Reportes de Programas Autómatas

Carlos Alejandro Monraga León

carlos.monraga@gmail.com

Instituto Tecnológico Superior de Huatusco

XXIV Verano de la Investigación Científica

ESCOM - IPN

Ciudad de México, D.F. - 7 de septiembre de 2014

Índice

Índice de figuras

1. Reporte Programa Autómata Que Valida Las Palabras: Instituto, Internet e Interconexión

1.1. Abstract

Abstact

El presente programa pretende demostrar a través de representación gráfica, como es que se representan los estados y transiciones durante la ejecución de un Autómata Finito Determinístico que válida si las palabras: instituto, internet e interconexión, se encuentran dentro de un texto contenido que el usuario ingresa por medio del teclado.

Keywords: Automaton, States, Transitions, Code in Java, GUI.

1.2. Introducción

Los Autómatas Finitos Determinísticos (AFD) forman parte de la familia de los Autómatas, sus aplicaciones varían, ya que en la práctica su implementación resulta ser de gran utilidad, como puede ser un analizador léxico componente importante de un compilador, programas que validan cadenas de caracteres que cierto campo debe contener, esto se ocupa mucho en el desarrollo de páginas web, entre muchas más usos.[2]

Dada la anterior introducción se tratará a continuación, el desarrollo de AFD para el tratamiento de las palabras: instituto, internet e interconexión, contenidas dentro de un texto.

1.3. Desarrollo

Como ya se ha visto la definición del problema, consta de encontrar las palabras: instituto, internet e interconexión dentro de un escrito, para esto, lo primero en realizar es construir el AFD:

Como pudo observarse, las transiciones solo avanzan al siguiente estado si y solo si, el siguiente carácter es válido, para todos los otros casos, concretamente si es “i” se regresa a Q1, sino retorna a Q0. Aunque si analizamos esto detenidamente el número de estados puede disminuir drásticamente, es decir, podemos realizar una menor cantidad de estados con más de una entrada, con lo cual imaginemos en el switcheo de focos, que van encendiendo conforme la validación del carácter es correcta.

Para tener una idea de esto obsérvese la siguiente figura:

Una vez conceptualizado la forma en cómo se habrán de validar las cadenas propuestas, comencemos a ver la parte de la codificación, donde el lenguaje de programación será Java y la herramienta utilizada para el desarrollo NetBeans.

En primer lugar se procederá a realizar una clase denominada “Automata” de la cual habrá que escribir sus atributos y métodos que a continuación se describen:

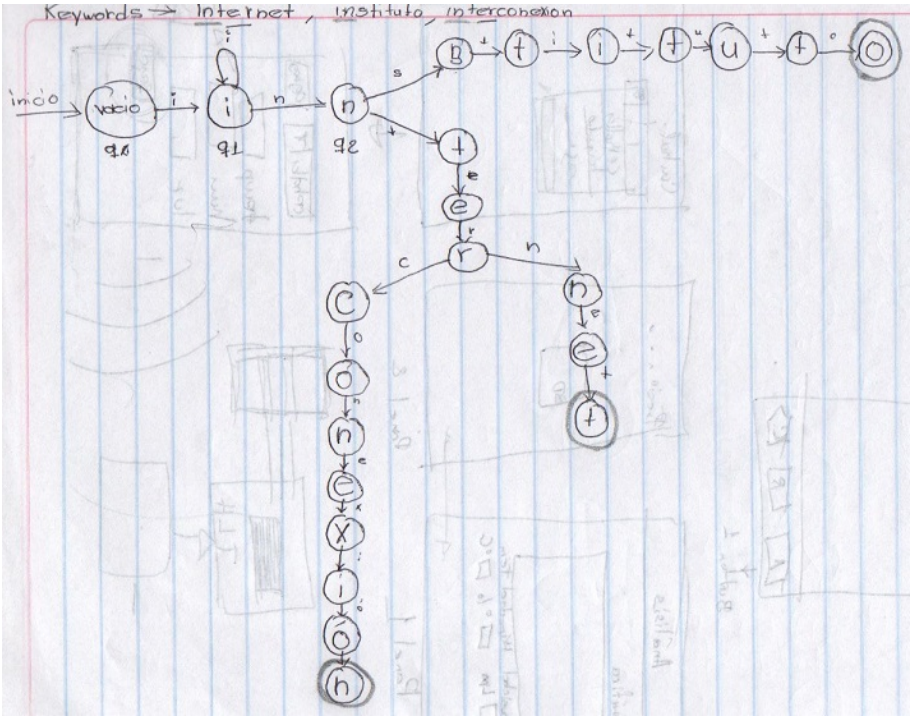

Figura 1: Diseño AFD.

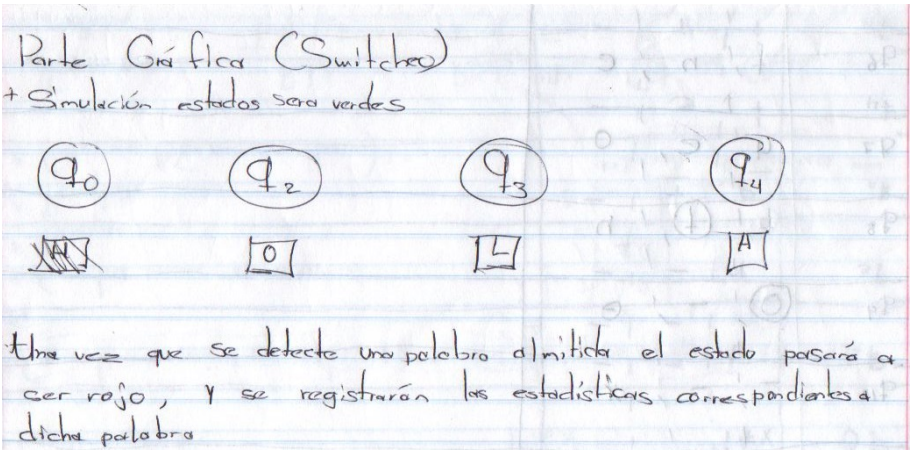

Figura 2: Switcheo.

1.3.1. Dependencias de la clase Automata

Como el nombre lo indica, son clases que necesitan importarse para utilizarlas en la clase "Automata".

- java.awt.Point: Esta es la clase que servirá para almacenar las coordenadas (renglón, columna), donde se encuentra cada palabra respectiva.
- java.util.ArrayList: Clase genérica que permite crear un arreglo de tamaño indefinido.
- java.util.Iterator: Clase que permite recorrer el arreglo anteriormente dicho.

1.3.2. Atributos de la clase Automata

- private final String cadena: Este atributo contendrá el este ingresado por el usuario.
- private final int longitud: Como su nombre lo dice almacenará la longitud de "cadena" (texto ingresado por el usuario).
- private final char caracteres[]: Arreglo de caracteres que contendrá a todos los caracteres que componen a "cadena".
- private final ArrayList<Point>p1, p2, p3: Estos atributos representan un arreglo de la clase Point de tamaño indefinido donde cada uno contendrá los puntos respectivos a la ubicación las palabras.
- private int contador, subcontador: Son variables de apoyo.
- private int frecuenciaP1, frecuenciaP2, frecuenciaP3: Contendrán los datos asociados a cada palabra, en este caso la frecuencia de las mismas.
- private char carácter: Variable de apoyo que contendrá el carácter en el momento específico cuando sale del arreglo "caracteres".

1.3.3. Métodos de la clase Automata

- public Automata(String texto): Constructor de la clase Automata, se crea una instancia de esta clase, inicializando el atributo "cadena" al pasarse como parámetro el texto que el usuario ingresó.
- public int obtenerLongitud(): Devuelva el valor de "longitud".
- public int obtenerContador(): Devuelve el valor de "contador".
- public int obtenerSubcontador(): Devuelve el valor de "subcontador".
- public void f0(int x): Realiza validación del estado Q0.
- public void f1(int x) : Realiza validación del estado Q1.

- public void f2(int x) : Realiza validación del estado Q2.
- public void f3(int x): Realiza validación del estado Q3.
- public void f4(int x) : Realiza validación del estado Q4.
- public void f5(int x) : Realiza validación del estado Q5.
- public void f6(int x) : Realiza validación del estado Q6.
- public void f7(int x) :Realiza validación del estado Q7.
- public void f8(int x) :Realiza validación del estado Q8.
- public void f9(int x) : Realiza validación del estado Q9.
- public void f10(int x): Realiza validación del estado Q10.
- public void f11(int x): Realiza validación del estado Q11.
- public void f12(int x): Realiza validación del estado Q12.

Como el programa requiere de representar los estados por medio de una interfaz gráfica de usuario (GUI), se necesita de una clase denominada "Graficos" que en esencia se encarga de dibujar y representar el flujo entre los estados durante la ejecución del programa, clase de la cual nuevamente se describen propiedades y métodos:

1.3.4. Dependencias de la clase Graficos

- java.awt.Color: Clase que proporciona diversos tipos de constantes que indican el color que se pueda requerir.
- java.awt.Font: Clase que proporciona diversos métodos y atributos para establecer una Fuente de texto.
- java.awt.Graphics: Utilizada para poder dibujar.

1.3.5. Atributos de la clase Graficos

- private static final Font fuente: Contendrá el tipo de fuente con el que dibujará el nombre de los estados.
- private static final Font fuenteCaracter: Fuente que imprimirá el carácter que en cierto momento especifico es sacado del arreglo "caracteres" y también con el cual se mostrará la palabra encontrada respectiva.

1.3.6. Métodos de la clase Graficos

Cabe resaltar que esta clase no se instanciará, en cambio se utilizarán los métodos estáticos.

- public static void representacionInicial(Graphics g). Se encarga de dibujar los estados que representa al AFD.
- public static void representacionProceso(Graphics g, int contador, int subcontador, char caracter): Representa el proceso que mostrará la actividad del autómata al evaluar el texto ingresado por el usuario.
- public static void clearWindow(Graphics g): Encargado de borrar una parte especifica del esquema.
- private static void representacionEstados(Graphics g): Pinta los estados dibujados en la pantalla.

La última clase que compone dicho programa es "Principal" de la cual basta solo decir que representa la interfaz gráfica de usuario (GUI) para ejecutar el programa, la cual a continuación se puede observar:

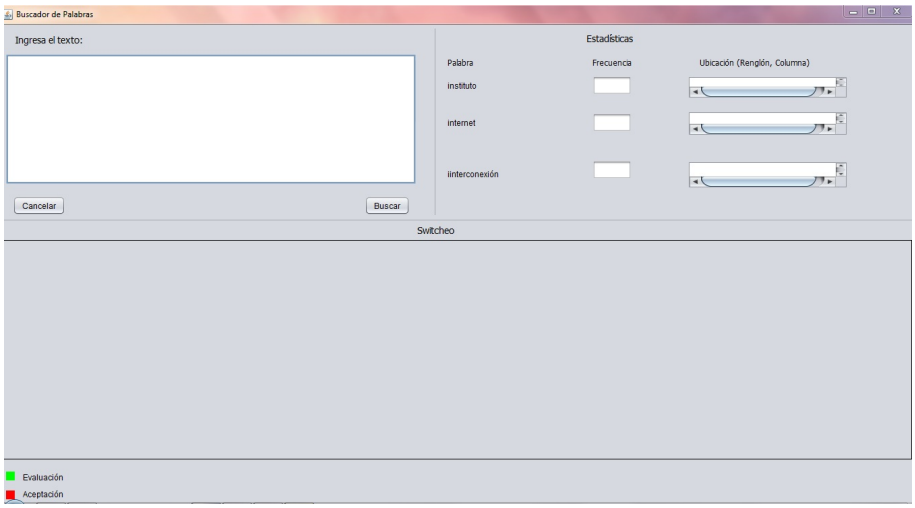

Figura 3: Pantalla principal del programa.

Lo siguiente es ejecutar el archivo "Automata.jar" (para eso se debe tener instalado Java en el equipo) como en la siguiente figura:

Figura 4: Archivo JAR.

Simplemente se abrir el archivo:

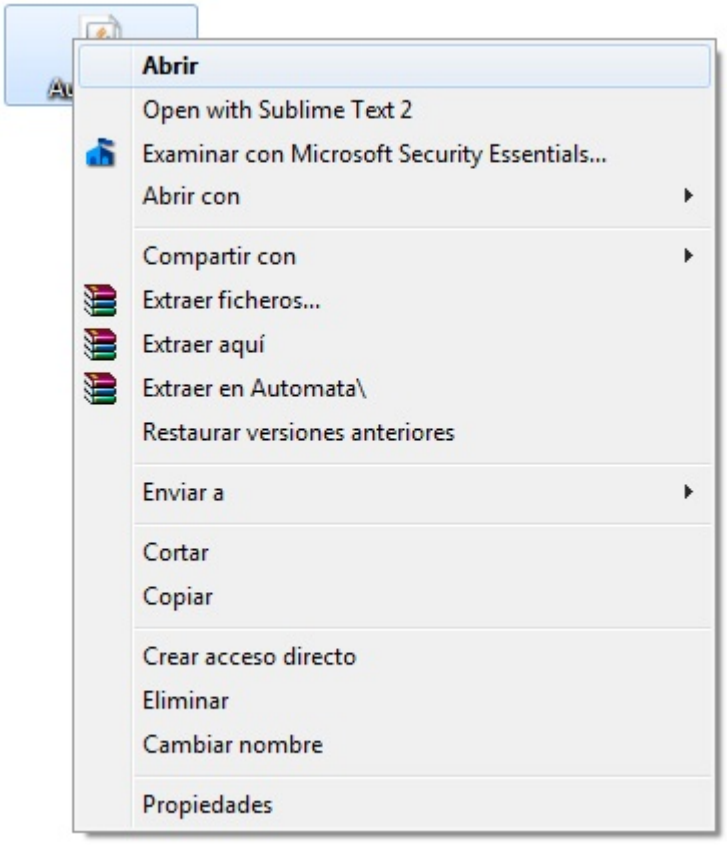

Figura 5: Ejecutar archivo JAR.

La ejecución del programa será más o menos similar a la que a continuación se muestra:

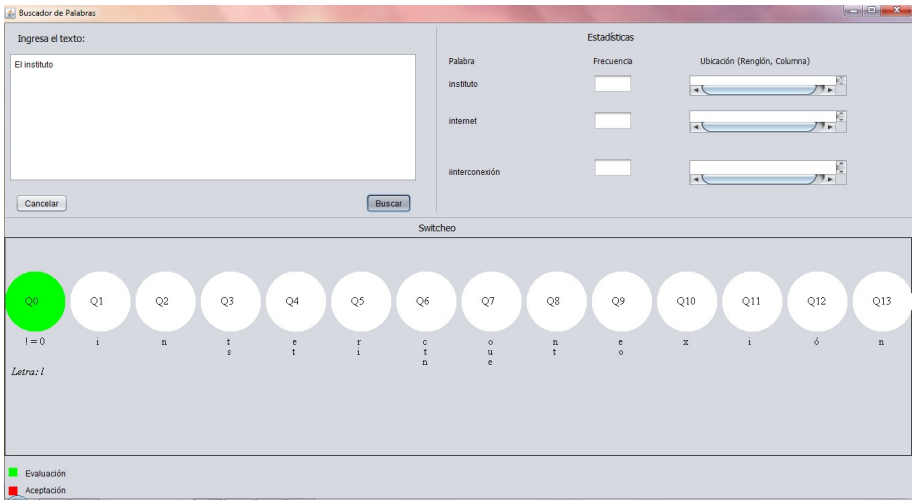

Figura 6: Ejecución del programa.

1.4. Conclusiones

La construcción de dicho programa implico el hecho de comprender, analizar y profundizar más a fondo en la Teoría de Autómatas, en conceptos como alfabeto, estados, transiciones, función de transición y estado inicial, además de que también implica teoría de conjuntos. Sin duda el conocimiento enriquecedor que me ha dejado la construcción de este programa ha sido de gran importancia para comprender los variados usos que tiene esta teoría en el proceso de la elaboración de software.

2. Reporte Programa Ejemplo Extendido De Planetas

2.1. Abstract

Abstact

El presente programa pretende encontrar una aproximación como solución a la propuesta del ejemplo de Autómata Extendido de Jay Misra en una lámina del libro "Introduction to Automata Theory, Languages, and Computation", parte del curso de Automatas en la universidad de Standford. El problema consiste en ver si un planeta puede fallar, al solo existir una población de las determinadas especies: A, B y C, dado que, para que nazca descendencia de una nueva especie (en todos los casos los nacimientos son dos hijos), un miembro de cada una de las otras especies debe morir.

Keywords: Deterministic Finite Automaton, Jay Misra, Example Planet, State.

2.2. Introducción

El ejemplo propuesto por Jay Misra[1] nos plantea que, en cierto planeta existen tres especies las cuales se denominan A, B y C, donde dos especies cualquiera se pueden aparear, pero al hacerlo estos miembros de las especies mueren y dos miembros de la tercera especie nacen, el planeta falla en un punto donde todos los individuos son de la misma especie, ya que ya ninguna cría puede nacer.

Una restricción a esta problemática es que el número de individuos nunca cambia.

Un estado es la representación de tres números enteros que hacen referencia al número de los individuos de las tres especies respectivamente, es así como también pueden definirse a la par los eventos que son de tres tipos: evento A, evento B y evento C, donde cada uno sucede cuando nacen dos hijos de la especie que por nombre lleva el evento y 1 de las especies restantes muere. En base a esto se plantea la pregunta fundamental del problema: ¿En un estado dado el planeta eventualmente puede fallar?

2.3. Desarrollo

Para conocer la respuesta a la pregunta hecha, se desarrolló un respectivo programa en lenguaje Java, que fuera capaz de ir iterando en los eventos a fin de encontrar de acuerdo a las combinaciones de estados, la respuesta a la pregunta.

2.3.1. Atributos de la clase Planeta

- private int poblacionA: Este campo representa el número de dicha población.

- private int poblacionB: Este campo representa el número de dicha población.
- private int poblacionC: Este campo representa el número de dicha población.
- private int n: Representa la cantidad total de individuos en el planeta, a partir de dicho número se realizan las combinaciones correspondientes, para poder ir iterando en los eventos, a fin de encontrar si el planeta falle.

2.3.2. Métodos de la clase Planeta

- private void eventoA(): Método que representa un evento en favor de la especie A.
- private void eventoB(): Este método representa un evento en favor de la especie B.
- private void eventoC(): Este método representa un evento en favor de la especie C.
- private void establecerCombinaciones(int n): Como su nombre lo indica realiza la acción de constituir las combinaciones, una vez establecido el valor de n. Esta combinaciones son de la forma siguiente:
  - ([n- 1], 1, 0)
  - ([n – 1], 0, 1)
  - (1, 0, [n – 1])
  - (0, [n – 1], 1)
  - (1, [n – 1], 0)
  - (0, 1, [n – 1])
- private void establecerCombinacionEntrada(int x): Constituye el estado en dicho momento especifico, dado el valor de n.
- private void print(): Imprime las combinaciones, consecuencias de las iteraciones sucesivas entre eventos.
- public void start(): Inicia el proceso de hallar si el planeta falla.

2.3.3. Métodos de la clase Automata

- public Automata(String texto): Constructor de la clase Automata, se crea una instancia de esta clase, inicializando el atributo “cadena” al pasarse como parámetro el texto que el usuario ingresó.
- public int obtenerLongitud(): Devuelva el valor de “longitud”.
- public int obtenerContador(): Devuelve el valor de “contador”.

- public int obtenerSubcontador(): Devuelve el valor de “subcontador”.
- public void f0(int x): Realiza validación del estado Q0.
- public void f1(int x) : Realiza validación del estado Q1.
- public void f2(int x) : Realiza validación del estado Q2.
- public void f3(int x): Realiza validación del estado Q3.
- public void f4(int x) : Realiza validación del estado Q4.
- public void f5(int x) : Realiza validación del estado Q5.
- public void f6(int x) : Realiza validación del estado Q6.
- public void f7(int x) :Realiza validación del estado Q7.
- public void f8(int x) :Realiza validación del estado Q8.
- public void f9(int x) : Realiza validación del estado Q9.
- public void f10(int x): Realiza validación del estado Q10.
- public void f11(int x): Realiza validación del estado Q11.
- public void f12(int x): Realiza validación del estado Q12.

2.3.4. Clase PrincipalPlanetas

Dicha clase es la encargada de ejecutar el programa, donde solamente se crea una instancia de la clase Planeta y manda a invocar el método “star”, encargado de realizar todo el proceso de encontrar si dada una combinación de entrada, el planeta puede fallar.

Cabe resaltar que el valor de n va desde; dos hasta catorce, esto en base a una petición.

2.4. Conclusiones

A continuación se presentan los resultados de forma resumida, en relación a las combinaciones de entrada, que simbolizan cuando un planeta falla:

- n = 2
  1. 110
  2. 101
  3. 011
- n = 3
  1. 111

- n = 4
    1. 310
    2. 301
    3. 103
    4. 031
    5. 130
    6. 013
- n = 5
    1. 410
    2. 401
    3. 104
    4. 041
    5. 140
    6. 014
- n = 7
    1. 610
    2. 601
    3. 106
    4. 061
    5. 160
    6. 016
- n = 8
    1. 710
    2. 701
    3. 107
    4. 071
    5. 170
    6. 017
- n = 10
    1. 910
    2. 901
    3. 109
    4. 091

5. 190
6. 019

- n = 11

1. 1010
2. 1001
3. 1010
4. 0101
5. 1100
6. 0110

- n = 13

1. 1210
2. 1201
3. 1012
4. 0121
5. 1120
6. 0112

- n = 14

1. 1310
2. 1301
3. 1013
4. 0131
5. 1130
6. 0113

Obsérvese que cuando n = 3, esta combinación no encaja dentro del patrón de combinaciones del programa, por lo que solo es meramente mencionada aquí. Por ultimo hágase ver que las combinaciones de entrada representan los “estados”.

3. Agradecimientos

Agradecimientos al Doctor Genaro Juárez Martínez miembro de la Universidad del Oeste de Inglaterra (University of the West of England), por al aporte de sus conocimientos y/o tiempo en el campo de la Teoría de Autómatas, dicho conocimiento fue de gran valor en la elaboración de los programas anteriormente descritos.

Referencias

[1] Motwani R. & Ullman J. D. Hopcroft, J. E. Introduction to automata. In *Introduction to Automata Theory, Languages, and Computation*. 2001.

[2] Motwani R. & Ullman J. D. Hopcroft, J. E. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.