Aula Teórica 9 (guião)

Semana de 10 a 14 de Novembro de 2025

José Carlos Ramalho

Sinopsis:

- Parsers Bottom-Up: LR0, SLR1 e outros.
- Imagens, alguns exemplos e outro conteúdo retirados da sebenta "Processamento de Linguagens: Reconhecedores Sintáticos" de José João Almeida e José Bernardo Barros, editada em Abril de 2022.

Uma última visita ao Top Down: Top Down dirigido por tabela

Considere a seguinte gramática retirada da sebenta:

Gramática:

```
p1: S      --> Exp '.'
p2: Exp    --> INT
p3:          | '(' Funcao ')'
p4: Funcao --> '+' Lista
p5:          | '*' Lista
p6: Lista  --> Exp Lista
p7:          |
```

Com a qual podemos escrever frases do tipo:

```
( * 2 3 ( + 1 1 ) ) .                 // daria 12
( + 1 2 ( * 1 2 3 ) ( + 11 11 ) ) .   // daria 31
```

Símbolos terminais: `T = {'(', ')', '+', '*', '.', INT}`

Lookaheads e condição LL(1)

```
la(p1) = First(Exp) = {INT, '('}

la(p2) = {INT}
la(p3) = {'('}

la(p4) = {'+'}
la(p5) = {'*'}
```

```
la(p6) = First(Exp) = {INT, '('}
la(p7) = {')'}
```

Estamos agora em condições de construir a tabela LL(1):

	INT	(	)	+	*	.	S	Exp	Funcao	Lista
INT	Av.						p1	p2		p6
(		Av.					p1	p3		p6
)			Av.							p7
+				Av.					p4	
*					Av.				p5	
.						Rec.				

- Fazer a animação com:

1. Input: "`(* 1 2 3).`"
2. Stack(início): `[S]`

Parser Bottom-Up: LR(0)

Vamos usar um exemplo mais simples, o que está na sebenta:

```
p1: S --> A a
p2:     | b
p3: A --> a A
        | c
```

No Bottom-Up, começamos por fazer a expansão da gramática (para garantir um terminador):

```
p0: Z --> S $
p1: S --> A a
p2:     | b
p3: A --> a A
        | c
```

Convencionou-se que o '$' representa o fim da frase, por exemplo, o fim de linha ou o fim de ficheiro.

Vamos agora:

- Desenhar o autómato LR(0): Estados, Kernel, Expansão, Items...
- Animar o algoritmo de reconhecimento.

Voltando à nossa gramática inicial(alteramos alguma coisa?):

```
p1: S      --> Exp '.'
p2: Exp    --> INT
p3:          | '(' Funcao ')'
p4: Funcao --> '+' Lista
p5:          | '*' Lista
p6: Lista  --> Exp Lista
p7:          |
```

- Desenhar o autómato LR(0): Estados, Kernel, Expansão, Items...
- Animar o algoritmo de reconhecimento.

Analisador Léxico (o que já usamos ou talvez não...)

```
# sexp_analex.py
# 2025-11-11 by jcr
# ----------------------
import ply.lex as lex

tokens = ['INT']
literals = ['(', ')', '.', '+', '*']

t_INT = r'\d+'

def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

t_ignore = '\t '

def t_error(t):
    print('Carácter desconhecido: ', t.value[0], 'Linha: ',
t.lexer.lineno)
    t.lexer.skip(1)

lexer = lex.lex()
```

Programa exemplo

- O programa usa a variável interna do parser `success` para determinar o sucesso da análise sintática;
- Neste exemplo, está a processar-se uma linha de cada vez.

```
# sexp_program.py
# 2025-11-04 by jcr
```

```
# ----------------------
from sexp_sin import parser
import sys

for linha in sys.stdin:
    # Análise do texto
    parser.parse(linha)

    if parser.success:
        print("Frase válida: ", linha)
    else:
        print("Frase inválida... Corrija e tente novamente!")
```

O parser em ply.yacc

```
import ply.yacc as yacc
from sexp_analex import tokens, literals

# Production rules
def p_gramatica(p):
    """
    S      --> Exp '.'
    Exp    --> INT
             | '(' Funcao ')'
    Funcao --> '+' Lista
             | '*' Lista
    Lista  --> Exp Lista
             |
    """

def p_error(p):
    print('Erro sintático: ', p)
    parser.success = False

# Build the parser
parser = yacc.yacc()

# Adicionar estado ao parser
parser.success = True
```