Resolução - Teste PL 2024

3 de junho de 2024 18:21

Questão 4: Parser a)

NT = { LingPaises, Pares, Par, Lingua, Paises, Pais }
T = { '.', ':', ',', DIC, CID, ORIG, pal }
P = {
p1: LingPaises : DIC Pares CID
p2: Pares : Pares Par '.'
p3: | &
p4: Par : Lingua ':' Paises
p5: Lingua : pal ORIG pal
p6: Paises : Pais ',' Paises
p7: : Pais
p8: Pais : pal
}

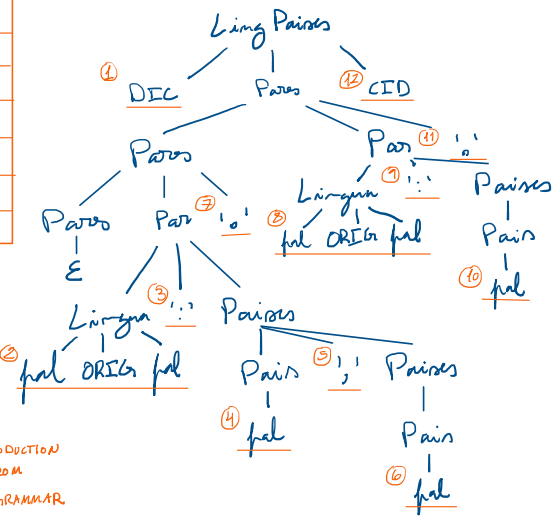
Recursividade à esquerda

First(Pares) = First(Pares) ∪ {ε} ?

DIC
PT ORIG latim: Portugal, Brasil.
FR ORIG latim: França
CID

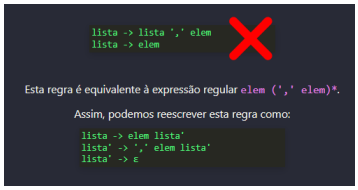
-	FIRST	FOLLOW
LingPaises -> DIC Pares CID	DIC	$
Pares -> Pares Par '.' | &	pal, &	pal, CID
Par -> Lingua ':' Paises	pal	'.'
Lingua -> pal ORIG pal	pal	':'
Paises -> Pais ',' Paises | Pais	pal	'.'
Pais -> pal	pal	',', '.'

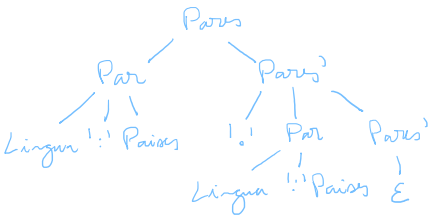
~~DIC~~
Pares
~~LingPaises~~ CID
$



-	.	:	,	DIC	CID	ORIG	pal
LingPaises				LingPaises -> DIC Pares CID			
Pares					Pares -> Pares Par '.'		Pares -> Pares Par '.' Pares -> ε
Par							Par -> Lingua ':' Paises
Lingua							Lingua -> pal ORIG pal
Paises							Paises -> Pais ',' Paises Paises -> Pais
Pais							Pais -> pal

TWO DIFFERENT PRODUCTION RULES TO CHOOSE FROM
⇒ NOT LL(1) GRAMMAR

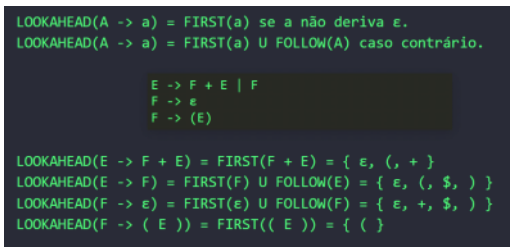




1	DIC
2	PT ORIG latim
3	:
4	Portugal
5	,
6	Brasil
7	.
8	FR ORIG latim
9	:
10	França
11	.
12	CID

Pares -> Par Pares'
Pares' -> '.' Par Pares' | &

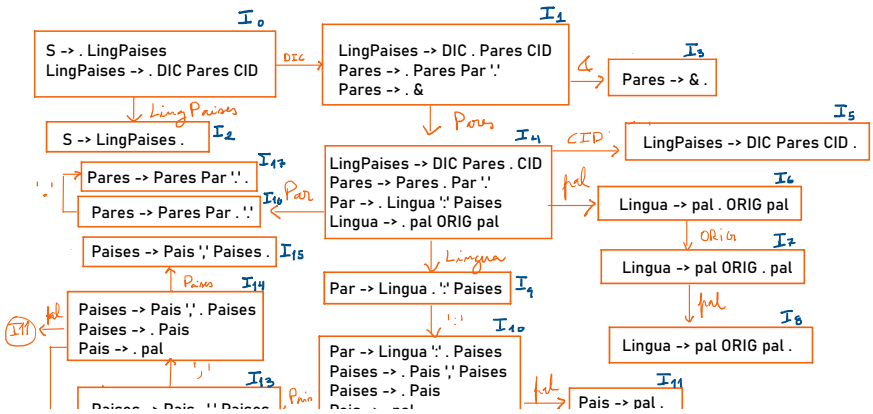
4 b)

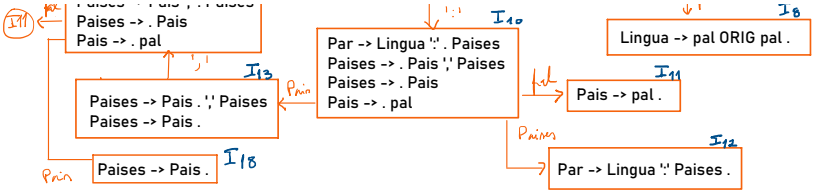


LOOKAHEAD(Pares -> Pares Par '.' | &) = Como calcular o FIRST(Pares) ?
LOOKAHEAD(Paises -> Pais ',' Paises | Pais) = { pal }
Justificação acima do porquê que não é LL1 (n sei usar o lookahead)

4 c)

Autómato LR(0)





SLR(1) Table

-	.	:	,	DIC	CID	ORIG	pal	&	$	LingPaises	Pares	Par	Lingua	Paises	Pais
0				S1						2					
1								S3			4				
2									AC						
3					R p3		R p3								
4					S5		S6					16	9		
5									R p1						
6						S7									
7							S8								
8		R p5													
9		S10													
10							S11							12	13
11	R p8		Rp8												
12	R p4														
13			S14												
14							S11							15	18
15	R p6														
16	S17														
17					R p2		R p2								
18	R p6														

DIC
PT ORIG latim: Portugal, Brasil.
FR ORIG latim: França
CID

(inventei aquele G4, essa parte provavelmente n está certa; also, isto n é pedido no exercício, só me apeteceu fazer)

Stack	Input	Action
0	DIC PT ORIG latim: Portugal, Brasil. FR ORIG latim: França CID	-
0Dic1	 PT ORIG latim: Portugal, Brasil. FR ORIG latim: França CID	Shift 1
0Dic1	 PT ORIG latim: Portugal, Brasil. FR ORIG latim: França CID	Goto 4
0Dic1pal6	 ORIG latim: Portugal, Brasil. FR ORIG latim: França CID	Shift 6
0Dic1pal6Orig7	 latim: Portugal, Brasil. FR ORIG latim: França CID	Shift 7
0Dic1pal6Orig7pal8	 : Portugal, Brasil. FR ORIG latim: França CID	Shift 8
0Dic1pal6Orig7pal8	 : Portugal, Brasil. FR ORIG latim: França CID	Shift 8
0Dic1Lingua9	 : Portugal, Brasil. FR ORIG latim: França CID	Reduce by Lingua -> pal ORIG pal
0Dic1Lingua9:10	 Portugal, Brasil. FR ORIG latim: França CID	Shift 10
0Dic1Lingua9:10pal11	 , Brasil. FR ORIG latim: França CID	Shift 11
0Dic1Lingua9:10Pais13	 , Brasil. FR ORIG latim: França CID	Reduce by Pais -> pal
0Dic1Lingua9:10Pais13,14	 Brasil. FR ORIG latim: França CID	Shift 14
0Dic1Lingua9:10Pais13,14pal11	 . FR ORIG latim: França CID	Shift 11
0Dic1Lingua9:10Pais13,14Pais18		Reduce by Pais -> pal

	FR ORIG latim: França CID	
0Dic1Lingua9:10Paises12	FR ORIG latim: França CID	Reduce by Paises -> Pais ',' Paises
0Dic1Par4	FR ORIG latim: França CID	Reduce by Par : Lingua ':' Paises
0Dic1Par4.17	FR ORIG latim: França CID	GOTO 16, Shift 17
0Dic1Pares4	FR ORIG latim: França CID	Reduce by Pares -> Pares Par '.'

. . .

4 d)

```
from enum import Enum
import re

""" GRAMÁTICA LL(1)
Paises -> Pais OutrosP
OutrosP -> ',' Pais OutrosP | &
Pais -> pal
"""

class Tokens(Enum):
    PAL = 0
    COMMA = 1
    EOF = -1


def tokenizer(string):
    tokens = []

    while len(string) > 0:
        if string[0] == ',':
            tokens.append((Tokens.COMMA, ','))
            string = string[1:]
        elif m := re.match(r"[a-zA-Z\-]+", string):
            tokens.append((Tokens.PAL, m.group()))
            string = string[m.end():]
        elif string[0] in " \n\t":
            string = string[1:]
        else:
            raise ValueError(f"Caracter inválido - {string[0]}")

    return tokens + [(Tokens.EOF, None)]

tokens = None

def current_token():
    global tokens
    return tokens[0]

def next_token():
    global tokens
    tokens.pop(0)

def parse_paises():
    pais = parse_pais()
    outros_p = parse_outros_p()
    return [pais] + outros_p

def parse_outros_p():
    match current_token():
        case (Tokens.COMMA, _):
            next_token()
            pais = parse_pais()
            outros_p = parse_outros_p()
            return [pais] + outros_p
        case (Tokens.EOF, _):
            return []
        case err:
            raise ValueError(f"Estava à espera de ',' ou EOF, mas recebi {err}")

def parse_pais():
    match current_token():
        case (Tokens.PAL, x):
            next_token()
            return x
        case err:
            raise ValueError(f"Estava à espera de um nome de país (palavra), mas recebi {err}")

def parse(tk):
    global tokens
    tokens = tk
    return parse_paises()


dados = "Portugal, Espanha, Franca"
tk = tokenizer(dados)
result = parse(tk)
print(result)
```

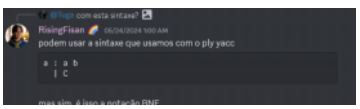
4 e)

Calcular o número total de línguas diferentes

LingPaises -> DIC Pares CID { return len(languages) }
Pares -> Pares Par ','
| ε
Par -> Lingua ':' Paises
Lingua -> pal ORIG pal { languages.add(pal1 + " " + pal2) }
Paises -> Pais ',' Paises { return 1 + paises_count }
| Pais { paises_count += 1 }
Pais -> pal

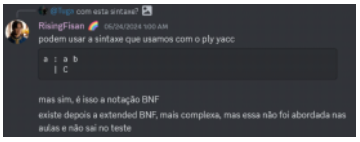
(também se podia utilizar transições de estados para descrever as ações (?))

Questão 3: Gramáticas (5v = 3 + 2)
3 a)



Note: uma gramática independente do contexto diz-se regular

Questão 3: Gramáticas (5v = 3 + 2)
3 a)



Note: uma gramática independente do contexto diz-se regular

GIC = <T, N, S, P>

T: (conjunto de símbolos terminais)
\n , : ( ) NAME { } [ ]

N: (conjunto de símbolos não terminais)
Paises, Pais, Paises', Loc, Locs, Pt, Pts, Pts'

S (símbolo inicial)

P (conjunto de produções)

S -> Paises

Paises -> Pais Paises'
Paises' -> \n Pais Paises' | &

Pais -> NAME { Loc Locs }
Locs -> , Loc Locs | &
Loc -> NAME : [Pts]

Pts -> Pt Pts'
Pts' -> , Pt Pts' | &
Pt -> (Ordem , Tipo , Custo)

```
import ply.lex as lex

""" Exemplo:
Portugal { Braga : [(1, Monumento, 15), (2, Exposição, 20)], Porto : [(3, Espetaculo, 40)] }
Espanha { Madrid : [(4, Panorama, 10)] }
"""

tokens = (
    'COLON',
    'COMMA',
    'LPAREN',
    'RPAREN',
    'LSBRACKET',
    'RSBRACKET',
    'LCBRACKET',
    'RCBRACKET',
    'NAME'
)

states = (
    ('country', 'exclusive'),
    ('location', 'exclusive'),
    ('points', 'exclusive'),
    ('point', 'exclusive')
)

t_ignore = ' \t\n'
t_location_points_point_country_ignore = ' \t\n'

"""Country"""

def t_INITIAL_NAME(t):
    r'\w+'
    t.lexer.push_state('country')
    return t

def t_country_LCBRACKET(t):
    r'\{'
    return t

"""Location"""

def t_country_NAME(t):
    r'\w+'
    t.lexer.push_state('location')
    return t

def t_location_COLON(t):
    r'\:'
    return t

def t_location_COMMA(t):
    r'\,'
    t.lexer.pop_state()
    return t

def t_location_RCBRACKET(t):
    r'\}'
    t.lexer.pop_state()  # location
    t.lexer.pop_state()  # country
    return t

"""Points"""

def t_location_LSBRACKET(t):
    r'\['
    t.lexer.push_state('points')
    return t

def t_points_COMMA(t):
    r'\,'
    return t
def t_points_RSBRACKET(t):
    r'\]'
    t.lexer.pop_state()
    return t

"""Point"""

def t_points_LPAREN(t):
    r'\('
    t.lexer.push_state('point')
    return t

def t_point_NAME(t):
    r'\w+'
    return t

def t_point_COMMA(t):
    r'\,'
    return t

def t_point_RPAREN(t):
    r'\)'
    t.lexer.pop_state()
    return t

"""Any"""
```

```
def t_ANY_error(t):
    print(f"Illegal character: {t.value[0]}")
    raise lex.LexError("Illegal character")

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

lexer = lex.lex()

def run_tests():
    tests = [
        {
            "name": "test 1",
            "input": """
                Portugal { Braga : [(1, Monumento, 15), (2, Exposição, 20)], Porto : [(3, Espetaculo, 40)] }
                Espanha { Madrid : [(4, Panorama, 10)] }
            """
        }
    ]

    for test in tests:
        print(f"Test: {test['name']}\n")
        lexer.input(test['input'])
        for tok in lexer:
            print(tok)
        print('\n---------------\n')

if __name__ == '__main__':
    run_tests()
```

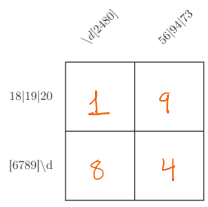
Também se poderia capturar um grupo todo com a expressão

```
\(((?:\w|\.|\d)+\,?\s?)+\)
```

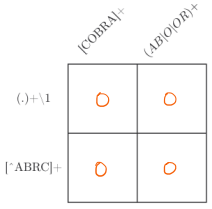
YACC: (a gramática foi modificada para ter recursividade à esquerda -- parsing ascendente)

```
    Paises : Paises Pais
           |
    Pais : NAME LCBRACKET Locs Loc RCBRACKET
    Locs : Locs Loc COMMA
         |
    Loc : NAME COLON LSBRACKET Pts Pt RSBRACKET
    Pts : Pts Pt COMMA
        |
    Pt : LPAREN NAME COMMA NAME COMMA NAME RPAREN


import ply.yacc as yacc
from viagem_lexer import tokens

def p_Paises(p):
    """
    Paises : Paises Pais
           |
    """
    if (len(p) == 1):
        p[0] = []
    else:
        p[0] = p[1] + p[2]


def p_Pais(p):
    """
    Pais : NAME LCBRACKET Locs Loc RCBRACKET
    """
    p[0] = [(p[1], p[3] + p[4])]

def p_Locs(p):
    """
    Locs : Locs Loc COMMA
         |
    """
    if (len(p) == 1):
        p[0] = []
    else:
        p[0] = p[1] + p[2]

def p_Loc(p):
    """
    Loc : NAME COLON LSBRACKET Pts Pt RSBRACKET
    """
    p[0] = [(p[1], p[4] + p[5])]

def p_Pts(p):
    """
    Pts : Pts Pt COMMA
        |
    """
    if (len(p) == 1):
        p[0] = []
    else:
        p[0] = p[1] + p[2]


def p_Pt(p):
    """
    Pt : LPAREN NAME COMMA NAME COMMA NAME RPAREN
    """
    p[0] = [(p[2], p[4], p[6])]

def p_error(p):
    print("Syntax error in input!", p)
    parser.exito = False

parser = yacc.yacc()
parser.exito = True
test = """
Portugal { Braga : [(1, Monumento, 15), (2, Exposição, 20)], Porto : [(3, Espetaculo, 40)] }
Espanha { Madrid : [(4, Panorama, 10)] }
"""
result = parser.parse(test)
print(result)
```

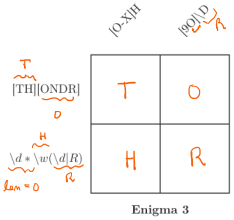
Questão 1: Expressões Regulares (6v = 1.5 + 1.5 + 1.5 + 1.5)



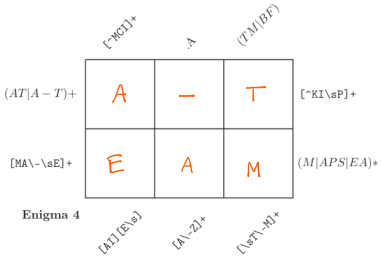
Enigma 1



Enigma 2



Enigma 3



Enigma 4

Questão 2: filtros (4v = 1.5 + 1.5 + 1)

```
import re
from typing import Tuple

def get_sorted_ids(html: str) -> list[str]:
    pattern = r'id="([^"]+)"'
    ids = re.findall(pattern, html)
    return sorted(ids)

def get_tag_counts(html: str) -> Tuple[int, int]:
    start_pattern = r'\<(\w+)([^<>\/]*?)\>'
    end_pattern = r'\<\/(\w+)\>'
    starts = re.findall(start_pattern, html)
    ends = re.findall(end_pattern, html)
    return len(starts), len(ends)

def get_tag_counts_2(html: str) -> Tuple[int, int]:
    start_pattern = r'\<(\w+)([^<>\/]*?)\>'
    end_pattern = r'\<\/(\w+)\>'
    num_start_tags = 0
    num_end_tags = 0
    for _ in re.finditer(start_pattern, html):
        num_start_tags += 1
    for _ in re.finditer(end_pattern, html):
        num_end_tags += 1
    return num_start_tags, num_end_tags


if __name__ == "__main__":

    input = " … "

    ids = get_sorted_ids(input)
    for value in ids:
        print(value)

    counts = get_tag_counts(input)
    print(counts)
```

2 c)

```
import re
def check_acronym(acronym: str, first_letters: str) -> bool:
    ptr = 0
    for l in first_letters:
        if l == acronym[ptr]:
            ptr += 1
    return ptr == (len(acronym) - 1)

def get_valid_names(text: str, acronym: str) -> list[str]:

    valid_names = []
    for line in text.splitlines():

        pattern = r"(?:\s*?)(?:[A-Z]\w*\s*)+(?:\s*?)"
        m = re.match(pattern, line)

        if m is not None:

            pattern = r"[A-Z]\w*\b"
            names = re.findall(pattern, line)
            first_letters = ''.join([name[0] for name in names])
            if check_acronym(acronym.lower(), first_letters.lower()):
                valid_names.append(line)

    return valid_names


if __name__ == "__main__":
    input = """
        Josefina Maria Carvalho Ramos
        asd asd asd
        João Rui Carvalho
        José Carlos Leite Ramalho
        Pedro Rangel Henriques
        Tiago João Fernandes Batista
    """

    valid_names = get_valid_names(input, "jrc")
    for name in valid_names:
        print(name)
```