PDA a CFG

May 28, 2021

1 PDA a CFG

Teoría de la Computación
Miguel Angel Soto Hernandez

2 Generar reglas de un PDA

2.1 Leer un PDA

[ ]:
```
pda = []
with open("pda.txt") as pushdown:
        for lineas in pushdown:
            pda.append(lineas.split())
print(pda)
```

```
[['f,c,/;g,c'], ['g,b,/;g,/'], ['g,c,c;h,/']]
```

2.2 Prcocesamiento PDA

[ ]:
```
def limpiar_pda(pda):
    pda_limpio = []
    for i in range(len(pda)):
        for j in range(len(pda[i])):
            sin_comas = pda[i][j].split(',')
            pda_limpio.append(sin_comas)

    punto_y_coma = []
    for i in range(len(pda_limpio)):
        bandera = 2
        for j in range(len(pda_limpio[i])):
            if len(pda_limpio[i][j]) > 1:
                punto_y_coma = pda_limpio[i][j].split(';')

        pda_limpio[i].pop(2)
        for j in range(len(punto_y_coma)):
            pda_limpio[i].insert(bandera, punto_y_coma[j])
            bandera += 1
```

```
    return pda_limpio
```

[ ]:
```
pda_limpio = limpiar_pda(pda)
print(pda_limpio)
```

```
[['f', 'c', '/', 'g', 'c'], ['g', 'b', '/', 'g', '/'], ['g', 'c', 'c', 'h',
'/']]
```

[ ]:
```
alfabeto = []

for i in range(len(pda_limpio)):
    bandera = 1
    for j in range(len(pda_limpio[i])):
        alfabeto.append(pda_limpio[i][bandera])

alfabeto = list(set(alfabeto))
alfabeto.sort()
print(alfabeto)
```

```
['b', 'c']
```

2.3 Regla 1

[ ]:
```
def regla_1(pda):
    estado_inicial = pda[0][0]
    estado_final = pda[len(pda) - 1][3]
    regla1 = [estado_inicial, '/', estado_final]
    print('Regla 1:')
    print(f'S -> <{estado_inicial}, /, {estado_final}>')
    return regla1
```

[ ]:
```
regla1 = regla_1(pda_limpio)
```

```
Regla 1:
S -> <f, /, h>
```

2.4 Regla 2

[ ]:
```
def regla_2(pda):
    estados = []
    for i in range(len(pda)):
        bandera = 0
        for j in range(2):
            estados.append(pda[i][bandera])
            bandera = 3
```

```
    estados = list(set(estados))
    estados.sort()

    regla2 = []
    print('Regla 2')
    for i in range(len(estados)):
        regla2.append([estados[i], '/', estados[i]])
        print(f'<{regla2[i][0]}, {regla2[i][1]}, {regla2[i][2]}> -> /')

    return regla2, estados
```

[ ]:
```
regla2, estados = regla_2(pda_limpio)
```

```
Regla 2
<f, /, f> -> /
<g, /, g> -> /
<h, /, h> -> /
```

2.5 Regla 3

[ ]:
```
def regla_3(pda, estados):
    regla3 = []
    for i in range(len(pda)):
        if pda[i][2] != '/':
            print('Regla 3')
            for j in range(len(estados)):
                regla3.append([[pda[i][0], pda[i][1], estados[j]],␣
 ↪[pda[i][2],[pda[i][3], '/', estados[j]]])
                print(f'<{pda[i][0]}, {pda[i][1]}, {estados[j]}> ->␣
 ↪{pda[i][2]}<{pda[i][3]}, /, {estados[j]}>')

    return regla3
```

[ ]:
```
regla3 = regla_3(pda_limpio, estados)
print(regla3)
```

```
Regla 3
<g, c, f> -> c<h, /, f>
<g, c, g> -> c<h, /, g>
<g, c, h> -> c<h, /, h>
[[['g', 'c', 'f'], ['c', ['h', '/', 'f']]], [['g', 'c', 'g'], ['c', ['h', '/',
'g']]], [['g', 'c', 'h'], ['c', ['h', '/', 'h']]]]
```

2.6 Regla 4

[ ]:
```
def regla_4(pda, estados):
    regla4 = []
    w = []
    bandera = 2

    for i in range(len(pda)):
        w.append(pda[i][bandera])
        w = list(set(w))
        w.sort()

    print('Regla 4')
    for i in range(len(pda)):
        if pda[i][2] == '/':
            for j in range(len(w)):
                for k in range(len(estados)):
                    for l in range(len(estados)):
                        regla4.append([[pda[i][0], w[j], estados[k]],␣
↪[pda[i][1], [pda[i][3], pda[i][4], estados[l]], [estados[l], w[j],␣
↪estados[k]]]])
                        print(f'<{pda[i][0]}, {w[j]}, {estados[k]}> ->␣
↪{pda[i][1]}<{pda[i][3]}, {pda[i][4]}, {estados[l]}><{estados[l]}, {w[j]},␣
↪{estados[k]}>')

    return regla4
```

[ ]:
```
regla4 = regla_4(pda_limpio, estados)
print(regla4)
```

```
Regla 4
<f, /, f> -> c<g, c, f><f, /, f>
<f, /, f> -> c<g, c, g><g, /, f>
<f, /, f> -> c<g, c, h><h, /, f>
<f, /, g> -> c<g, c, f><f, /, g>
<f, /, g> -> c<g, c, g><g, /, g>
<f, /, g> -> c<g, c, h><h, /, g>
<f, /, h> -> c<g, c, f><f, /, h>
<f, /, h> -> c<g, c, g><g, /, h>
<f, /, h> -> c<g, c, h><h, /, h>
<f, c, f> -> c<g, c, f><f, c, f>
<f, c, f> -> c<g, c, g><g, c, f>
<f, c, f> -> c<g, c, h><h, c, f>
<f, c, g> -> c<g, c, f><f, c, g>
<f, c, g> -> c<g, c, g><g, c, g>
<f, c, g> -> c<g, c, h><h, c, g>
<f, c, h> -> c<g, c, f><f, c, h>
<f, c, h> -> c<g, c, g><g, c, h>
```

<f, c, h> -> c<g, c, h><h, c, h>
<g, /, f> -> b<g, /, f><f, /, f>
<g, /, f> -> b<g, /, g><g, /, f>
<g, /, f> -> b<g, /, h><h, /, f>
<g, /, g> -> b<g, /, f><f, /, g>
<g, /, g> -> b<g, /, g><g, /, g>
<g, /, g> -> b<g, /, h><h, /, g>
<g, /, h> -> b<g, /, f><f, /, h>
<g, /, h> -> b<g, /, g><g, /, h>
<g, /, h> -> b<g, /, h><h, /, h>
<g, c, f> -> b<g, /, f><f, c, f>
<g, c, f> -> b<g, /, g><g, c, f>
<g, c, f> -> b<g, /, h><h, c, f>
<g, c, g> -> b<g, /, f><f, c, g>
<g, c, g> -> b<g, /, g><g, c, g>
<g, c, g> -> b<g, /, h><h, c, g>
<g, c, h> -> b<g, /, f><f, c, h>
<g, c, h> -> b<g, /, g><g, c, h>
<g, c, h> -> b<g, /, h><h, c, h>
[[['f', '/', 'f'], ['c', ['g', 'c', 'f'], ['f', '/', 'f']]], [['f', '/', 'f'],
['c', ['g', 'c', 'g'], ['g', '/', 'f']]], [['f', '/', 'f'], ['c', ['g', 'c',
'h'], ['h', '/', 'f']]], [['f', '/', 'g'], ['c', ['g', 'c', 'f'], ['f', '/',
'g']]], [['f', '/', 'g'], ['c', ['g', 'c', 'g'], ['g', '/', 'g']]], [['f', '/',
'g'], ['c', ['g', 'c', 'h'], ['h', '/', 'g']]], [['f', '/', 'h'], ['c', ['g',
'c', 'f'], ['f', '/', 'h']]], [['f', '/', 'h'], ['c', ['g', 'c', 'g'], ['g',
'/', 'h']]], [['f', '/', 'h'], ['c', ['g', 'c', 'h'], ['h', '/', 'h']]], [['f',
'c', 'f'], ['c', ['g', 'c', 'f'], ['f', 'c', 'f']]], [['f', 'c', 'f'], ['c',
['g', 'c', 'g'], ['g', 'c', 'f']]], [['f', 'c', 'f'], ['c', ['g', 'c', 'h'],
['h', 'c', 'f']]], [['f', 'c', 'g'], ['c', ['g', 'c', 'f'], ['f', 'c', 'g']]],
[['f', 'c', 'g'], ['c', ['g', 'c', 'g'], ['g', 'c', 'g']]], [['f', 'c', 'g'],
['c', ['g', 'c', 'h'], ['h', 'c', 'g']]], [['f', 'c', 'h'], ['c', ['g', 'c',
'f'], ['f', 'c', 'h']]], [['f', 'c', 'h'], ['c', ['g', 'c', 'g'], ['g', 'c',
'h']]], [['f', 'c', 'h'], ['c', ['g', 'c', 'h'], ['h', 'c', 'h']]], [['g', '/',
'f'], ['b', ['g', '/', 'f'], ['f', '/', 'f']]], [['g', '/', 'f'], ['b', ['g',
'/', 'g'], ['g', '/', 'f']]], [['g', '/', 'f'], ['b', ['g', '/', 'h'], ['h',
'/', 'f']]], [['g', '/', 'g'], ['b', ['g', '/', 'f'], ['f', '/', 'g']]], [['g',
'/', 'g'], ['b', ['g', '/', 'g'], ['g', '/', 'g']]], [['g', '/', 'g'], ['b',
['g', '/', 'h'], ['h', '/', 'g']]], [['g', '/', 'h'], ['b', ['g', '/', 'f'],
['f', '/', 'h']]], [['g', '/', 'h'], ['b', ['g', '/', 'g'], ['g', '/', 'h']]],
[['g', '/', 'h'], ['b', ['g', '/', 'h'], ['h', '/', 'h']]], [['g', 'c', 'f'],
['b', ['g', '/', 'f'], ['f', 'c', 'f']]], [['g', 'c', 'f'], ['b', ['g', '/',
'g'], ['g', 'c', 'f']]], [['g', 'c', 'f'], ['b', ['g', '/', 'h'], ['h', 'c',
'f']]], [['g', 'c', 'g'], ['b', ['g', '/', 'f'], ['f', 'c', 'g']]], [['g', 'c',
'g'], ['b', ['g', '/', 'g'], ['g', 'c', 'g']]], [['g', 'c', 'g'], ['b', ['g',
'/', 'h'], ['h', 'c', 'g']]], [['g', 'c', 'h'], ['b', ['g', '/', 'f'], ['f',
'c', 'h']]], [['g', 'c', 'h'], ['b', ['g', '/', 'g'], ['g', 'c', 'h']]], [['g',
'c', 'h'], ['b', ['g', '/', 'h'], ['h', 'c', 'h']]]]

3 Leer un string y determinar si esaceptado por el PDA

[ ]:
```
string =  'cbbc'
```

[ ]:
```
def procesar_cadena_texto(string):
    string = list(string)
    return string
```

[ ]:
```
string = procesar_cadena_texto(string)
cadena_inicial = string
string
```

[ ]:
```
['c', 'b', 'b', 'b', 'c']
```

[ ]:
```
def agregar_regla_2(string, opcion):
    opciones = []
    ultima_opcion = []

    for i in range(len(regla2)):
        ultima_opcion = string[len(string) - 1]
        if opcion == 1:
            opciones.append([ultima_opcion, regla2[i]])
        elif opcion == 2:
            opciones.append([ultima_opcion[0], regla2[i], ultima_opcion[1]])
        elif opcion == 3:
            opciones.append([ultima_opcion[0], ultima_opcion[1], regla2[i]])
        else:
            break

    return opciones


def coincidencia_regla_3_4(opciones, regla):
    coincidencia = []
    opcion_correcta = []

    for i in range(len(opciones)):
        for j in range(len(regla)):
            if opciones[i] == regla[j][1]:
                coincidencia = regla[j][0]
                opcion_correcta = regla[j][1]
                break

    return coincidencia, opcion_correcta


def buscar_regla_4(string):
    coincidencia_r4 = []
    coincidencia_r4_bool = False
```

```
    for i in range(len(regla4)):
        if string[len(string) - 1] == regla4[i][1]:
            coincidencia_r4 = regla4[j][0]
            coincidencia_r4_bool = True
            return coincidencia_r4, coincidencia_r4_bool
            break
        else:
            coincidencia_r4_bool = False
            return coincidencia_r4, coincidencia_r4_bool


def obtener_cadena_actualizada(string, valor_a_insertar):
    string.pop()
    string.append([string[len(string) - 1], valor_a_insertar])
    string.pop(len(string) - 2)
    print(f'Ahora tenemos una nueva cadena: {string}\n')
    return string


def imprimir_coincidencia(opcion_correcta, arreglo_coincidencia):
    if arreglo_coincidencia:
        print(f'Hubo coincidencia, ahora sustituiremos nuestra opcion que␣
 ↪coincidió, en este caso {opcion_correcta} y la sustituimos por␣
 ↪{arreglo_coincidencia}\n')
    else:
        print('No se encontro coincidencia')


def procesamiento_principal(string):
    alfabeto_cadena = list(set(string))
    alfabeto_cadena.sort()

    if alfabeto_cadena != alfabeto:
        print('Hay letras que no corresponden al alfabeto')
    else:
        print(f'Cadena ingresada: {string}')
        print(f'Tomamos el ultimo valor, en este caso "{string[len(string) -␣
 ↪1]}" y probamos con todas los valores de la regla 2\n')

        print(f'Regla 2: {regla2}')
        opciones = agregar_regla_2(string, 1)
        print(f'Opciones con nuestra regla 2: {opciones}\n')

        print('Ahora comparamos nuestras opciones con nuetras regla 3')
        print(f'Regla 3: {regla3}')
```

```
        coincidencia_r3, opcion_correcta = coincidencia_regla_3_4(opciones,␣
↪regla3)
        imprimir_coincidencia(opcion_correcta, coincidencia_r3)

        string = obtener_cadena_actualizada(string, coincidencia_r3)

        for i in range(len(string) - 1):
            coincidencia_r4 = []
            coincidencia_r4_bool = True
            opciones1 = []
            nueva_coincidencia_r4 = []
            opcion_correcta1 =[]

            print(f'Con esta nueva cadena {string[len(string) - 1]}␣
↪compararemos si hay alguna coincidencia con la regla 4')
            coincidencia_r4, coincidencia_r4_bool = buscar_regla_4(string)
            print(f'Coincidencia: {coincidencia_r4_bool}')
            print('Dado que no hubo coincidencia, ahora agregaremos otro lambda␣
↪de la regla 2, y buscaremos una coincidencia nuevamente con la regla 4\n')

            if coincidencia_r4_bool == False:
                opciones1 = agregar_regla_2(string, 2)
                print(f'Ahora las opciones a comparar con la regla 4 son las␣
↪siguientes: {opciones1}')
                nueva_coincidencia_r4, opcion_correcta1 =␣
↪coincidencia_regla_3_4(opciones1, regla4)
                imprimir_coincidencia(opcion_correcta1, nueva_coincidencia_r4)
                string = obtener_cadena_actualizada(string, coincidencia_r3)

            if len(string) == 1:
                opciones1 = agregar_regla_2(string, 3)
                print(f'Ahora las opciones a comparar con la regla 4 son las␣
↪siguientes: {opciones1}')
                nueva_coincidencia_r4, opcion_correcta1 =␣
↪coincidencia_regla_3_4(opciones1, regla4)
                imprimir_coincidencia(opcion_correcta1, nueva_coincidencia_r4)

                if nueva_coincidencia_r4 == regla1:
                    print(f'El regla final {nueva_coincidencia_r4} es igual a␣
↪la regla 1 {regla1}, por lo tanto la cadena es aceptada por el PDA')
                else:
                    print(f'El regla final {nueva_coincidencia_r4} es diferente␣
↪a la regla 1 {regla1}, por lo tanto la cadena NO es aceptada por el PDA')
```

[ ]:
```
aceptado = procesamiento_principal(string)
```

```
Cadena ingresada: ['c', 'b', 'b', 'b', 'c']
```

```
Tomamos el ultimo valor, en este caso "c" y probamos con todas los valores de la
regla 2

Regla 2: [['f', '/', 'f'], ['g', '/', 'g'], ['h', '/', 'h']]
Opciones con nuestra regla 2: [['c', ['f', '/', 'f']], ['c', ['g', '/', 'g']],
['c', ['h', '/', 'h']]]

Ahora comparamos nuestras opciones con nuetras regla 3
Regla 3: [[['g', 'c', 'f'], ['c', ['h', '/', 'f']]], [['g', 'c', 'g'], ['c',
['h', '/', 'g']]], [['g', 'c', 'h'], ['c', ['h', '/', 'h']]]]
Hubo coincidencia, ahora sustituiremos nuestra opcion que coincidió, en este
caso ['c', ['h', '/', 'h']] y la sustituimos por ['g', 'c', 'h']

Ahora tenemos una nueva cadena: ['c', 'b', 'b', ['b', ['g', 'c', 'h']]]

Con esta nueva cadena ['b', ['g', 'c', 'h']] compararemos si hay alguna
coincidencia con la regla 4
Coincidencia: False
Dado que no hubo coincidencia, ahora agregaremos otro lambda de la regla 2, y
buscaremos una coincidencia nuevamente con la regla 4

Ahora las opciones a comparar con la regla 4 son las siguientes: [['b', ['f',
'/', 'f'], ['g', 'c', 'h']], ['b', ['g', '/', 'g'], ['g', 'c', 'h']], ['b',
['h', '/', 'h'], ['g', 'c', 'h']]]
Hubo coincidencia, ahora sustituiremos nuestra opcion que coincidió, en este
caso ['b', ['g', '/', 'g'], ['g', 'c', 'h']] y la sustituimos por ['g', 'c',
'h']

Ahora tenemos una nueva cadena: ['c', 'b', ['b', ['g', 'c', 'h']]]

Con esta nueva cadena ['b', ['g', 'c', 'h']] compararemos si hay alguna
coincidencia con la regla 4
Coincidencia: False
Dado que no hubo coincidencia, ahora agregaremos otro lambda de la regla 2, y
buscaremos una coincidencia nuevamente con la regla 4

Ahora las opciones a comparar con la regla 4 son las siguientes: [['b', ['f',
'/', 'f'], ['g', 'c', 'h']], ['b', ['g', '/', 'g'], ['g', 'c', 'h']], ['b',
['h', '/', 'h'], ['g', 'c', 'h']]]
Hubo coincidencia, ahora sustituiremos nuestra opcion que coincidió, en este
caso ['b', ['g', '/', 'g'], ['g', 'c', 'h']] y la sustituimos por ['g', 'c',
'h']

Ahora tenemos una nueva cadena: ['c', ['b', ['g', 'c', 'h']]]

Con esta nueva cadena ['b', ['g', 'c', 'h']] compararemos si hay alguna
coincidencia con la regla 4
Coincidencia: False
```

```
Dado que no hubo coincidencia, ahora agregaremos otro lambda de la regla 2, y
buscaremos una coincidencia nuevamente con la regla 4

Ahora las opciones a comparar con la regla 4 son las siguientes: [['b', ['f',
'/', 'f'], ['g', 'c', 'h']], ['b', ['g', '/', 'g'], ['g', 'c', 'h']], ['b',
['h', '/', 'h'], ['g', 'c', 'h']]]
Hubo coincidencia, ahora sustituiremos nuestra opcion que coincidió, en este
caso ['b', ['g', '/', 'g'], ['g', 'c', 'h']] y la sustituimos por ['g', 'c',
'h']

Ahora tenemos una nueva cadena: [['c', ['g', 'c', 'h']]]

Ahora las opciones a comparar con la regla 4 son las siguientes: [['c', ['g',
'c', 'h'], ['f', '/', 'f']], ['c', ['g', 'c', 'h'], ['g', '/', 'g']], ['c',
['g', 'c', 'h'], ['h', '/', 'h']]]
Hubo coincidencia, ahora sustituiremos nuestra opcion que coincidió, en este
caso ['c', ['g', 'c', 'h'], ['h', '/', 'h']] y la sustituimos por ['f', '/',
'h']

El regla final ['f', '/', 'h'] es igual a la regla 1 ['f', '/', 'h'], por lo
tanto la cadena es aceptada por el PDA
```

[ ]: