Aula Teórica 6 (guião)

Semana de 10 a 14 de Março de 2025

José Carlos Ramalho

Sinopsis:

- Parsers Top-Down: o Recursivo Descendente;
- A condição LL(1).

Gramáticas

- Gramáticas Regulares;
- Gramáticas Independentes de Contexto;
- Gramáticas Dependentes de Contexto;
- Gramáticas Livres.

Gramáticas

Uma gramática G é um tuplo (N, T, S, P), onde:

- N - conjunto de símbolos não terminais;
- T - conjunto de símbolos terminais;
- S - símbolo inicial;
- P - as produções.

Exemplificar: uma turma de alunos

```
Turma PL2025 .
Alunos{
    ("A23876", "Ana Maria")
    ("A78654", "Paulo Azevedo")
    ...
}
```

Qual seria a gramática?

```
T = {}
S =
N =
P =
```

(Dar exemplos)

LISP + MarkDown

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

Expressões S

```
(+ 5 4)
(+ (- 7 2) (+ 12 19))
```

Revisitando os parentesis

```
()
(()())()
```

Expressões aritméticas básicas

```
4 + 15
7 - 12 * 3
7 + 8 * 2 / 3
```

Voltando às listas

Exercício: a linguagem das listas

Exemplos:

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

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

Produções:

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

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

Analisador Léxico

```
# listas_analex.py
# 2023-03-21 by jcr
# ---------------------
import ply.lex as lex

tokens = ('NUM', 'PA', 'PF', 'VIRG')

t_NUM = r'[+\-]?\d+'
t_PA = r'\['
t_PF = r'\]'
t_VIRG = r','

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

```
# listas_program.py
# 2023-03-21 by jcr
# ---------------------

linha = input("Introduza uma lista: ")
rec_Parser(linha)
```

Analisador Sintático: recursivo descendente

```
# listas_anasin.py
# 2023-03-21 by jcr
```

```
# ----------------------
import listas_analex

prox_simb = ('Erro', '', 0, 0)

# Lista --> '[' ']'
#         | '[' Conteudo ']'
def rec_Lista():
    global prox_simb
    ... Temos um problema!!!

def rec_Parser(data):
    global prox_simb
    lexer.input(data)
    prox_simb = lexer.token()
    rec_Lista()
    print("That's all folks!")
```

Com a alteração das produções de Lista

```
def parserError(simb):
    print("Erro sintático, token inesperado: ", simb)

def rec_term(simb):
    global prox_simb
    if prox_simb.type == simb:
        prox_simb = lexer.token()
    else:
        parserError(prox_simb)

# P4: Conteudo --> num
# P5:            | num ',' Conteudo
# É preciso alterar para:
# P4: Conteudo --> num Cont2
# P5: Cont2    -->
# P6: Cont2    --> ',' Conteudo

def rec_Cont2():
    global prox_simb
    if prox_simb.type == 'VIRG':
        rec_term('VIRG')
        rec_Conteudo()
        print("Reconheci P6: Cont2    --> ',' Conteudo")
    elif prox_simb.type == '???':
        print("Reconheci P5: Cont2 -->")
    else:
        parserError(prox_simb)

def rec_Conteudo():
    rec_term('NUM')
    rec_Cont2()
```

```
    print("Reconheci P4: Conteudo --> num Cont2")

def rec_LCont():
    global prox_simb
    if prox_simb.type == 'PF':
        rec_term('PF')
        print("Reconheci P2: LCont --> ']'")
    elif prox_simb.type == 'NUM':
        rec_Conteudo()
        rec_term('PF')
        print("Reconheci P3: LCont --> Conteudo ']'")
    else:
        parserError(prox_simb)

# P1: Lista --> '[' LCont
# P2: LCont --> ']'
# P3:         | Conteudo ']'
def rec_Lista():
    global prox_simb
    rec_term('PA')
    rec_LCont()
    print("Reconheci P1: Lista --> '[' LCont")
```

Gramática Concreta Final

```
P1: Lista --> '[' LCont
P2: LCont --> ']'
P3:         | Conteudo ']'
P4: Conteudo --> num Cont2
P5: Cont2    -->
P6: Cont2    --> ',' Conteudo
```

Gramática Abstrata

```
P1: Lista -->
P2:         | Conteudo
P3: Conteudo --> num Conteudo
P4:            |
```

Modelo em Python

```
class Lista
```