Aula Teórica 7 (guião)

Semana de 27 a 31 de Outubro de 2025

José Carlos Ramalho

Sinopsis:

- Parsers Top-Down: o Recursivo Descendente;
- A condição LL(1).
- Imagens e algum conteúdo retirados da sebenta "Processamento de Linguagens: Reconhecedores Sintáticos" de José João Almeida e José Bernardo Barros, editada em Abril de 2022.

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)
```

Reconhecedores (Parsers) Top-Down: o recursivo descendente

Condição LL(1)

Para quaisquer duas derivações com o mesmo lado esquerdo, a **interseção dos seus lookaheads tem que ser vazia**.

Exemplo: cálculo dos lookaheads para o caso das listas

Gramática:

```
p1: Lista --> '[' ']'
p2:         | '[' Conteudo ']'

p3: Conteudo --> num
p4:            | num ',' Conteudo
```

Lookaheads:

```
la(p1) = First('[' ']') = {'['}
la(p2) = First('[' Conteudo ']') = {'['}

la(p1) ∩ la(p2) = {'['} ==> Conflito LL(1)

la(p3) = First(num) = {num}
la(p4) = First(num ',' Conteudo) = {num}

la(p3) ∩ la(p4) = {num} ==> Conflito LL(1) ❌
```

Quando existem conflitos não é possível reconhecer a linguagem com um parser TopDown que esteja a ver apenas um símbolo à frente. A solução passa por transformar a gramática.

Vamos supor que tínhamos especificado a gramática com outra forma:

```
p1: Lista --> '[' ']'
p2:         | '[' Conteudo ']'

p3: Conteudo --> num
p4:            | Conteudo ',' num
```

O conflito entre `p1` e `p2` continua a existir. E entre `p3` e `p4`?

```
la(p1) = First('[' ']') = {'['}
la(p2) = First('[' Conteudo ']') = {'['}

la(p1) ∩ la(p2) = {'['} ==> Conflito LL(1)

la(p3) = ...
la(p4) = ...

la(p3) ∩ la(p4) = ...
```

Patologias identificáveis em gramáticas LL(1)

Recursividade à esquerda

```
p1: A --> A X
p2:     | Y

la(p1) = First(A) = First(A X) ∪ First(Y)
la(p2) = First(Y)
la(p1) ∩ la(p2) = First(Y) ==> Conflito LL(1) ❌
```

Transformação a aplicar:

```
p1: A    --> Y cont
p2: cont --> X cont
p3:        | ε

la(p2) = First(X)
la(p3) = Follow(cont) = Follow(A)
la(p2) ∩ la(p3) =  {}
```

Prefixo Comum

```
p1: A --> X Y
p2:     | X Z

la(p1) = First(X Y) = First(X)
la(p2) = First(X Z) = First(X)
la(p1) ∩ la(p2) = First(X) ==> Conflito LL(1) ❌
```

Transformação a aplicar:

```
p1: A    --> X cont
p2: cont --> Y
p3:        | Z

la(p2) = First(Y)
la(p3) = First(Z)
la(p2) ∩ la(p3) =  {}
```

Voltando ao caso das listas

```
p1: Lista --> '[' ']'
p2:         | '[' Conteudo ']'

p3: Conteudo --> num
p4:            | Conteudo ',' num
```

Vamos aplicar a transformação do prefixo comum entre `p1` e `p2` e a transformação da recursividade à esquerda entre `p3` e `p4`:

```
p1: Lista --> '[' Lcont
p2: Lcont --> ']'
p3:         | Conteudo ']'

p4: Conteudo --> num Cont2
```

```
p5: Cont2    --> ',' num Cont2
p6:            | ε

la(p2) = {']'}
la(p3) = First(Conteudo) = {num}
la(p1) ∩ la(p2) =  {} ✅

la(p5) = {','}
la(p6) = Follow(Cont2) = Follow(Conteudo) = {']'}
la(p1) ∩ la(p2) =  {} ✅
```

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")
```

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 Concreta Final

```
p1: Lista --> '[' LCont
p2: LCont --> ']'
p3:         | Conteudo ']'
```

```
p4: Conteudo --> num Cont2
p5: Cont2    -->
p6: Cont2    --> ',' Conteudo
```

Gramática Abstrata

Vamos retirar o que é constante da gramática abstrata:

```
p1: Lista -->
p2:         | Conteudo
p3: Conteudo --> num Conteudo
p4:            |
```

Simplificando:

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

O que fica dá-nos o modelo a construir como representação intermédia, neste caso, uma lista simples.

Modelo em Python

```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

O programa principal

```
# listas_program_ri.py
# 2023-03-21 by jcr
# ---------------------
from listas_anasin_ri import rec_Parser

# Exemplo de uma travessia externa à classe: o maior da lista
def maxLista2(m, asa):
    if asa.type == 'vazia':
        res = m
    else:
        if m > int(asa.num):
            res = maxLista2(m, asa.lista)
        else:
            res = maxLista2(int(asa.num), asa.lista)
    return res

def maxLista(asa):
    if asa.type == 'vazia':
        res = None
    else:
        res = maxLista2(int(asa.num), asa.lista)
    return res


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

```
print("\n----------------------\n")
ast.pprev()
print("\n", ast.count())
print("\n", ast.sum())
print("\nO Maior é: ", maxLista(ast))
```

O parser recursivo descendente

```
# listas_anasin_ri.py
# 2023-03-21 by jcr
# ----------------------
from listas_analex import lexer
from listas_ast import Lista, Vazia

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

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)
        prox_simb = ('erro', '', 0, 0)

# 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':
        print("Derivando por P6: Cont2    --> ',' Conteudo")
        rec_term('VIRG')
        res = rec_Conteudo()
        print("Reconheci P6: Cont2    --> ',' Conteudo")
    elif prox_simb.type == 'PF':
        print("Derivando por P5: Cont2 -->")
        res = Vazia('vazia')
        print("Reconheci P5: Cont2 -->")
    else:
        parserError(prox_simb)
        res = Vazia('vazia')
    return res

def rec_Conteudo():
```

```
    global prox_simb
    print("Derivando por P4: Conteudo --> num Cont2")
    num = prox_simb.value
    rec_term('NUM')
    cont2 = rec_Cont2()
    print("Reconheci P4: Conteudo --> num Cont2")
    return Lista('lista', num, cont2)

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

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


def rec_Parser(data):
    global prox_simb
    lexer.input(data)
    prox_simb = lexer.token()
    return rec_Lista()
```

Desafio final

- Especifique um parser recursivo descendente para o caso da agenda de contactos:
    1. Uma agenda é uma lista de entradas ou grupos de entradas;
    2. Uma entrada tem um identificador, um tipo, e uma lista de campos: nome, email e telefone;
    3. Um grupo é composto por um identificador e uma lista de entradas ou referências a entradas previamente declaradas.

Exemplo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<agenda>
    <entrada id="e1" tipo="pessoa">
        <nome>José Martins Costa</nome>
        <email>josemartinscosta@di.uminho.pt</email>
        <telefone>253-131244</telefone>
    </entrada>

    <entrada id="e2" tipo="pessoa">
        <nome>Rui Silva</nome>
        <email>ruisilva@universidade.pt</email>
        <telefone>911-948749</telefone>
    </entrada>
    ...
    <grupo gid="g10">
        <entrada id="e199" tipo="pessoa">
            <nome>Rui Gomes</nome>
            <email>ruigomes@di.uminho.pt</email>
            <telefone>912-826420</telefone>
        </entrada>
        <entrada id="e200" tipo="pessoa">
            <nome>António Ferreira</nome>
            <email>antonioferreira@contactos.org</email>
            <telefone>218-769737</telefone>
        </entrada>
        <ref entref="e29"/>
        <ref entref="e16"/>
        <ref entref="e141"/>
    </grupo>
</agenda>
```