UDESC

Compiladores

Prof. Ricardo

ricardo.martins@udesc.br

UDESC

Analisador Sintático

Obtém uma sequência de *tokens* fornecida pelo analisador léxico e verifica se a mesma pode ser gerada pela gramática.

Os métodos de análise sintática comumente usados em compiladores são classificados como:

- Métodos *top-down* (descendente).
- Métodos *bottom-up* (ascendente).

Os métodos eficientes, tanto *top-down* quanto *bottom-up*, trabalham com subconjuntos das gramáticas livres de contexto.

UDESC

Métodos *top-down*

Podem ser vistos como a tentativa de encontrar a derivação mais a esquerda para uma cadeia de entrada. Partindo do símbolo inicial da gramática são aplicadas sucessivas derivações tentado produzir a cadeia que se deseja reconhecer.

Exemplos:

- Método descendente recursivo
- Método LL(1)

Método Descendente Recursivo

<expr> → + <expr> <expr>

| - <expr> <expr>

<expr> → <const>

<const> → 0 | 1| 2 | 3| 4 | 5 | 6 | 7 | 8| 9

Método Descendente Recursivo

```
void cons()
{
        if (isdigit(lookahead))
            nextToken();
        else
            erro("Erro sintático");
}
void expr ()
{
        if (lookahead == '+' || lookahead == '-')
        {
            nextToken();  expr(); expr();
        }
        else
            cons(); // Todos os erros serão tratados em cons()...
}
```

Analisadores Sintáticos Preditivos

Escrevendo a gramática de forma cuidadosa, podemos obter uma gramática processável por um analisador sintático que não necessite de retrocesso. Dado um símbolo de entrada *a* e um não terminal *A*, a ser expandido, a gramática deve possuir uma única produção que leve ao reconhecimento da cadeia iniciada com *a*.

Analisadores sintáticos não preditivos (ou não deterministas) necessitam de retrocesso (*backtraking*) e, em geral, são ineficientes.

Fatoração à Esquerda

As vezes é necessário fazer alterações na gramática que possibilitem a implementação de um reconhecedor preditivo:

<cmd> → if <expr> then <cmd> else <cmd>

| if <expr> then <cmd>

<cmd> → if <expr> then <cmd> <cmd'>

<cmd'> → else <cmd>

| ε

UDESC

Fatoração à Esquerda

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m$$

$$A \rightarrow \alpha A' \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m$$

$$A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

Eliminação da Recursividade à Esquerda

$E \rightarrow E + T$

$\quad | \; E - T$

$\quad | \; T$

$T \rightarrow c$

$\quad | \; (E)$

E

$\Rightarrow E - T$

$\Rightarrow E + T - T$

$\Rightarrow T + T - T$

$^{*}\Rightarrow c + c - c$

Eliminação da Recursividade à Esquerda

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$$

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A'$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \varepsilon$$

$$E \rightarrow E + T \mid E - T \mid T$$
$$T \rightarrow c \mid (E)$$

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' \mid -TE' \mid \varepsilon$$
$$T \rightarrow c \mid (E)$$

Análise Sintática Preditiva não Recursiva LL(1)

$E \rightarrow TE'$

$E' \rightarrow +TE'$

$\quad | \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow * FT'$

$\quad | \varepsilon$

$F \rightarrow c$

$\quad | (E)$

Não Terminal	c	+	*	(	)	#
E	TE’			TE’		
E’		+TE’			ε	ε
T	FT’			FT’		
T’		ε	* FT’		ε	ε
F	c			(E)		

E

$\Rightarrow$ TE’

$\Rightarrow$ FT’E’

$\Rightarrow$ cT’E’

$\Rightarrow$ cE’

$\Rightarrow$ c+TE’

$\Rightarrow$ c+FT’E’

$\Rightarrow$ c+cT’E’

$\Rightarrow$ c+c*FT’E’

$\Rightarrow$ c+c*cT’E’

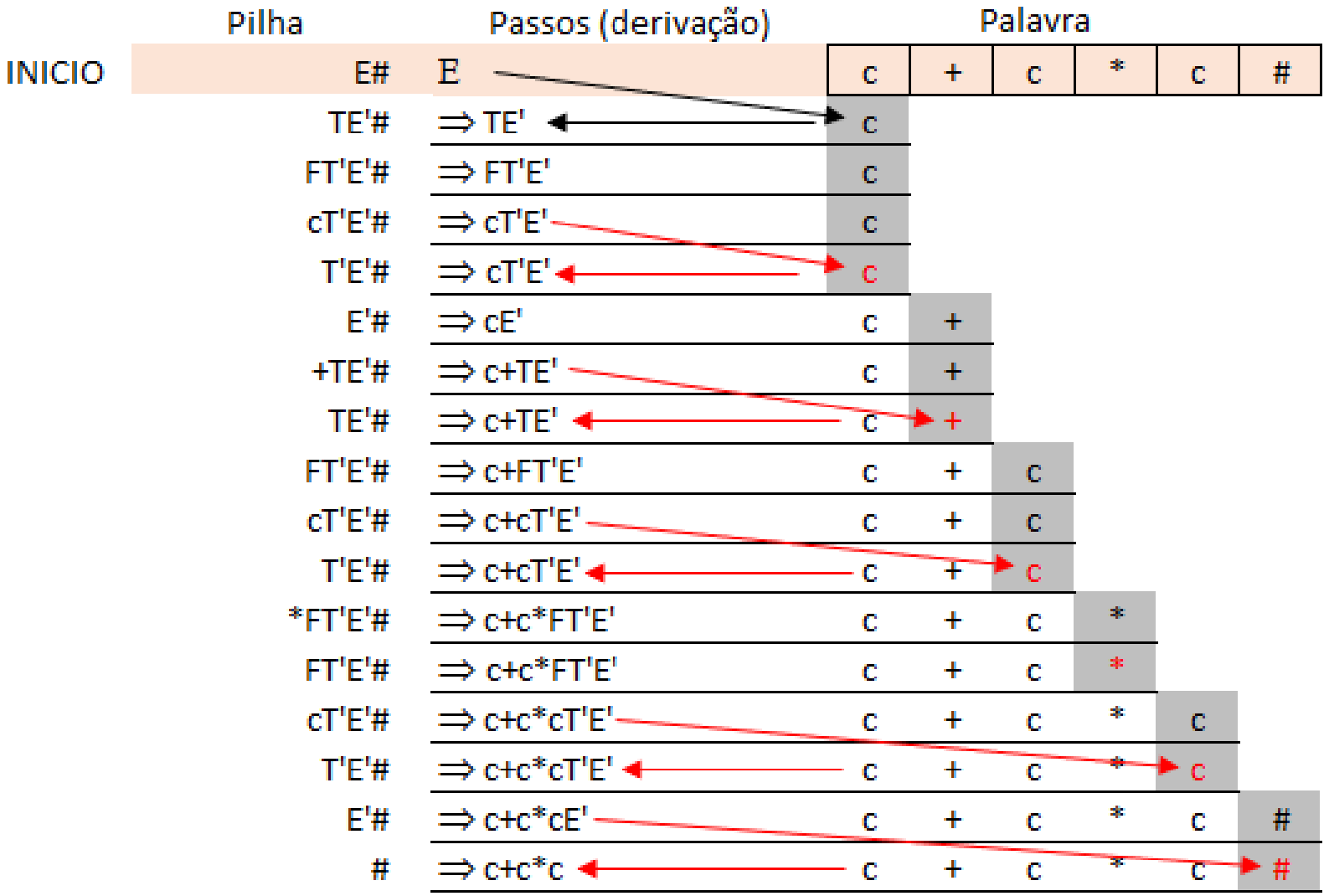
$\Rightarrow$ c+c*cE’

$\Rightarrow$ c+c*c

Analisador Sintático LL(1)

Considerando *w* a cadeia de entrada.
Empilhar #, Empilhar o símbolo inicial da gramática.
Faça *p* apontar para o primeiro símbolo de w#
Repetir
    Seja *X* o símbolo no topo da pilha e *a* o símbolo apontado por *p*;
    Se *X* for um terminal ou # então
        Se *X* = *a* então
            Remover *X* da pilha e avançar *p*;
        Senão erro.
    Senão /* *X* não é um terminal */
        Se M[*X*, *a*] = $X \rightarrow Y_1 Y_2 ... Y_k$ então
            Remover *X* da Pilha
            Empilhar $Y_k ... Y_2 Y_1$
        Senão
            erro
Até que *X* = #

Analisador Sintático LL(1)



UDESC

Analisador Sintático LL(1)

Uma gramática cuja tabela não possui entradas multiplamente definidas é dita LL(1). O primeiro L indica a varredura da cadeia de entrada, que e feita da esquerda para a direita (*left to right*) o segundo L indica que são aplicadas derivações mais a esquerda (*left linear*). O número 1 indica que é necessário apenas um símbolo para decidir qual produção aplicar (*1 lookahead*).

Construção da Tabela LL(1)

A construção de um analisador sintático preditivo e auxiliada por duas funções associadas a gramática: PRIMEIROS e SEGUINTES (FIRST e FOLLOW).

Seja α uma cadeia qualquer de símbolos gramaticais, PRIMEIROS(α) representa o conjunto de símbolos terminais que começam as cadeias derivadas a partir de α.

Se $\alpha \ ^*\Rightarrow \varepsilon$, então ε também é um elemento de PRIMEIROS(α).

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow c$

Caso 1:

$E \Rightarrow T$
$\Rightarrow F$
$\Rightarrow (E)$

Caso 2:

$E \Rightarrow T$
$\Rightarrow F$
$\Rightarrow c$

Logo:

PRIMEIROS(E) = { (, c }

Construção da Tabela LL(1)

SEGUINTES(A), para um não terminal A, é o conjunto de terminais *a* tais que existe uma derivação $S \overset{*}{\Rightarrow} \alpha A a \beta$, para alguma cadeia α e alguma cadeia β, onde S é o símbolo inicial da gramática. Ou seja o conjunto de símbolos que podem ocorrer após o não terminal A em alguma forma sentencial da gramática.

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow c$

SEGUINTES(F) = { +, #, *,) }

$E \Rightarrow E + T$
$\Rightarrow T + T$
$\Rightarrow F + T$

$E \Rightarrow E + T$
$\Rightarrow E + T$
$\Rightarrow E + F\ \#$

$E \Rightarrow T$
$\Rightarrow T * F$
$\Rightarrow F * F$

$E \Rightarrow T$
$\Rightarrow F$
$\Rightarrow (E)$
$\Rightarrow (E + T)$
$\Rightarrow (E + F)$

UDESC

Construção da Tabela LL(1)

Entrada: Gramática

Saída: Tabela *M*

Para cada produção $A \rightarrow \alpha$ da gramática faça:

- Para cada terminal *a* em PRIMEIROS(α), adicione $A \rightarrow \alpha$ em M[*A*, *a*].
- Se ε estiver em PRIMEIROS(α), adicione $A \rightarrow \alpha$ em M[*A*, *b*], para cada terminal *b* em SEGUINTES(*A*).

Cada entrada indefinida em *M* indica uma situação de erro.

Construção da Tabela LL(1)

E →	TE’	→ PRIMEIROS (TE’)	= {c, (}
E’ →	+TE’	→ PRIMEIROS (+TE’)	= {+ }
|	ε	→ SEGUINTES (E’)	= {), # }
T →	FT’	→ PRIMEIROS (FT’)	= {c, (}
T’ →	* FT’	→ PRIMEIROS (*FT’)	= { * }
|	ε	→ SEGUINTES (T’)	= { +,), # }
F →	c	→ PRIMEIROS (c)	= {c}
|	(E)	→ PRIMEIROS((E))	= { (}

Não Terminal	c	+	*	(	)	#
E	TE’			TE’		
E’		+TE’			ε	ε
T	FT’			FT’		
T’		ε	* FT’		ε	ε
F	c			(E)		