SIMULACION DE AFD

TEORIA DE AUTÓMATAS

Integrantes: Javier Castro

Leandro Calloguerea

Diego Rojas

Docente: María Eliana de la Maza

Valdivia – Chile *14/04/2016*

INTRODUCCIÓN

Se requiere hacer una simulación de un Autómata Finito Determinista, del cual tenga como propósito ingresar una serie de transiciones, y verificar si la palabra es aceptada o no por este autómata.

El programa está basado en la Teoría de Autómatas estudiado en clases para su implementación, debiendo ser esta en algún lenguaje de programación a gusto por el estudiante, que realice las operaciones pertinentes en su funcionamiento, entendiendo como teoría base que: "*un lenguaje se dirá regular si existe un AFD que lo acepte*".

En base a lo anterior, el programa realizará la acción de recibir los parámetros de entrada en este caso las transiciones, donde procederá a descomponerlas, encontrar las transiciones que ocupa el autómata, los símbolos del alfabeto que ocupa, y posteriormente determinar que palabra genera, para verificar si acepta o no por estado final.

DESARROLLO

Para llevar a cabo esta tarea escogimos el lenguaje de programación computacional Python basándonos en la versatilidad de este lenguaje del cual podemos rescatar 2 grandes razones que avalan nuestra elección.

LENGUAJE INTERPRETADO: Al ser Python un lenguaje interpretado facilita y optimiza de manera sustancial la programación de alguna aplicación, ya que es capaz de ejecutar instrucciones por parte sin necesidad de que el programa este completo, esta cualidad permite ir verificando la correcta implementación de funciones en el momento y así sortear errores rápidamente a diferencia de otros lenguajes que requieren compilar la totalidad de la aplicación.

FÁCIL DE USAR: Este lenguaje posee la particularidad de ser absolutamente orientado a objeto de modo que en Python todo es manejado como objeto, esto hace que el uso de funciones y métodos sean fáciles de implementar y heredar.

Basándonos en las fortalezas de Python optamos por definir las siguientes **estructuras de datos**:

TRANSICIONES:

- **(Variable tipo String de transiciones)** Cada transición será una 3-tupla ingresada por el usuario, y constará de:

  ="ESTADO_ACTUAL,SIMBOLO_ACTUAL,ESTADO_SIGUIENTE".

  Separando cada elemento de esta por "," y cada tupla del String separada por ";".

- **(Arreglo Bidimensional de transiciones:[1..m][1..n] de tipo String)** Cada elemento del arreglo será una transición obtenida del String de transiciones, usando solo un 3-tupla para albergar la transición, separados por comas.

- **[m]**: tipo entero, almacena la cantidad de estados.
- **[n]**: tipo entero, almacena la cantidad de letras del alfabeto (Ambos obtenidos a partir del análisis de las 3 tuplas ingresadas).

Análogamente para el arreglo bidimensional de transiciones, se usará un:

- **Arreglo unidimensional** de estados:[1..m] de tipo String.
- **Arreglo unidimensional** de letras del alfabeto:[1..n] tipo String
  (Ambos obtenidos de las transiciones ingresadas por el usuario).

PALABRA DE ENTRADA:

- Almacenado en un arreglo de tipo String.

ESTADO INICIAL:

- Almacenado en una variable de tipo String.

ESTADO FINAL:

- Almacenado en una variable de tipo String.

Para desarrollar esta aplicación nos apoyamos del modelo de desarrollo Vista – Controlador para asociar entornos gráficos a Python, usando Qt como diseñador de la interfaz y PySide como intérprete entre ambas plataformas.

Nuestro esquema de desarrollo cuenta de 4 archivos:

- AFDUi.py: Archivo que genera pyside con el comando pyside-uic usando como fuente la ui creada con Qt Designer.
- AFDController.py: el cual interpreta las funciones graficas con los métodos asociados.
- CreateAFD.py: El cual consta de funciones primarias de cálculos lógicos a ser usados en la instanciación del AFD en cuestión.
- AFDFuntions.py: El cual contiene la estructura principal del programa y métodos de validaciones necesarias.

MÉTODOS CLAVE :

- obtEstadosAlfabeto(entradao): Filtra la entrada de transiciones obteniendo un arreglo de estados y de letras del alfabeto pertenecientes al automata con el que se quiere trabajar

- llenarTransiciones(estados, alfabeto, ent): Usando los el arreglo de estados, el arreglo de letras del alfabeto obtenido anterior mente y por último las tuplas ingresadas. Se crea y llena la matriz bidimensional ubicando correctamente las transiciones con sus respectivos estados actuales y símbolos de lectura.

- aceptaPalabra(palabra): En el se encuentra el algoritmo de trabajo para aceptar una palabra que pertenezca al lenguaje definido por el AFD, usando las transiciones, estados y el alfabeto para lograrlo.

PROGRAMA

NOMBRE: Simulador de AFD by CCR

LENGUAJE:

- Python 2.7.6 - compilador Versión GCC 4.8.2

  Adicionalmente:

  - IDE: Ninja – IDE 2.3
  - Plataforma de desarrollo: Linux distribución Ubuntu/Kubuntu

REQUISITOS MÍNIMOS:

Software:

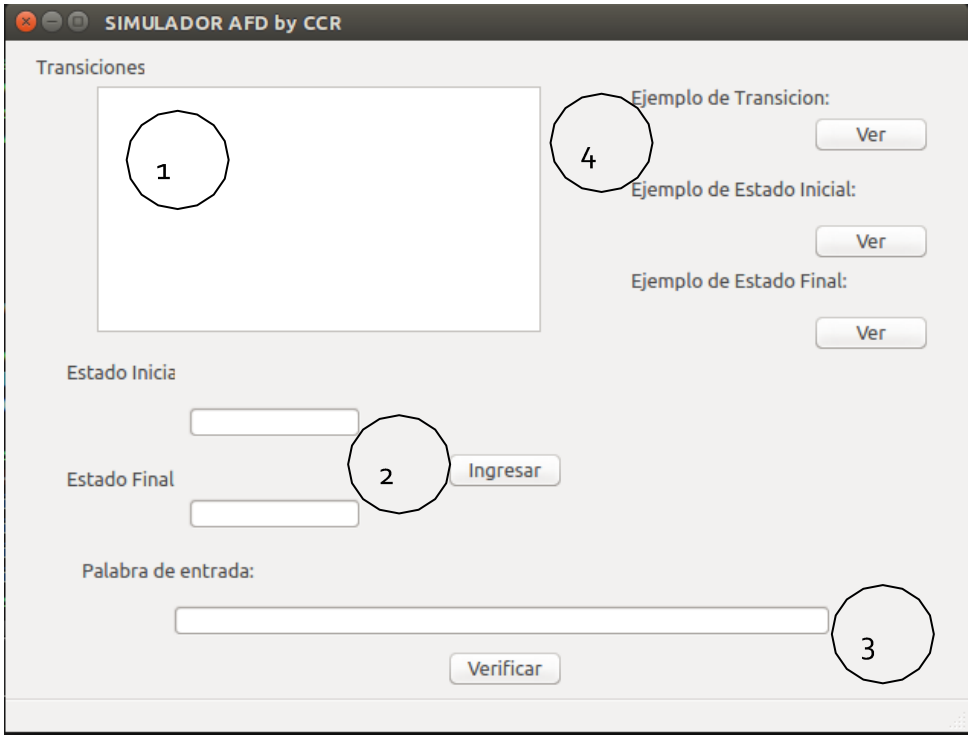
- Sistema operativo (de preferencia) Linux distribución debian Ubuntu/Kubuntu
- Tener instalado Python 2.7.6
- Tener la librería PySide para visualizar ventanas creadas en QT

Hardware:

- 512mb de memoria RAM
- 1ghz frecuencia de trabajo de CPU
- Periféricos: Monitor, dispositivo señalador(Mouse) y teclado para las entradas.

COMO USAR:

Para ejecutar la aplicación es necesario abrir el archivo AFDController.py con el intérprete y ejecutarlo. O en su defecto mediante consola, una vez estando en la carpeta raíz del proyecto ejecutar "python AFDController.py". Una vez hecho esto vera una ventana como esta:



En la imagen podemos apreciar los siguientes aspectos:

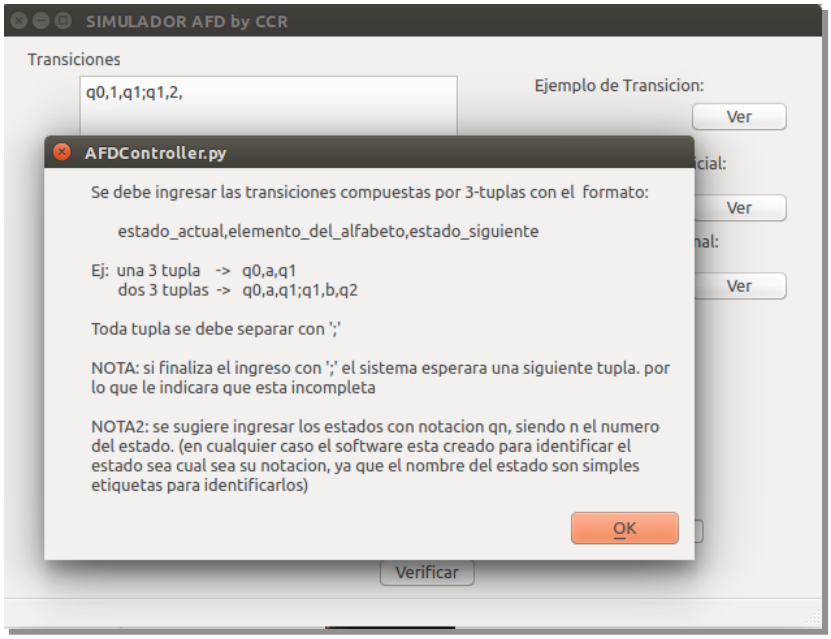
1 – Corresponde a un recuadro para ingresar las respectivas tuplas ("TextBox")

2 – Acá tenemos 2 recuadros ambos con sus respectivos etiquetas explicativas de que es lo que solicitan (Estado inicial y Estado final).

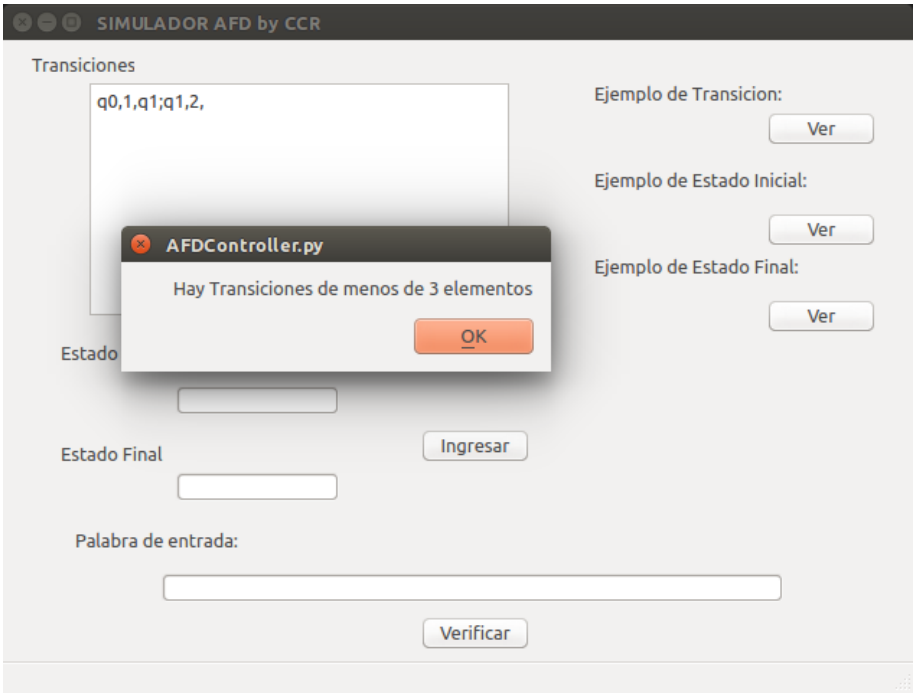
3 – En este apartado se nos solicitara ingresar la cadena completa que contendrán cada tuplas y a su vez los estados y transiciones correspondientes. Al ingresarlas el programa verificara que siga la correcta sintaxis de separación de elementos por comas y tuplas por punto y coma. En caso de error se indicara mediante ventana emergente solicitando el ingreso nuevamente.

4 – En caso de que el usuario tenga alguna duda de cómo interactuar con este programa hemos dejado a disposición en el apartado (4) el segmento de ayuda, el cual le proporcionara la información requerida de como se debe ingresar el dato solicitado.

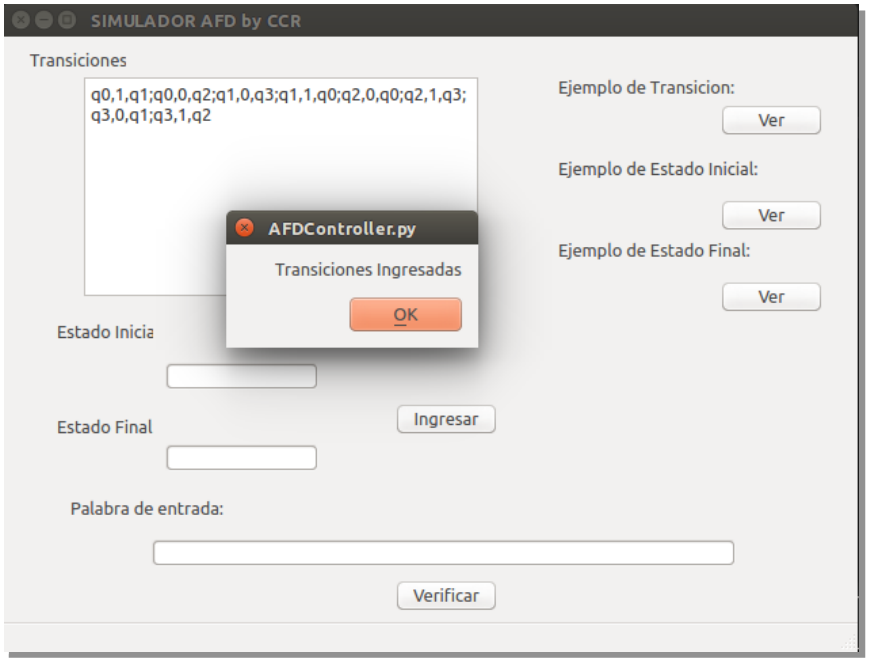
EJEMPLO DE FUNCIONAMIENTO:



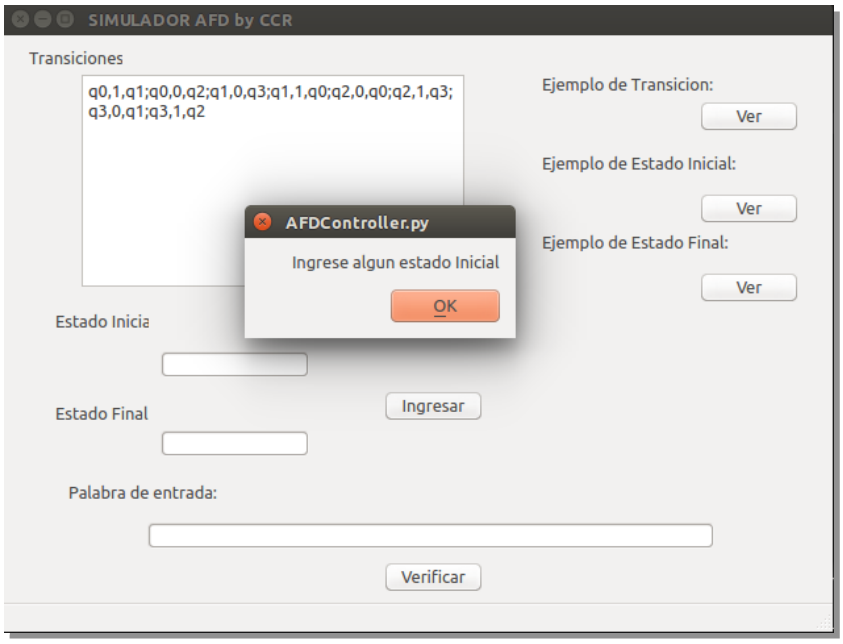
Podemos ver un ejemplo del apartado 4 al consultar por como ingresar las Transiciones.



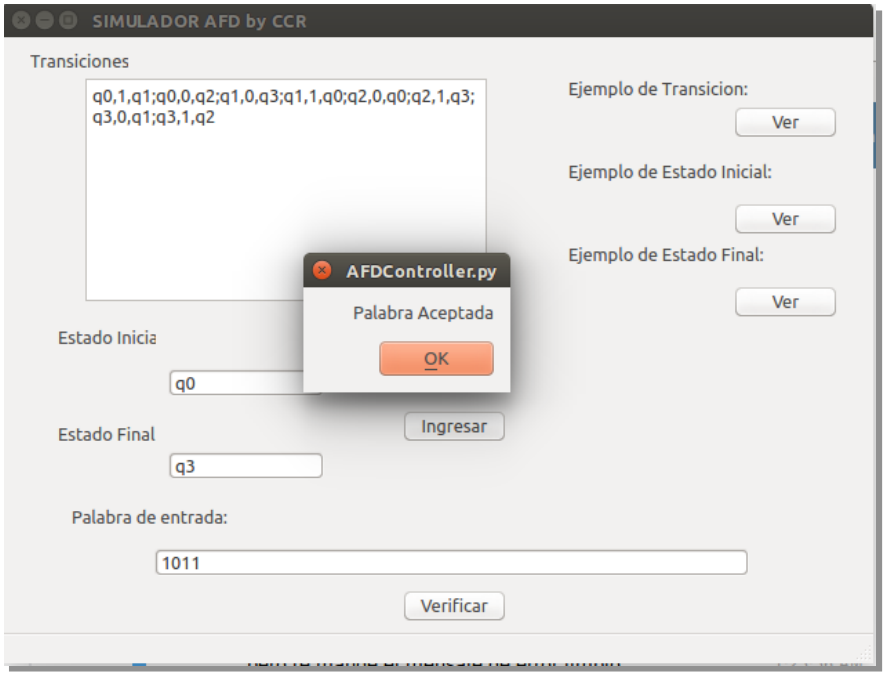
El realizar el sistema de ingreso de transiciones incorrectamente, el programa alerta para el ingreso correcto.



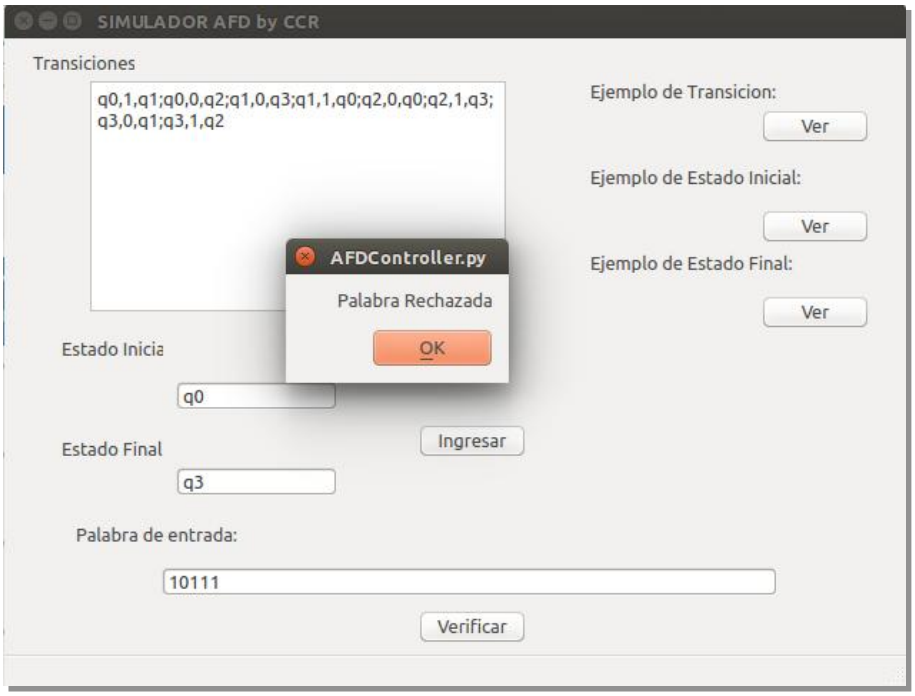
En este ejemplo podemos ver como muestra las transiciones ingresadas y una ventana emergente validando su ingreso.



En la imagen anterior nos muestra que seguido de la validación este solicitara los datos necesarios respectivamente como por ejemplo el estado inicial. Análogamente hará lo mismo con estado final.



Ejemplo de salida 1: Acá podemos apreciar como el programa al ejecutarse con la sentencia "1011" arroja la ventana emergente validando que es aceptada.



Ejemplo de salida 2: En esta imagen podemos ver el resultado de ingresar una cadena que no es aceptada en la simulación y su respectiva ventana emergente señalándolo.

BIBLIOGRAFIA

http://repositorio.utp.edu.co/dspace/bitstream/11059/1727/1/5113G643.pdf

CONCLUSIÓN

Hoy por hoy todo tipo de sistema automatizado está basado en procesos, de diferentes tipos de estados que realizan variadas tareas, o en algunos casos no realizan procesos, para todo esto se necesita tener un manejo sobre este tipo de procesos que contemplan el funcionamiento de máquina, llevando no solo un control, sino que también su eficiencia e interacción con el usuario.

Logramos el cómo poder realizar la simulación de un sistema de autómatas finitos deterministas, donde por diferentes tipos de estados, realiza tareas básicas de lectura para un determinado lenguaje, entregando resultados de su funcionamiento. Nos ayuda a comprender él como una maquina realiza distintos tipos de tareas en variados tiempos de ejecución en cada proceso, que para la actualidad basados en complejidad, supera con creces lo que se puede lograr sistemas autómatas día a día.

MEJORAS:

- Se añadió una interfaz gráfica que facilita el uso y comprensión de la ejecución de nuestro programa
- Validaciones de Transiciones con cantidad de elementos, donde las tuplas se particionan mediante comas "," y ";" lo que causaba fallas.
- Validación de palabra de entrada
- Validación estado inicial y estado final

LIMITANTES:

- Debido a falta de librerías no fue posible crear el ejecutable con compatibilidad para Windows

NOTA:

Por recomendación se sugiere usar notación q0,q1...qn. para los estados, de cualquier modo el software está diseñado para identificar estados independiente de su notación, ya que el nombre del/los estado(s) son simples etiquetas para identificarlos en sus distintas transiciones de funcionamiento.