Aula Teórica 8 (guião)

Semana de 3 a 7 de Novembro de 2025

José Carlos Ramalho

Sinopsis:

- Parsers Bottom-Up: LR0, SLR1 e outros.

Exercício: Revisitando a linguagem das listas

Exemplos:

```
[]
[2]
[ 2, 4, 5]
[ 2, 4, [ 5, 7, 9], 6]
```

Símbolos terminais: `T = {'[', ']', ',', num}`

Produções:

No reconhecimento Bottom-UP iremos tentar especificar as gramáticas com recursividade à esquerda:

```
Lista --> '[' ']'
        | '[' Conteudo ']'

Conteudo --> num
           | Conteudo ',' num
```

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

```
# listas_analex.py
# 2025-11-04 by jcr
# ----------------------
import ply.lex as lex

tokens = ['NUM']
literals = ['[', ']', ',']

t_NUM = 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.

```
# listas_program.py
# 2025-11-04 by jcr
# ----------------------
from listas_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!")
```

Reconhecedores (Parsers) Bottom-Up: a ferramenta ply.yacc

- Existe uma manual detalhado online

Parser para as listas (MSP - mais simples possível)

- Nesta versão, estamos apenas a indicar se a frase é válida ou não, podemos colocar a gramática toda numa única função.

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

# Production rules
def p_gramatica(p):
```

```
    """
    Lista : '[' ']'
          | '[' Conteudo ']'

    Conteudo : NUM
             | Conteudo ',' NUM
    """

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

# Build the parser
parser = yacc.yacc()

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

Ações Semânticas

- Para colocarmos ações semânticas diferenciadas devemos separar as produções em várias funções;
- Desta forma, podemos ter ações diferenciadas para cada produção.

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

# Production rules
def p_listaVazia(p):
    "Lista : '[' ']'"

def p_lista(p):
    "Lista : '[' Conteudo ']'"

def p_conteudoNum(p):
    "Conteudo : NUM"

def p_conteudo(p):
    "Conteudo : Conteudo ',' NUM"

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

# Build the parser
parser = yacc.yacc()

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

Contar o número de elementos na lista

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

# Production rules
def p_listaVazia(p):
    "Lista : '[' ']'"
    p[0] = 0

def p_lista(p):
    "Lista : '[' Conteudo ']'"
    p[0] = p[2]

def p_conteudoNum(p):
    "Conteudo : NUM"
    p[0] = 1

def p_conteudo(p):
    "Conteudo : Conteudo ',' NUM"
    p[0] = p[1] + 1

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

# Build the parser
parser = yacc.yacc()

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

Resultado

- O valor final retornado pelo parser é o valor que ficar associado ao axioma da gramática:

```
def p_listaVazia(p):
    "Lista : '[' ']'"
    p[0] = 0

def p_lista(p):
    "Lista : '[' Conteudo ']'"
    p[0] = p[2]
```

- Que no programa pode ser obtido da seguinte forma:

```
res = parser.parse(linha)
```

```
if parser.success:
    print("Frase válida: ", linha)
    print(f"Número de elementos na lista: {res}")
```

Desafio:

- Altera o parser anterior para calcular o somatório da lista.

Representação Intermédia

- Um parser pode ir calculando um resultado à medida que vai reconhecendo;
- Esse resultado poderá ser uma representação abstrata da árvore de derivação ou algo ligeiramente mais simples contendo apenas a informação relevante.

Gramática Abstrata

Já tínhamos visto que, neste caso, o modelo é uma lista simples.

```
p1: Lista -->
p2:         | num Lista
```

Modelo em Python

```
# ----------------------
# P1: Lista -->  num Conteudo
# P2:          |
# ----------------------------
class Lista:
    def __init__(self, type, num, lista):
        self.type = type
        self.num = num
        self.lista = lista

    def pp(self):
        print('(', end="")
        print(self.num, " ", end="")
        self.lista.pp()
        print(')', end="")

    def pprev(self):
        print('(', end="")
        self.lista.pprev()
        print(self.num, " ", end="")
        print(')', end="")

    def count(self):
        return 1 + self.lista.count()
```

```
    def sum(self):
        return int(self.num) + self.lista.sum()

class Vazia:
    def __init__(self, type):
        self.type = type

    def pp(self):
        print('()', end="")

    def pprev(self):
        print('()', end="")

    def count(self):
        return 0

    def sum(self):
        return 0
```

Juntando tudo

Vamos juntar o cálculo da representação intermédia.

O programa principal

```
# listas_program_ri.py
# 2025-11-04 by jcr
# ----------------------
from listas_sin_ri import parser
import sys

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

    if parser.success:
        print("Frase válida: ", linha)
        res.pp()
        print("\n----------------------\n")
        res.pprev()
        print("\n", res.count())
        print("\n", res.sum())
    else:
        print("Frase inválida... Corrija e tente novamente!")
```

O parser

```
import ply.yacc as yacc
from listas_analex import tokens, literals
from listas_ast import Lista, Vazia

# Production rules
def p_listaVazia(p):
    "Lista : '[' ']'"
    p[0] = Vazia('vazia')

def p_lista(p):
    "Lista : '[' Conteudo ']'"
    p[0] = p[2]

def p_conteudoNum(p):
    "Conteudo : NUM"
    p[0] = Lista('lista', p[1], Vazia('vazia'))

def p_conteudo(p):
    "Conteudo : Conteudo ',' NUM"
    p[0] = Lista('lista', p[3], p[1])

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

# Build the parser
parser = yacc.yacc()

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

Desafio final

- Especifique um parser usando o ply yacc para exemplos deste tipo:

```
(doc
    (tit "Primeiro exemplo")
    (subtit "Aula 6: 2025-03-14")
    "Este é um primeiro exemplo."
)
```