Analisador Sintático – Algoritmo

Cada símbolo não-terminal vai possuir uma função que recebe como parâmetro o token corrente.

A função deve tratar os elementos na ordem que aparecem nas regras.

Analisador Sintático – Algoritmo

$A \rightarrow aB$

```
função A(Token t){
        tratar terminal a;
        tratar não-terminal B;
}
```

Analisador Sintático – Algoritmo

O tratamento de cada regra depende do tipo do símbolo:

- **Terminal** :

  verifica se o token atual é igual ao terminal esperado;

  avança para token atual para o próximo token na lista.

- **Não-terminal** :

  chama função do símbolo com o token como parâmetro.

Analisador Sintático – Algoritmo

```
função tratarTerminal(Token t, Classificação c){
        se t.classificação é igual c então
                avança(t);
}
```

Analisador Sintático – Algoritmo

$A \rightarrow aB$

```
função A(Token t){
        tratarTerminal( t , 'a' );
        B( t )
}
```

Analisador Sintático – Algoritmo

Símbolos não-terminais com mais de uma regra devem possuir comando de decisão para saber qual regra deve ser seguida.

Comparam o token atual com o terminal que inicia a regra.

Caso a regra inicie com não-terminal, Usa o conjunto de símbolos terminais que iniciam o lado direito das regras desse não-terminal.

Analisador Sintático – Conjunto First

Definido para todo não-terminal.

Formado pelos símbolos terminais que iniciam o lado direito das regras.

Ex.:

$D \rightarrow eF$	First(D) = { e }
$D \rightarrow eF \mid dC$	First(D) = { e, d }

Analisador Sintático – Conjunto First

Caso o lado direito inicie por um não-terminal, seu conjunto First fará parte do outro.

Ex.:

D → eF | dC | E First(D) = { e, d } U First(E) = { e, d, c }

E → cF | e First(E) = { c, e }

Analisador Sintático – Algoritmo

$A \rightarrow aB \mid D$

```
função A(Token t){
        tratarTerminal( t , 'a' );
        B( t )
}
```

Analisador Sintático – Algoritmo

$A \rightarrow aB \mid D$

```
função A(Token t){
    se t equivale a 'a' então
        tratarTerminal( t , 'a' );
        B( t );
    senão se t está no First( D ) então
        D( t );
}
```

Analisador Sintático – Erro

Será criada uma função única para tratar erro.

Função erro deve receber o token atual como parâmetro.

Função tratarErro será chamada quando:

- Símbolo terminal esperado não é reconhecido no token.
- Token não corresponde a nenhum regra do não-terminal.

Analisador Sintático – Algoritmo

```
função tratarTerminal(Token t, Classificação c){
        se t.classificação é igual c então
                avança(t);
        senão
                tratarErro(t);
}
```

Analisador Sintático – Algoritmo

$A \rightarrow aB \mid D$

```
função A(Token t){
    se t equivale a 'a' então
        tratarTerminal( t , 'a' );
        B(t);
    senão se t está no First(D) então
        D(t);
    senão
        tratarErro(t);
}
```

Analisador Sintático – Tratamento de erros

Procedimentos podem ser:

Parar Compilação

Ao encontrar um erro interrompe a análise.

Continuar Compilação

- Modo Notificação: Apenas notifica sobre o erro.
- Modo Recuperação: Tenta encontrar estado seguro.

Analisador Sintático – Algoritmo

Parar Compilação:

```
função tratarErro(Token t, Símbolo NT){
        mensagem de erro.
        encerrar Programa.
}
```

Problema: Informar um erro de cada vez.

Analisador Sintático – Algoritmo

Continuar Compilação no modo **Notificação:**

```
função tratarErro(Token t, Símbolo NT){
        mensagem de erro.
}
```

Problema: Possibilidade de informar falsos erros.

Analisador Sintático – Recuperação de erros

Não garantem + de 1 erro mas não devem gerar falsos erros.

Sincronização (modo Pânico)

Pula tokens até encontrar um seguro para prosseguir.

Mais fácil de implementar.

Reparação (recuperação local)

Tenta supor o erro e inserir/remover tokens.

Mais complexo de implementar.

Analisador Sintático – Sincronização

Procurar token dentro de conjunto de tokens de sincronização.

Definição do conjunto é determinante para o sucesso da técnica.

Uma possibilidade:

Encerrar a avaliação do não-terminal.

montar conjuntos que garantam isso.

Analisador Sintático – Conjunto Follow

- Conjunto de terminais que **sucedem** o não-terminal nas regras sintáticas.
- Montado para cada não-terminal.
- Deve olhar todos os lugares onde o não-terminal aparece no lado direito das regras.

Ex.:

D → eFa | dC Follow(F) = { a, b }

C → Fb | Ef

Analisador Sintático – Conjunto Follow

Caso seja um não-terminal que sucede, deve-se adicionar o conjunto **first** deste no follow.

Ex.:

D → eFa | dC Follow(F) = { a, b } U First(D) = { a, b, e, d }

C → Fb | Ef

E → eFD

Analisador Sintático – Conjunto Follow

Caso o não-terminal seja o último símbolo na regra, deve-se adicionar o conjunto **follow** do não-terminal na esquerda.

Ex.:

D → eFa | dC Follow(F) = { a, b } U First(D) = { a, b, e, d }

C → Fb | Ef Follow(E) = { f } U Follow(F) = { a, b, e, d, f }

E → eFD

F → dE

Analisador Sintático – Algoritmo

Continuar compilação com **Sincronização:**

```
função tratarErro(Token t, Símbolo NT){
        mensagem de erro.
        faça
                avança(t)
        enquanto t não está no Follow(NT)
}
```

Analisador Sintático – Função

E → TE’

E’ → +TE’ | ε

T → FT’

T’ → *FT’ | ε

F → (E) | a

```
função F(Token t) {
        Se t está no First(‘(‘) então
                tratarTerminal( t , ‘(‘ );
                E(t);
                tratarTerminal( t , ‘)‘ );
        Senão se t está no First(a) então
                tratarTerminal( t , ‘a‘ );
        Senão tratarErro();
}
```

Analisador Sintático – Função

E → TE’

E’ → +TE’ | ε

T → FT’

T’ → *FT’ | ε

F → (E) | a

```
função E(Token t) {
        Se t está no First(T) então
                T(t);
                Elinha(t);
        Senão
                tratarErro();
}
```

Analisador Sintático – Função

E → TE’

E’ → +TE’ | ε

T → FT’

T’ → *FT’ | ε

F → (E) | a

```
função Elinha(Token t) {
        Se t é o ‘+’ então
                tratarTerminal( t , ‘+’ );
                T(t);
                Elinha(t);
}
```