

Práctica de Teoría de autómatas y computación

Entrega 1



Enrique albalate prieto, Lucía de ancos VILLA, PABLO del hoyo abad

24/03/2023

1. Descripción de la práctica.

El objetivo de la práctica consiste en crear un programa que tome como entrada un archivo de texto en un formato creado por la profesora para la representación gráfica de un grafo y transformarlo al lenguaje DOT. El lenguaje DOT es más complejo ya que permite especificar de manera más precisa la representación visual del grafo. Asimismo, existen numerosas herramientas para la creación de grafos que aceptan ese lenguaje.

Lo que se nos pide, por tanto, es desarrollar un procesador de lenguaje que se capaz de traducir el lenguaje definido por la profesora al lenguaje DOT. Si bien las fases concretas en las que se divide un procesador de lenguajes depende de lenguaje de entrada y el lenguaje de salida, todas estas pueden pertenecen a una de estas tres: análisis léxico, análisis sintáctico y generación de código.

Para esta entrega, se ha desarrollado la parte que concierne al análisis léxico. En esta fase, el programa lee una cadena de caracteres y genera una secuencia de elementos léxicos o tokens. Un token es un nombre que se asocia a un conjunto de secuencias de caracteres. Se trata de un elemento indivisible que suele tener un significado por sí mismo. Esta secuencia de tokens es la que recibe la fase de análisis sintáctico. Para especificar formalmente el conjunto de secuencias caracteres que pueden ser parte de token, se emplean expresiones regulares.

Para la construcción del analizador léxico, se ha empleado la herramienta JFlex. JFlex permite principalmente ejecutar una serie de sentencias cada vez que encuentra una cadena de caracteres que pertenezca a un lenguaje definido por la expresión regular de un token. Si una cadena de texto perteneciera a varios lenguajes, se seguirían leyendo caracteres hasta que la cadena de texto perteneciese a solo uno. Si aun así siguieran existiendo varias opciones, JFlex ejecutará el código asociado a la primera expresión regular por orden de aparición.

1. Tabla de elementos léxicos.

|  |  |  |  |
| --- | --- | --- | --- |
| EXPRESIÓN REGULAR | DESCRIPCIÓN | NOMBRE TOKEN | EJEMPLO CADENA |
| arbol | Palabra reservada | ARBOL | árbol |
| node | Palabra reservada | NODE | node |
| shape | Palabra reservada | SHAPE | shape |
| label | Palabra reservada | LABEL | label |
| color | Palabra reservada | COLOR | color |
| fontcolor | Palabra reservada | FONTCOLOR | fontcolor |
| style | Palabra reservada | STYLE | style |
| edge | Palabra reservada | EDGE | edge |
| dir | Palabra reservada | DIR | dir |
| hijos | Palabra reservada | HIJOS | hijos |
| ; | Signo de puntuación | SEMI | ; |
| [ | Signo de puntuación | LSQB | [ |
| ] | Signo de puntuación | RSQB | ] |
| = | Signo de puntuación | EQUAL | = |
| , | Signo de puntuación | COMMA | , |
| “ | Signo de puntuación | QUOTE | “ |
| { | Signo de puntuación | LBRACE | { |
| } | Signo de puntuación | RBRACE | } |
| \/\/.\*(\r\n)? | Cualquier combinación de caracteres que comience con “//” (hasta el final de la línea) | COMMENT\_1 | // comment\_1 |
| #.\*(\r|\n)? | Cualquier combinación de caracteres que comience con “#” (hasta el final de la línea) | COMMENT\_2 | # comment\_2 |
| \/\\*(.\*|\r\n)\*\\*\/ | Cualquier combinación de caracteres que comience con “/\*” y acabe por “\*/” (incluso en diferentes líneas) | COMMENT\_3 | /\* comment\_3 \*/ |
| \"[a-zA-Z0-9\_ ]+\" | Cualquier cadena formada por números o letras (debe tener al menos un caracter) situada en comillas (“”) | STR\_QUOTES | “cadena” |
| [a-zA-Z0-9\_]+ | Cualquier cadena formada por números o letras (debe tener al menos un caracter) | IDENTIFIER | grafo1 |
| . | Resto de casos | - | @ |

1. Manual de usuario.

3.1 Ejecución del programa con eclipse

Para poder realizar la práctica, previamente es necesario haber integrado Jflex en Eclipse, para ello, creamos un workspace y un proyecto para Jflex, además de crear la configuración externa de ejecución.

Una vez preparado el entorno, en la raíz del proyecto implementamos en Jflex un fichero “practica.fle” con cada una de las expresiones regulares del lenguaje propuesto en la práctica. Para probar su funcionamiento, ejecutamos dicho fichero con la herramienta externa Jflex, lo cual generará el fichero “Detectatokens.java” dentro de la carpeta src, a esto le consideramos el Main.

Para ejecutar el Main, pinchamos sobre el propio fichero y en Run – Run configurations añadimos como argumento el fichero con el que vayamos a hacer la prueba, después pulsamos en run y se imprimirán por pantalla las cadenas reconocidas.

3.2 Ejecución desde la terminal

Lo primero que hay que hacer es descargar JFlex desde su página web oficial. Una vez descargado, se deberá añadir el ejecutable que corresponda según el sistema operativo a un de los directorios que se encuentre en el PATH. A continuación, deberá ejecutar los siguientes comandos

$ *jflex practica.fle*

$ *javac DetectaToken.java*

Esto generará un archivo de bytecode de Java que se puede ejecutar con

$ *java DetectaToken*

Por defecto, JFlex genera un programa que lee el programa desde los argumentos de entrada, por lo que para "tokenizar" una secuencia de caracteres que se encuentre en un archivo, se hace lo siguiente:

$ *java DetectaToken archivo.*txt

1. Ejemplo de fichero de entrada y su salida.

Ejemplo de fichero de entrada:

arbol ejemplo {

node [shape = square, label = "a", color = "green", fontcolor = blue , style = bold]

edge [dir = forward, label = "0", color = "blue", fontcolor = red , style = dashed]

a hijos = {b [label = "6", color = "red"],c [label = "100"]} [label = "A", color = "blue", shape = circle];

b hijos = {d [label = "80"] } [label = "B"];

c hijos = {f [label = "20"], g [label = "22"]} [label = "C"];

d hijos = {e [label = "14"] } [label ="D"];

e [label = "E 14", shape = circle];

f [label = "F 5", shape = circle];

g [label = "G 4", shape = circle];

}

# hello

//HOLA

/\*\*\*\*\*

adios

bye

\*/

Salida del fichero:

ARBOL => arbol

IDENTIFIER => ejemplo

LBRACE => {

NODE => node

LSQB => [

SHAPE => shape

EQUAL => =

IDENTIFIER => square

COMMA => ,

LABEL => label

EQUAL => =

STR\_QUOTES => "a"

COMMA => ,

COLOR => color

EQUAL => =

STR\_QUOTES => "green"

COMMA => ,

FONTCOLOR => fontcolor

EQUAL => =

IDENTIFIER => blue

COMMA => ,

STYLE => style

EQUAL => =

IDENTIFIER => bold

RSQB => ]

EDGE => edge

LSQB => [

DIR => dir

EQUAL => =

IDENTIFIER => forward

COMMA => ,

LABEL => label

EQUAL => =

STR\_QUOTES => "0"

COMMA => ,

COLOR => color

EQUAL => =

STR\_QUOTES => "blue"

COMMA => ,

FONTCOLOR => fontcolor

EQUAL => =

IDENTIFIER => red

COMMA => ,

STYLE => style

EQUAL => =

IDENTIFIER => dashed

RSQB => ]

IDENTIFIER => a

HIJOS => hijos

EQUAL => =

LBRACE => {

IDENTIFIER => b

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "6"

COMMA => ,

COLOR => color

EQUAL => =

STR\_QUOTES => "red"

RSQB => ]

COMMA => ,

IDENTIFIER => c

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "100"

RSQB => ]

RBRACE => }

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "A"

COMMA => ,

COLOR => color

EQUAL => =

STR\_QUOTES => "blue"

COMMA => ,

SHAPE => shape

EQUAL => =

IDENTIFIER => circle

RSQB => ]

SEMI => ;

IDENTIFIER => b

HIJOS => hijos

EQUAL => =

LBRACE => {

IDENTIFIER => d

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "80"

RSQB => ]

RBRACE => }

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "B"

RSQB => ]

SEMI => ;

IDENTIFIER => c

HIJOS => hijos

EQUAL => =

LBRACE => {

IDENTIFIER => f

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "20"

RSQB => ]

COMMA => ,

IDENTIFIER => g

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "22"

RSQB => ]

RBRACE => }

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "C"

RSQB => ]

SEMI => ;

IDENTIFIER => d

HIJOS => hijos

EQUAL => =

LBRACE => {

IDENTIFIER => e

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "14"

RSQB => ]

RBRACE => }

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "D"

RSQB => ]

SEMI => ;

IDENTIFIER => e

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "E 14"

COMMA => ,

SHAPE => shape

EQUAL => =

IDENTIFIER => circle

RSQB => ]

SEMI => ;

IDENTIFIER => f

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "F 5"

COMMA => ,

SHAPE => shape

EQUAL => =

IDENTIFIER => circle

RSQB => ]

SEMI => ;

IDENTIFIER => g

LSQB => [

LABEL => label

EQUAL => =

STR\_QUOTES => "G 4"

COMMA => ,

SHAPE => shape

EQUAL => =

IDENTIFIER => circle

RSQB => ]

SEMI => ;

RBRACE => }

COMMENT\_2 => # hello

COMMENT\_1 => //HOLA

IDENTIFIER => adios

IDENTIFIER => bye