



Judson Santos Santiago

Análise Sintática

Compiladores

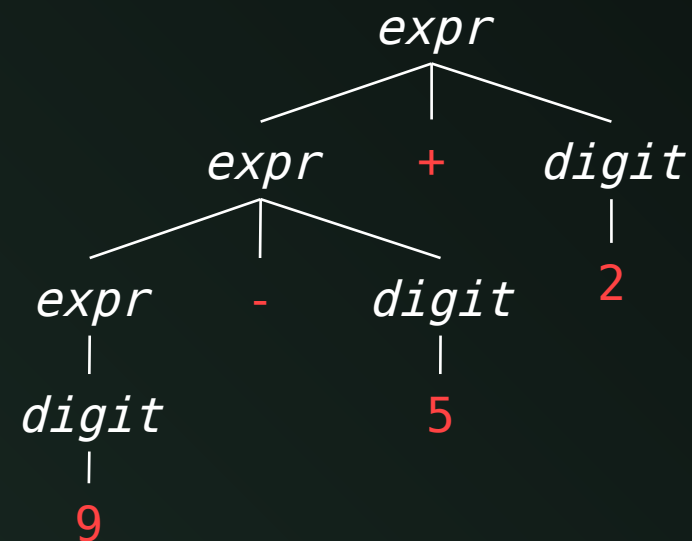
Introdução

- Uma **gramática** é formada por um conjunto de produções que descrevem as construções das linguagens
 - Sentenças são válidas se for possível derivá-las

Gramática para
expressões
aritméticas em
notação infixada

expr ::= *expr* +
 digit
 | *expr* -
 digit
 | *digit*
digit ::= 0 | ... | 9

Derivação de
9-5+2

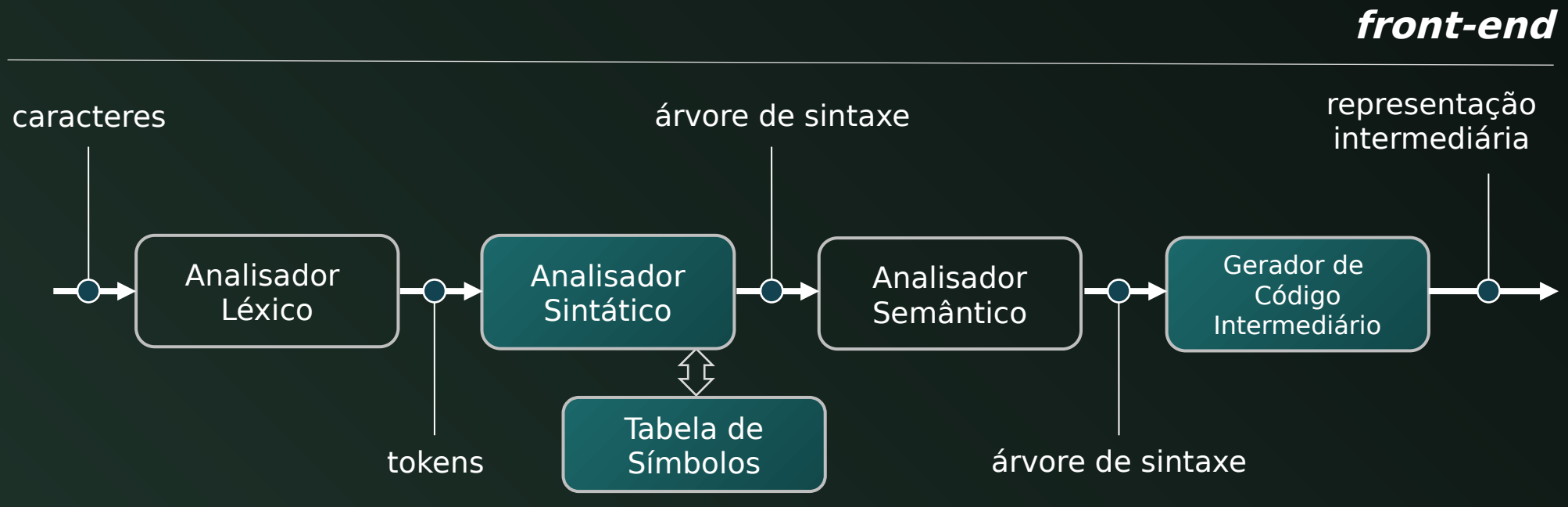


Introdução

- A **tradução dirigida por sintaxe** permite construir um tradutor usando uma busca em profundidade em uma **árvore de derivação**:
 - Anotada com **atributos**
(definição dirigida por sintaxe)
 - Modificada por **ações semânticas**
(esquema de tradução dirigido por sintaxe)
- Mas a tradução foi feita **sem realizar nenhuma análise**
 - Os processadores de linguagem idealmente realizam:
 - Análise léxica, sintática e semântica

Introdução

- Um **esquema de tradução** pode ser usado para a **análise sintática**
 - A análise léxica e semântica serão omitidas por enquanto

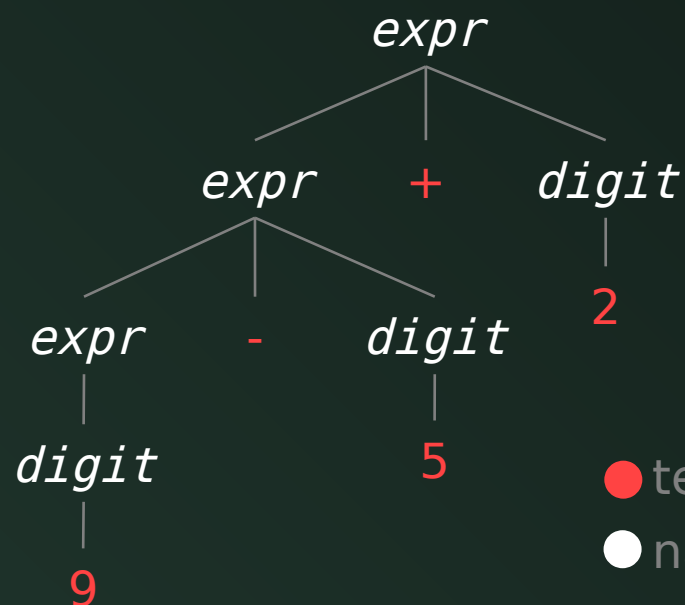


Análise Sintática

- Um analisador sintático pode ser **construído**:
 - **Manualmente** através de um programa em alguma linguagem (C++)
 - Usando uma **ferramenta** apropriada (Yacc, Bison, etc.)
- Para qualquer gramática livre de contexto, existe um analisador sintático capaz de analisar **n símbolos** terminais em **$O(n^3)$**
 - No entanto, **na prática**, as linguagens de programação conseguem ser analisadas **em tempo linear**
 - É feita uma única leitura da esquerda para a direita sobre a entrada

Análise Sintática



- A **análise sintática** é o processo que determina se uma cadeia de terminais (sentença) é gerada por uma gramática



● terminais
● não-terminais

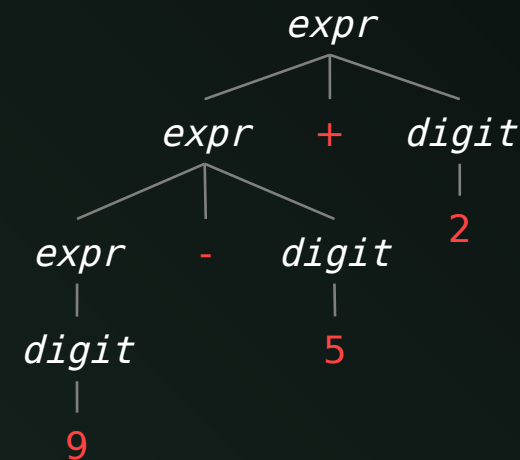
árvore de derivação
para a cadeia
9-5+2

Gramática

<i>expr</i>	 <i>expr</i> + <i>digit</i>	(<i>p</i> ₁)
	<i>expr</i> - <i>digit</i>	(<i>p</i> ₂)
	<i>digit</i>	(<i>p</i> ₃)
<i>digit</i> <i>t</i>	 0 ... 9	(<i>p</i> ₄)


Análise Sintática

- A análise sintática **constrói a árvore de derivação** de uma cadeia
 - A ordem em que os nós são construídos depende do método
 - Analisadores **descendentes**
 - A árvore é construída de cima para baixo (da raiz para as folhas)
 - Popular entre analisadores sintáticos **construídos à mão**
 - Analisadores **ascendentes**
 - A árvore é construída de baixo para cima (das folhas para a raiz)
 - Pode tratar uma classe maior de gramáticas
 - Mais comum para analisadores sintáticos criados por **geradores automáticos**



Análise Sintática Descendente

- Considere a gramática de uma **linguagem simplificada**
 - Com **expr** sendo, excepcionalmente, um símbolo terminal

inst  **expr;**
| **if (expr)** *inst*
| **for (optexpr; optexpr; optexpr)**
 inst

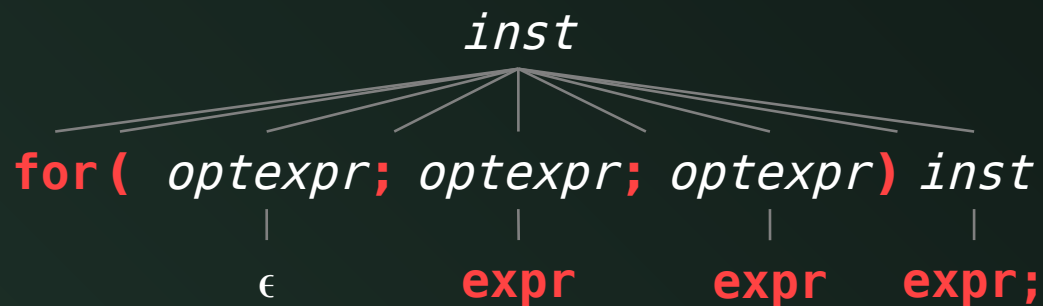
optexpr  **expr**
r
| ϵ

● terminais
● não-terminais
 ϵ vazio

Ex.: **if (expr)** **for (; expr; expr)**
 expr; **expr;**

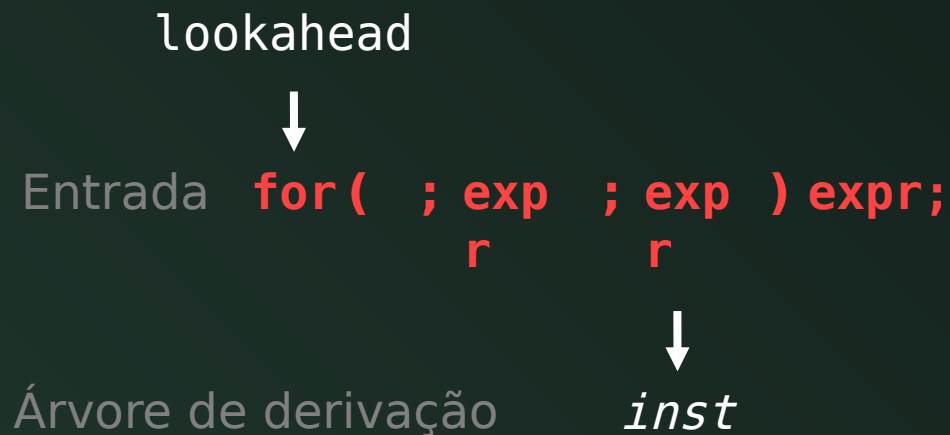
Análise Sintática Descendente

- A **construção descendente** de uma árvore de derivação é feita:
 - Iniciando no símbolo inicial da gramática:
 1. Rotule um nó N com um símbolo não-terminal S
 2. Selecione uma das produções de S e construa nós filhos em N, um para cada símbolo no corpo da produção
 3. Encontre o próximo nó mais a esquerda da árvore, correspondente a um símbolo não-terminal não expandido, e repita o processo



Análise Sintática Descendente

- A **construção da árvore** pode ser feita **em uma única leitura** da sentença de entrada, da esquerda para a direita
 - O símbolo corrente da sentença é chamado de símbolo **lookahead**
 - No início, o símbolo **lookahead** é o primeiro da entrada



inst

```

| expr;
| if (expr) inst
| for (optexpr; optexpr; optexpr)
  inst

```

optexpr

```

| expr
| €

```

Análise Sintática Descendente

- Seleccionando uma das produções de *inst* e construindo nós filhos para os símbolos no corpo da produção




Análise Sintática Descendente

- Encontrando o próximo nó mais a esquerda da árvore correspondente a um não-terminal não expandido



Entrada **for (; exp ; exp) expr;**
 r r

inst  **expr;**
| **if (expr)** *inst*
| **for (optexpr; optexpr; optexpr)**
 inst


Árvore de derivação



optexpr  **expr**
| €

Análise Sintática Descendente



- A **seleção de uma produção** pode envolver **tentativa e erro**
 - Se houver mais de uma produção viável
 - Tentamos a produção e se ela falhar recuamos e tentamos outra

```
inst       expr;  
          | if (expr) inst  
          | for (optexpr; optexpr; optexpr)  
              inst  
          | for (init : container) inst
```

- Esse processo de recuo não é desejável e é possível evitá-lo através de um **analisador sintático preditivo**

Analizador Sintático Preditivo

- Um **analizador sintático preditivo** é um tipo simples de analisador descendente recursivo
 - **Análise descendente recursiva** é um método de análise sintática:
 - Funções recursivas são usadas para processar a entrada
 - Uma função é associada a cada não-terminal da gramática

```
inst       expr;  
          | if (expr) inst  
          | for (optexpr; optexpr;  
                optexpr)  
            inst  
optexp   expr  
r  
          |  $\epsilon$ 
```

Funções:

```
void inst();  
void  
optexpr();
```


Analizador Sintático Preditivo

- As funções **imitam o corpo** de uma produção
 - Os símbolos do corpo são processados da esquerda para a direita:
 - Um **não-terminal**:
 - Se torna uma chamada de função
 - Um **terminal**:
 - Se casar com o símbolo **lookahead** provoca a leitura do próximo símbolo da entrada
 - Se não casar com o símbolo **lookahead** gera um erro de sintaxe

Gramática

```
inst  ::= expr;  
      | if (expr) inst  
      | for (optexpr; optexpr;  
            inst)  
optexpr ::= expr  
r      ::= €
```

Analizador Sintático Preditivo

- Em um analisador sintático preditivo:

- O símbolo ***lookahead*** determina a escolha da próxima produção (de forma não ambígua)



lookahead
↓
Entrada **for (; exp ; exp) expr;**
 r r

```
void inst()
{
    switch(lookahead)
    {
        case expr: ... break;
        case if: ... break;
        case for: ... break;
        default: print("syntax
error");
    }
}
```

Analizador Sintático Preditivo

- Implementação do analisador

Gramática

```
inst       expr;  
          | if (expr) inst  
          | for (optexpr; optexpr;  
                optexpr)  
              inst  
  
optexpr   expr  
r        |  $\epsilon$ 
```

```
void inst()  
{  
    switch(lookahead)  
    {  
        case expr:  
            match(expr); match(';'); break;  
        case if:  
            match(if); match('('); match(expr);  
            match(')'); inst(); break;  
        case for:  
            match(for); match('('); optexpr();  
            match(';'); optexpr(); match(';');  
            optexpr(); match(')'); inst();  
            break;  
        default:  
            print("syntax error");  
    }  
}
```

Analizador Sintático Preditivo

- Implementação do analisador

Gramática

```
inst      ::= expr;  
           | if (expr) inst  
           | for (optexpr; optexpr;  
                 inst  
optexpr ::= expr  
r        |  $\epsilon$ 
```

```
void optexpr()  
{  
    if (lookahead == expr)  
        match(expr);  
}  
  
void match(terminal t)  
{  
    if (t == lookahead)  
        lookahead =  
next_terminal();  
    else  
        print("syntax error");  
}
```

Analizador Sintático Preditivo

- A **sequência de chamadas das funções** define implicitamente uma árvore de derivação para uma cadeia de entrada
 - Pode ser usada para construir a árvore de derivação

Entrada: **for (; exp ; exp) expr;**
 r **r**

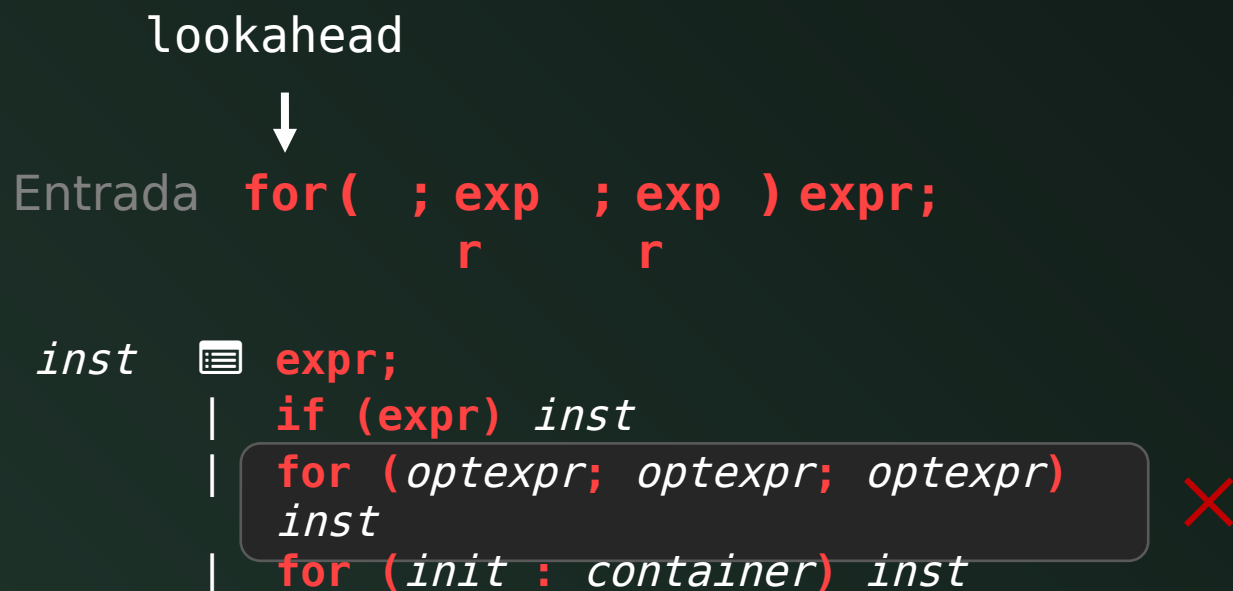
Sequência de chamadas: inst(), optexpr(), optexpr(), optexpr(), inst()

Árvore de derivação:



Analizador Sintático Preditivo

- Para que o analisador preditivo funcione **não podem haver dúvidas** na seleção da produção



O analisador sintático preditivo **não funciona** com **todo** tipo de gramática

Analizador Sintático Preditivo

- O analisar preditivo deve conhecer os primeiros símbolos gerados pelo corpo de uma produção
 - Seja $\text{FIRST}(\rightarrow)$ o conjunto de símbolos que aparecem no início do corpo de uma produção que tem \rightarrow como cabeça:
 1. Ou \rightarrow começa com um terminal e este faz parte de $\text{FIRST}(\rightarrow)$
 2. Ou então começa com um não-terminal cujos corpos da produção começam com terminais que, neste caso, são membros de $\text{FIRST}(\rightarrow)$
 3. Se a produção gerar ϵ , então ele também está em $\text{FIRST}(\rightarrow)$

Analizador Sintático Preditivo

- Encontrando os conjuntos FIRST para a gramática:


```
inst      ::= expr;  
           | if (expr) inst  
           | for (optexpr; optexpr; optexpr)  
             inst  
optexpr ::= expr  
           |  $\epsilon$ 
```


$\text{FIRST}(\textit{inst}) = \{ \text{expr}, \text{if}, \text{for} \}$
 $\text{FIRST}(\textit{optexpr}) = \{ \text{expr}, \epsilon \}$







Analizador Sintático Preditivo

- Os conjuntos **FIRST** precisam ser analisados
 - Se houverem produções do tipo:

Exemplo:

S  
| 

```
inst     if (expr) inst  
        | for (optexpr; optexpr;  
              optexpr)
```

- O analisador preditivo exige que **FIRST()** e **FIRST()** sejam conjuntos disjuntos
 - Permite que o símbolo **lookahead** seja usado para decidir:
 - Se o símbolo **lookahead** estiver em **FIRST()** então  é usado
 - Se o símbolo **lookahead** estiver em **FIRST()** então  é usado

Tradutor Dirigido por Sintaxe

- Retornando para a nossa gramática de **expressões aritméticas**

Gramática para
expressões
aritméticas em
notação infixada

```
expr ::= expr +  
        digit  
        | expr -  
        digit  
        | digit  
digit ::= 0 | ... | 9
```

Esquema de tradução
dirigido por sintaxe

```
expr ::= expr + digit { print('+') }  
        | expr - digit { print('-') }  
        | digit  
digit ::= 0 { print('0') }  
        | 1 { print('1') }  
        | ...  
        | 9 { print('9') }
```

Tradutor Dirigido por Sintaxe


- Assim como um esquema de tradução é formado estendendo-se uma gramática, um **tradutor dirigido por sintaxe** pode ser formado estendendo-se um **analisador preditivo**
 - Constrói-se um analisador preditivo para a gramática
 - Copia-se as ações semânticas para o analisador:
 - A **posição da ação semântica** na função deve ser a mesma da produção:
 - Se ela aparecer depois do símbolo X na produção p, deve ser copiada após a implementação de X na função f

expr 

expr **+** *digit* { print(**'+'**) }

Recursão à Esquerda

- Um analisador sintático preditivo pode ficar em um laço infinito ao tratar produções recursivas à esquerda:
 - O procedimento `expr()` vai chamar a si mesmo na primeira linha de código

$\begin{matrix} \text{exp} \\ r \end{matrix}$  `expr + digit` | `void expr() { expr(); match('+'); digit(); }`

- Uma produção recursiva à esquerda pode ser eliminada reescrevendo-se a produção problemática

`expr`  `expr + digit`
| `digit`

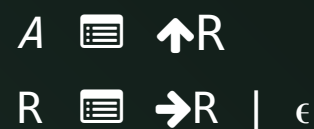
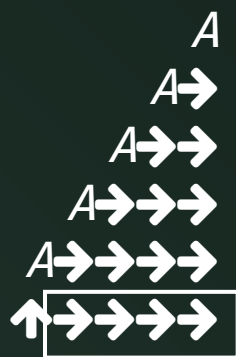
$A = \text{expr}$
 $\alpha = + \text{digit}$
 $\uparrow = \text{digit}$

A  $A \rightarrow$
| \uparrow

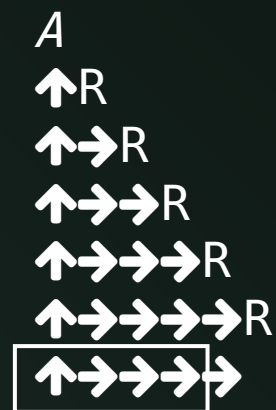
A  $\uparrow R$
 R  $\rightarrow R$ | ϵ

Recursão à Esquerda

- A aplicação da produção:
 - Acumula uma sequência de \rightarrow 's à direita de A
 - Quando A é substituído por \uparrow , produz um \uparrow seguido por zero ou mais \rightarrow 's



O não-terminal R é
recursivo à direita e
produz o mesmo
resultado



Exercícios

1. Construa funções que implementem **analísadores sintáticos preditivos** para as seguintes gramáticas:

$$\begin{array}{l} \text{a) } S \rightarrow +SS \\ \quad \quad | -SS \\ \quad \quad | a \end{array}$$

$$\begin{array}{l} \text{b) } S \rightarrow (S)S \\ \quad \quad | \epsilon \end{array}$$

$$\begin{array}{l} \text{c) } S \rightarrow 0S1 \\ \quad \quad | 01 \end{array}$$

✗ Os conjuntos FIRST não são disjuntos

Exercícios

1. Construa funções que implementem **analísadores sintáticos preditivos**:

a) $S \rightarrow +SS$
 $\quad | -SS$
 $\quad | a$

```
void S()
{
    switch(lookahead)
    {
        case '+': match('+'); S(); S(); break;
        case '-': match('-'); S(); S(); break;
        case 'a': match('a'); break;
        default: print("syntax error");
    }
}
```

Exercícios

1. Construa funções que implementem **analísadores sintáticos preditivos** para as seguintes gramáticas:

b) $S \rightarrow (S)S$
 $\quad \quad \quad | \epsilon$

```
void S()
{
    if (lookahead == '(')
    {
        match('('); S(); match(')'); S();
    }
    else
    {
        // vazio
    }
}
```

Resumo

- Um tradutor pode ser construído a partir de um esquema de tradução dirigido por sintaxe usando:
 - Análise sintática descendente
percorre a árvore de derivação de cima para baixo
 - Evitando o processo de tentativa e erro
através de um analisador sintático preditivo
 - Trabalhando com uma gramática apropriada
sem recursão à esquerda e usando conjuntos FIRST disjuntos
- Essa técnica é apropriada para implementar um tradutor