28/11/23

ITESO, Universidad Jesuita de Guadalajara

Lenguajes Formales

MINI PROYECTO 2. GRAMÁTICAS LC Y PILAS - IMPLEMENTACIÓN CON LEX Y YACC O PILAS Y ER PARA EXTRACCIÓN DE INFORMACIÓN DE ARCHIVOS TIPO JSON O XML

ALUMNOS: DÍAZ CAMPOS JOSÉ JUAN

AMADOR GUDIÑO EDGAR EDUARDO

PROFESORA: ALCARAZ MEJÍA MILDRETH ISADORA

SEMESTRE: OTOÑO 2023

Contenido

Propósito

Poner en práctica los conceptos, propiedades y procedimientos aprendidos con respecto a: Obtención de gramáticas o Autómatas de Pila e implementación de reglas de producción.

Descripción del sistema al que está apoyando la estructura JSON y cómo es que el sistema lo usa

El sistema que está siendo apoyado por la estructura JSON proporcionada, que es una estructura de estandarizada de diferentes CVs. Cada elemento en el array JSON representa un perfil de una persona con información específica, dividida en varias secciones como "basics", "work", etc.

A continuación, se describe la estructura del JSON:

Descripción del Sistema:

1. **Datos Básicos (basics):**
   - name: Nombre completo de la persona.
   - label: Una etiqueta asociada al nombre, como el lugar de estudio o el trabajo.
   - picture: Enlace a una imagen de perfil.
   - email: Dirección de correo electrónico.
   - phone: Número de teléfono.
   - website: Enlace al sitio web personal o profesional.
   - summary: Un resumen breve o descripción de la persona.
   - location: Información sobre la ubicación, con el código de país y la dirección.
2. **Perfiles en Redes Sociales (profiles):**
   - network: Nombre de la red social (por ejemplo, GitHub, LinkedIn).
   - username: Nombre de usuario en la red social.
   - url: Enlace al perfil en la red social.
3. **Trabajo (work):**
   - company: Nombre de la empresa donde trabaja.
   - position: Puesto laboral o cargo.
   - startDate: Fecha de inicio en el trabajo.
   - endDate: Fecha de finalización del trabajo (puede haber errores en el formato de fecha).
   - summary: Resumen de las responsabilidades y experiencia laboral.
4. **Datos Meta (meta):**
   - theme: Tema asociado, en este caso, se utiliza el término "modern" o "default" para indicar preferencias de presentación.

Cómo el Sistema Utiliza el JSON:

El sistema está enfocado en la extracción primordial para hacer la conexión con N numero de personas que han aplicado para la empresa extrayendo los datos necesarios para el primer acercamiento del equipo de recursos humanos. La información que se extrajo fue:

- Identificación Única (id): Cada individuo tiene un identificador único.
- Información Básica del Usuario (basics):

- name: Nombre completo de la persona.
- email: Dirección de correo electrónico.
- phone: Número de teléfono.

La información fue extraída mediante dos archivos: un Lex y un Yacc. Yacc y Lex son herramientas clásicas en la construcción de compiladores y analizadores sintácticos. Se utilizan comúnmente en conjunto para analizar y procesar archivos de texto según una gramática dada.

Lex es una herramienta que genera analizadores léxicos (también conocidos como escáneres o tokenizadores). Su función principal es dividir el flujo de entrada en "tokens" o unidades léxicas. Yacc es una herramienta que genera analizadores sintácticos (también conocidos como parsers). Su tarea principal es reconocer la estructura sintáctica del código fuente.

GLC con su tupla

Gramática:

S->	A	|	B										
A->	SBL B SBR												
B->	BL C BR D												
BB->	BL C BR E												
C->	idNombre valueNombre E	|	trivial_key text E	|	trivial_key BB	|	idEmail valueEmail E	|	keyId idValue E	|	idTelefono valueTelefono E	
D->	ε	|	comma B										
E->	ε	|	comma C										

Tupla:

V	=	{S,A,B,BB,C,D,E}
T	=	{D,E}
P	=	S-> A
		S->B
		A-> SBL B SBR
		B-> BL C BR D
		BB-> BL C BR E
		C-> idNombre valueNombre E
		C-> trivial_key text E
		C-> trivial_key BB
		C-> idEmail valueEmail E
		C-> keyId idValue E
		C-> idTelefono valueTelefono E
		D-> ε
		D-> comma B
		E-> ε
		E-> comma C
S	=	S

Imágenes de la pantalla que muestra algunas pruebas de validación de la implementación

```
def p_nodeB_A(p):
    'C : idNombre valueNombre E '
    p[0] = p[1]
    print(p[2])
    names.insert(0,p[2])
```

```
def p_nodeB_F(p):
    'C : idEmail valueEmail E'
    p[0] = p[1]
    print(p[2])
    emails.insert(0,p[2])
```

```
def p_nodeB_J(p):
    'C : keyId idValue E'
    p[0] = p[1]
    print(p[2])
    ids.insert(0,p[2])

def p_nodeB_K(p):
    'C : idTelefono valueTelefono E'
    p[0] = p[1]
    print(p[2])
    phones.insert(0,p[2])
```

Todas las reglas donde sacamos la info relevante tienen prints, los pueden apreciar aquí:

html_yacc.py M

html_yacc.py > p_nodeB_C

```
83      p[0] = p[1]
84
85
86
87  def p_nodeB_E(p):
88      'C : trivial_key BB '
89      p[0] = p[1]
90
91
92  def p_nodeB_F(p):
93      'C : idEmail valueEmail E'
94      p[0] = p[1]
95      print(p[2])
96      emails.insert(0,p[2])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMI

```
(310) 1277512"
tscotcher0@weather.com"
Tamarah Scotcher"
1
(695) 9027257"
kfruchon1@desdev.cn"
Kissee Fruchon"
2
(526) 5137065"
philbin2@wikipedia.org"
Sheff Philbin"
3
(611) 1951855"
zferminger3@businessweek.com"
Zeb Ferminger"
4
(164) 7251527"
bathowe4@facebook.com"
Brockie Athowe"
5
(160) 7359480"
nfallowes5@example.com"
Nessy Fallowes"
6
(181) 8052848"
dlathe6@sogou.com"
```

Y el CSV exportado queda:

```
Se ha creado el archivo CSV: output.csv
```

html_yacc.py M | output.csv

original > LF_Proyect_2 > output.csv

```
ID,NAME,EMAIL,PHONE
1,Tamarah Scotcher,tscotcher0@weather.com,(310) 1277512
2,Kissee Fruchon,kfruchon1@desdev.cn,(695) 9027257
3,Sheff Philbin,sphilbin2@wikipedia.org,(526) 5137065
4,Zeb Ferminger,zferminger3@businessweek.com,(611) 1951855
5,Brockie Athowe,bathowe4@facebook.com,(164) 7251527
6,Nessy Fallowes,nfallowes5@example.com,(160) 7359480
7,Dottie Lathe,dlathe6@sogou.com,(181) 8052848
8,Carey Dracksford,cdracksford7@hubpages.com,(589) 7370335
9,Shalne Amberger,samberger8@nytimes.com,(754) 6380458
10,Jolynn Lillow,jlillow9@imdb.com,(324) 1573614
11,Roxie Culver,rculvera@about.me,(478) 7435348
12,Fraze Raynton,frayntonb@sourceforge.net,(605) 8877905
13,Katleen MacCallion,kmaccallionc@furl.net,(505) 3142390
14,Martita Mara,mmarad@altervista.org,(733) 4244482
15,Hali Jans,hjanse@cdc.gov,(524) 1843493
16,Ansell Greger,agregerf@redcross.org,(447) 7880041
17,Jena Cheasman,jcheasmang@dailymail.co.uk,(776) 7692969
18,Cordie Mariner,cmarinerh@xing.com,(274) 8133951
19,Honoria Okroy,hokroyi@sciencedaily.com,(680) 7564387
20,Lev Oehme,loehmej@chronoengine.com,(964) 7467641
```

Código de la implementación de la GLC

Código del lexer:

```
# PARTE 1: TOKENIZAR LA ENTRADA USANDO LEX
import ply.lex as lex

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import re

""" with open('test.json') as doc1:
    text = doc1.read()     # por default se abre como sólo lectura doc =
open("Data/Example1.txt", 'r')

doc1.close() # qué pasa si abrimos y no cerramos el archivo? """

# Paso 1: Proporcione una lista de tokens que defina
# todos los posibles nombres de token que puede producir
# el lexer.
# La lista de tokens también es utilizada YACC
# para identificar terminales.

tokens = (
    'SBL',
    'SBR',
    'BL',
    'BR',
    'keyId',
    'idValue',
    'idNombre',
    'valueNombre',
    'idEmail',
    'valueEmail',
    'idTelefono',
    'valueTelefono',
    "trivial_key",
    "comma",
    'text',
)

# Paso 2. Cada token se especifica escribiendo una regla
# de expresión regular, mediante declaraciones que usan
# un prefijo especial *t_* para indicar que define un token.
```

```
# Definición de tokens simples:

t_SBL = r"\["
t_SBR = r"\]"
t_BL = r"\{"
t_BR = r"\}"
t_keyId= r"\"id\":[ ]*"
t_idValue=r"[0-9]+"
t_idNombre=r"\"name\":[ ]*\""
t_valueNombre="[A-Z]?[A-Z,a-z,á-ú]*[ ]*[A-Z][A-Z,a-z,á-ú]*\""
t_idEmail=r"\"email\":[ ]*\""
t_valueEmail=r"[A-Z,a-z,0-9]*@[A-Z,a-z,\.]*\""
t_idTelefono=r"\"phone\":[ ]*\""
t_valueTelefono=r"\([0-9]*\)[ ,0-9]*\""
t_trivial_key=r"\"[A-z]+\":"
t_comma = r","
t_text=r'".+"'




# Definición de tokens que incluye código para
# complementar, por ejemplo, para la definición de número,
# se incluye su conversión a entero (para este caso):

##def t_NUMBER(t):
##    r'\d+'
##    t.value = int(t.value)
##    return t


# Para incluir caracteres a ignorar, en este caso,
# tabuladores y espacios.
t_ignore  = '/s \t \n'


# Para el manejo de errores en las entradas
def t_error(t):
    print('Illegal character', t.value[0])
    t.lexer.skip(1)


# Paso 3. Crear el objeto tipo lex (el tokenizador)
```

```
lexer = lex.lex()

# Hasta aquí se requiere sólo para tokenizar antes del parseo.


# DE AQUI EN ADELANTE ES PARA VALIDAR ESTE SCRIPT

# # Paso 4. Definir o solicitar la entrada al usuario
# # Se define una cadena de entrada, para este ejemplo.
# # data puede ser una cadena obtenida de un archivo o directamente
# # dada por un usuario desde la línea de comandos.


""" input_str = text


print('Para terminar introduce ENTER --- ')
 ##
 ### Paso 5. Usar el objeto tipo lex para tokenizar la entrada
 ### Hacer que el objeto tipo lex identifique en la cadena de
 ### entrada los tokens definidos.
 ##
if input_str != '':
    lexer.input(input_str)
 ##
 ### los tokens quedan en la variable lexer de tipo objeto Lexer,
 ### y se puede acceder a esos elementos de la siguiente manera:
 ##
for tok in lexer:
    valx=1
    print(tok)
 ##
 ### La salida de cada tok tiene el siguiente formato:
 ### LexToken(type of token, token, line number, position)
 """
```

Código del yacc:

```
# PARTE 2: ANALISIS GRAMATICAL (PARSING) USANDO YACC
# DE LA ENTRADA YA TOKENIZADA CON LEX

#Se importan los elementos del Yacc
import ply.yacc as yacc
#Se obtienen los tokesn del archivo lex
from html_lex import tokens
#Se importan los elementos para poder generar el CVS
import csv
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
#Se importa el modulo que permite la evalucion de expresiones regulares
import re

#Se abre el archivo JSON de donde se obtiene la información
with open('CV-Json.json') as doc1:
    # por default se abre como sólo lectura doc = open("Data/Example1.txt",
'r')
    text = doc1.read()
# Se cierra el documento
doc1.close()


#Se establecen los tokens que se usaran
precedence = (
    (
        'SBL',
        'SBR',
        'BL',
        'BR',
        "comma",
        'keyId',
        'idValue',
        'idNombre',
        'valueNombre',
        'idEmail',
        'valueEmail',
        'idTelefono',
        'valueTelefono',
        "trivial_key",
        'text'
    ),
)
```

```
#Se crean las listas donde guardaremos los datos que extraeremos del JSON
titles=['ID','NAME','EMAIL','PHONE']
ids=[]
names=[]
emails=[]
phones=[]

#Se generan las reglas
def p_start(p):
    'S : A'
    p[0] = p[1]


def p_start2(p):
    'S : B'
    p[0]=p[1]

def p_array(p):
    'A : SBL B SBR'
    p[0] = p[1]


def p_brackets(p):
    'B : BL C BR D'
    p[0] = p[1]

def p_brackets2(p):
    'BB : BL C BR E'
    p[0] = p[1]


def p_nodeB_A(p):
    'C : idNombre valueNombre E '
    p[0] = p[1]
    print(p[2])
    names.insert(0,p[2])

def p_nodeB_C(p):
    'C : trivial_key text E '
    p[0] = p[1]



def p_nodeB_E(p):
    'C : trivial_key BB '
```

```
    p[0] = p[1]


def p_nodeB_F(p):
    'C : idEmail valueEmail E'
    p[0] = p[1]
    print(p[2])
    emails.insert(0,p[2])



def p_nodeB_J(p):
    'C : keyId idValue E'
    p[0] = p[1]
    print(p[2])
    ids.insert(0,p[2])

def p_nodeB_K(p):
    'C : idTelefono valueTelefono E'
    p[0] = p[1]
    print(p[2])
    phones.insert(0,p[2])

def p_nodeD_A(p):
    'D : '


def p_nodeD_B(p):
    'D : comma B'
    p[0] = p[1]




def p_nodeD_C(p):
    'E : comma C'
    p[0] = p[1]


def p_nodeD_C2(p):
    'E : '



# Error rule for syntax errors
def p_error(p):
```

```
    print("Syntax error in input!")


# Build the parser
parser = yacc.yacc()

x=0
while x==0:
   try:
       s = text
   except EOFError:
       break
   if not s: continue
   result = parser.parse(s)
   x=1


# Combinar las listas de datos en una lista de tuplas
ids=[s.replace('"', '').replace("'", '') for s in ids]
names =[s.replace('"', '').replace("'", '') for s in names]
emails = [s.replace('"', '').replace("'", '') for s in emails]
phones = [s.replace('"', '').replace("'", '') for s in phones]
combined_data = list(zip(ids, names, emails, phones))

# Nombre del archivo CSV
csv_file = 'output.csv'

# Escribir el archivo CSV
with open(csv_file, 'w', newline='') as file:
    writer = csv.writer(file)

    # Escribir la cabecera
    writer.writerow(titles)

    # Escribir los datos
    writer.writerows(combined_data)

print(f'Se ha creado el archivo CSV: {csv_file}')
```

Conclusiones

José Juan Díaz Campos:

Para mí, fue hasta después de hacer el examen que tuve una idea clara de cómo proceder con el proyecto. Honestamente, estaba un poco perdido, pero después de darle muchas vueltas y consultar con mi compañero de equipo, que me mostró el avance que él tenía, me quedó muchísimo más claro.

Ahora me siento más cómodo hablando de gramáticas libres de contexto, y, sobre todo aplicándolas, pues este proyecto fue la práctica definitiva no para usarlas, si no para poder entenderlas a fondo. Creo que una vez pasado ese intervalo de confusión y estancamiento (y dedicándole suficiente tiempo), el proyecto marchó mucho más fácilmente.

En fin, fue un proyecto entretenido y didáctico. 😊

Edgar Eduardo Amador:

A lo largo de este ultimo periodo se me ha dificultad más las evaluaciones que hemos estado desarrollando tales como las gramáticas libres de contexto o las maquinas de Turing, sin embargo, gracias a este proyecto me he dado cuenta de la forma lógica de las estructuras anteriormente mencionadas. Mis habilidades analíticas y lógicas se han desarrollado a lo largo de este proyecto. En primer lugar, ya que se desarrollo la solución en tres maneras diferentes utilizando diferentes métodos que culminaron en la presentación final.

El objetivo principal era la extracción específica del JSON, como identificación (ID), nombre, correo electrónico y número de teléfono, y luego almacenar esos datos en un archivo CSV.

El código utiliza reglas gramaticales definidas por las funciones para reconocer y analizar la estructura del JSON. Los datos extraídos se almacenan en listas correspondientes a cada tipo de información, y luego se combinan en una lista de tuplas. Finalmente, esta información se escribe en un archivo CSV con encabezados apropiados.

Para culminar, me gustaría agregar que ahora me siento libre para proyectos que necesiten una evaluación sintáctica y me gusta como me he desarrollado en la clase. Bastante amena y aprendí bastante en el semestre.