Análise Sintática
Métodos Top Down

Prof. Leandro I. Pinto

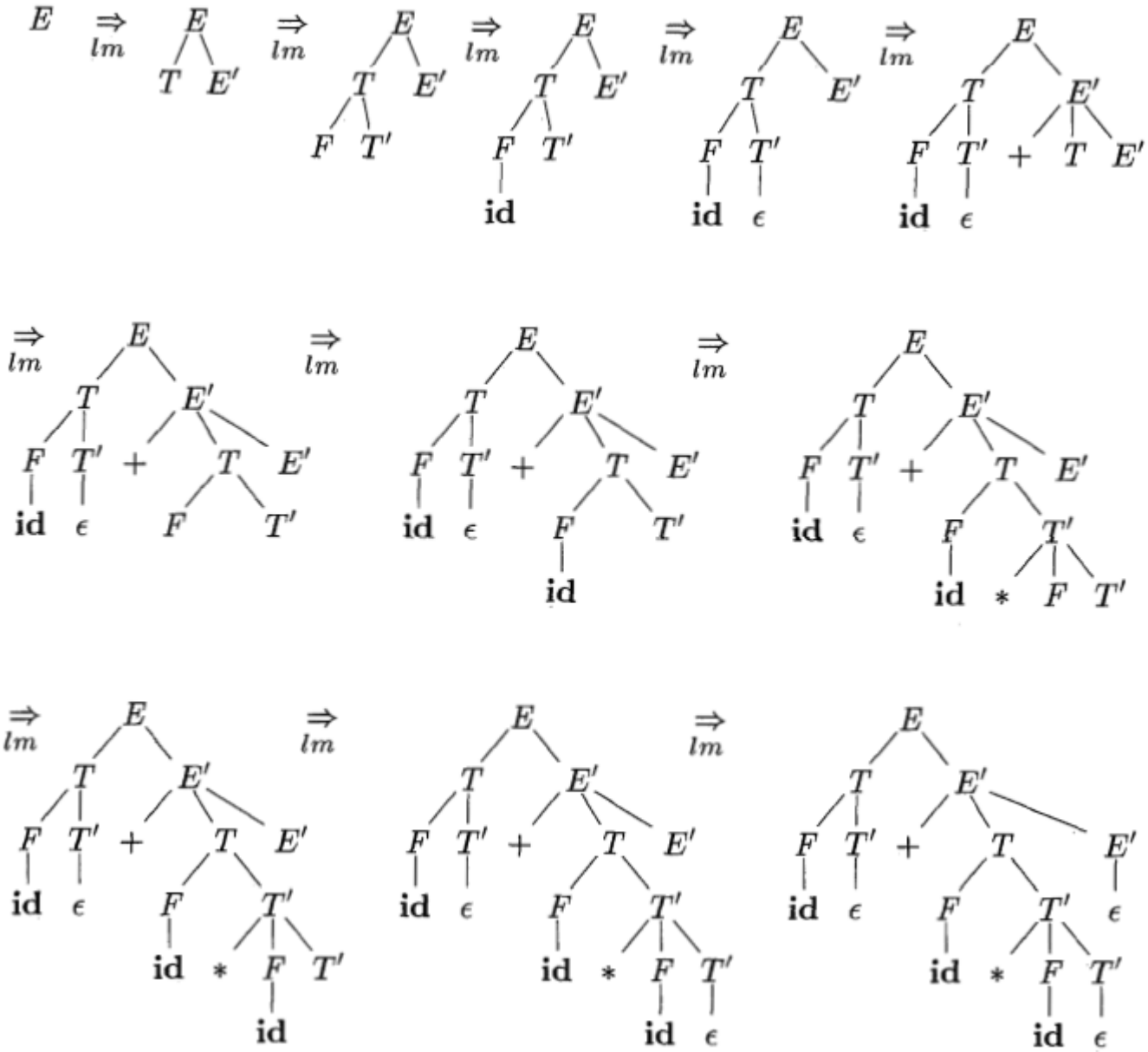
Análise Sintática Descendente

- Constrói a árvore de derivação de cima para baixo;
- Produz a derivação mais a esquerda para a cadeia de entrada;
- O problema é escolher a produção correta;
  - Resolve-se examinando 1 ou mais elementos da cadeia adiante;
  - Geralmente examina-se 1 a frente;
- A classe de gramáticas para as quais podemos construir analisadores preditivos examinando k símbolos adiante as vezes é chamada de classe LL(k)

Análise Sintática Descendente

$$
\begin{aligned}
E &\rightarrow T\,E' \\
E' &\rightarrow +T\,E' \mid \epsilon \\
T &\rightarrow F\,T' \\
T' &\rightarrow *\,F\,T' \mid \epsilon \\
F &\rightarrow (\,E\,) \mid \mathbf{id}
\end{aligned}
$$

Análise Sintática Descendente



Análise Sintática de Descida Recursiva

- Chama-se um método para cada não-terminal;
  - Se encontrar outro não terminal, chama-se o método deste;
- Pode exigir retrocesso;
  - Retorna e escolhe outra produção se a anterior não deu certo;
  - Faz repetidas leituras da entrada;

Análise Sintática de Descida Recursiva

```
     void A() {
1)       Escolha uma produção-A, A → X1 X2 ··· Xk;
2)       for ( i = 1 até k ) {
3)               if ( Xi é um não-terminal )
4)                       ativa procedimento Xi();
5)               else if ( Xi igual ao símbolo de entrada a )
6)                       avance na entrada para o próximo símbolo terminal;
7)               else /* ocorreu um erro */;
         }
     }
```

FIRST e FOLLOW

- A construção de analisadores descendentes é auxiliada por duas funções: FIRST e FOLLOW;
- Nos permite escolher qual produção aplicar durante a análise;
  - Com base na próxima entrada;
  - FOLLOW ajuda na recuperação de erros;

FIRST

- FIRST(X) retorna os símbolos terminais que podem ocorrer por primeiro ao derivar X (inclusive ϵ);

$$
\begin{aligned}
E &\rightarrow T\,E' \\
E' &\rightarrow +\,T\,E' \mid \epsilon \\
T &\rightarrow F\,T' \\
T' &\rightarrow *\,F\,T' \mid \epsilon \\
F &\rightarrow (\,E\,) \mid \mathbf{id}
\end{aligned}
$$

```
FIRST(F)=FIRST(T)=FIRST(E)={(,id}
FIRST(E')={+,e}
FIRST(T')={*,e}
FOLLOW(E)=FOLLOW(E')={),$}
FOLLOW(T)=FOLLOW(T')={+,),$}
FOLLOW(F)={+,*,),$}
```

FOLLOW

- FOLLOW(X) retorna os símbolos terminais que podem ocorrer por primeiro a direita de X
  - Para obter FOLLOW, coloca-se um símbolo $ no final da cadeia a ser reconhecida para representar fim de entrada (não faz parte da gramática);
  - Se X é não-terminal mais a direita em alguma produção então $ está em FOLLOW(X);
  - Se $A \rightarrow \alpha X \beta$, então tudo em $FIRST(\beta)$ está em FOLLOW(B), exceto o ϵ.
  - Se houver $A \rightarrow \alpha B$ ou $A \rightarrow \alpha B \beta$ onde $FIRST(\beta)$ contem ϵ, então inclua FOLLOW(A) em FOLLOW(B).

Gramáticas LL(1)

- Analisadores de descida que não precisam de retrocesso podem ser construídos para a classe de gramáticas LL(1);
  - L: Cadeia de entrada analisada da esquerda para a direita; L: Derivação mais a esquerda; O 1 é pelo uso de um símbolo a frente para prever a derivação;
- Reconhece a maioria das construções das linguagens de programação;
- Escrever uma gramática adequada não é simples;
  - Nenhuma gramática com recursão a esquerda ou ambígua pode ser LL(1);

Gramáticas LL(1)

- Uma gramática G é LL(1) se e somente se, sempre que $A \rightarrow \alpha | \beta$ forem duas produções distintas de G e as seguintes condições forem verdadeiras:
  - Para um terminal a, tanto α quanto β não derivam cadeias começando com a.
  - No máximo um dos dois, α e β, pode derivar a cadeia vazia;
  - Se $\beta \overset{*}{\Rightarrow} \epsilon$, então α não deriva nenhuma cadeia começando com um terminal em FOLLOW(A). Se $\alpha \overset{*}{\Rightarrow} \epsilon$, então β não deriva nenhuma cadeia começando com um terminal em FOLLOW(A).

Tabela de reconhecimento sintático preditivo

Para cada produção $A \rightarrow \alpha$, faça:

1. Para cada terminal a em $FIRST(\alpha)$, inclua $A \rightarrow \alpha$ em $M[A, a]$
2. Se $\epsilon \in FIRST(\alpha)$, inclua $A \rightarrow \alpha$ em $M[A, b]$ para cada terminal b em $FOLLOW(A)$. Se $\epsilon \in FIRST(\alpha)$ e $\$ \in FOLLOW(A)$, acrescente também $A \rightarrow a$ em $M[A, \$]$.

$$
\begin{array}{lcl}
E & \rightarrow & T\,E' \\
E' & \rightarrow & +T\,E' \mid \epsilon \\
T & \rightarrow & F\,T' \\
T' & \rightarrow & *\,F\,T' \mid \epsilon \\
F & \rightarrow & (\,E\,) \mid \mathbf{id}
\end{array}
$$

Qual a tabela?

$$
\begin{aligned}
E &\rightarrow T\,E' \\
E' &\rightarrow +T\,E' \mid \epsilon \\
T &\rightarrow F\,T' \\
T' &\rightarrow *\,F\,T' \mid \epsilon \\
F &\rightarrow (\,E\,) \mid \mathbf{id}
\end{aligned}
$$

```
FIRST(TE')     = {(,id}
FIRST(+TE')    = {+}
FOLLOW(E')     = {),$}
FIRST(FT')     = {(,id}
FIRST(*FT')    = {*}
FOLLOW(T')     = {+,),$}
FIRST(E)       = {(,id}
```

	+	*	(	)	id	$
E						
E'						
T						
T'						
F						

Qual a tabela?

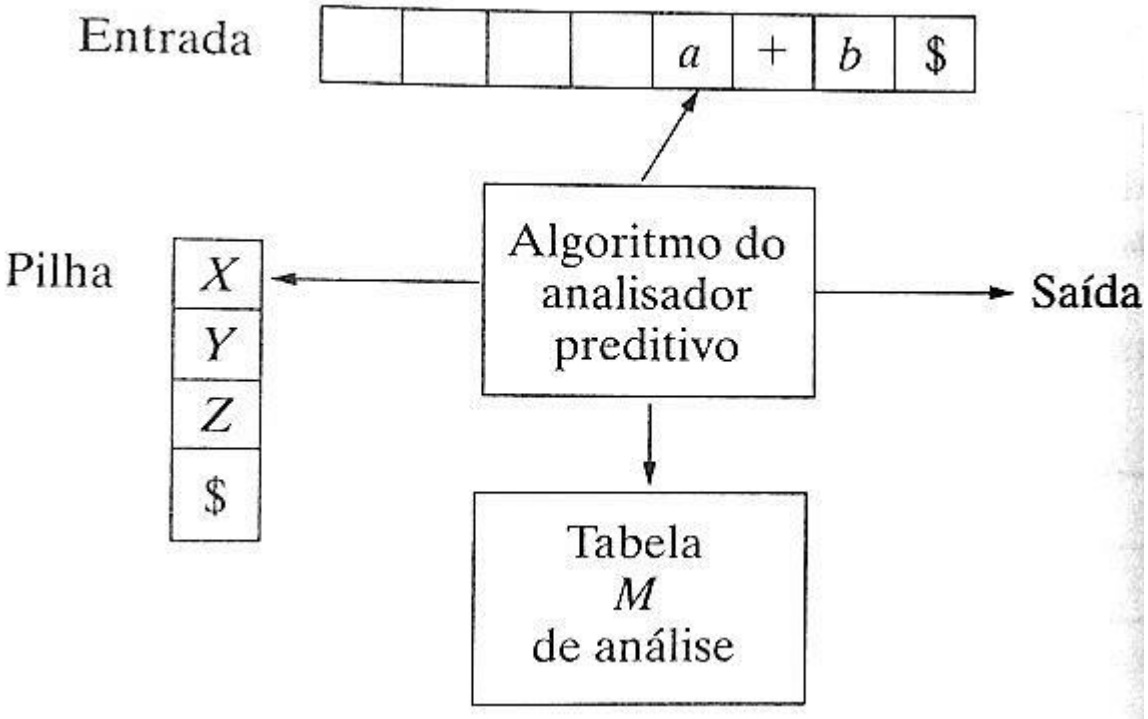
$$
\begin{aligned}
E &\rightarrow T\,E' \\
E' &\rightarrow +T\,E' \mid \epsilon \\
T &\rightarrow F\,T' \\
T' &\rightarrow *\,F\,T' \mid \epsilon \\
F &\rightarrow (\,E\,) \mid \mathbf{id}
\end{aligned}
$$

```
FIRST(TE’)     = {(,id}
FIRST(+TE’)    = {+}
FOLLOW(E’)     = {),$}
FIRST(FT’)     = {(,id}
FIRST(*FT’)    = {*}
FOLLOW(T’)     = {+,),$}
FIRST(E)       = {(,id}
```

	+	*	(	)	id	$
E			E -> T E'		E -> T E'	
E'	E' -> + T E'			E' -> ''		E' -> ''
T			T -> F T'		T -> F T'	
T'	T' -> ''	T' -> * F T'		T' -> ''		T' -> ''
F			F -> (E)		F -> id	

Analisador sintático sem recursão

- Mantem uma pilha explicitamente, em vez de implicitamente via chamadas recursivas.



Analisador sintático sem recursão

```
define ip para que aponte para o primeiro símbolo de w;
define X para ser o símbolo no topo da pilha;
while ( X ≠ $ ){ /* pilha não está vazia */
        if ( X é a ) desempilha e avança ip;
        else if ( X é um terminal ) error();
        else if ( M[X, a] é uma entrada de erro ) error();
        else if ( M[X, a] = X → Y1 Y2 ··· Yk ) {
                imprime a produção X → Y1 Y2 ··· Yk;
                desempilha X;
                empilha Yk , Yk−1 ,... , Y1 na pilha, com Y1 no topo;
        }
        define X como o símbolo no topo da pilha;
}
```

Referências

- [Ref 1] AHO, Alfred V et al. (). Compiladores: princípios, técnicas e ferramentas. 2. ed. São Paulo: Pearson/Addison Wesley, 2007. 634 p. ISBN 9788588639249 (broch.).