

# Sintaxe de LP

**Linguagens de Programação  
2017.2018**

*Teresa Gonçalves*  
[tcg@uevora.pt](mailto:tcg@uevora.pt)

Departamento de Informática, ECT-UÉ

# Sumário

**Processo de compilação**

**Léxico, Sintaxe e Semântica da Linguagem**

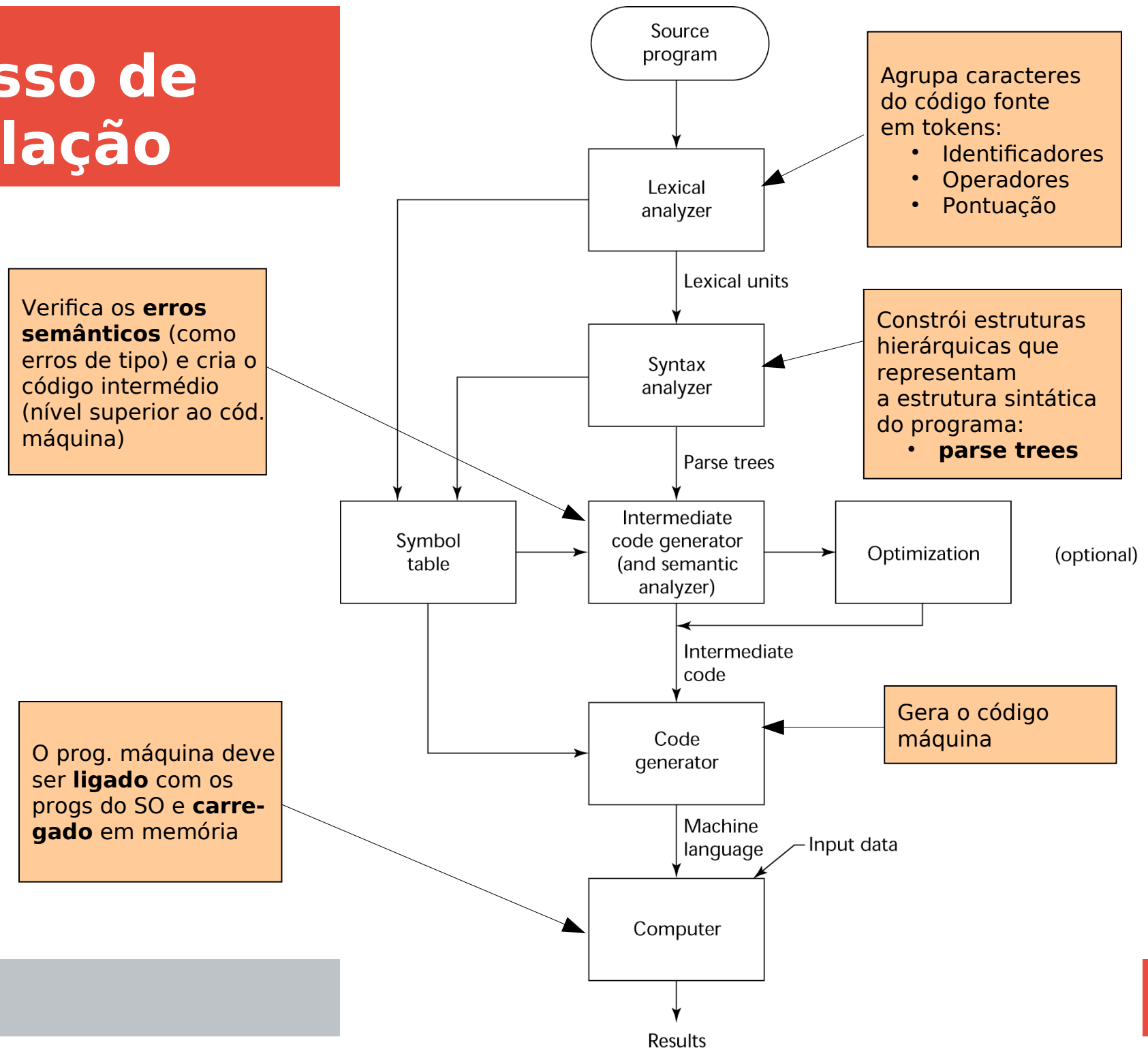
**Gramáticas formais**

**Notação BNF**

**Derivação e árvore sintáctica**

**Ambiguidade**

# Processo de compilação



# Exemplo (2)

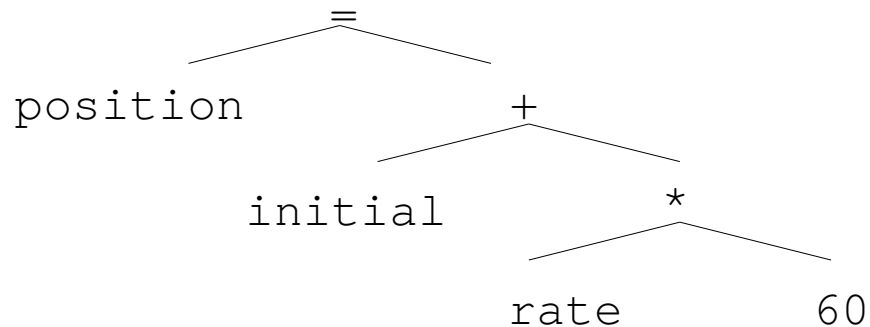
## 1.Input

position = initial + rate\*60

## 2.Análise lexical

position, initial, +, rate, \*, 60

## 3.Análise sintática



## Exemplo (2)

### 4. Análise semântica

adiciona ações semânticas à análise sintática

### 5. Código intermédio

```
temp1 = convert_int_to_double(60)
temp2 = mult(rate, temp1)
temp3 = add(initial, temp2)
position = temp3
```

### 6. Código otimizado

```
temp1 = mult(rate, 60.0)
position = add(initial, temp1)
```

## Exemplo (3)

### 7.Código máquina

```
movf rate, fp2  
mulf #60.0, fp2  
movf initial, fp1  
addf fp2, fp1  
movf fp1, position
```

# Léxico da linguagem

## Identifica

As unidades de construção

Tokens (palavras-chave, operadores, identificadores)

Delimitadores

Formato dos comentários

## Especificação

Informal

Formal

Regras lexicais

# Sintaxe da linguagem

## Indica

Quais as construções corretas na linguagem

Que relações podem existir entre as “unidades de construção”

## Especificação

Informal

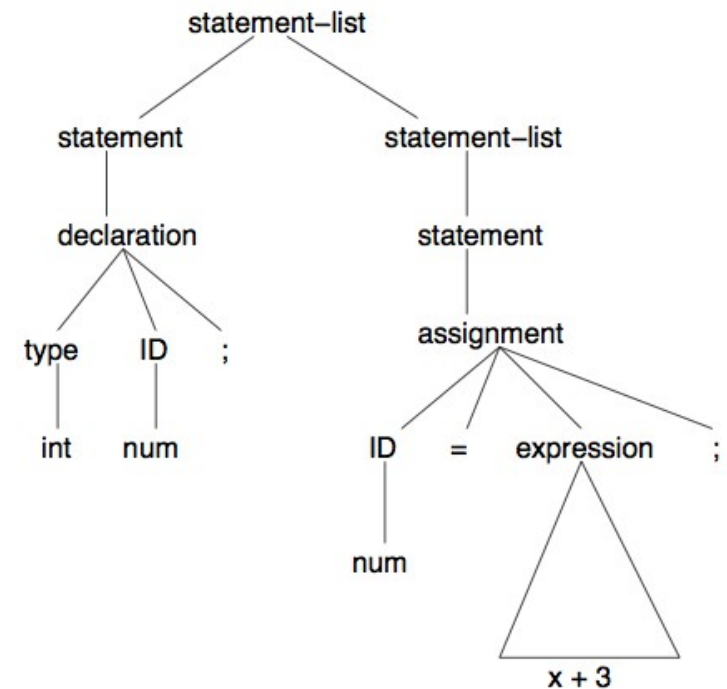
Formal

Existem vários formalismos

## Exemplo

```
int num;
```

```
num = x+3;
```





# Semântica da linguagem

## Indica

Qual o significado das construções corretas

Qual o efeito da execução do programa

## Especificação

Informal

Descrição em língua natural

Formal

Existem vários formalismos

São de utilização difícil

Raramente adotados

# Gramática formal

## Formalismo da sintaxe

Especifica que *strings* são possíveis numa linguagem

## Existem várias categorias

Hierarquia de Chomsky

## Categorias

Estrutura de frase

Sensível ao contexto

Livre de contexto

Regular (podem ser descritas  
por expressões regulares)



mais expressiva

menos expressiva

# Características das categorias

## **Linguagem regular**

Apenas pode descrever estruturas lineares não estruturadas

## **Linguagem livre de contexto**

Pode descrever construções “aninhadas”, ligando pares de itens

# Expressão regular

## Notação

Fecho Kleene: \*

0 ou mais repetições

Fecho Positivo: +

1 ou mais repetições

Alternância: |

Escolha

Outros

“(“ e “)” utilizadas para agrupar

$\varepsilon$  denota a *string* vazia (ou nula)

$\emptyset$  denota a linguagem sem *strings*

## Exemplos

$(0|1)^*$

$(a|b)^*aa(a|b)^*$

# Gramática livre de contexto (GLC)

## Composta por 4 tipos de elementos

Terminal (ou *token*)

Símbolo atômico da linguagem

Não-terminal

Variável utilizada na gramática

Símbolo inicial (um não-terminal especial)

Representa a construção de topo da linguagem

Regra (ou produção)

Especifica uma forma de construir um não terminal a partir de uma sequência de tokens e não terminais

# Notação para GLC

## Forma Backus-Naur (BNF)

Terminal

Escrito entre as regras

Não-terminal

Representado entre “<” e “>”

`<empty>` representa a *string* vazia

Símbolo inicial

Usualmente o primeiro símbolo não-terminal listado

Produção

Não-terminal seguido de “→” e depois a lista de terminais e não terminais que a podem formar

# Extensões ao BNF

**Tornam o BNF mais conciso, mas não mais poderoso**

## Exemplos

`{blah}` : 0 ou mais repetições de `blah`

`[blah]` : denota que `blah` é opcional

`+` : denota 1 ou mais repetições

`num` : denota `num` repetições

`( )` : utilizados para agrupar

# Exemplo

## BNF

```
<expr> → <expr> + <term>  
        | <expr> - <term>  
        | <term>  
  
<term> → <term> * <factor>  
        | <term> / <factor>  
        | <factor>
```

## EBNF

```
<expr> → <term> { (+|-) <term> }  
<term> → <factor> { (*|/) <factor> }
```



# Derivação

## O que é?

Sequência de aplicação de produções

Começando do símbolo inicial, aplicar as regras até existirem apenas símbolos terminais

## Uma frase pertence à linguagem

se e só se existir uma **derivação** para ela

# Árvore sintática

**Mostra a estrutura de uma frase da linguagem**

## **Análise sintática**

Processo de produção da árvore sintática

## **Composição**

A **raíz** é o símbolo inicial

As **folhas** são terminais

Cada nó interno e os seus filhos correspondem, por ordem, ao lado esquerdo e direito de uma produção da gramática

## **Uma frase pertence à linguagem**

se e só se existir uma árvore sintática para ela

# Exemplo

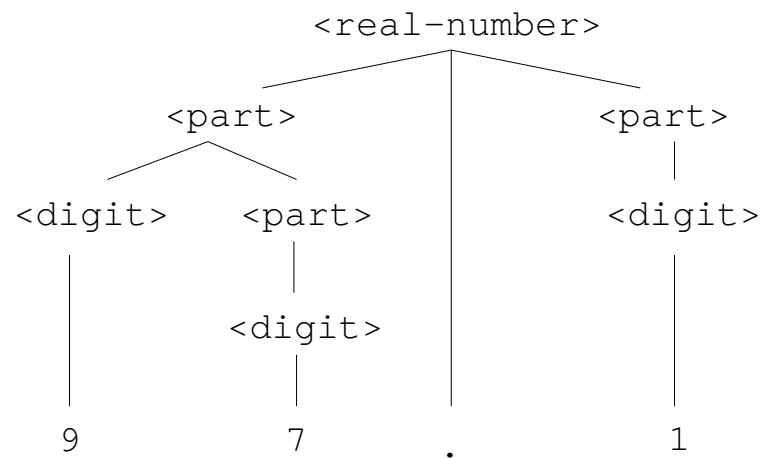
## Gramática

`<real-number> → <part>.<part>`

`<part> → <digit> | <digit><part>`

`<digit> → 0|1|2|3|4|5|6|7|8|9`

**Qual a árvore sintática de 97.1?**



# Técnicas de análise sintática

## ***Bottom-up***

Começa com as folhas (*string* a analisar) e trabalha de forma ascendente de modo à chegar à raíz (símbolo inicial)

## ***Top-down***

Começa pela raíz (símbolo inicial) e trabalha de forma descendente até chegar às folhas (*string* a analisar)

Também designada **análise sintática recursiva descendente**

# Análise top-down

## Funcionamento

Cada não-terminal

é representado por um sub-programa que analisa as strings geradas por esse não-terminal, de acordo com as regras de produção

Quando precisa analisar outro não-terminal

Chama o sub-programa correspondente

## Problema

Requer **não recursividade à esquerda**

Para permitir saber qual o lado direito sem ver à frente (*look-ahead*)

# Retirar a recursividade à esquerda

## Exemplo

`<expr> → <expr> + <term> | <term>`

## Soluções

Colocar a recursividade à direita

`<expr> → <term> + <expr> | <term>`

Remover a recursividade à esquerda

`<expr> → <term> {+ <term>}`

Fatorizar à esquerda

`<expr> → <term> [+ <expr>]`

# Ambiguidade

## Gramática ambígua

Uma gramática é ambígua se e só se gerar uma frase ambígua

## Frase ambígua

Uma frase é ambígua (com respeito a uma gramática) se e só se a gramática permitir gerar duas ou mais árvores sintáticas **distintas**

## Atenção

*Duas derivações distintas não tornam uma frase ambígua!*

# Exemplo 1

## Gramática

`<stm> → <asg-stm> | <loop-stm> | <if-stm>`

`<if-stm> → if <bool-exp> then <stm>  
| if <bool-exp> then <stm> else <stm>`

## Frase

```
if( xodd ) then
if( x==1 ) then
    print "bleep";
else
    print "bloop";
```

Desenhar 2 árvores sintáticas



# Exemplo 2

## Gramática

`<stm> → <asg-stm> | <loop-stm> | <if-stm>`

`<if-stm> → if <bool-exp> then <stm>  
| if <bool-exp> then <stm> else <stm>`

## Frase

```
if( xodd ) then  
    print "bleep";
```

## Desenhar

árvore sintática

2 derivações

# Gramática e linguagem

## **$L(G)$**

linguagem gerada pela gramática  $G$

**$G$  é ambígua se  $L(G)$  contém uma *string* com**

mais que uma árvore sintática, ou

mais que uma derivação mais à esquerda (canónica)

**A gramática de uma linguagem não deve ser ambígua!**

Caso contrário, existem “strings” cuja semântica (significado) não é única!

# Como retirar a ambiguidade?

## Alterar a linguagem

### Como?

Incluir **delimitadores**

Exemplo

```
<stm> → <asg-stm> | <loop-stm> | <if-stm>
```

```
<if-stm> → if <bool-exp> then <stm> fi
```

```
      | if <bool-exp> then <stm> else <stm> fi
```

Impôr **precedência** e **associatividade**

# Precedência e associatividade

## Impor precedência

Introduzir um não-terminal para cada nível de precedência

Na gramática, ordenar da menor para a maior precedência

## Impor associatividade

Operador associativo à esquerda

Nas regras de produção, colocar o termo recursivo **antes** do termo não recursivo

Operador associativo à direita

Nas regras de produção, colocar o termo recursivo **depois** do termo não recursivo

# Exemplos

## Precedência

```
<expr> → <expr> + <term>  
        | <expr> - <term>  
        | <term>
```

```
<term> → <term> * <factor>  
        | <term> / <factor>  
        | <factor>
```

## Associatividade à esquerda

```
<term> → <term> * <factor>
```

## Associatividade à direita

```
<expo> → <number> ^ <expo>
```

# Lidar com a ambiguidade

## A remoção da ambiguidade nem sempre é possível!

Uma linguagem ambígua não possui uma gramática não ambígua

### Exemplo

$$L = \{a^i b^j c^k \mid i, j, k \geq 1, i = j \text{ ou } j = k\}$$

Escreva uma gramática

Desenhe 2 árvores sintáticas para  $a^i b^j c^i$

# Sintaxe das LP

## Alguns aspetos da sintaxe não conseguem ser especificados com GLC

### Exemplos

Não declarar o mesmo identificador duas vezes no mesmo bloco

Obrigar a declaração do identificador antes da sua utilização

Obrigar que o nº de parâmetros actuais seja igual ao nº de parâmetros formais

$A[i,j]$  ser válido apenas se  $A$  for bidimensional

### Solução

Especificar estes aspectos informalmente, separadamente da gramática formal

# Implementação

## **Análise Lexical**

gramáticas regulares

autômatos de estados finitos

## **Análise Sintática**

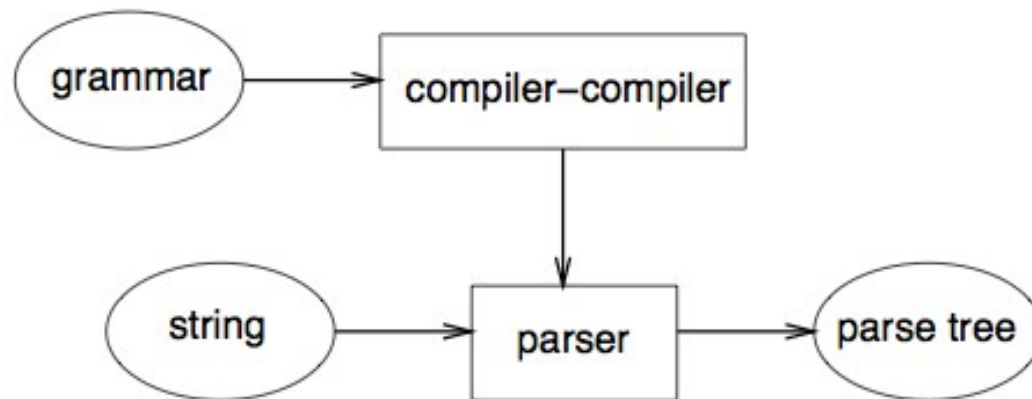
gramáticas livres de contexto

algoritmos de análise sintática



# Ferramentas

## Compiler-compiler



## Exemplos

Yacc / Bison

JavaCC

CUP