UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL RESISTENCIA

# Trabajo Práctico Integrador

## Diseño e implementación de Lexer y Parser
## Lenguaje DocBook/XML

**Carrera:** Ingeniería en Sistemas de Información

**Asignatura:** Sintaxis y Semántica de los Lenguajes

**Curso académico:** 2023

**Primer cuatrimestre**

**Grupo Nº:** 12

**Integrantes:**

- Beron de Astrada, Santiago Agustín
- Bregant, Joaquín Conrado
- Niveiro, Gianfranco

Tercer entrega: 03/07/2023
Versión de la documentación: 3.0

# ÍNDICE

# Introducción

Este es un trabajo realizado durante el cursado de la asignatura *Sintaxis y Semántica de los Lenguajes* correspondiente a la carrera de Ingeniería en Sistemas de Información de la Universidad Tecnológica Nacional, Facultad Regional Resistencia. El mismo tiene como objetivo el diseño, creación e implementación de un Lexer y un Parser que pueda interpretar correctamente el lenguaje *Docbook/XML*, para luego traducirlo al lenguaje *HTML*.

Para la realización de esta tarea se consideró adecuado por los miembros del grupo, utilizar el lenguaje de programación de alto nivel *Python* debido a su versatilidad y al hecho de que todos los integrantes poseen conocimiento alguno sobre el mismo. Adicionalmente se utilizará la dependencia PLY, la cual permitirá una fácil integración con la gramática definida.

# Lenguaje Docbook/XML

Es un lenguaje de marcas que se utiliza para definir solo la estructura de un documento, y no su formato. Los documentos DocBook no describen ni la apariencia ni la presentación de sus contenidos, sino únicamente el sentido de dichos contenidos. Contiene básicamente texto. Ahora bien, hay texto que sirve para indicar elementos, describir el contenido y propiedades de la página. Es decir, un documento contiene lo que se denominan etiquetas o *tags*. Las etiquetas sirven para delimitar y determinar el tipo de elemento que representa un fragmento de texto en la página; por ejemplo, un párrafo o una tabla son elementos que pueden conformar una página. Incluso hay elementos que contienen otros elementos; por ej, las tablas constan de filas y las filas de celdas.

## Componentes léxicos o tokens

<u>Observaciones</u>

- El texto dentro de las páginas webs no respeta los espacios en blanco ni tabulaciones que se coloquen en el código a la hora de mostrar el contenido por pantalla. Solo se considera el primer espacio en blanco, el resto se elimina.
- Las etiquetas distinguen entre mayúsculas y minúsculas.

Tipos de Datos

El tipo de dato Cadena y URL se tratarán de manera especial

- Cadena:  Estará compuesta por letras, números, signos de puntuación, caracteres especiales,
- URL: Los únicos caracteres permitidos en URL son letras, números, guión medio, guión bajo y punto. además de los caracteres reservados: # , /, :, &, ?, =

Reglas generales de las Etiquetas

- La mayoría de etiquetas reservadas requieren una apertura y un cierre de la misma. Al inicializarse estarán encerradas por los símbolos *<nombreDeEtiqueta>*  y al finalizar su uso deberán cerrarse con *</nombreDeEtiqueta>*. En caso de que la etiqueta no requiera cierre, se colocará solo la de apertura.
- Algunas etiquetas podrán contener determinados atributos que se colocarán luego del nombre de la etiqueta y antes del símbolo “>” separados por espacios con el formato:
  *nombreAtributo=”valorAtributo”*
- Las etiquetas serán utilizadas en minúscula.

Etiqueta **DOCTYPE:** Indica el tipo de archivo XML que se utiliza. Para el desarrollo de este trabajo es *article.*

- **Article**: Todo artículo está encerrado dentro de la apertura y el cierre de esa etiqueta. Marca el principio y fin del mismo (marca el elemento raíz de un documento). Aparece por única vez, puede o no poseer información o título, y a su vez debe obligatoriamente contener alguna otra etiqueta de texto.

- **Info:** Lo usamos para agregar si necesitamos la información de “encabezado”. Tiene que contener como mínimo alguna información sobre el autor

- **Title**: Se utiliza para indicar los títulos de una sección en un documento o de un bloque dentro del mismo.
- **MediaObject:** Imágenes y multimedia
- **ItemizedList:** crea una lista no ordenada, la cual agrupa párrafos alineados a través de viñetas.

- **Important:** indica que un fragmento de texto es importante.

# Gramática

## Símbolos NO terminales (N)

NT_ARTICLE: sentencia de tipo *article.*
NT_SECTION: sentencia de tipo *section.*
NT_SIMPLESEC: sentencia de tipo *simplesec.*
CONT_A_S: elementos opcionales y obligatorios para *article* y *section.*
CONT_SS: elementos opcionales y obligatorias para *simplesec.*
SECTIONS: permite agregar *section* y *simplesec* recursivamente.
CONT_1: conjunto de etiquetas permitidas para *article*, *section*, *simplesec*, *itemizedlist* e *important*.
NT_INFO: sentencia de tipo *info*.
CONT_INFO: permite agregar etiquetas recursivamente para *info*.
ELEM_INFO: etiquetas permitidas para *info*.
NT_TITLE: sentencia de tipo *title.*
CONT_TITLE: permite agregar etiquetas recursivamente para *title.*
ELEM_TITLE: etiquetas permitidas para *title.*
PARAS: permite agregar recursivamente uno o más *para* y *simpara.*
NT_PARA: sentencia de tipo *para.*
CONT_PARA: permite agregar etiquetas recursivamente para la sentencia de tipo *para.*
ELEM_PARA: etiquetas permitidas para la sentencia de tipo *para.*
NT_SIMPARA: sentencia de tipo *simpara.*
NT_ABSTRACT: sentencia de tipo *abstract.*
NT_ITEMIZEDLIST: sentencia de tipo *itemizedlist.*
LISTITEM: permite agregar recursivamente etiquetas permitidas para *itemizedlist.*
NT_MEDIAOBJECT: sentencia de tipo *mediaobject*.
CONT_MEDIAOBJECT: etiquetas permitidas para *mediaobject*.
NT_IMAGEOBJECT: sentencia de tipo *imageobject*.
NT_VIDEOOBJECT: sentencia de tipo *videoobject*.
NT_AUTHOR: sentencia de tipo *author.*
CONT_AUTHOR: etiquetas permitidas para *author.*
NT_ADDRESS: sentencia de tipo *address.*
CONT_ADDRESS: etiquetas permitidas para *address*.

NT_COPYRIGHT: sentencia de tipo *copyright.*
CONT_2: conjunto de etiquetas permitidas para *simpara*, *emphasis*, *link* y *comment*.
CONT_SECL: permite agregar recursivamente etiquetas permitidas para *emphasis*, *comment*, *simpara* y *link.*
NT_SIMPARA: sentencia de tipo *simpara.*
NT_EMPHASIS: sentencia de tipo *emphasis.*
NT_COMMENT: sentencia de tipo *comment.*
NT_LINK: sentencia de tipo *link*.
NT_IMPORTANT: sentencia de tipo *important.*
CONT_IMPORTANT: permite agregar recursivamente etiquetas permitidas para *important*.
CONT_3: conjunto de etiquetas permitidas para *firstname*, *surname*, *street*, *city*, *state*, *phone*, *email*, *date, year* y *holder*.
CONT_VAR: permite agregar recursivamente etiquetas que pertenecen al conjunto CONT_3.
NT_FIRSTNAME: sentencia de tipo *firstname*.
NT_SURNAME: sentencia de tipo *surname*.
NT_STREET: sentencia de tipo *street*.
NT_CITY: sentencia de tipo *city*.
NT_STATE: sentencia de tipo *state*.
NT_PHONE: sentencia de tipo *phone*.
NT_EMAIL: sentencia de tipo *email*.
NT_DATE: sentencia de tipo *date.*
NT_YEAR: sentencia de tipo *year.*
NT_HOLDER: sentencia de tipo *holder.*
NT_INFORMALTABLE: sentencia de tipo *informaltable*.
TABLE_MEDIA: permite agregar recursivamente etiquetas *mediaobject*.
TABLE_GROUP: permite agregar recursivamente etiquetas *tgroup*.
NT_TGROUP: sentencia de tipo *tgroup*.
NT_THEAD: sentencia de tipo *thead*.
NT_TFOOT: sentencia de tipo *tfoot*.
NT_TBODY: sentencia de tipo *tbody*.
CONT_T: permite agregar recursivamente etiquetas *row* para las sentencias de tipo *thead*, *tfoot* y *tbody*.
NT_ROW: sentencia de tipo *row*.
CONT_ROW: etiquetas permitidas dentro de *row*.
NT_ENTRY: sentencia de tipo *entry*.

NT_ENTRYTBL: sentencia de tipo *entrytbl*.
CONT_ENTRY: permite agregar recursivamente las etiquetas permitidas para *entry*.

## Símbolos terminales (T)

- Cadena
- <!DOCTYPE article>
- <article>
- </article>
- <section>
- </section>
- <simplesec>
- </simplesec>
- <info>
- </info>
- <title>
- </title>
- <para>
- </para>
- <simpara>
- </simpara>
- <itemizedlist>
- </itemizedlist>
- <author>
- </author>
- <firstname>
- </firstname>
- <surname>
- </surname>
- <address>
- </address>
- <copyright>
- </copyright>
- <emphasis>
- </emphasis>
- <comment>
- </commento>
- <link>
- </link
- <important>
- </important>
- <street>
- </street>
- <city>
- </city>
- <state>
- </state>
- <phone>
- </phone>
- <email>
- </email>
- <date>
- </date>
- <year>
- </year>
- <holder>
- </holder>
- <informaltable>
- </informaltable>
- <tgroup>
- </tgroup>
- <thead>
- </thead>
- <tfoot>
- </tfoot>
- <tbody>
- </tbody>
- <row>
- </row>
- <entrytbl>
- </entrytbl>
- <entry>
- </entry>
- fileref:xlink
- href

## Producciones (P)

Los símbolos escritos en mayúsculas se corresponden con símbolos no terminales de la gramática, y los escritos en minúsculas, con símbolos terminales, excepto por *DOCTYPE* que también es un terminal.

**Σ** → <!DOCTYPE article>**NT_ARTICLE**

**NT_ARTICLE** → <article>**NT_INFO NT_TITLE CONT_A_S**</article> |
<article>**NT_TITLE CONT_A_S**</article> |
<article>**NT_INFO CONT_A_S**</article> |
<article>**CONT_A_S**</article>

**NT_SECTION** → <section>**CONT_A_S**</section> |
<section>**NT_INFO CONT_A_S**</section> |
<section>**NT_TITLE CONT_A_S**</section> |
<section>**NT_INFO NT_TITLE CONT_A_S**</section>

**NT_SIMPLESEC** → <simplesec>**CONT_SS**</simplesec> |
<simplesec>**NT_INFO CONT_SS**</simplesec> |
<simplesec>**NT_TITLE CONT_SS**</simplesec> |
<simplesec>**NT_INFO NT_TITLE CONT_SS**</simplesec>

**CONT_A_S** → **CONT_1** | **CONT_1 CONT_A_S** | **CONT_1 SECTIONS** |
**CONT_1 CONT_A_S SECTIONS**

**SECTIONS** → **NT_SECTION** | **NT_SECTION SECTIONS** | **NT_SIMPLESEC** |
**NT_SIMPLESEC SECTIONS**

**CONT_1** → **NT_ITEMIZEDLIST** | **NT_IMPORTANT** | **NT_PARA** | **NT_SIMPARA** |
**NT_ADDRESS** | **NT_MEDIAOBJECT** | **NT_INFORMALTABLE** | **NT_COMMENT** |
**NT_ABSTRACT**

**NT_INFO** → <info>**CONT_INFO**</info>

**CONT_INFO** → **ELEM_INFO** | **ELEM_INFO CONT_INFO**

**ELEM_INFO** → **NT_MEDIAOBJECT** | **NT_ABSTRACT** | **NT_ADDRESS** | **NT_AUTHOR** |
**NT_DATE** | **NT_COPYRIGHT** | **NT_TITLE**

**NT_ABSTRACT** → **NT_TITLE** | **NT_TITLE PARAS**

**NT_TITLE** → <title>**CONT_TITLE**</title>

**CONT_TITLE** → **ELEM_TITLE** | **ELEM_TITLE CONT_TITLE**

**ELEM_TITLE** → Cadena | **NT_EMPHASIS** | **NT_LINK** | **NT_EMAIL**

**PARAS** → **NT_PARA** | **NT_SIMPARA** | **NT_PARA PARAS** | **NT_SIMPARA PARAS**

**NT_PARA** → <para>**CONT_PARA**</para>

**CONT_PARA** → **ELEM_PARA** | **ELEM_PARA CONT_PARA**

ELEM_PARA → Cadena | NT_EMPHASIS | NT_LINK | NT_EMAIL | NT_AUTHOR | NT_COMMENT | NT_ITEMIZEDLIST | NT_IMPORTANT | NT_ADDRESS | NT_MEDIAOBJECT | NT_INFORMALTABLE

NT_ITEMIZEDLIST → <itemizedlist>LISTITEM</itemizedlist>

NT_MEDIAOBJECT → <mediaobject>NT_INFO CONT_MEDIAOBJECT</mediaobject>| <mediaobject>CONT_MEDIAOBJECT</mediaobject>

CONT_MEDIAOBJECT → NT_IMAGEOBJECT | NT_VIDEOOBJECT | NT_IMAGEOBJECT CONT_MEDIAOBJECT | NT_VIDEOOBJECT CONT_MEDIAOBJECT

NT_IMAGEOBJECT → <imageobject>NT_INFO<imagedata fileref="URL"/></imageobject> | <imageobject><imagedata fileref="URL" /></imageobject>

NT_VIDEOOBJECT → <videoobject>NT_INFO<videodata fileref="URL"/></videoobject> | <videoobject><videodata fileref="URL"/></videoobject>

LISTITEM → <listitem>CONT_ITEM</listitem>

CONT_ITEM → CONT_1 | CONT_1 CONT_ITEM

NT_AUTHOR → <author>CONT_AUTHOR</author>

CONT_AUTHOR → NT_FIRSTNAME | NT_SURNAME | NT_EMAIL | NT_FIRSTNAME NT_SURNAME | NT_FIRSTNAME NT_EMAIL | NT_SURNAME NT_EMAIL | NT_FIRSTNAME NT_SURNAME NT_EMAIL

CONT_SS → CONT_1 | CONT_1 CONT_SS

NT_ADDRESS → <address></address> | <address>CONT_ADDRESS</address>

CONT_ADDRESS → ELEM_ADDRESS | ELEM_ADDRESS CONT_ADDRESS

ELEM_ADDRESS → NT_STREET | NT_CITY | NT_STATE | NT_PHONE | NT_EMAIL |

NT_COPYRIGHT → <copyright>NT_YEAR</copyright> | <copyright>NT_YEAR NT_HOLDER</copyright>

CONT_2 → Cadena | NT_EMPHASIS | NT_LINK | NT_EMAIL | NT_AUTHOR | NT_COMMENT

NT_SIMPARA → <simpara>CONT_SECL</simpara>

NT_EMPHASIS → <emphasis>CONT_SECL</emphasis>

NT_COMMENT → <comment>CONT_SECL</comment>

NT_LINK → <link xlink:href="NT_URL">CONT_SECL</link>

CONT_SECL → CONT_2 | CONT_2 CONT_SECL

NT_IMPORTANT → <important>NT_TITLE CONT_IMPORTANT</important> | <important>CONT_IMPORTANT</important>

CONT_IMPORTANT → CONT_1 | CONT_1 CONT_IMPORTANT

CONT_3 → Cadena | NT_LINK | NT_EMPHASIS | NT_COMMENT

CONT_VAR → CONT_3 | CONT_3 CONT_VAR

NT_FIRSTNAME → <firstname>CONT_VAR</firstname>

NT_SURNAME → <surname>CONT_VAR</surname>

NT_STREET → <street>CONT_VAR</street>

NT_CITY → <city>CONT_VAR</city>

NT_STATE → <state>CONT_VAR</state>

NT_PHONE → <phone>CONT_VAR</phone>

NT_EMAIL → <email>CONT_VAR</email>

NT_DATE → <date>CONT_VAR</date>

NT_YEAR → <year>CONT_VAR</year>

NT_HOLDER → <holder>CONT_VAR</holder>

NT_INFORMALTABLE → <informaltable>TABLE_MEDIA</informaltable> |
<informaltable>TABLE_GROUP</informaltable>

TABLE_MEDIA → NT_MEDIAOBJECT | NT_MEDIAOBJECT TABLE_MEDIA

TABLE_GROUP → NT_TGROUP | NT_TGROUP TABLE_GROUP

NT_TGROUP → <tgroup>NT_THEAD NT_TFOOT NT_TBODY</tgroup> |
<tgroup>NT_THEAD NT_TBODY</tgroup> |
<tgroup>NT_TFOOT NT_BODY</tgroup> | <tgroup>NT_TBODY</tgroup>

CONT_T → NT_ROW | NT_ROW CONT_T

NT_THEAD → <thead>CONT_T</thead>

NT_TFOOT → <tfoot>CONT_T</tfoot>

NT_TBODY → <tbody>CONT_T</tbody>

NT_ROW → <row>CONT_ROW</row>

CONT_ROW → NT_ENTRY | NT_ENTRY CONT_ROW | NT_ENTRYTBL |
NT_ENTRYTBL CONT_ROW

NT_ENTRYTBL → <entrytbl>NT_THEAD NT_TBODY</entrytbl> |
<entrytbl>NT_TBODY</entrytbl>

NT_ENTRY → <entry>CONT_ENTRY</entry>

CONT_ENTRY → Cadena | Cadena CONT_ENTRY | NT_ITEMIZEDLIST |
NT_ITEMIZEDLIST CONT_ENTRY | NT_IMPORTANT | NT_PARA | NT_SIMPARA |
NT_IMPORTANT CONT_ENTRY | NT_PARA CONT_ENTRY | NT_COMMENT |
NT_SIMPARA CONT_ENTRY | NT_COMMENT CONT_ENTRY | NT_ABSTRACT |
NT_ABSTRACT CONT_ENTRY | NT_MEDIAOBJECT CONT_ENTRY |
NT_MEDIAOBJECT

# Documentación

Como bien se explicó en la introducción del presente trabajo, se utilizó la librería PLY del lenguaje de programación Python para el desarrollo del Analizador Léxico y el Analizador Sintáctico.

## Analizador Léxico

### Funcionamiento

Un analizador léxico o “Lexer” es una parte fundamental de un compilador o intérprete utilizado en el proceso de reconocimiento y traducción de un lenguaje específico. Su principal función es escanear el código de entrada (conjunto de sentencias) y dividirlo en unidades léxicas o lexemas, que son elementos más pequeños del lenguaje con significado propio, como palabras claves, identificadores, números, símbolos, operadores, entre otros. Durante el desarrollo de este trabajo se hará referencia a estas unidades léxicas como **tokens** o **etiquetas**.

El analizador léxico realiza esta tarea definiendo un conjunto de reglas específicas, conocidas como expresiones regulares, para cada uno de las etiquetas que forman el lenguaje, describiendo la estructura específica que puede tener cada uno de ellas. Cada vez que encuentran una etiqueta al analizar el código de entrada, el analizador lo convierte en un token, que es una representación interna del lexema con información adicional sobre su tipo y valor. Con la librería PLY esto se hace definiendo previamente el nombre con el que se hará referencia a estas (“etiqueta”) y su correspondiente función con el nombre de “t_etiqueta”. Dicha función contiene la expresión regular que define la forma de la etiqueta.

Una vez definidos los tokens, estos son enviados al componente del Analizador Sintáctico.

### Desarrollo

Una vez comprendido el funcionamiento del analizador léxico se procede con su construcción. Estas son algunas especificaciones o pasos que se consideraron necesarias:

1. **Definición de tokens:** Se identificó el conjunto de etiquetas de Docbook que serán reconocidas por el analizador y se definieron en el archivo del lexer.

```
tokens = [
    # Etiquetas Docbook/XML

    'doctype',

    'article',
    'cierreArticle',

    'section',
    'cierreSection',

    'info',
    'cierreInfo',

    etc...

]
```

2. **Definición de expresiones regulares:** Para cada token definido, se especificó la expresión regular que capture su estructura.

```
def t_doctype(t): r'<!DOCTYPE article>'; return(t);

def t_article(t): r'<article>'; return(t);
def t_cierreArticle(t): r'</article>'; return(t);

def t_section(t): r'<section>'; return(t);
def t_cierreSection(t): r'</section>';  return(t);

def t_info(t): r'<info>'; return(t);
def t_cierreInfo(t): r'</info>'; return(t);

# etc...
```

3. **Ignorar espacios en blanco y comentarios:** El analizador ignora los espacios en blanco y comentarios, ya que no se consideran de una importancia significante en la estructura del programa. Con la librería PLY esto se hace de manera sencilla ya que solo hace falta definir un token con los elementos a ignorar.

```
# token que ignora los saltos de linea y tabulaciones
t_ignore = ' \t\n'
```

4. **Función de análisis:** Toma el código fuente como entrada y produce un archivo de salida con la secuencia de tokens reconocidos, identificando el token definido y su valor.

```python
# Exportacion de TOKENS ENCONTRADOS a un archivo .txt
def exportarTokens(arrAnalizar):
    global contadorErrores
    fileNameExport = f'tokens-analizados.txt'
    with open(fileNameExport, 'w', encoding='UTF8') as f:
        f.write('TOKEN | VALOR\n')
        f.write('-------------\n')
        contador = 0
        for line in arrAnalizar:
            contador += 1
            f.write(f'{contador}- {line[0]}: {line[1]}')
            f.write('\n')
        f.write('-------------\n')
        f.write(f'Total de tokens válidos analizados: {contador}.\n')
        if (contadorErrores > 0):
            f.write(f'Total de tokens NO válidos: {contadorErrores}.')
    f.close()
    if (contadorErrores > 0):
        print('(X) El lexer NO acepta este archivo.')
    else:
        print('(✅) El lexer ACEPTA este archivo.')
    print('(!) Se exportó un .txt con los tokens analizados.')
```

5. **Control y manejo de errores:** Se consideran los errores que pueden ocurrir durante el funcionamiento del analizador, producidos por etiquetas inexistentes, mal escritas, entre otras situaciones. Esto se hace también con una función.

```python
# Funcion que se ejecuta al encontrar un error lexico
def t_error(t):
    global contadorErrores
    print(f'Caracter ilegal! : \'{t.value[0]}\'.')
    print(f'En linea: {t.lineno}. Posición: {t.lexpos}')
    contadorErrores += 1
    t.lexer.skip(1)
```

6. **Pruebas:** Se desarrollaron diferentes pruebas utilizando archivos de entrada variados en contenido de etiquetas, con el objetivo de garantizar el correcto funcionamiento del analizador.

## Analizador Sintáctico

### Funcionamiento

El analizador sintáctico, también conocido como “Parser” es la parte principal en el funcionamiento de un compilador o intérprete, el cual se encarga de analizar la estructura gramatical de un programa o una secuencia de texto, para determinar si sigue las reglas sintácticas del lenguaje en el que está escrito. El parser utiliza como entrada el

conjunto de tokens reconocidos por el lexer para la aplicación de las reglas gramaticales en él definidas.

Existen diferentes tipos de analizadores sintácticos utilizados en el ámbito de la compilación y el procesamiento de lenguajes de programación. Si bien la elección de cada uno de ellos depende de la naturaleza de la gramática del lenguaje y de los requisitos específicos del sistema en el que se implementará el compilador, dos de los mas utilizados son:

- Analizadores sintácticos descendentes (LL): Analizan la entrada de izquierda a derecha y construyen una derivación de arriba hacia abajo (top-down). Son populares debido a su simplicidad y eficiencia. Son fáciles de implementar pero suelen utilizarse para gramáticas sencillas, ya que no suelen ser muy eficientes para gramáticas complejas, además de que pueden sufrir recursividad por izquierda.
- Analizadores sintácticos ascendentes (LR): Analizan la entrada de izquierda a derecha y construyen una derivación de abajo hacia arriba (bottom-up). Son más complejos que los descendientes, pero tienen la ventaja de poder trabajar con gramáticas más generales, ya que poseen mayor poder de cómputo.

La librería PLY utilizada, basa su funcionamiento en un análisis ascendente (LR). Para que el analizador reconozca una producción, esta se define como una función con el nombre del símbolo no terminal a la izquierda de esta, de la forma "*p_noTerminal*", la cual contiene dentro las diferentes derivaciones que pueda tener ese símbolo dentro de la gramática.

Cada función intenta hacer coincidir el texto de entrada con la parte correspondiente de la gramática y, si tiene éxito, avanza al siguiente símbolo de entrada, Si no devuelve un error. Este proceso se realiza hasta que se procesa toda la entrada y el parser logra construir un árbol sintáctico de derivación desde el símbolo distinguido o sucede un error mediante el cual la derivación no puede continuar.

### Desarrollo

Se consideraron los siguientes pasos para la construcción del analizador sintáctico:

1. **Definición de producciones gramaticales:** Para cada símbolo no terminal de la gramática, se construye una función la cual contiene dentro sus derivaciones correspondientes. Se adiciona una línea de código que exporta a un archivo de control todas las ocurrencias de cada producción. Dentro de las funciones, los terminales (etiquetas) son referenciados con el mismo nombre con el que fueron definidos en el lexer (por convención del grupo, en minúsculas) y los no terminales como fueron definidos en el nombre de su función correspondiente (en mayúsculas).

```
def p_SIGMA(p): # Simbolo distinguido
    '''SIGMA : doctype ARTICLE
    '''
    print("Ejecución completa!")
    exportarTxt.append(['Prod. SIGMA -->', p.slice])


def p_ARTICLE(p):
    '''ARTICLE : article INFO TITLE CONT_A_S cierreArticle
               | article INFO TITLE CONT_A_S SECTIONS cierreArticle
               | article TITLE CONT_A_S cierreArticle
               | article TITLE CONT_A_S SECTIONS cierreArticle
               | article INFO CONT_A_S cierreArticle
               | article INFO CONT_A_S SECTIONS cierreArticle
               | article CONT_A_S cierreArticle
    '''
    exportarTxt.append(['Prod. ARTICLE -->', p.slice])
```

2. **Control y manejo de errores:** Se consideran los errores que pueden ocurrir durante el funcionamiento del analizador, producidos por derivaciones imposibles o inexistentes, entre otras situaciones. Esto se hace con una función.

```
def p_error(p):
    # p regresa como un objeto del Lexer.
    # p.__dict__ -> ver propiedades del objeto.
    global contadorErrores
    if (p):
        print(f'Error en el parser --> Tipo: {p.type} | Valor: {p.value}')
        print('Error sintáctico en la LINEA:', p.lineno)
        exportarTxt.append(['!!! Error parser -->', p])

    contadorErrores += 1
```

3. **Función de análisis:** Se desarrollaron dos funciones independientes para los casos de análisis de texto de entrada a través de archivos y de la terminal. Ambas toman el código de entrada y lo evalúan de acuerdo a las producciones gramaticales definidas, devolviendo un mensaje con el estado final del análisis.

```
> def analizarPorRuta():···

> def analizarPorLinea():···
```

4. **Pruebas:** Se llevaron a cabo diferentes pruebas a través de ambos métodos de entrada de código, considerando situaciones con variada estructura y presencia de tokens con el objetivo de obtener siempre el resultado esperado por parte del analizador.

## Funciones auxiliares

***pedirRuta()*:** Solicita al usuario que ingrese la ruta del archivo que desea analizar en el modo de ejecución a través de archivo. Se encuentra definida en el archivo *helpers.py*.

```
def pedirRuta():
  # Pedir ruta del archivo por input
  pathFile = input('Ingrese la ruta del archivo que desea analizar: ')
  # Remover comillas
  pathClean = re.sub(
      r'\'|"',
      '',
      pathFile
  )
  return pathClean.strip()
```

***cls()*:** Limpia la terminal

```
def cls():
    os.system('cls' if os.name == 'nt' else 'clear')
```

***printMenu(*menuOptions*)*:** Muestra las opciones disponibles del menú al usuario.

```
def printMenu(menuOptions):
    for key in menuOptions.keys():
        print(key, '--', menuOptions[key])
```

***logicaMenu()*:** Maneja la lógica principal del menú interactivo. Muestra el menú, solicita al usuario una opción, y ejecuta la función correspondiente a esa opción. El ciclo continúa hasta que se elija la opción de salida.

```python
def logicaMenu(
    nombrePrograma: str,
    opcionesMenu: 'dict[int, str]',
    opcionUna: 'function',
    opcionDos: 'function'
):
    cls()
    print(f'{nombrePrograma} de Docbook Article | Grupo 12. SSL 2023.')
    while(True):
        printMenu(opcionesMenu)
        option = ''
        try:
            option = int(input('Ingrese la opción: '))
        except:
            cls()
            print('Opción inválida. Por favor, ingrese una de las opciones disponibles...')
        cls()
        if option == 1:
            opcionUna()
        elif option == 2:
            opcionDos()
        elif (option == 3):
            print('Ejecucion finalizada.')
            exit()
        else:
            print('Opción incorrecta. Por favor, ingresar un número del 1 al 3.')
```

## Modo de obtención del intérprete

- ***lexer.py***: Módulo que contiene el analizador léxico.
- ***parser.py***: Módulo que contiene el analizador sintáctico.
- ***logicaMenu.py***: Contiene las funciones requeridas para la funcionalidad del menú de usuario interactivo.
- ***helpers.py***: Funciones adicionales.
- ***obtenerHtml.py***: Contiene las funciones requeridas para la traducción del archivo de entrada *.xml* a *.html*.
- ***parsetab.py***: Archivo generado por la librería PLY.
- ***parser.out***: Archivo generado por la librería PLY.

## Modo de ejecución del intérprete

- **Interactiva:** Una vez iniciado el programa, la opción “2” del menú de opciones permite analizar la línea de código ingresada a través de la terminal.
- **A partir de un archivo:** Una vez iniciado el programa, la opción “1” del menú de opciones permite analizar código proveniente de un archivo externo. Se debe ingresar la ruta de dicho archivo el cual debe tener extensión *.xml*.

## Ejemplos de funcionamiento

Se utilizaron diferentes archivos de ejemplo para la verificación del correcto funcionamiento del analizador. Los mismos se encuentran almacenados en el directorio *doc*.

- *ejemplo_catedra.xml*: ejemplo brindado por la cátedra al final del documento de requerimientos para el TPI.
- *ejemplo_simple.xml*: ejemplo brindado por la cátedra al principio del documento de requerimientos para el TPI.
- *pruebaTabla.xml*: ejemplo realizado para realizar pruebas sobre la etiqueta *informalTable*.

# Conclusión

Se pudo completar el desarrollo de un analizador léxico sintáctico que reconozca un conjunto determinado de etiquetas del lenguaje Docbook/XML y traduzca alguna de estas a su correspondiente en el lenguaje HTML.

Para el desarrollo del presente trabajo, fue necesario aplicar los conocimientos adquiridos durante el cursado de la materia sobre expresiones regulares, lenguajes y gramáticas. En primer lugar, se pudo poner en práctica la construcción de una gramática que permita analizar el lenguaje Docbook/XML y visualizar cómo se realizan las derivaciones. Además, también se pudo aplicar la construcción de expresiones regulares que reconozcan cada una de las etiquetas del lenguaje.

Un resultado adicional del desarrollo del trabajo utilizando el lenguaje de programación Python y la librería PLY fue una mayor familiarización con estos, así como también una mayor eficiencia del trabajo en equipo junto con el uso de *Git* y *GitHub* para el control de versiones del proyecto.

# Bibliografía

https://tdg.docbook.org/tdg/4.5/docbook.html