Compiladores – FIRST e FOLLOW (Gramáticas LL(1))

Carlos Henrique Vieira Marques Veeck
Baseado na apresentação de Leopoldo Teixeira (IF688)

O que são FIRST e FOLLOW?

- Usados para construir **parsers top-down preditivos** (como LL(1)).
- Ajudam o parser a escolher **qual produção aplicar**, com base no **próximo token de entrada**.
- Também ajudam em **tratamento de erros**.

FIRST(α)

- Conjunto de terminais que podem iniciar uma derivação da string α.
- Se α pode gerar ε (vazio), então ε ∈ FIRST(α).

Regras para calcular FIRST(X):

1. Se X é terminal → FIRST(X) = { X }
2. Se X → ε → ε ∈ FIRST(X)
3. Se X → $Y_1Y_2…Y$□:
    - Se ε ∉ FIRST(Y_1), então FIRST(Y_1) ⊆ FIRST(X)
    - Se ε ∈ FIRST(Y_1), então FIRST(X) inclui:
        - FIRST(Y_1) – {ε} ∪ FIRST($Y_2Y_3…Y$□)
    - Se ε ∈ FIRST(Yj) para todo j → ε ∈ FIRST(X)

FOLLOW(A)

- Conjunto de terminais que podem aparecer **imediatamente após o não-terminal A** em alguma derivação.

- Se A pode ser o **último símbolo** de uma produção → $ ∈ FOLLOW(A)

Regras para calcular FOLLOW(A):

1. $ ∈ FOLLOW(S), onde S é o símbolo inicial.
2. Se A → αBβ, então tudo de FIRST(β), exceto ε, vai para FOLLOW(B).
3. Se A → αB ou A → αBβ e ε ∈ FIRST(β), então tudo de FOLLOW(A) vai para FOLLOW(B).

Exemplo de gramática:

S → aABe
A → bK
K → bcK | ε
B → d

FIRST:

- FIRST(S) = { a }
- FIRST(A) = { b }
- FIRST(K) = { b, ε }
- FIRST(B) = { d }

FOLLOW:

- FOLLOW(S) = { $ }
- FOLLOW(A) = { d }
- FOLLOW(K) = { d }
- FOLLOW(B) = { e }

Gramáticas LL(1)

- Uma gramática é LL(1) se o parser pode decidir a produção a ser usada olhando **apenas 1 token de lookahead**.

- Não pode haver **ambiguidade** nem **recursão à esquerda**.

Tabela Preditiva LL(1)

- Tabela M[A, a] indica qual produção usar para não-terminal A com símbolo de entrada a.
- Construção:
    - Para cada produção A → α:
        - Para cada terminal a ∈ FIRST(α), adicione A → α em M[A, a]
        - Se ε ∈ FIRST(α), para cada b ∈ FOLLOW(A), adicione A → α em M[A, b]
- Células vazias indicam **erro**.

Parsing LL(1) sem recursão

- Utiliza **pilha explícita** e **tabela preditiva**.
- Algoritmo:
    1. Inicialize a pilha com $ e o símbolo inicial S.
    2. Leia o primeiro token da entrada.
    3. Enquanto o topo da pilha ≠ $:
        - Se topo == token → consome token e desempilha.
        - Se topo é terminal ≠ token → erro.
        - Se topo é não-terminal:
            - Consulte M[topo, token]:
                - Se erro → erro.

- Se produção A → $Y_1Y_2…Y_n$ → desempilha e empilha $Y_n...Y_1$.

Exemplo de Gramática LL(1):

E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε
F → (E) | id

🔹 FIRST

Objetivo: identificar com quais **terminais uma derivação pode começar**.

Regras:

1. Se a produção começa com um **terminal**, ele entra no FIRST.
2. Se começa com um **não-terminal**, adicione os símbolos do **FIRST desse não-terminal** (exceto ε).
    - Se esse FIRST contém **ε**, continue verificando os próximos símbolos da produção.
3. Se **toda a produção pode gerar ε**, então **ε entra no FIRST**.

🔹 FOLLOW

Objetivo: identificar quais **terminais podem aparecer imediatamente após um não-terminal**.

Regras:

1. Para o símbolo inicial (ex: S), o **símbolo $ sempre entra no FOLLOW(S)**.
2. Em uma produção A → αBβ, adicione ao FOLLOW(B):
    - **Todos os símbolos de FIRST(β)** (exceto ε).
3. Se β pode derivar ε (ou se B é o último da produção),

- então **tudo de FOLLOW(A)** entra no FOLLOW(B).