Autómatas y Lenguajes 2011

# Primer práctico de máquina - Análisis Lexicográfico

### Consigna:

#### Se pidió diseñar e implementar un analizador lexicográfico para la gramática JAVA y otro para la gramática HTML.

### Grupo:

#### Bringas, Germán y Perez, Federico

# Resumen:

Para llevar a cabo el presente práctico, se decidió utilizar autómatas finitos determinísticos (uno para cada lenguaje).

El primer paso fue establecer los autómatas necesarios en papel para tener una aproximación de lo que sería el autómata final en cada caso.

Luego se implementó en lenguaje JAVA, una aplicación de consola capaz de realizar la ejecución de un autómata a partir de una configuración dada.

Se realizaron varias iteraciones sobre la aplicación, lo que implicó muchas veces modificar el programa para poder soportar tanto características que no se habían tenido en cuenta como también respetar todas los lineamientos de la consigna.

Luego, se creó la configuración del autómata correspondiente a HTML y se corrió sobre la aplicación construida, dando como resultado un nuevo conjunto de pruebas que desembocaron en la versión final.

Por último se refactorizó el código, se comentó y se preparó para la entrega.

# Implementación:

Para la implementación del analizador se diseñó una aplicación dividida en 3 capas:

* Capa principal o invocante: Encargada de levantar el archivo de entrada, el parámetro de lenguaje a analizar (JAVA o HTML), generar una entrada al analizador propiamente dicho y procesar la salida para luego escribirla en el archivo “salida.tok”.

Para esto se diseñó el archivo “config.yaml” (obligatoriamente en el raíz del proyecto java) el cual debe contener la configuración por lenguaje de:

* + El archivo de entrada.
  + El archivo de salida.
  + El archivo con la definición del autómata.
  + El archivo con la configuración de los tokens.
* Capa de análisis: Aquí es donde habita la lógica del analizador, se toma la entrada, y se procesa a través de un autómata. En cada iteración, se recupera el lexema reconocido y se busca el token correspondiente al mismo, de esta manera se logra una colección de pares (token, lexema) como salida.

Es también aquí donde se realiza el reconocimiento, tratamiento y recupero de errores.

* Capa de Procesamiento: Un autómata es el encargado de procesar una cadena de texto, carácter a carácter, hasta llegar a un estado final.

# Configuración de los tokens:

La configuración de los tokens se puede modificar en el archivo descrito en el ya nombrado “config.yaml”, de igual manera, se detalla a continuación:

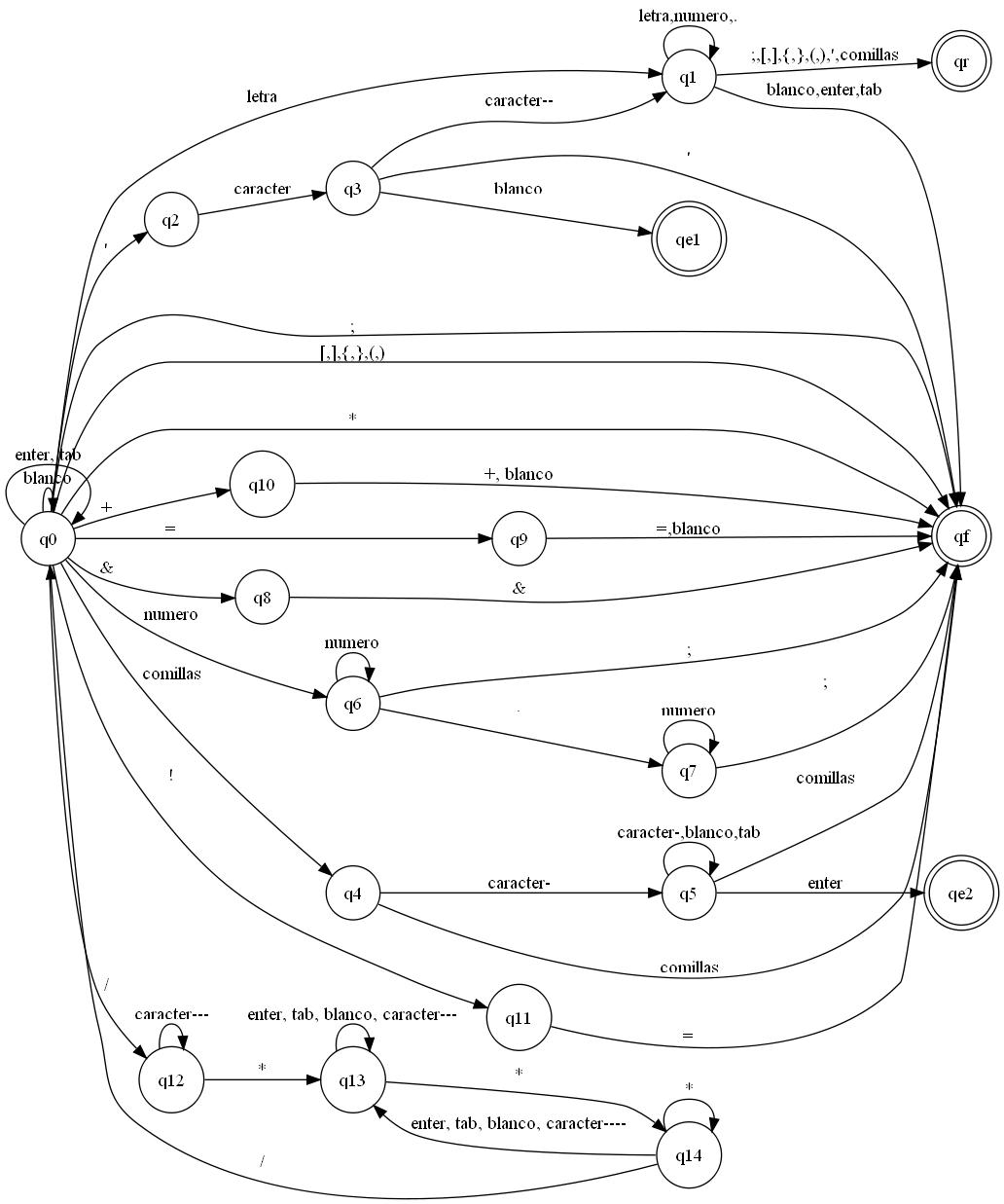
|  |  |
| --- | --- |
| JAVA | HTML |
| 0 - Palabra reservada | **0 - Palabra reservada** |
| 1 - Operador aritmético | **1 - Corchete angular** |
| 2 - Operador de asignación | **2 - Barra** |
| 3 - Operador de comparación |  |
| 4 - Agrupador |  |
| 5 - Operador lógico |  |
| 6 - Separador |  |
|  |  |

# Referencia para los diagramas:

Por una cuestión de comodidad, tanto en los archivos de configuración de cada autómata como en los diagramas (a continuación) se utilizaron palabras para agrupar conjunto de caracteres. A continuación la referencia de los mismos.

|  |  |  |  |
| --- | --- | --- | --- |
| JAVA | | HTML | |
| caracter | ASCII | ASCII | ASCII sin < > / |
| caracter- | ASCII sin \ | ASCII- | ASCII sin < > / = |
| caracter-- | ASCII sin ‘ | ASCII-- | ASCII sin ! |
| caracter--- | ASCII sin \* | ASCII--- | ASCII sin - |
| caracter---- | ASCII sin \* / | ASCII---- | ASCII sin - > |
|  |  | ASCII----- | ASCII sin <>/= |

# Diagrama autómata JAVA



# Diagrama autómata HTML

