Asignatura:
Sintaxis y Semántica de los Lenguajes

Primer cuatrimestre
Ciclo Lectivo 2022

Trabajo Práctico Integrador:
Entrega Final – Lexer + Parser + Traductor

Universidad Tecnológica Nacional
Facultad Regional Resistencia
Ingeniería en Sistemas de Información

Alumnos:

Claro, Michelle Andrea
Saucedo, Gonzalo Nicolás
Paredes, Samuel Octavio
Ramírez, Eduardo Manuel

Docentes:

Tomaselli, Gabriela
Torre, Juliana
Tortosa, Nicolás
Vigil, Rodrigo

Comisión: K2.1

Ciudad de Resistencia, Chaco
Abril - Julio de 2022

Índice

Introducción

Objetivo

En el presente Trabajo Práctico Integrador se pretende desarrollar las siguientes competencias de la asignatura, las cuales son:
Capacidad para reconocer los elementos propios de la Sintaxis y Semántica de los Lenguajes de Programación, pudiendo así comprender los conceptos y procedimientos léxicos/sintácticos asociados, para así poder aplicarlos en la creación de las gramáticas necesarias para detallar los lenguajes que luego se utilizaran en las correspondientes máquinas.

Descripción

(se actualiza antes de cada entrega)

Contenido

El documento estará divido en las siguientes partes:

- Primero detallaremos la definición del RSS que vamos a utilizar, declarando cuales son los tokens o los componentes léxicos que acepta (palabras reservadas como terminales) y como deben ser la estructura de las mismas.

- Luego describiremos las producciones de la gramática que utilizara el compilador, detallando los símbolos que utilizara y el significado de cada uno de ellos. El tipo de gramática que utilizamos es libre de contexto.

- Posteriormente profundizaremos con lo que respecta al analizador léxico, mencionando que herramientas utilizamos y una breve descripción de las mismas, en qué consiste el mismo, qué funciones posee, cómo se obtuvo y se ejecuta el intérprete y ejemplos.

Herramientas

XML (Extensible Markup Language)

El XML se preocupa por estructurar la información que se pretende almacenar, es decir, define un conjunto de reglas para la codificación de documentos. Este lenguaje de marcado es un conjunto de códigos que se pueden aplicar en el análisis de datos o la lectura de textos creados por computadoras o personas. Se utiliza generalmente para definir elementos, crear un formato y

generar un lenguaje personalizado. El diseño XML se centra en la simplicidad, la generalidad y la facilidad de uso y, por lo tanto, se utiliza para varios servicios web, entre otras funcionalidades.

RSS (Really Simple Syndication)

En simples palabras, es un formato que cumple con el estándar XML.

Este formato es muy estructurado debido a que posee cierto orden o forma.

RSS se usa principalmente para compartir contenido web actualizado desde un sitio web a otros, dado que es un formato estandarizado que facilita la forma de distribuir el contenido y además estos archivos se actualizan automáticamente.

Como todo documento XML se usa para describir, almacenar y transmitir o transportar información. La estructura está marcada por la propia lógica de la información.

Python

De forma resumida podemos decir que, es un lenguaje de alto nivel de programación (facilidad de utilizar para el programador) interpretado, cuya filosofía hace hincapié en la legibilidad de su código, ya que es lo más parecido al lenguaje humano y se puede aprender de manera sencilla y rápida.

Se utiliza para desarrollar aplicaciones de todo tipo y se trata de un lenguaje de programación multiparadigma (resultado de integrar dos o más paradigmas en un mismo sistema). Es un lenguaje interpretado (no se traduce realmente a un formato legible por el ordenador en tiempo de ejecución), dinámico y multiplataforma (todas las herramientas necesarias están disponibles en todas las plataformas principales).

Gramática

Reglas de Producción

Σ → XML //Llamado XML primero y luego a RSS (obligatoriamente).

XML→ `t_xml` RSS //Token (terminal) XML.

RSS→ `t_rss` CANAL `tc_rss` //Token RSS. Abre y cierra RSS

CANAL→ `t_canal` TITULO LINK DESC ITEM `tc_canal` | `t_canal` TITULO LINK DESC OPC ITEM `tc_canal` //Token Channel. /Del canal se va a los 3 elementos obligatorios: Título, Link y Descripción y luego Item./ O: Elementos opcionales.

TITULO → `t_titulo` t_txt `tc_titulo` //título genérico principal del canal.

LINK → `t_link` URL `tc_link` //link genérico principal del canal.

DESC → `t_desc` t_txt `tc_desc`

OPC → CAT|COPY|IMAGE|LENG|WEBMAST|ULTeDIT||CAT OPC|

// Del canal se va 3 elementos opcionales: Category, Copyright, Image. /Puede recursionar hacia el elemento opcional Categoría.

CAT→ `t_cat` TXT `tc_cat`

COPY→ `t_copy` TXT `tc_copy`

IMAGEN→ `t_imagen` URL TITULO LINK | URL TITULO OPIMAG `tc_imagen` //3 elementos obligatorios: Url, Título y Link.

OPIMAG→HW | H | W //2 elementos opcionales de la imagen: Width y Height.

H→ `t_height` N `tc_height`

W→ `t_width` N `tc_width`

LENG→ t_leng t_txt tc_leng

WEBMAST→ t_webmast t_txt tc_webmast

ULTeDIT→ t_ult_edit t_txt tc_ult_edit

ITEM→ `t_item` TITULO LINK DESC `tc_item` | `t_item` TITULO LINK DESC `tc_item` ITEM // 3 elementos obligatorios en Ítem: Título, Link y Descripción. /Recursión de items.

ITEM→ `t_item` TITULO LINK DESC OPI `tc_item` | `t_item` TITULO LINK DESC OPI `tc_item` ITEM // Elementos opcionales de Ítem./ Recursión de ítems con opciones adicionales.

OPI→ CAT | COPY | AUTOR | FECHA | CAT OPI // El ítem contiene 4 elementos opcionales**:** Category, Copyright, Autor y Fecha. /Puede recursionar hacia el elemento opcional Categoría.

AUTOR→ t_autor t_txt tc_autor
FECHA→ t_date t_fecha tc_date

Gramática para Links

URL→ t_url PRO SUB DOM EXT : PUERTO tc_url

//del URL pasa al protocolo (parte de su estructura)

PRO→ http:// **|** https:// **|** ftps:// **|** ftp://

//protocolo

○ http:// (para recursos de la web)

○ https:// (para recursos de la web contenidos en un servidor seguro)

○ ftp:// (recursos contenidos en un servidor de ficheros)

○ ftps:// (recursos contenidos en un servidor de ficheros seguro)

PUERTO→ t_num **|** /DIR **|** t_num PUERTO //puerto (opcional)

SUB→ t_txt. **|** www. **|** ww1 **|** blog. //subdominio

DOM→ t_txt/ **|** miweb //dominio

EXT→ t_txt **|** com **|** ar **|** org **/**/extensión

DIR→ t_txt **|** t_txt DIR/ **|** LOC //ruta o directorio (opcional)

LOC→#t_txt **|** #t_txt LOC //localizador interno (opcional)

Lista de Tokens

Lista de los Símbolos No Terminales:

· XML : Inicio del XML

· RSS : Inicio del RSS

· CANAL : Channel (canal)

. TITULO : Título

. LINK : Link

· DESC : Descripción

· ITEM : Item

· OPC : Opciones del Canal

· CAT : Categoría

· COPY : Copyright

· IMAG : Imagen

· OPIMAG : Opciones de la Imagen

· H : Height

· W : Width

· OPI : Opciones del Ítem

· TXT: Texto Genérico (terminales)

· WWW : Link Genérico (terminales)

· URL : URL

· PRO : Protocolo

· SUB : Subdominio

· DOM : Dominio

· EXT : Extensión

· DIR : Ruta o directorio (opcional)

. LOC : Localizador (opcional)

· PUERTO : Puerto (opcional)

Lista de los Símbolos No Terminales Adicionales:

. LENG: Lenguaje del canal

. WEBMAST: El administrador de la página

. ULTeDIT: Última edición de página

. AUTOR: Autor del ítem

. FECHA: Fecha del ítem

Lista de los Símbolos Terminales/Tokens:

· t_xml : <?xml version="t_num.t_num" encoding="t_txt-t_num"?>

· t_rss: <rss version=”t_num.t_num”>

· tc_rss:</rss>

· t_canal : <channel>

. tc_canal: </channel>

· t_titulo : <title>

. tc_titulo: </title>

· t_link : <link>

. tc_link: </link>

· t_desc : <description>

. tc_desc : </description>

· t_cat : <category>

. tc_cat: </category>

· t_copy : <copyright>

· tc_copy : </copyright>

· t_imag : <image>

· tc_imag : </image>

· t_url : <url>

· tc_url : </url>

· t_height : <height>

· tc_height : </height>

· t_idth : <width>

. tc_width : </width>

· t_item : <item>

. tc_item : </item>

. tc_date : </pubDate>

. tc_date : </pubDate>

. t_txt: Texto Génerico

. t_num: Número Genérico

. t_fecha: Fecha Genérica

. t_url: <url>

. tc_url: </url>

Lista de los Símbolos Terminales/Tokens Adicionales:

. t_language: <language>

. tc_languaje: </language>

. t_webMaster: <webMaster>

. tc_webMaster: </webMaster>

. t_ultEdit: <lastBuildDate>

. tc_ultEdit: </lastBuildDate>

. t_autor: <author>

. tc_autor: </author>

. t_fecha: <pubDate>

. tc_fecha: </pubDate>

Analizador Léxico (Lexer o Scanner)

Definición de Token, Lexema y Patrón (con ejemplos)

Creemos conveniente para una mejor comprensión del tema, dejar una breve definición con nuestra palabras de:

- *Token Léxico o Token:* Es una unidad indivisible con significado único dentro del lenguaje, por lo cual, es la salida del analizador léxico.
  Ejemplos de Tokens:

Categoría	Ejemplos
Delimitadores	( ) , ; : [ ]
Palabras reservadas	while true do if for
Identificadores	Index f isEven
Números enteros	3 -4 55 0 7658
Números flotantes	4.5 .3 0.5 8.4e-5
Simbolos especiales	+ -* / . = < > <= >= != ++
Cadenas	"Hola mundo!"

- *Patrón:* Es una expresión regular que permite identificar un tipo de token y referenciar al conjunto de todos los tokens que se ajustan a él.
- Lexema: Es una cadena de caracteres que encaja con un patrón que describe un componente léxico, es decir, es como una instancia de un patrón.
  Ejemplo de patrón y lexema:

Tipo de token	Patrón léxico	Ejemplos de lexema	Cadena propia
While	while	while	SI
Enteros	digito+	12 123 0 45	NO
División	/	/	SI
Identificador	letra (letra |digito)*	Index f isEven	NO

Definición del AL

El ***analizador léxico*** es la primera fase un de compilador (programa que traduce código escrito en un lenguaje de programación a otro lenguaje) y lo que hace es recibir como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de tokens (componentes léxicos) o símbolos.
Es decir, es un programa capaz de descomponer una entrada de caracteres (generalmente contenidos en un fichero) en una secuencia ordenada de tokens.

Características del AL

Por lo cual, las acciones que realiza el lexer son:

- Procesar el lenguaje fuente como una secuencia de caracteres, es decir, convertir la secuencia de caracteres en una colección de componentes léxicos con significado único dentro del lenguaje llamados tokens.
- Agrupar la secuencia de caracteres en palabras con significado propio y después lo transforma en una secuencia de terminales.
- Buscar los componentes léxicos o palabras que componen el programa fuente, según reglas o patrones.

Procedimiento del AL

1. Se aplica un método para reconocer los posibles patrones, es decir que, en este caso las expresiones regulares son útiles para describir formalmente los patrones de los tokens.
2. Se reconocen los tokens, es decir que, los autómatas finitos reconocen los lenguajes por medio de dichas expresiones regulares.
3. Se realizan acciones preestablecidas al reconocer el token correspondiente.

Errores detectables por el AL

Algunos ejemplos de errores que detecta nuestro analizador léxico son:

- Caracteres ilegales en el programa fuente, es decir, caracteres inválidos, números malformados, entre otros.
- Errores de ortografía en palabras reservadas.

Por lo cual se puede decir que detecta etiquetas faltantes o mal escritas y también utilización de símbolos no permitidos.

- Control en el número de caracteres que aparecen en un identificador, por ejemplo, exceder el número de caracteres máximo.
- Fin de archivo a mitad de componente léxico.

Modo de Obtención del Interprete

Luego de elegir el lenguaje de programación a utilizar (Python), recurrimos a varios videos tutoriales para así poder obtener una librería llamada PLY, la cual nos permitió realizar el analizador léxico. Después de eso, modificamos el formato y añadimos los tokens específicos de la estructura RSS para que sean reconocidos por nuestro Lexer. Posteriormente añadimos funciones específicas para una mejor ejecución del programa.

Modo de Ejecución del Interprete

Los pasos para la ejecución del interprete son:

1. El usuario ingresará por consola la dirección de la carpeta que contiene los archivos.rss a analizar. Por ejemplo: “C:\Users\Desktop\Lexer\prueba\”.
2. Se interpretarán las sentencias de un archivo pasado como argumento desde la línea de comandos. El archivo deberá tener la extensión “.rss” rss.exe ejemplo.rss.

Logrando que el intérprete:

- Funcione correctamente tanto de forma interactiva como ejecutando las instrucciones desde los archivos de ejemplo.
- Indique como salida si el análisis fue exitoso (archivo correctamente codificado, sin errores), en otro caso indicar los errores existentes (indicando tipo de error, número de línea y cadena que generó el error).

Código del Lexer

```
#importar librerias
import ply.lex as lex
import re
import codecs
import os
import sys
import pathlib

reservadas = ['APERTURA_XML', 'VERSION_XML', 'ENCODING_VERSION', 'CIERRE_XML', 'APERTURA_RSS',
'VERSION_RSS', 'CIERRE_RSS', 'APERTURA_CHANNEL', 'CIERRE_CHANNEL', 'APERTURA_TITLE',
'CIERRE_TITLE', 'APERTURA_LINK', 'CIERRE_LINK', 'APERTURA_DESC', 'CIERRE_DESC',
'APERTURA_CAT', 'CIERRE_CAT', 'APERTURA_COPY', 'CIERRE_COPY', 'APERTURA_IMAG', 'CIERRE_IMAG',
'APERTURA_URL', 'CIERRE_URL', 'APERTURA_HEIGHT', 'CIERRE_HEIGHT', 'APERTURA_WIDTH',
'CIERRE_WIDTH', 'APERTURA_ITEM', 'CIERRE_ITEM', 'APERTURA_LANGUAGE', 'CIERRE_LANGUAGE',
'APERTURA_WEBMASTER', 'CIERRE_WEBMASTER', 'URL', 'APERTURA_ULTEDIT', 'CIERRE_ULTEDIT',
'APERTURA_AUTOR','CIERRE_AUTOR', 'APERTURA_BD', 'CIERRE_BD', 'APERTURA_GUID', 'CIERRE_GUID',
'APERTURA_TTL', 'CIERRE_TTL',
 ]

tokens = reservadas+['TXT', 'NUM','IGUAL', 'MENOR','MAYOR','DOBLE_COMILLA','COMA',
'PUNTO','DOBLE_PUNTO','DIVISOR', 'ESPACIO', 'MULT','PUNTO_COMA','PARENT_I',
'PARENT_D','PREG_I','PREG_D', 'SALTO', 'MAS', 'A','E','I','O','U', 'SANGRIA', 'MENOS',
]

#tokens xml
t_APERTURA_XML = r'\<\?xml'
t_VERSION_XML = r'\ version="1.0"'
t_ENCODING_VERSION= r'\ encoding="UTF-8"'
t_CIERRE_XML= r'\?\>'
#tokens rss
t_APERTURA_RSS=r'\<rss'
t_VERSION_RSS=r'\ version="2.0">'
t_CIERRE_RSS=r'\</rss>'
#tokens tags generales
t_APERTURA_CHANNEL = r'\<channel>'
t_CIERRE_CHANNEL = r'\</channel>'
t_APERTURA_TITLE = r'\<title>'
t_CIERRE_TITLE = r'\</title>'
t_APERTURA_LINK=r'\<link>'
t_CIERRE_LINK=r'\</link>'
t_APERTURA_DESC=r'\<description>'
t_CIERRE_DESC=r'\</description>'
```

```
t_APERTURA_CAT=r'\<category>'
t_CIERRE_CAT=r'\</category>'
t_APERTURA_COPY=r'\<copyright>'
t_CIERRE_COPY=r'\</copyright>'
t_APERTURA_IMAG=r'\<image>'
t_CIERRE_IMAG=r'\</image>'
t_APERTURA_URL=r'\<url>'
t_CIERRE_URL=r'\</url>'
t_APERTURA_HEIGHT=r'\<height>'
t_CIERRE_HEIGHT=r'\</height>'
t_APERTURA_WIDTH=r'\<width>'
t_CIERRE_WIDTH=r'\</width>'
t_APERTURA_ITEM=r'\<item>'
t_CIERRE_ITEM=r'\</item>'
t_APERTURA_LANGUAGE=r'\<language>'
t_CIERRE_LANGUAGE=r'\</language>'
t_APERTURA_WEBMASTER=r'\<webMaster>'
t_CIERRE_WEBMASTER=r'\</webMaster>'
t_APERTURA_ULTEDIT=r'\<pubDate>'
t_CIERRE_ULTEDIT=r'\</pubDate>'
t_APERTURA_AUTOR=r'\<author>'
t_CIERRE_AUTOR=r'\</author>'
t_APERTURA_BD=r'\<lastBuildDate>'
t_CIERRE_BD=r'\</lastBuildDate>'
t_APERTURA_GUID=r'\<guid>'
t_CIERRE_GUID=r'\</guid>'
t_APERTURA_TTL=r'\<ttl>'
t_CIERRE_TTL=r'\</ttl>'

t_ignore = '\t'
#definición de tokens simbolos
t_IGUAL = r'='
t_MENOR = r'<'
t_MAYOR = r'>'
t_DOBLE_COMILLA = r'"'
t_COMA = r','
t_PUNTO = r'\.'
t_DOBLE_PUNTO = r':'
t_DIVISOR  = r'/'
t_ESPACIO = r'\ '
t_MULT = r'\*'
t_PUNTO_COMA = r';'
t_PARENT_I = r'\('
t_PARENT_D = r'\)'
```

```
t_PREG_I = r'\¿'
t_PREG_D = r'\?'
t_MAS = r'\+'
t_MENOS = r'\-'
t_A= r'\á'
t_E= r'\é'
t_I= r'\í'
t_O= r'\ó'
t_U= r'\ú'
t_SALTO= r'\n'
t_SANGRIA= r'\t'


#definición de tokens de cadena de texto
def t_TXT(t):
      r'[a-zA-Z][a-zA-Z]*'
      if t.value.upper() in reservadas:
            t.value = t.value.upper()
            t.type = t.value

      return t

#definición de tokens de cadena de numeros
def t_NUM(t):
      r'\d+'
      t.value = int(t.value)
      return t
#definición de tokens de cadena de URL
def t_URL(t):
  r'http[s]?://(?:[a-zA-Z.0-9/_=?:#&$-]*)'
  t.type='URL'
  return t

#definición de salto de pagina
def t_newline(t):
      r'\n+'
      t.lexer.lineno += len(t.value)

def t_COMMENT(t):
      r'\#.*'
      pass

def t_error(t):
      print (" LexToken(SALTODEPAG)%s'" % t.value[0])
```

```
      t.lexer.skip(1)

def buscarFicheros(directorio):
      ficheros = []
      numArchivo = ''
      respuesta = False
      cont = 1
  #lo que haya dentro del directorio
      for base, dirs, files in os.walk(directorio):
            ficheros.append(files)

      for file in files:
            print (str(cont)+". "+file)
            cont = cont+1

      while respuesta == False:
            numArchivo = input('\nIngrese numero de archivo de prueba: ')
            for file in files:
                  if file == files[int(numArchivo)-1]:
                        respuesta = True
                        break

      return files[int(numArchivo)-1]

directorio = '/home/runner/Lexer/test/'
archivo = buscarFicheros(directorio)
test = directorio+archivo

#permite leer archivos con tildes
fp = codecs.open(test,"r","utf-8")

#lee fp
cadena = fp.read()
fp.close()

analizador = lex.lex()

analizador.input(cadena)

while True:
      tok = analizador.token()
      if not tok : break
      print (tok)
```

Ejemplos

```
Ingresar carpeta donde se encuentran las pruebas rss
/home/runner/Lexer/test/
1. prueba1.rss
2. prueba2.rss
3. prueba2 (copy).rss

Ingrese numero de archivo de prueba: 1
LexToken(APERTURA_XML,'<?xml',1,0)
LexToken(VERSION_XML,' version="1.0"',1,5)
LexToken(ENCODING_VERSION,' encoding="UTF-8"',1,19)
LexToken(ESPACIO,' ',1,36)
LexToken(CIERRE_XML,'?>',1,37)
'LexToken(SALTODEPAG)
LexToken(APERTURA_RSS,'<rss',2,41)
LexToken(VERSION_RSS,' version="2.0">',2,45)
'LexToken(SALTODEPAG)
LexToken(APERTURA_CHANNEL,'<channel>',3,62)
'LexToken(SALTODEPAG)
LexToken(APERTURA_TITLE,'<title>',4,73)
LexToken(TXT,'RSS',4,80)
LexToken(ESPACIO,' ',4,83)
LexToken(TXT,'de',4,84)
LexToken(ESPACIO,' ',4,86)
LexToken(TXT,'la',4,87)
LexToken(ESPACIO,' ',4,89)
LexToken(TXT,'c',4,90)
LexToken(A,'á',4,91)
LexToken(TXT,'tedra',4,92)
LexToken(ESPACIO,' ',4,97)
LexToken(TXT,'de',4,98)
LexToken(ESPACIO,' ',4,100)
LexToken(TXT,'Sintaxis',4,101)
LexToken(ESPACIO,' ',4,109)
LexToken(TXT,'y',4,110)
LexToken(ESPACIO,' ',4,111)
LexToken(TXT,'Sem',4,112)
```

```
LexToken(A,'á',4,115)
LexToken(TXT,'ntica',4,116)
LexToken(ESPACIO,' ',4,121)
LexToken(TXT,'de',4,122)
LexToken(ESPACIO,' ',4,124)
LexToken(TXT,'Lenguajes',4,125)
LexToken(ESPACIO,' ',4,134)
LexToken(CIERRE_TITLE,'</title>',4,135)
'LexToken(SALTODEPAG)
LexToken(APERTURA_LINK,'<link>',5,145)
LexToken(TXT,'https',5,151)
LexToken(DOBLE_PUNTO,':',5,156)
LexToken(DIVISOR,'/',5,157)
LexToken(DIVISOR,'/',5,158)
LexToken(TXT,'frre',5,159)
LexToken(PUNTO,'.',5,163)
LexToken(TXT,'cvg',5,164)
LexToken(PUNTO,'.',5,167)
LexToken(TXT,'utn',5,168)
LexToken(PUNTO,'.',5,171)
LexToken(TXT,'edu',5,172)
LexToken(PUNTO,'.',5,175)
LexToken(TXT,'ar',5,176)
LexToken(DIVISOR,'/',5,178)
LexToken(TXT,'course',5,179)
LexToken(DIVISOR,'/',5,185)
LexToken(TXT,'view',5,186)
LexToken(PUNTO,'.',5,190)
LexToken(TXT,'php',5,191)
LexToken(PREG_D,'?',5,194)
LexToken(TXT,'id',5,195)
LexToken(IGUAL,'=',5,197)
LexToken(NUM,399,5,198)
LexToken(CIERRE_LINK,'</link>',5,201)
'LexToken(SALTODEPAG)
LexToken(APERTURA_DESC,'<description>',6,210)
```

Analizador Sintáctico (Parser)

Definición

Creemos conveniente para una mejor comprensión del tema, dejar una breve definición:

Un ***analizador sintáctico*** o parser es un programa que normalmente es parte de un compilador. El compilador se asegura de que el código se traduce correctamente a un lenguaje ejecutable. La tarea del analizador es, en este caso, la descomposición y transformación de las entradas en un formato utilizable para su posterior procesamiento. Se analiza una cadena de instrucciones en un lenguaje de programación y luego se descompone en sus componentes individuales.

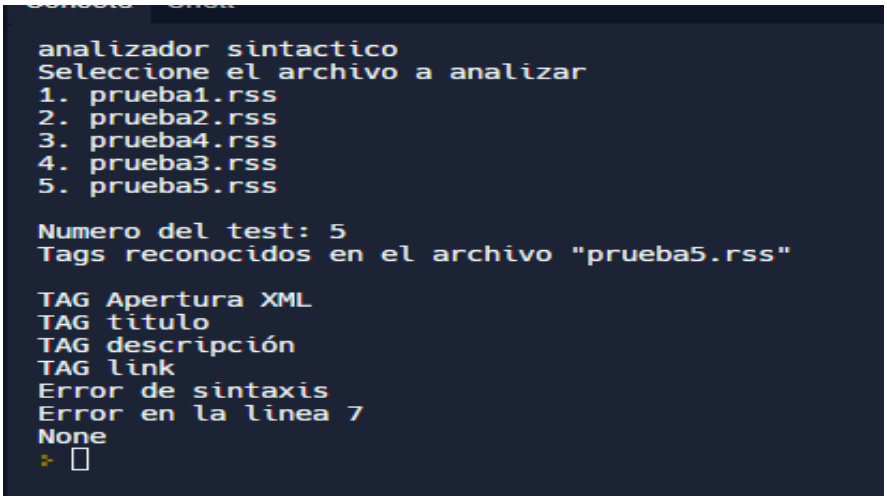
Cómo funciona un AS

Para analizar un texto, por ejemplo, los analizadores suelen utilizar un analizador léxico separado (llamado lexer), que descompone los datos de entrada en fichas (símbolos de entrada como palabras). Los Lexers son por lo general máquinas finitas, que siguen la gramática regular y por lo tanto aseguran un desglose adecuado. Los tokens obtenidos de esta manera sirven como caracteres de entrada para el analizador sintáctico.

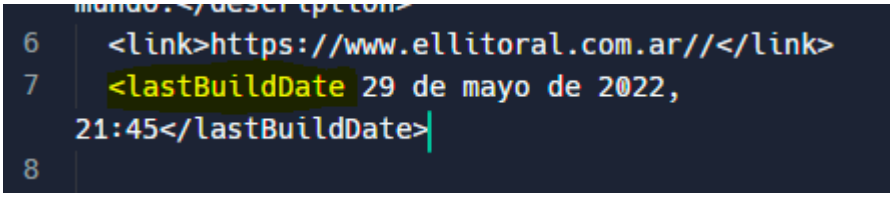
Errores detectables por el AS

Un ejemplos de error que detecta nuestro ***Analizador Sintáctico*** es:

- Al intentar realizar la prueba numera 5, nos detecta un error en la línea 7.



- Si vamos al código que contiene la prueba número 5, podremos comprobar que efectivamente existe un error de sintaxis en la línea 7: No se cerró el Tag de "lastBuildDate"



Modo de Obtención del Intérprete

Luego de realizar y corregir el lexer, ya que en la segunda entrega tuvo algunos errores de ejecución, proseguimos con la codificación del parser utilizando la misma librería (PLY) la cual nos permitió realizar el analizador sintáctico. De esta forma, adaptamos y definimos las reglas de gramática en RSS.

Modo de Ejecución del Intérprete

Para ejecutar el analizador sintáctico, se lo puede ejecutar directamente a través del archivo PARSERV1.exe, y del mismo modo que el leer, debemos ingresar por teclado la dirección de donde se encuentran los archivos de prueba, en la cual luego de realizar el análisis léxico, nos da el resultado de si está correctamente ordenado. Además, como adicional, genera un archivo 'prueba.html' donde se traduce los tags RSS a HTML.

Código del Parser

```
import ply.yacc as yacc
#importo las mismas librerias que el lexer
import os
import codecs
import re
from lex import tokens
from sys import stdin
import msvcrt


#Sigma
def p_sigma(p):
     '''sigma : aperturaxml aperturarss'''
     print ("\nFelicidades")
     print ("Su codigo es sintacticamente correcto")

#xml
def p_aperturaxml(p):
     '''aperturaxml : APERTURA_XML VERSION_XML ENCODING_VERSION CIERRE_XML'''
     print ("TAG Apertura XML")


#rss
def p_aperturarss(p):
     '''aperturarss : APERTURA_RSS VERSION_RSS canal CIERRE_RSS'''
     print ("TAG Apertura RSS")

#tag CHANNEL
def p_canal_basico(p):
     '''canal : APERTURA_CHANNEL titulo LINK desc item CIERRE_CHANNEL'''
     print ("TAG channel")

def p_canal_basico2(p):
     '''canal : APERTURA_CHANNEL titulo desc LINK item CIERRE_CHANNEL'''
     print ("TAG channel")
```

```
def p_canal_opcionales(p):
     '''canal : APERTURA_CHANNEL titulo LINK desc opc item CIERRE_CHANNEL'''
     print ("TAG channel")

def p_canal_opcionales2(p):
     '''canal : APERTURA_CHANNEL titulo desc LINK opc item CIERRE_CHANNEL'''
     print ("TAG channel")

def p_item(p):
  '''item : APERTURA_ITEM titulo LINK desc CIERRE_ITEM'''
  print("TAG ITEM")

def p_item2(p):
  '''item : APERTURA_ITEM titulo desc LINK CIERRE_ITEM'''
  print("TAG ITEM")
def p_item_recursivo(p):
  '''item : APERTURA_ITEM titulo LINK desc CIERRE_ITEM item'''
  print("TAG ITEM")

def p_item_recursivo2(p):
  '''item : APERTURA_ITEM titulo desc LINK CIERRE_ITEM item'''
  print("TAG ITEM")

def p_item_opcionales(p):
  '''item : APERTURA_ITEM titulo LINK desc opc CIERRE_ITEM'''
  print("TAG ITEM")

def p_item_opcionales2(p):
  '''item : APERTURA_ITEM titulo desc LINK opc CIERRE_ITEM'''
  print("TAG ITEM")

def p_item_opcionales_recursivo(p):
  '''item : APERTURA_ITEM titulo LINK desc opc CIERRE_ITEM item'''
  print("TAG ITEM")
def p_item_opcionales_recursivo2(p):
  '''item : APERTURA_ITEM titulo desc LINK opc CIERRE_ITEM item'''
  print("TAG ITEM")
```

```
def p_titulo(p):
     '''titulo : APERTURA_TITLE TXT CIERRE_TITLE'''
     print ("TAG titulo")

def p_LINK(p):
     '''LINK : APERTURA_LINK URL CIERRE_LINK'''
     print ("TAG link")

def p_desc(p):
     '''desc : APERTURA_DESC TXT CIERRE_DESC'''
     print ("TAG descripción")

#                    TAGS OPCIONALES
#-------------------------------------------------------------#
#TAG CATEGORIA
def p_opc1(p):
     '''opc : categoria'''

def p_categoria(p):
     '''categoria : APERTURA_CAT TXT CIERRE_CAT'''
     print ("TAG categoria")

#TAG COPYRIGHT
def p_opc2(p):
     '''opc : copyright'''

def p_copyright(p):
     '''copyright : APERTURA_COPY TXT CIERRE_COPY'''
     print ("TAG copyright")

#TAG IMAGE
def p_opc3(p):
     '''opc : image'''

def p_image1(p):
     '''image : APERTURA_IMAG URL titulo LINK CIERRE_IMAG'''
     print ("TAG image")
```

```
#imagen con elementos opcionales
def p_image2(p):
     '''image : APERTURA_IMAG URL titulo LINK opcimag CIERRE_IMAG'''
     print ("TAG image")

#elementos opcionales de imagen (largo-ancho, largo, ancho)
def p_opcimag1(p):
     '''opcimag : height width'''

def p_opcimag2(p):
     '''opcimag : height'''

def p_opcimag3(p):
     '''opcimag : width'''

def p_height(p):
     '''height : APERTURA_HEIGHT NUM CIERRE_HEIGHT'''

def p_width(p):
     '''width : APERTURA_WIDTH NUM CIERRE_WIDTH'''

def p_opc4(p):
     '''opc : lenguaje'''

def p_lenguaje(p):
     '''lenguaje : APERTURA_LANGUAGE TXT CIERRE_LANGUAGE'''
     print ("TAG language")

def p_opc5(p):
     '''opc : webmaster'''

def p_WEBMASTER(p):
     '''webmaster : APERTURA_WEBMASTER TXT CIERRE_WEBMASTER'''
     print ("TAG webmaster")

def p_opc6(p):
     '''opc : ultedit'''
```

```
def p_pubdate(p):
     '''ultedit : APERTURA_ULTEDIT TXT CIERRE_ULTEDIT'''
     print ("TAG pubdate")

def p_opc7(p):
     '''opc : autor'''

def p_autor(p):
     '''autor : APERTURA_AUTOR TXT CIERRE_AUTOR'''
     print ("TAG author")

def p_opc8(p):
     '''opc : lastbuilddate'''

def p_lastbuilddate(p):
     '''lastbuilddate : APERTURA_BD TXT CIERRE_BD'''
     print ("TAG lastBuildDate")

def p_opc9(p):
     '''opc : guid'''

def p_guid(p):
     '''guid : APERTURA_GUID TXT CIERRE_GUID'''
     print ("TAG guid")

def p_opc10(p):
     '''opc : ttl'''

def p_ttl(p):
     '''ttl : APERTURA_TTL TXT CIERRE_TTL'''
     print ("TAG ttl")
# fin elementos opcionales ----------------------------------------#



def p_error(p):
     print ("Error de sintaxis "), p
```

```
      print ("Error en la linea "+str(p.lineno))

def buscarFicheros(directorio):
      ficheros = []
      numArchivo = ''
      respuesta = False
      cont = 1

      for base, dirs, files in os.walk(directorio):
            ficheros.append(files)

      for file in files:
            print (str(cont)+". "+file)
            cont = cont+1

      while respuesta == False:
            numArchivo = input('\nNumero del test: ')
            for file in files:
                  if file == files[int(numArchivo)-1]:
                        respuesta = True
                        break

      print ("Tags reconocidos en el archivo \"%s\" \n" %files[int(numArchivo)-1])

      return files[int(numArchivo)-1]

print ("analizador sintactico")
print ("Por favor, ingrese por teclado la dirección de la carpeta en donde se encuntra
los archivos.rss")
directorio = input() +'/'
archivo = buscarFicheros(directorio)
test = directorio+archivo
fp = codecs.open(test,"r","utf-8")
cadena = fp.read()
fp.close()

parser = yacc.yacc()
result = parser.parse(cadena)
```

```
print (result)


# archivo input rss
fin = open(test)
# archivo output html
fout = open(("prueba.html"), "wt")
# iteracion
buffer = fin.read()
     # reemplazo
buffer = buffer.replace('<title>', '<h1>')
buffer = buffer.replace('</title>', '</h1>')
buffer = buffer.replace('<description>', '<p>')
buffer = buffer.replace('</description>', '</p>')
buffer = buffer.replace('<link>', '<a>')
buffer = buffer.replace('</link>', '</a>')
buffer = buffer.replace('<url>', '<a>')
buffer = buffer.replace('</url>', '</a>')

fout.write(buffer)
fin.close()
fout.close()



msvcrt.getch()
```

Reglas de gramática del Parser

Rule 0 S' -> sigma

Rule 1 sigma -> aperturaxml aperturarss

Rule 2 aperturaxml -> APERTURA_XML VERSION_XML ENCODING_VERSION CIERRE_XML

Rule 3 aperturarss -> APERTURA_RSS VERSION_RSS canal CIERRE_RSS

Rule 4 canal -> APERTURA_CHANNEL titulo LINK desc item CIERRE_CHANNEL

Rule 5 canal -> APERTURA_CHANNEL titulo LINK desc opc item CIERRE_CHANNEL

Rule 6 item -> APERTURA_ITEM titulo LINK desc CIERRE_ITEM

Rule 7 item -> APERTURA_ITEM titulo LINK desc CIERRE_ITEM item

Rule 8 item -> APERTURA_ITEM titulo LINK desc opc CIERRE_ITEM

Rule 9 item -> APERTURA_ITEM titulo LINK desc opc CIERRE_ITEM item

Rule 10 titulo -> APERTURA_TITLE TXT CIERRE_TITLE

Rule 11 LINK -> APERTURA_LINK URL CIERRE_LINK

Rule 12 desc -> APERTURA_DESC TXT CIERRE_DESC

Rule 13 opc -> categoria

Rule 14 categoria -> APERTURA_CAT TXT CIERRE_CAT

Rule 15 opc -> copyright

Rule 16 copyright -> APERTURA_COPY TXT CIERRE_COPY

Rule 17 opc -> image

Rule 18 image -> APERTURA_IMAG URL titulo LINK CIERRE_IMAG

Rule 19 image -> APERTURA_IMAG URL titulo LINK opcimag CIERRE_IMAG

Rule 20 opcimag -> height width

Rule 21 opcimag -> height

Rule 22 opcimag -> width

Rule 23 height -> APERTURA_HEIGHT NUM CIERRE_HEIGHT

Rule 24 width -> APERTURA_WIDTH NUM CIERRE_WIDTH

Rule 25 opc -> lenguaje

Rule 26 lenguaje -> APERTURA_LANGUAGE TXT CIERRE_LANGUAGE

Rule 27 opc -> webmaster

Rule 28 webmaster -> APERTURA_WEBMASTER TXT CIERRE_WEBMASTER

Rule 29 opc -> ultedit

Rule 30 ultedit -> APERTURA_ULTEDIT TXT CIERRE_ULTEDIT

Rule 31 opc -> autor

Rule 32 autor -> APERTURA_AUTOR TXT CIERRE_AUTOR

Rule 33 opc -> lastbuilddate

Rule 34 lastbuilddate -> APERTURA_BD TXT CIERRE_BD

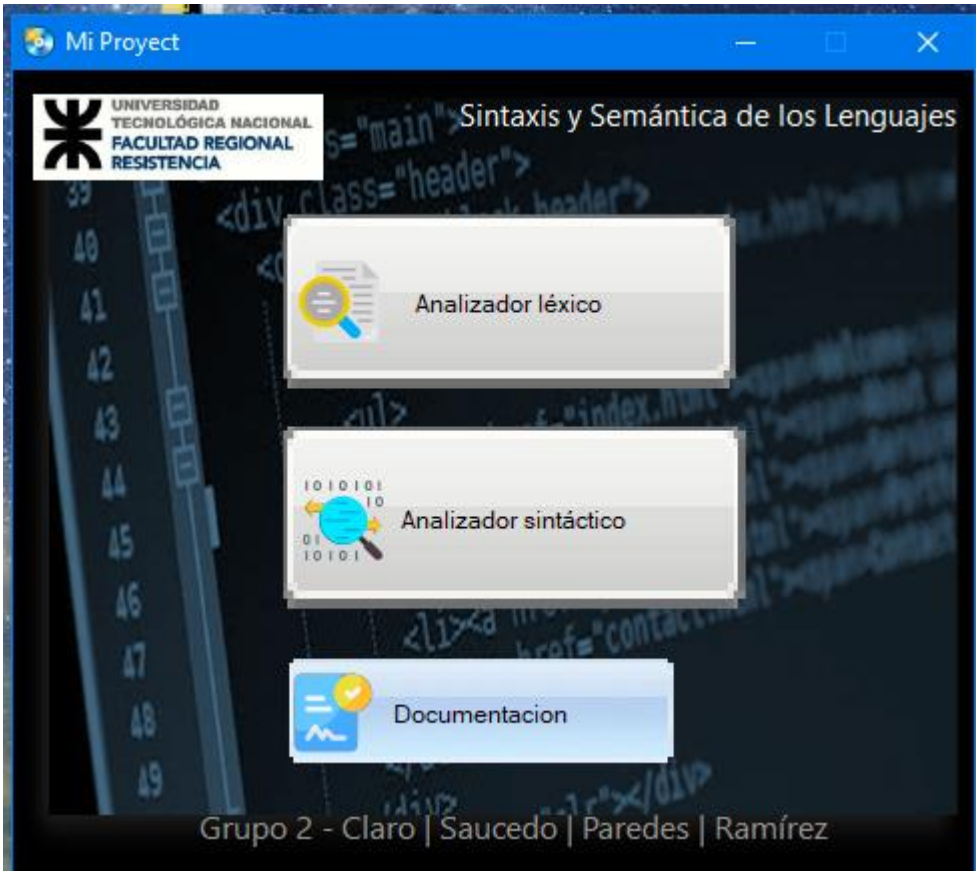
Rule 35 opc -> guid

Rule 36 guid -> APERTURA_GUID TXT CIERRE_GUID

Rule 37 opc -> ttl

Rule 38 ttl -> APERTURA_TTL TXT CIERRE_TTL

Interfaz gráfica



Como adicional en esta entrega, decidimos implementar una simple interfaz gráfica ejecutable, que funciona como acceso directo de ambos archivos (LEXERV1.exe y PARSERV1.exe). La misma se encuentra en la carpeta ‘’bin’.

Conclusión

En la primera entrega, aprendimos a aplicar el diseño de gramáticas visto en clase a un ejemplo concreto, en este caso, el del formato RSS. Este formato nos resultó sencillo de entenderlo, ya que todos los integrantes del grupo teníamos, al menos un pequeño conocimiento previo del mismo. No tuvimos grandes dificultades a la hora de diseñar la gramática, salvo en la aplicación de la recursión en las producciones gramaticales y de organizar un horario en común para dedicarnos al desarrollo del trabajo práctico.

En la segunda entrega aprendimos a diseñar un analizador léxico mediante un lenguaje de programación, en este caso Python, ya que nos pareció el más práctico al momento de realizar la codificación. En el proceso aprendimos a utilizar dicho lenguaje, ya que la mayoría de integrantes del grupo no sabían utilizarlo y fue una experiencia muy satisfactoria ya que pudimos observar la aplicación de lo aprendido en clase ese momento. Algunas complicaciones que se nos presentaron fueron con respecto a las instalaciones de las diferentes librerías a utilizar, como así también errores de sintaxis con respecto a algunas líneas del código. Para solucionar esto recurrimos a diferentes fuentes y vídeos en donde pudimos corregir dichos problemas. Un problema puntual que se nos presentó y no supimos cómo solucionarlo es el de obtener el directorio donde se encuentran los archivos .rss de prueba automáticamente, por lo que en el programa dicho directorio se debe ingresar manualmente.

A lo largo de la construcción de este trabajo final, se pudo poner en práctica lo aprendido este cuatrimestre. Empezando con la realización de una gramática LDC hasta implementar todo para la construcción de un compilador.

En el proceso de codificación tuvimos varios inconvenientes, debido a no tener mucha experiencia previa Py. Por lo anterior mencionado hemos perdido mucho tiempo en investigar cómo funcionaba el lenguaje.

Luego de muchas pruebas y errores pudimos realizar el proyecto, lo cual se creyó que sería imposible.

La realización de este proyecto nos permitió comprender el funcionamiento de los analizadores e intérpretes, logrando así ampliar nuestro conocimiento en programación/desarrollo de software y familiarizarnos aún más con ello.

Bibliografía o Referencias Web

Wikipedia - Comparison of parser generators:
https://en.wikipedia.org/wiki/Comparison_of_parser_generators

YouTube - ¿Qué es HTML? - 10 cosas que debes saber:

https://www.youtube.com/watch?v=tPzq8IufGxE

YouTube - Curso Básico de HTML desde 0 – Introducción:

https://www.youtube.com/watch?v=cqMfPS8jPys&list=PLhSj3UTs2_yVHt2DgHky_MzzRC58UHE4z

YouTube - ¿Qué es RSS?:

https://www.youtube.com/watch?v=t5m5lKx6rEo vídeo de qué es RSS

W3 Schools - XML RSS:

https://www.w3schools.com/xml/xml_rss.asp

Youtube - Tutorial #Como crear un RSS:

https://www.youtube.com/watch?v=0uNZDzi_jYs

W3C - W3C Feed Validation Service3:

https://validator.w3.org/feed/

YouTube - ¿QUÉ ES XML Y PARA QUÉ SE USA?:

https://www.youtube.com/watch?v=AZihBEg8VBk

YouTube - What is XML | XML Beginner Tutorial | Learn XML with Demo in 10 min:

https://www.youtube.com/watch?v=1JblVElt5K0

YouTube - Introducción a XML - 1 - Tutorial XML básico en español:

https://www.youtube.com/watch?v=PxGICnkFZJU&list=PLM-p96nOrGcYb96AMy3VdUN8fo-sVAx7K

YouTube - How to Create Simple XML Document:

https://www.youtube.com/watch?v=i9E_mOptuEo

Landofcode - RSS tags reference:

http://www.landofcode.com/rss-reference/rss-tags.php

https://youtu.be/TG0qRDrUPpA Qué es el Lexer, para que sirve, cómo empezar, ejemplos de lexer

https://www.youtube.com/watch?v=gWrmCOTrtrs Lexer en Python

https://programmerclick.com/article/4860986790/ Escribir Analizador Léxico a mano
https://programmerclick.com/article/4860986790/ AL en Python
http://acodigo.blogspot.com/2014/02/analizador-lexico.html Analizador Léxico en C
https://www.youtube.com/watch?v=cRvmSMWLZlY
https://es.wikipedia.org/wiki/Lex_(inform%C3%A1tica) YACC Analizador Léxico Programa en C
https://youtu.be/Hh49BXmHxX8 Calculadora en PY (analizador léxico)
https://youtu.be/orI232lQv6U Calculadora en PY (analizador léxico)
https://youtu.be/Zbk0lic04SI Calculadora en PY (analizador léxico)
https://youtu.be/DOuWQNln9wc Calculadora en PY (analizador léxico)
https://youtu.be/WrYTi90ey0E Calculadora en PY (analizador léxico)