



Judson Santos Santiago

# Expansão do Tradutor

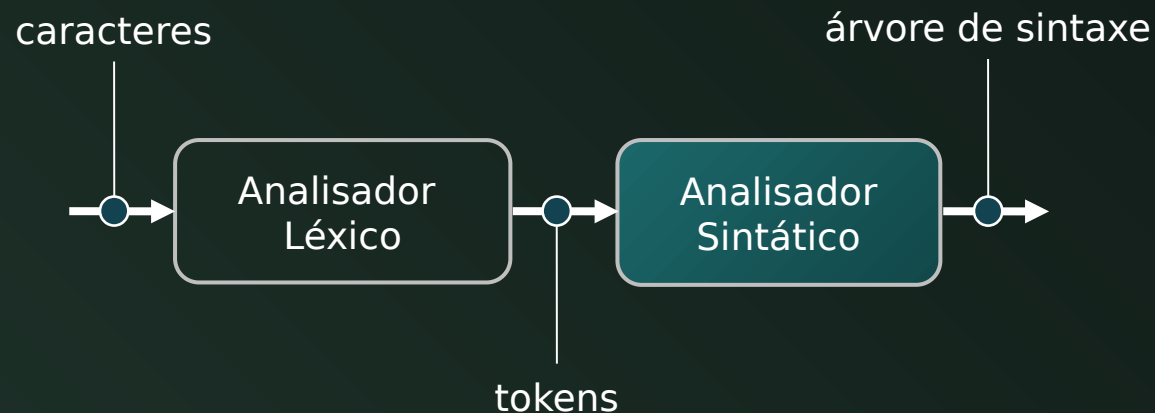
Compiladores

# Introdução

- Um esquema de tradução dirigido por sintaxe pode ser usado para construir um tradutor:
  - Usando análise sintática descendente  
percorre a árvore de derivação de cima para baixo
  - Através de um analisador sintático preditivo  
evita o processo de tentativa e erro
  - Trabalhando com uma gramática apropriada  
sem recursão à esquerda e usando conjuntos FIRST disjuntos
- Essa técnica é ideal para a construção manual

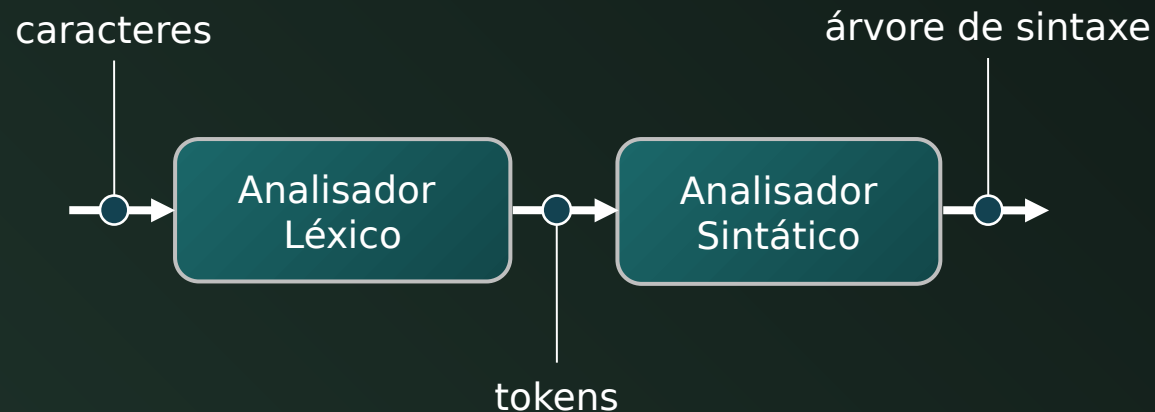
# Introdução

- O tradutor implementado anteriormente estava incompleto
  - Trabalhava diretamente com os caracteres individuais da entrada:
    - Caracteres estranhos (ex.: espaços, tabulações, etc.) provocavam erros
    - Números com mais de um dígito (ex.: 25, 84, etc.) também






# Introdução

- O **analizador léxico** implementado permite processar:
  - Números com qualquer quantidade de dígitos
  - Identificadores e palavras-chave
  - Espaços em branco



# Gramática

- Vamos **expandir o tradutor** usando um analisador léxico e o novo esquema de tradução mostrado abaixo:

```
expr  expr + term { print('+') }  
    | expr - term { print('-') }  
    | term  
  
term  term * fact { print('*') }  
    | term / fact { print('/') }  
    | fact  
  
fact  (expr)  
    | num { print(num.valor) }  
    | id { print(id.nome) }
```

O **esquema de tradução** considera operações de **multiplicação** e **divisão**.


Ele também trabalha com **tokens** para números (**num**)

e identificadores (**id**).

# Recursão à Esquerda

- As **recursões à esquerda** precisam ser eliminadas

```

expr  expr + term { print('+') }
    | expr - term { print('-') }
    | term
  
```

$A = \text{expr}$


$\rightarrow = + \text{ term } \{ \text{print}('+') \}$

$\uparrow = - \text{ term } \{ \text{print}('-') \}$

$+$  = term




```

expr  term add
add  + term { print('+') } add
    | - term { print('-') } add
    |  $\epsilon$ 
  
```

# Recursão à Esquerda

- As **recursões à esquerda** precisam ser eliminadas

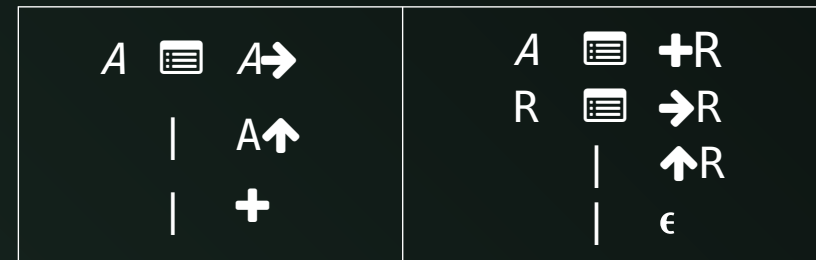
```
term  term * fact { print('*') }  
| term / fact { print('/') }  
| fact
```



$A = \text{term}$

$\rightarrow = * \text{ fact } \{ \text{print}('*') \}$

$\uparrow = / \text{ fact } \{ \text{print}('/') \}$




$+$  = fact








```
term  fact mult  
mult  * fact { print('*') } mult  
| / fact { print('/') } mult  
|  $\epsilon$ 
```

# Recursão à Esquerda

- Resultado da eliminação:

```
expr  expr + term { print('+') }  
    | expr - term { print('-') }  
    | term  
  
term  term * fact { print('*') }  
    | term / fact { print('/') }  
    | fact  
  
fact  (expr)  
    | num { print(num.valor) }  
    | id { print(id.nome) }
```

## Esquema de Tradução

```
expr  term add  
add  + term { print('+') } add  
    | - term { print('-') } add  
    | €  
  
term  fact mult  
mult  * fact { print('*') } mult  
    | / fact { print('/') } mult  
    | €  
  
fact  (expr)  
    | num { print(num.valor) }  
    | id { print(id.nome) }
```








# Tradutor Dirigido por Sintaxe

- O **lookahead** e a função **match** agora terão que lidar com tokens

```
void expr()
{
    term();

    while(true)
    {
        if (lookahead == <+>)
        {
            match(<+>); term(); print('+');
        }
        else if (lookahead == <->)
        {
            match(<->); term(); print('-');
        }
        else return; // vazio
    }
}
```

## Esquema de Tradução

<i>expr</i>		<i>term add</i>
<i>add</i>		<i>+ term</i> { print('+') } <i>add</i>
		<i>- term</i> { print('-') } <i>add</i>
		$\epsilon$
<i>term</i>		<i>fact mult</i>
<i>mult</i>		<i>* fact</i> { print('*') } <i>mult</i>
		<i>/ fact</i> { print('/') } <i>mult</i>
		$\epsilon$
<i>fact</i>		<i>(expr)</i>
		<i>num</i> { print(num.valor) }
		<i>id</i> { print(id.nome) }






# Tradutor Dirigido por Sintaxe

- A função *term* possui a **mesma estrutura** da função *expr*

```
void term()
{
    fact();

    while(true)
    {
        if (lookahead == <*>)
        {
            match(<*>); fact(); print('*');
        }
        else if (lookahead == </>)
        {
            match(</>); fact(); print('/');
        }
        else return; // vazio
    }
}
```

## Esquema de Tradução






<i>expr</i>		<i>term add</i>
<i>add</i>		<i>+</i> <i>term</i> { print('+') } <i>add</i>
		<i>-</i> <i>term</i> { print('-') } <i>add</i>
		$\epsilon$
<i>term</i>		<i>fact mult</i>
<i>mult</i>		<i>*</i> <i>fact</i> { print('*') } <i>mult</i>
		<i>/</i> <i>fact</i> { print('/') } <i>mult</i>
		$\epsilon$
<i>fact</i>		<i>(expr)</i>
		<i>num</i> { print(num.valor) }
		<i>id</i> { print(id.nome) }

# Tradutor Dirigido por Sintaxe

- Os tokens **num** e **id** possuem **atributos**

```
void fact()
{
    if (lookahead == <(>)
    {
        match(<(>); expr(); match(<(>);
    }
    else if (lookahead == <num>)
    {
        match(<num>); print(num.valor);
    }
    else if (lookahead == <id>)
    {
        match(<id>); print(id.nome);
    }
    else
        print("syntax error");
}
```

## Esquema de Tradução

<i>expr</i>		<i>term add</i>
<i>add</i>		<i>+</i> <i>term</i> { print('+') } <i>add</i>
		<i>-</i> <i>term</i> { print('-') } <i>add</i>
		$\epsilon$
<i>term</i>		<i>fact mult</i>
<i>mult</i>		<i>*</i> <i>fact</i> { print('*') } <i>mult</i>
		<i>/</i> <i>fact</i> { print('/') } <i>mult</i>
		$\epsilon$
<i>fact</i>		<i>(expr)</i>
		<b>num</b> { print(num.valor) }
		<b>id</b> { print(id.nome) }

# Integração do Analisador Léxico

- O **analisador léxico** será responsável por:
  - Ler caracteres da entrada
  - Retornar tokens, que podem conter atributos
  - Ignorar espaços, tabulações e saltos de linha
- O **analisador sintático** sofrerá algumas alterações:
  - Deverá usar a **função scan**, no lugar de ler caracteres da entrada
  - O símbolo **lookahead** será um token e não um caractere
  - A **função match** deve casar tokens

# Resumo

- Um **tradutor** mais completo pode ser obtido:
  1. Escrevendo uma **gramática** para a linguagem fonte
  2. Criando um **esquema de tradução** para a linguagem destino
  3. Adaptando o esquema para um analisador preditivo:
    - Removendo recursão à esquerda
    - Garantindo que os conjuntos FIRST sejam disjuntos
  4. Implementando um **analisador léxico**
  5. Implementando um **analisador sintático preditivo**