cadeia de símbolos da gramática, como sendo o conjunto de símbolos terminais que iniciam as cadeias derivadas de α. Se $\alpha \rightarrow^{*} \varepsilon$, então ε está no First(α).

Calculando First(X)

Para calcular o First(X) para todo símbolo X da gramática, aplique as seguintes regras até que não haja mais terminais ou e que possam ser acrescentados a algum dos conjuntos First.

1. Se X é um símbolo terminal, então First(X)= {X}
2. Se X é um símbolo não-terminal e $X \rightarrow Y_1 Y_2 .. Y_k$ é uma produção para k ≥ 1, então acrescente α ao First(X) se, para algum i, α estiver em First(Y_i), e ε estiver em todos os First(Y_1),...,First(Y_{i-1}); ou seja, $Y_1 ... Y_{i-1} \rightarrow \varepsilon$. Se ε está em First($Y_j$) para todo j=1, 2, …, k então ε está em First(X).
3. Se X → e é uma produção de X então ε está em First(X).

Follow(A)

Defina Follow(A), para o não-termianl A, como sendo o conjunto de terminais α que podem aparecer imediatamente à direita de A em uma forma sentencial; ou seja, o conjunto de terminais a tais que exista uma derivação na forma $S \rightarrow \alpha A \alpha \beta$, para algum α, β.

Calculando o Follow(A)

Para calcular o Follow(A) para todos os não-terminais da gramática aplique as seguintes regras até que nada mais possa ser acresncentado a nenhum dos conjuntos Follow.

1. Coloque $ (símbolo de final de cadeia) em Follow(S), onde S é o símbolo inicial da gramática
2. se houver uma produção $A \rightarrow \alpha B \beta$, então tudo no First($\beta$) exceto ε está em Follow(B).
3. se houver uma produção $A \rightarrow \alpha B$, ou uma produção $A \rightarrow \alpha B \beta$ onde Firts(β) contém ε , então inclua Follow(A) ao Follow(B).

Exemplo

Gramática

E → TE'

E' → +TE' | ε

T → FT'

T' → *FT' | ε

F → (E) | id

First(+) = {+}
First(*) = {*}
First(() = {(}
First()) = {)}
First(id) = {id}

First(E) = FIRST(T) = FIRST(F) = { (,id }
First(E') = { +, ε }
First(T') = { *, ε }

Follow(E) = {$} U First()) = { $,) }
Follow(E') = Follow(E) = { $,) }
Follow(T)= First(+TE') U Follow(E) = { +, $,) }
Follow(T') = Follow(T) = { +, $,) }
Follow(F)= First(T') U Follow(T') = { *, +, $,) }