**Universidad Internacional de La Rioja**

Nombre:

Arnaldo Quintero Segura

Asignatura:

Gestión de Proyectos

Profesor:

Dr. Ing. Fernando López Hernández

Lab02 - Analizador sintáctico

Fecha:

03 de mayo de 2020

ESPAÑA - LA RIOJA

[Lab01 - Analizador léxico 1](#_Toc36314631)

[Ejecución del código 1](#_Toc36314632)

[test.minipy 2](#_Toc36314633)

[ejemplos-pdf.minipy 2](#_Toc36314634)

[Bibliografía 3](#_Toc36314635)

# Lab02 - Analizador sintáctico

Para la realización de este analizador léxico, se ha utilizado la herramienta JFlex, junto con la herrramienta CUP, siguiendo los pasos expuestos en las indicaciones.

## Problemas encontrados:

Como principal problema, se ha encontrado el parseo de los ficheros cuando existen muchas cadenas vacías. Ya que el lenguaje debe tomar en cuenta los saltos de línea como delimitadores. Fue un poco difícil lograr que la gramática aceptara ficheros vacíos, ficeros con instrucciones, ficheros con líneas vacías (solo contiene el salto de línea), y la mezcla de todos estos.

Además de la posibilidad de tener bloques anidados de grupos de control, tales como definiciones de funciones con bloques for dentro, que dentro incluyen bloques if y demás. Veamos a continuación cómo fueron resueltos estos problemas:

### Parseo de ficheros:

Para la resolución de este problema, simplemente se plantea esta solución gramatical:

(1)program ::= actions

| new\_lines

;

(2)actions ::= new\_lines statement statements

;

(3)new\_lines ::= LINE\_END new\_lines

|

;

(4)statements ::= LINE\_END new\_lines statement statements

| new\_lines

;

Esto nos permite, que el programa pueda ser vacío, o símplemente lleno de líneas vacías. Si se utiliza la segunda expansión de la regla (1). De lo contrario, si se sigue por la primera expansión, significa que tendremos al menos una línea de código. Bien sea terminada con el fin de línea o con el fin del fichero.

En este caso, se debe prever que antes de esta línea de código, existan o no líneas vacías. Y luego, una acción puede estar seguida de una o más acciones, siempre teniendo un fin de línea antes de la nueva acción. Esto se logra con la ayuda de la regla (4).

### Bloques anidados:

Para la resolución de este problema, fue necesario cambiar el fichero de análisis Léxico, siguiendo lo hecho por (Syrion89, 2016) en el cuál se ha visto que se posee una pila dentro del lexer, junto con un par de variables de tipo entero, que controlan si el código se entcuentra actualmente dentro de un bloque anidado o no.

Haciendo así que al analizar un bloque anidado, se pueda ver un token de "TAB" al iniciar un bloque indentad

# Ejecución del código

Para la ejecución del código, se utiliza la herramienta JFlex para compilar el fichero "lexicon.java" adjunto. Que nos da como resultado:

Reading "./lexicon.flex"

Constructing NFA : 222 states in NFA

Converting NFA to DFA :

...................................................................................................

101 states before minimization, 52 states in minimized DFA

Old file "./Lexer.java" saved as "./Lexer.java~"

Writing code to "./Lexer.java"

Ahora, teniendo el fichero Lexer.java, se procede a compilar con el comando "javac Lexer.java". El cual nos genera el binario Java ejecutable.

Por ultimo se utiliza el ejecutable Lexer para correr los dos ficheros de prueba adjuntos:

## test.minipy

Dado el tamaño del fichero de prueba y el tamaño de su resultado, se adjuntan los ficheros "test.minipy" y "test.minipy.result.txt", obtenido al ejecutar el comando "java Lexer ./test.minipy > test.minipy.result.txt", el cual transporta a el fichero de salida todo lo que imprime dicho output.

Al ver el archivo adjunto, podemos ver que hay gran cantidad de pruebas respecto a comentarios, los distintos tipos de números reales y enteros que soporta el lenguaje, se prueban las constantes enteras, los posibles identificadores. Y por ultimo, unas pruebas de caracteres de texto, terminando con un carácter de texto invalido, con una comilla doble extra.

Se debe notar que este fichero no posee ningún carácter de tabulador, y utiliza un formato CRLF como saltos de línea.

## ejemplos-pdf.minipy

Dado el tamaño del fichero de prueba y el tamaño de su resultado, se adjuntan los ficheros "ejemplos-pdf.minipy " y "ejemplos-pdf.minipy.result.txt", obtenido al ejecutar el comando "java Lexer ./ejemplos-pdf.minipy > ejemplos-pdf.minipy.result.txt", el cual transporta a el fichero de salida todo lo que imprime dicho output.

Este fichero consta de todos los pedazos de código extraídos del fichero pdf de la especificación propuesta. Aquí se puede ver cómo las distintas operaciones se manejan, los separadores paréntesis, comas, llaves y demás. Así como la detección de los tabuladores y las palabras reservadas. Se debe nota que este archivo utiliza un formato de fin de línea LF.

# Bibliografía

Foundation, T. P. S. (a). *10. full grammar specification — python 3.8.3rc1 documentation* docs.python.org. Recuperado de <https://docs.python.org/3/reference/grammar.html>

Foundation, T. P. S. (b). *8. compound statements — python 3.8.3rc1 documentation* docs.python.org. Recuperado de <https://docs.python.org/3/reference/compound_stmts.html>

Foundation, T. P. S. (c). *9. top-level components — python 3.8.3rc1 documentation* docs.python.org. Recuperado de <https://docs.python.org/3/reference/toplevel_components.html>

Syrion89. (2016). *Syrion89/LexerParserPython* GitHub. Recuperado de <https://github.com/Syrion89/LexerParserPython>