TALFi

Una herramienta de aprendizaje para
Teoría de Autómatas y Lenguajes Formales

PROYECTO DE SISTEMAS INFORMÁTICOS

Autores:

Miguel Ballesteros Martínez
José Antonio Blanes García
Samer Nabhan Rodrigo

Director: Alberto de la Encina Vara

Facultad de Informática.
Universidad Complutense de Madrid.
Curso 2008-2009
Junio 2009

INDICE

Capítulo 1.

Resumen

1.1 TALFi (Versión en castellano)

TALFi es una aplicación cuyo objetivo es ser una herramienta para el aprendizaje y el uso de diversos algoritmos aplicados al tratamiento de autómatas. Con TALFi podemos crear autómatas y ver sus transformaciones en otros autómatas, gracias a los diferentes algoritmos que se pueden aplicar sobre ellos. Es una herramienta de fácil uso, capaz de ser usada en un terminal o bajo una interfaz gráfica. La aplicación se puede ejecutar bajo previa instalación en un equipo o vía internet.

TALFi dispone de una base de datos con diversos ejemplos de autómatas, la cual puede ser ampliada por el usuario con nuevos autómatas creados por él mismo. Además la aplicación posee una base de datos con ejercicios que también puede ser extendida, pero en este caso sólo por el administrador.

La aplicación está pensada para ser una herramienta de ayuda para los estudiantes que cursen asignaturas donde se traten temas de lenguajes formales y autómatas.

1.2 TALFi (English version)

TALFi is an application whose goal is to be a tool for learning and using various algorithms dealing with automata. With TALFi you can create automata and view the automata transformations into other automata by using the different algorithms that can be applied to them. The program is easy to use and it can be used in a shell or a graphical interface. The application can be used online through the Internet or it can be installed in any computer to be used offline.

TALFi has a database with examples of automata. These examples can be extended by the user with new automata created by himself. The application also has a database with exercises that can be extended by an administrator.

The application is designed to be used by students learning subjects about formal languages and automata.

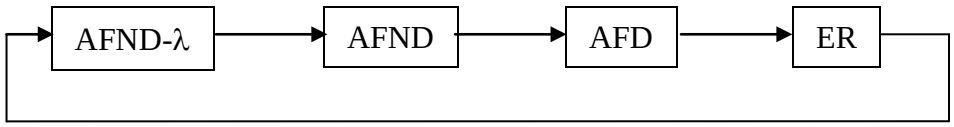
Capítulo 2.

Objetivos

El objetivo principal de la aplicación es ser una herramienta útil de estudio para los estudiantes de asignaturas que traten sobre autómatas y las diferentes interacciones entre ellos. Como el motivo docente es evidente, hemos implementado una herramienta visual, donde el usuario puede interaccionar en una interfaz gráfica, en donde se muestra todo el potencial de la aplicación.

Otro de los objetivos es que los alumnos, o usuarios, puedan con *TALFi* resolver ejercicios de autómatas de manera automática y servirse de dicha herramienta para averiguar la corrección o incorrección de sus soluciones. Por tanto, la solución que les mostrará nuestra herramienta se podrá utilizar tanto para averiguar cómo se realiza un ejercicio, como para comprobar la solución propia averiguando en que paso se ha cometido un error o si ésta es correcta.

Para poder cumplir estos objetivos creemos conveniente que la aplicación implemente todos los algoritmos automáticos de la parte de lenguajes regulares. Es decir, cualquier transformación entre los distintos de autómatas finitos y las expresiones regulares. Por lo tanto, nuestra aplicación permitirá generar un ciclo, pasando por todos los autómatas y expresiones regulares. Esta transformación se realizará siguiendo el siguiente esquema:



También nos parece interesante que nuestra aplicación permita minimizar los AFD. De esta forma se podrá comprobar la equivalencia entre iguales. Esta equivalencia se logrará a través de la transformación de cualquiera de ellos en el correspondiente AFD mínimo y se comprobará la igualdad entre los autómatas finitos deterministas correspondientes.

La codificación de la aplicación se enfocara en dos partes diferenciadas: una que corresponde con la lógica de los algoritmos y de la aplicación en general y otra que corresponderá con la parte de interacción con el usuario y muestra de resultados. En una primera etapa bajo una vista por consola y a posteriori con una interfaz gráfica de usuario amigable y de fácil manejo.

La consola utilizará el enfoque UNIX ya que es un estándar muy usado en la actualidad y conocido por todo el mundo. La aplicación gráfica generará órdenes de consola automáticamente en función de la selección del usuario. De esta manera será posible realizar mejoras en la codificación de la interfaz sin tener que modificar la lógica de la aplicación, ya sea creando diferentes interfaces gráficas sobre la misma consola o modificando la existente.

Hay que tener en cuenta que pretendemos que la aplicación sea fácilmente mejorada y ampliada con nuevos autómatas, máquinas de Turing y otros conocimientos que se estudian en Teoría de Autómatas que no se han tenido en esta. Por tanto, implementaremos el código de los algoritmos e interfaz gráfica con vistas a posibles mejoras y ampliaciones futuras. Además dicho código deberá estar documentado con el

fin de que pueda ser fácilmente reutilizable en futuros proyectos de ampliación y mejora.

Una de nuestras pretensiones es que dicha aplicación no sólo sea utilizada por los alumnos de nuestra universidad, sino que pueda ser utilizada en otras universidades. Para ello, y con el fin de ampliar el público que pueda utilizarla, la aplicación debe ser traducible a diferentes idiomas. Además, deberá ser posible cambiar de un idioma a otro dinámicamente en cualquier momento durante el uso de la aplicación y también deberá ser sencilla la traducción de la aplicación o otros idiomas.

Por último, pretendemos que la aplicación pueda ser utilizada para el desarrollo de ejercicios a distancia y permita un control individualizado sobre los conocimientos de cada uno de sus usuarios. Para ello, permitirá un seguimiento a los alumnos en base a la resolución de ejercicios, que se corrigen automáticamente. Esto se logrará guardando información en la base de datos de los ejercicios que cada alumnos haya realizado correcta e incorrectamente.

Capítulo 3.

Información

3.1 Antecedentes

No conocemos ningún antecedente directo o indirecto de una aplicación que trate el tema de los autómatas con la intención docente que le hemos dado nosotros. Precisamente por ello elegimos éste proyecto, ya que nos gustaba la idea de crear algo que ayudara a los alumnos en su estudio de autómatas.

Si bien no existe ninguna herramienta con marcado carácter docente de este tipo, si que existe una llamada **JFLAP** de la que nuestro proyecto toma ciertas ideas intuitivas a la hora del diseño de la interfaz gráfica, ya que en sí la que plantea dicha herramienta nos parece sencilla en cuanto a su manejo. Sin embargo hemos añadido mucha funcionalidad a la parte gráfica y desarrollado los algoritmos desde cero con la intención docente de que los alumnos puedan visualizar los pasos al aplicar cualquier algoritmo, opción no disponible en JFLAP.

Existen otras aplicaciones desarrolladas en la misma línea, como por ejemplo:

- dk.brics.automaton

Es un paquete de Java que contiene una implementación de autómatas con estados finitos que soportan operaciones con expresiones regulares.

Se puede descargar gratuitamente desde la página web:

http://www.brics.dk/automaton/

- Visual automata Simulator

Herramienta para la simulación, visualización y transformación de autómatas finitos y máquinas de Turing.

Se puede descargar gratuitamente desde la página web:

http://www.versiontracker.com/dyn/moreinfo/win/35508

- Proyecto SEPa!

Proyecto que busca la creación de una herramienta conjunta de simulación de autómatas y traductores de lenguajes formales.

Se puede descargar gratuitamente desde la página web:

http://www.ucse.edu.ar/fma/sepa/

3.2 Introducción teórica a la aplicación

En la aplicación hemos tratado el estudio de los autómatas y el uso de diferentes algoritmos para interactuar con ellos. Para un correcto estudio de la aplicación, es necesario conocer algunos conceptos teóricos.

- **Lenguaje formal**

En matemáticas, lógica, y ciencias de la computación, un lenguaje formal es un conjunto de palabras (cadenas de caracteres) de longitud finita en los casos más simples o expresiones válidas (formuladas por palabras) formadas a partir de un alfabeto

(conjunto de caracteres) finito. El nombre lenguaje se justifica porque las estructuras que con éste se forman tienen reglas de sintaxis (gramática) e interpretación semántica (significado) en una forma muy similar a los lenguajes hablados o naturales.

Un posible alfabeto sería, digamos, $\{a, b\}$, y una cadena cualquiera sobre este alfabeto sería, por ejemplo, $ababba$. Un lenguaje sobre este alfabeto, que incluyera esta cadena, sería por ejemplo el conjunto de todas las cadenas que contienen el mismo número de símbolos a que b.

La palabra vacía (esto es, la cadena de longitud cero) se permite en este tipo de lenguajes. Para referirnos a ella usamos el símbolo λ, o simplemente la llamaremos "lambda". A diferencia de lo que ocurre con el alfabeto (que es un conjunto finito) y con cada palabra (que tiene una longitud también finita), un lenguaje puede estar compuesto por un número infinito de palabras.

- **El concepto de autómata**

Un autómata es un modelo matemático para una máquina de estado finita (Finite State Machines, FSM sus siglas en inglés).

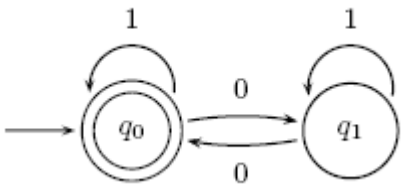
Una FSM es una máquina que, dada una entrada de símbolos, "salta" a través de una serie de estados de acuerdo a una función de transición (que puede ser expresada como una tabla). En la variedad común "Mealy" de FSMs, esta función de transición dice al autómata a qué estado cambiar dados unos determinados estado y símbolo.

En este proyecto, únicamente hemos tratado autómatas finitos. Hay 3 tipos de autómatas finitos:

→ **Autómata finito determinista (AFD)**

Cada estado de un autómata de este tipo tiene una transición por cada símbolo del alfabeto desde cada estado.

Así, por ejemplo, si partimos del alfabeto {0,1}, el autómata que se presenta a continuación sería un autómata finito determinista que reconoce la palabras que contienen un número par de ceros.



→ **Autómata finito no determinista (AFND)**

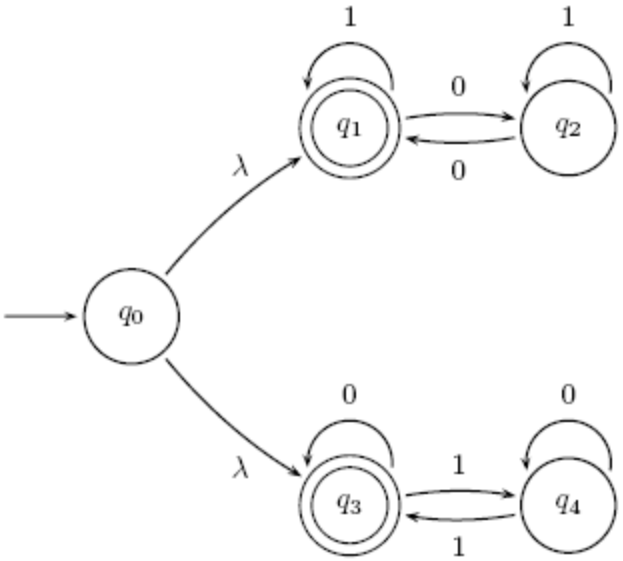
Los estados de un autómata de este tipo pueden tener varias transiciones por cada símbolo del alfabeto desde el mismo estado. El autómata acepta una palabra si existe al menos un camino desde el estado inicial a un estado final *etiquetado* con la palabra de entrada. Si una transición no está definida, de manera que el autómata no puede saber cómo continuar leyendo la entrada, la palabra es rechazada.

→ **Autómata finito no determinista, con transiciones λ (AFND-λ)**

Además de ser capaz de alcanzar más de un estado leyendo un símbolo, igual que el AFND, permite alcanzar otro estado sin leer ningún símbolo. Si un estado tiene transiciones etiquetadas con λ, entonces el AFND- **λ** puede *encontrarse* en cualquier de

los estados alcanzables por las transiciones λ, directamente o a través de otros estados con transiciones λ. El conjunto de estados que pueden ser alcanzados mediante este método desde un estado q, se denomina el cierre lambda de q.

El autómata que se presenta a continuación parte del alfabeto de entrada {0,1} y reconoce las palabras que contienen una cantidad par de ceros o de unos.



Todos estos tipos de autómatas (AFD, AFND y AFND-λ) **aceptan los mismos lenguajes**, es decir, los lenguajes regulares. Por lo tanto, siempre se puede construir un AFD que acepte el mismo lenguaje que el dado por un AFND o que un AFND-λ y al revés.

- **Expresión regular sobre un alfabeto (ER)**

Una expresión regular describe un conjunto de cadenas sin enumerar sus elementos. Toda expresión regular básicamente se puede ver como una cadena formada por las letras del alfabeto y ciertos operadores. Para que la cadena sea considerada una expresión regular deberá seguir una construcción gramatical correcta.

A continuación presentamos las reglas para la construcción de expresiones regulares de manera recursiva:

1. λ, la palabra vacía, es una expresión regular. En nuestra aplicación la palabra vacía se representa con el símbolo
2. Ø es una expresión regular también, que identifica el conjunto vacío de palabras que se reconocen, es decir, ninguna.
3. Cualquier letra perteneciente al alfabeto es una expresión regular en sí misma.

Dadas las anteriores expresiones regulares, consideremos ahora que E y F son expresiones regulares correctamente formadas entonces:

4. E + F (operación de suma) es también una expresión regular.
5. E·F o EF (operación de concatenación) es una expresión regular correcta. En nuestra aplicación optaremos por la segunda representación.

6. E* (operación de clausura) es una expresión regular correcta (E^+ no es estrictamente una expresión regular pero se acepta ya que es una abreviación de EE*).
7. (E) es una expresión regular correcta.

Y dadas estas reglas, no existe ninguna expresión regular correcta que no pueda formarse con algunas de las siete reglas anteriores.

3.3 Viabilidad

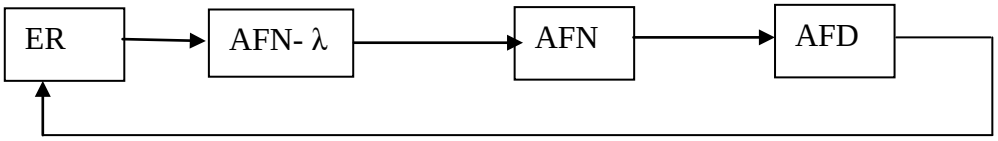
La viabilidad del proyecto ha sido lo primero que hemos tenido que "acotar", dado que es una materia donde se puede entrar en muchos temas distintos. Como el objetivo de la aplicación es ser una herramienta docente, hemos querido centrarnos en uno de los temas principales del estudio de autómatas: la interacción entre los diferentes autómatas finitos y el paso de una expresión regular a su equivalente autómata.

Una vez acotado nuestro proyecto, no hemos encontrado ningún punto teórico que haya sido insalvable.

3.4 Ejercicios posibles

Los algoritmos implementados en TALFi permiten resolver un buen conjunto de ejercicios. Se permiten resolver ejercicios que tienen un algoritmo de resolución automático y además se puede comprobar la solución de ejercicios no automáticos.

Los ejercicios automáticos permiten transformar desde la expresión regular hasta el autómata finito no determinista. Cerrando el círculo de ejercicios posibles de la temática, como se observa en el siguiente gráfico:



En el gráfico, cada flecha es un algoritmo implementado.

3.4.1 Ejercicios automáticos

1. Ejercicio de minimización de autómatas (AFD-->AFDMinimo)

El algoritmo de minimización de autómatas que hemos desarrollado, consiste en lo siguiente:

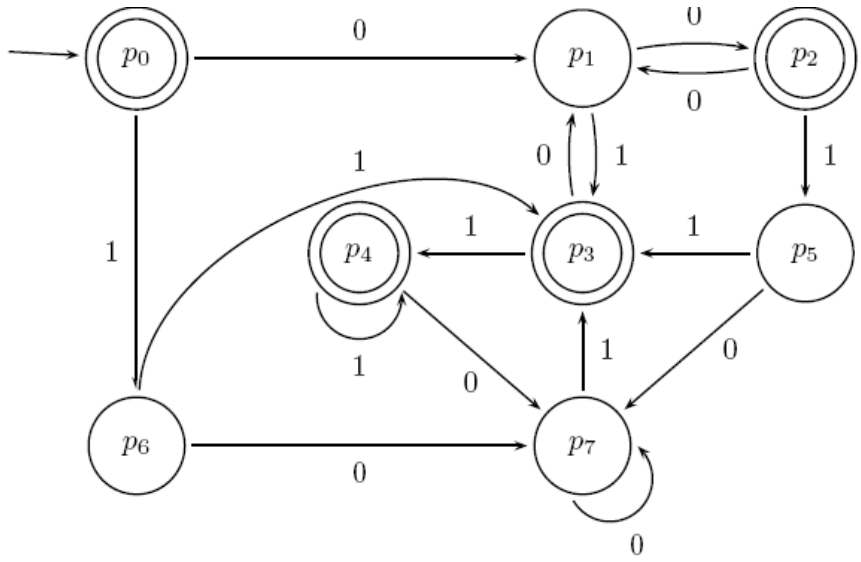
Entrada: autómata finito determinista.
Salida: autómata finito determinista mínimo.

Se realiza una tabla de distinguibilidad de autómatas y se colapsan estados indistinguibles.

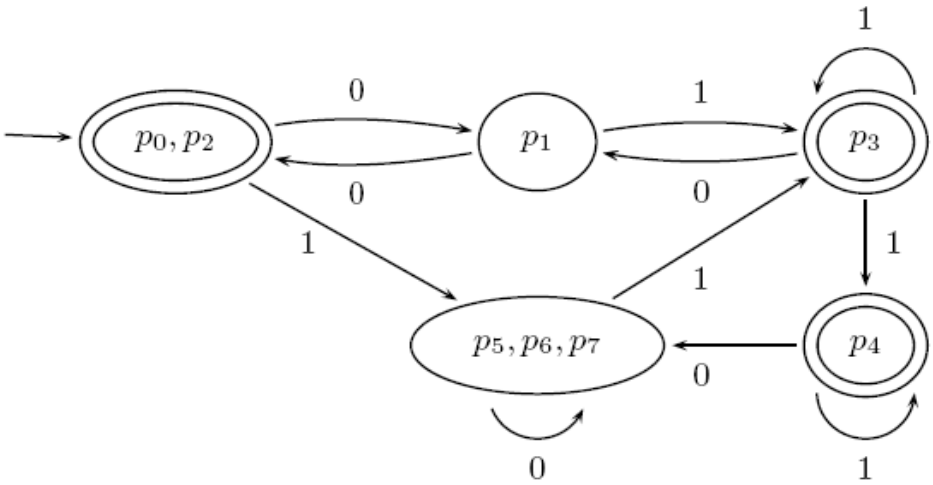
No se ha implementado el algoritmo existente de autómata mínimo desde un autómata finito con transiciones lambda.

Ejemplo de ejecución del algoritmo:

Minimiza el siguiente autómata finito determinista:

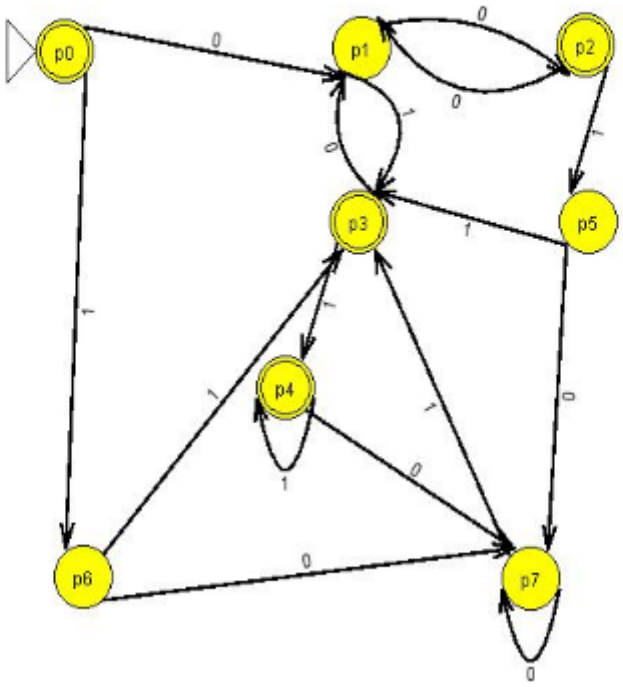


p_1	X_1						
p_2		X_1					
p_3	$X_2(p_6, p_4)$	X_1	$X_2(p_5, p_4)$				
p_4	$X_2(p_6, p_4)$	X_1	$X_2(p_5, p_4)$	$X_3(p_7, p_1)$			
p_5	X_1	$X_2(p_7, p_2)$	X_1	X_1	X_1		
p_6	X_1	$X_2(p_7, p_2)$	X_1	X_1	X_1		
p_7	X_1	$X_2(p_7, p_2)$	X_1	X_1	X_1		
	p_0	p_1	p_2	p_3	p_4	p_5	p_6



En TALFi obtenemos el mismo resultado como puede verse ejecutando la aplicación y capturando por pantalla la ejecución por pasos:

Entrada



Las X indican no distinguibilidad de estados, y los + indican distinguibilidad de estados.

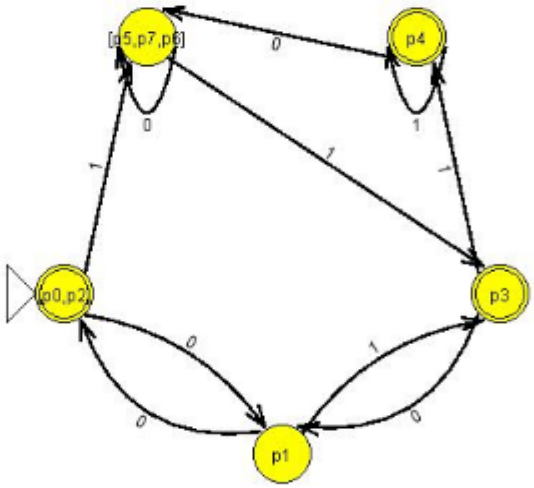
Pasos-Tabla de minimizacion

p1	X						
p2	+	X					
p3	X	X	X				
p5	X	X	X	X			
p7	X	X	X	X	+		
p4	X	X	X	X	X	X	
p6	X	X	X	X	+	+	X
	p0	p1	p2	p3	p5	p7	p4

Como se puede ver en la imagen los estados p0 y p2 son indistinguibles, por tanto pueden colapsarse en uno solo. Lo mismo pasa con las siguientes parejas de estados: (p5,p6), (p5,p7), (p7,p6). En este caso pueden colapsarse en nuevo estado: (p5,p6,p7).

Aquí podemos observar el autómata final obtenido tras la minimización. Donde se ven las nuevas transiciones y los nuevos estados que lo conforman.

Resultado del algoritmo de minimizacion



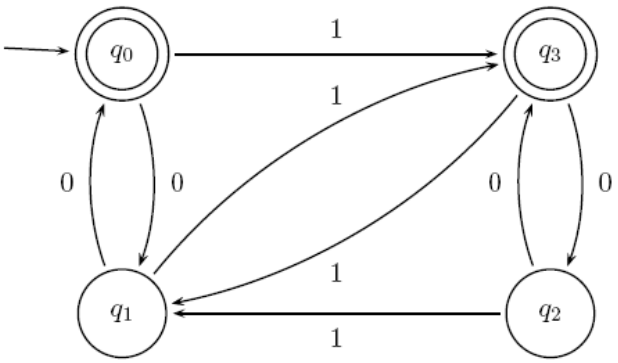
2. Ejercicio de transformación de AFD a Expresión regular (AFD-->ER)

El algoritmo desarrollado para resolver este ejercicio consiste en lo siguiente:

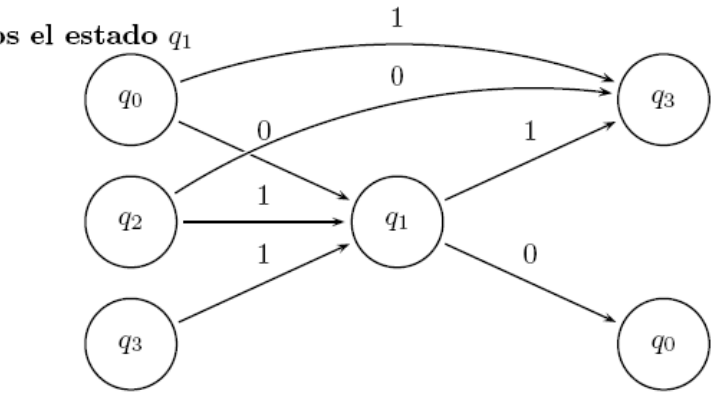
Entrada: autómata finito determinista.
Salida: Expresión regular.

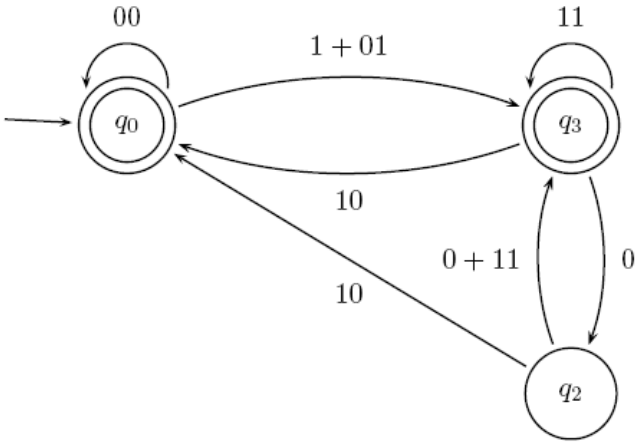
Paso 1: Eliminar todos aquellos estados que no son finales (salvo el inicial) y añadir su parte de expresión regular a las transiciones entre estados finales.
Paso 2: Calcular para cada estado final su expresión regular.
Paso 3: Eliminar todos los estados finales (salvo el inicial) y devolver la expresión regular resultante en el estado inicial.

Dado el siguiente autómata finito, se pide obtener la expresión regular equivalente:

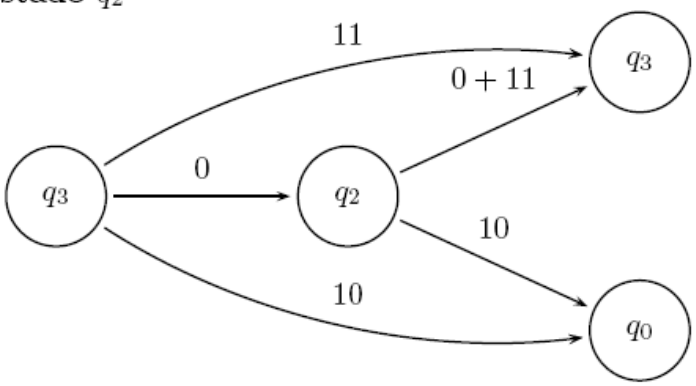


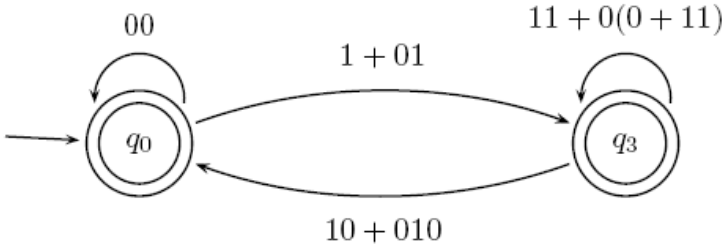
Eliminamos el estado q_1



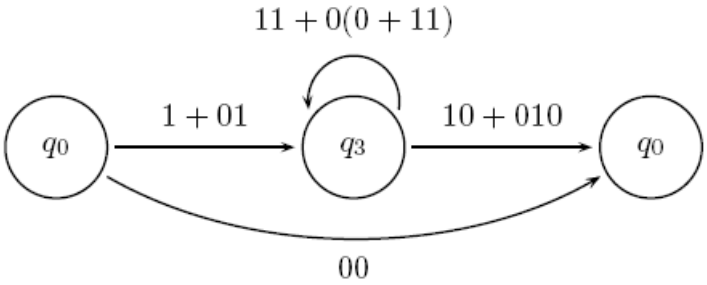


Eliminamos el estado q_2

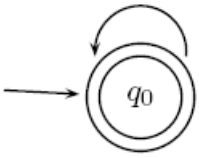




Para calcular la expresión regular para q_0 eliminamos q_3



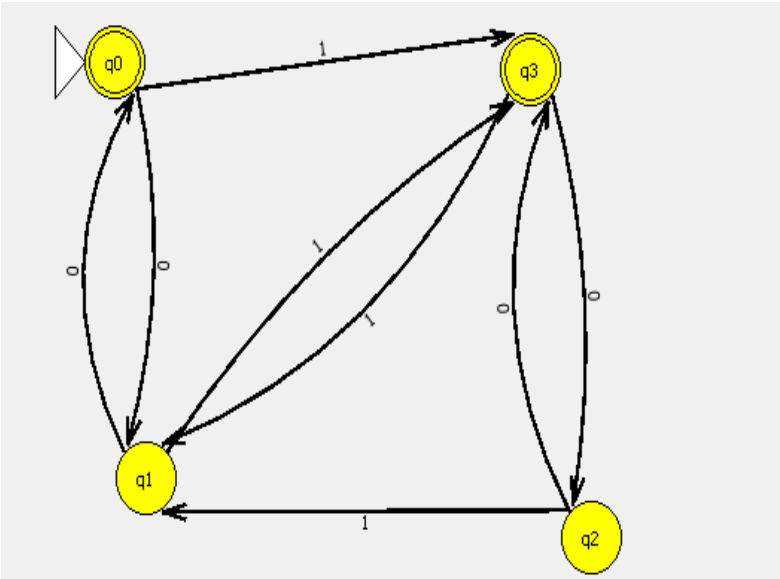
$$00 + (1 + 01)(11 + 0(0 + 11))^*(10 + 010)$$



Finalmente Su expresión regular asociada es:

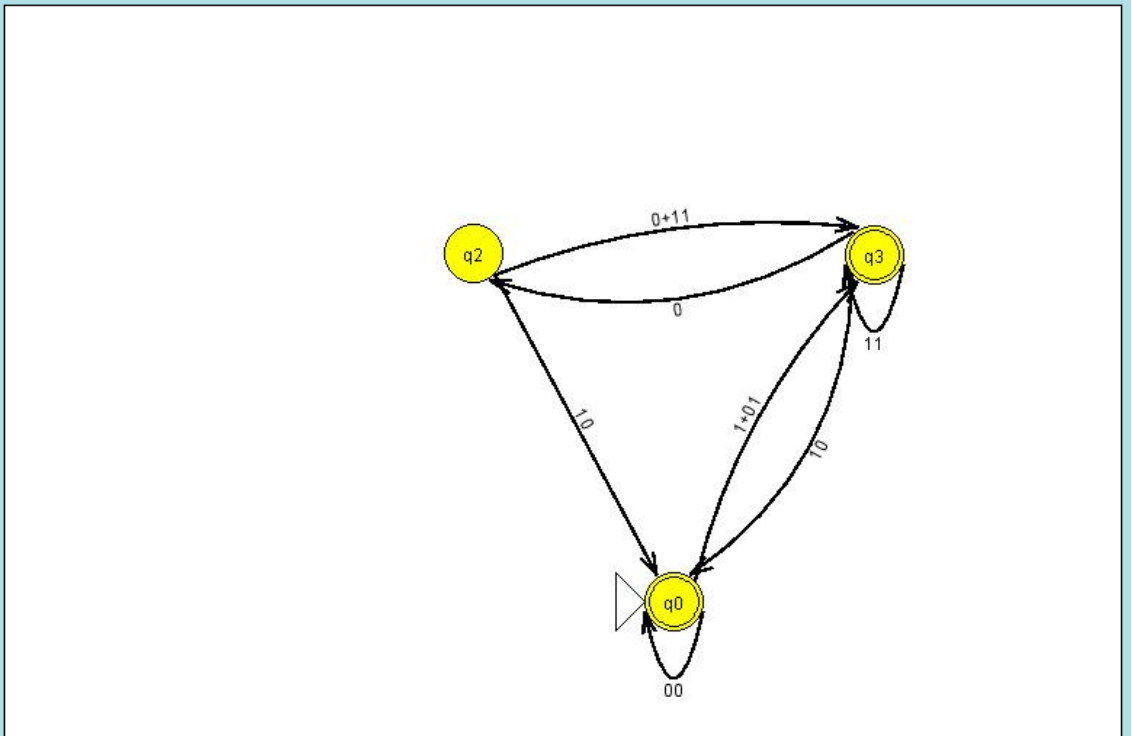
((00+(1+01)(11+0(0+11))*(10+010))*)*+(00+(1+01)(11+0(0+11))*(10+010))*(1+01)(11+0(0+11))*

En la aplicación TALFI obtenemos el mismo resultado:

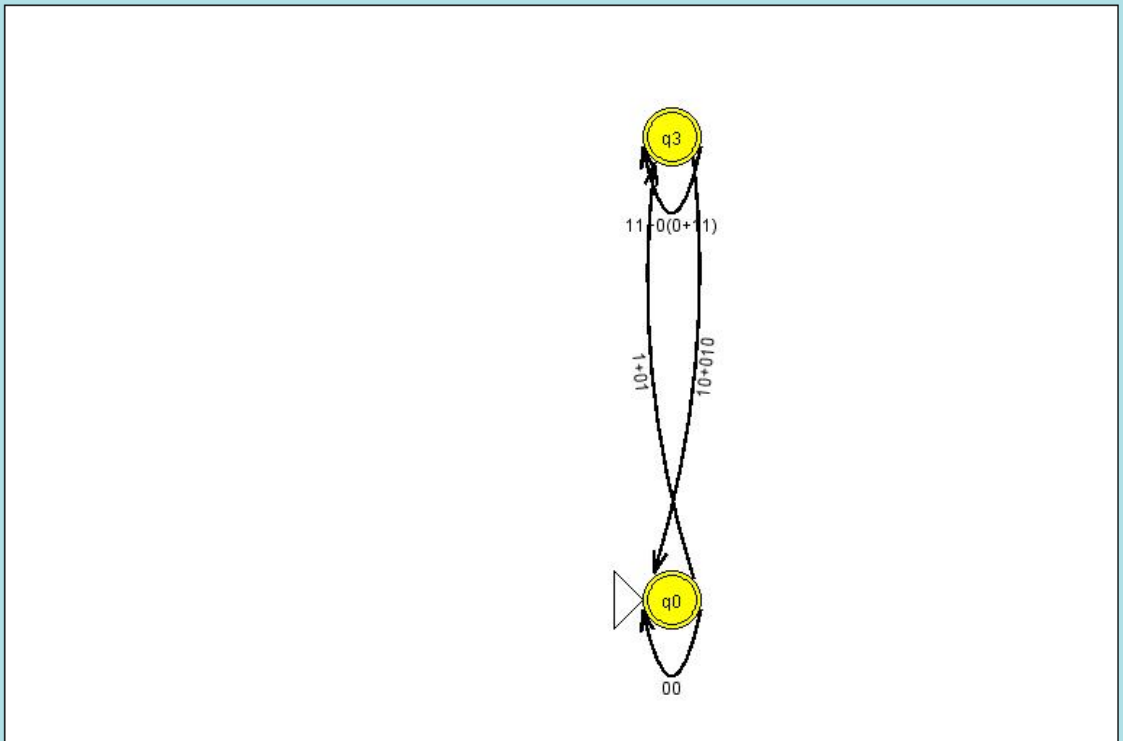


Obteniendo la salida por pasos del algoritmo en la aplicación sobre el mismo autómata finito, obtenemos:

ELIMINAMOS EL ESTADO: Q1



ELIMINAMOS EL ESTADO: Q2



Expresion regular: ((00+(1+01)(11+0(0+11))*(10+010))*)*+(00+(1+01)(11+0(0+11))*(10+010))*(1+01)(11+0(0+11))*

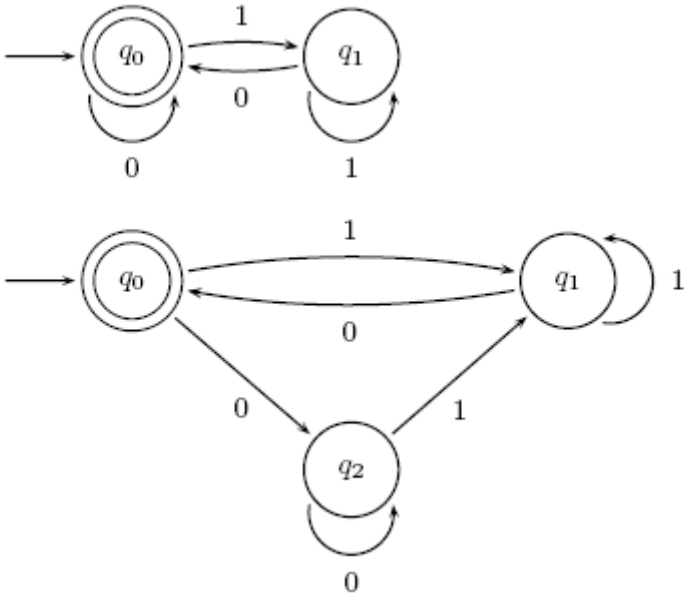
3. Ejercicio de equivalencia de autómatas ((AFD, AFD) -->Resultado)

El algoritmo desarrollado para resolver este ejercicio consiste en lo siguiente:

Entrada: Dos autómatas finitos deterministas.
Salida: Resultado cierto si los autómatas son indistinguibles, falso en caso contrario.

El algoritmo genera un autómata unión entre los dos autómatas de entrada, y genera la tabla del algoritmo de minimización. La condición de equivalencia es la siguiente: si cada uno de los estados de uno de los autómatas son indistinguibles con al menos uno de los estados del otro autómata, los autómatas son indistinguibles.

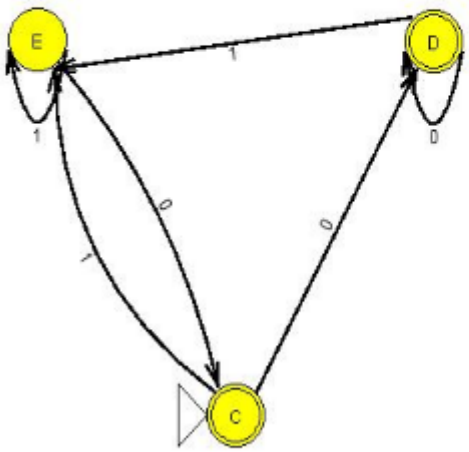
Decide la equivalencia entre estos dos autómatas finitos:

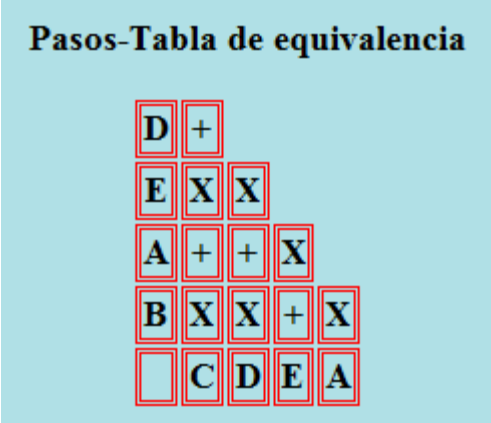


Si ejecutamos sobre TALFi la equivalencia entre estos dos autómatas obtenemos:

Automata 1

Automata 2





Los estados A y B pertenecen al primer autómata, los estados C, D y E pertenecen al segundo autómata.

Observamos que:

→A es indistinguible con C.

→B es indistinguible con E.

→A es indistinguible con D.

Los autómatas son equivalentes.
El resultado es este ya que todos los estados son indistinguibles con al menos uno de los estados del otro autómata, por tanto los autómatas son equivalentes.

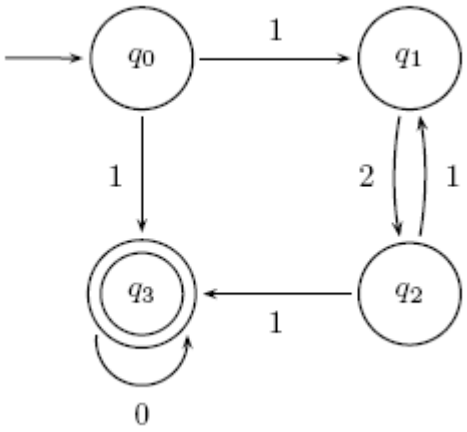
4. Ejercicio de transformación de autómata finito no determinista a autómata finito determinista. (AFN-->AFD)

El algoritmo desarrollado para resolver este ejercicio consiste en lo siguiente:

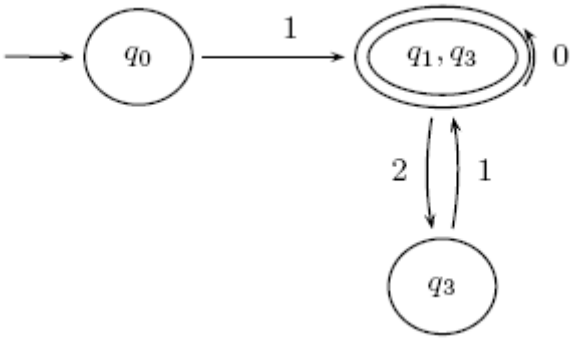
Entrada: autómata finito no determinista.
Salida: autómata finito determinista.

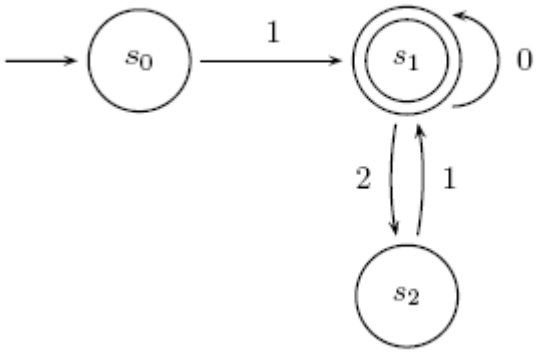
Se va creando un autómata finito nuevo recorriendo los estados y colapsando aquellos a los que se puede llegar con las mismas palabras, el resultado final es un autómata finito determinista.

Obtén el autómata finito determinista a partir de este autómata finito no determinista.



Se obtiene el siguiente autómata aplicando el algoritmo:

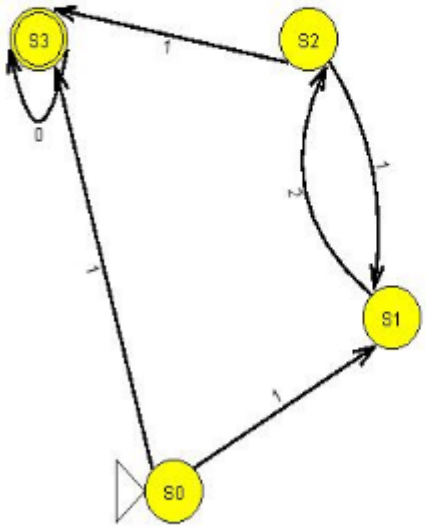




Renombrando:

Si ejecutamos el ejemplo en la aplicación obtenemos:

Entrada

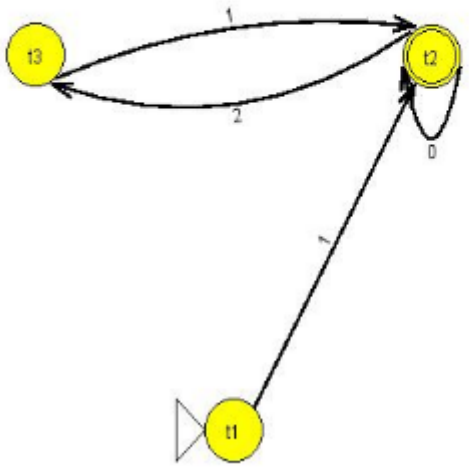


Pasos

	0	2	1
t1=[S0]	[ ]	[ ]	[S1, S3]
t2=[S1, S3]	[S1, S3]	[S2]	[ ]
t3=[S2]	[ ]	[ ]	[S1, S3]

El algoritmo va recorriendo los estados y generando nuevos a partir de las transiciones correspondientes, generando un nuevo autómata pero en este caso, determinista.

Salida



El autómata obtenido no contiene todas las transiciones que tendría un autómata finito determinista puro. Faltaría incluir un estado basura, de no aceptación, al que lleguen las transiciones que faltan.
Hemos decidido no incluir dicho estado por simplicidad y facilidad de comprensión del resultado devuelto.

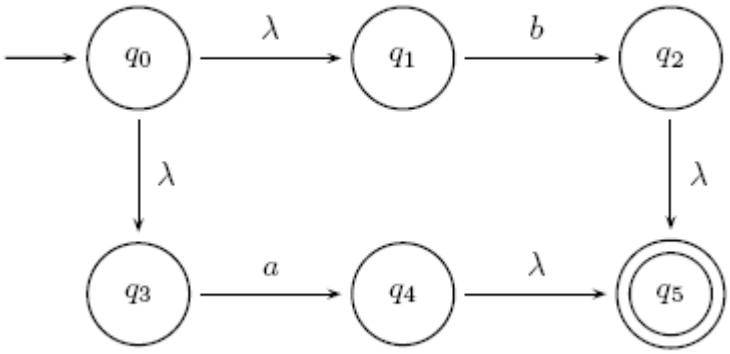
5. Ejercicio de transformación de AFND-λ a AFND

El algoritmo desarrollado para resolver este ejercicio consiste en lo siguiente:

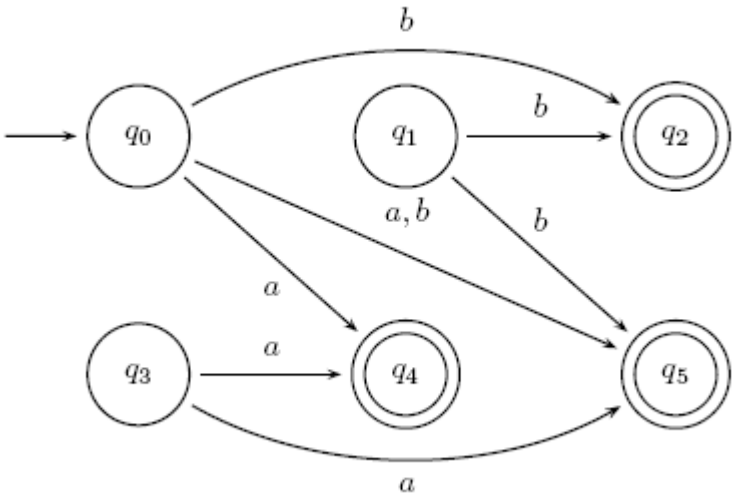
Entrada: autómata finito no determinista con transiciones lambda.
Salida: autómata finito no determinista.

Se eliminan las transiciones lambda y para cada una de ellas se añaden todas las transiciones que se encuentran en el cierre lambda del estado origen.

Dado el siguiente autómata con transiciones lambda:

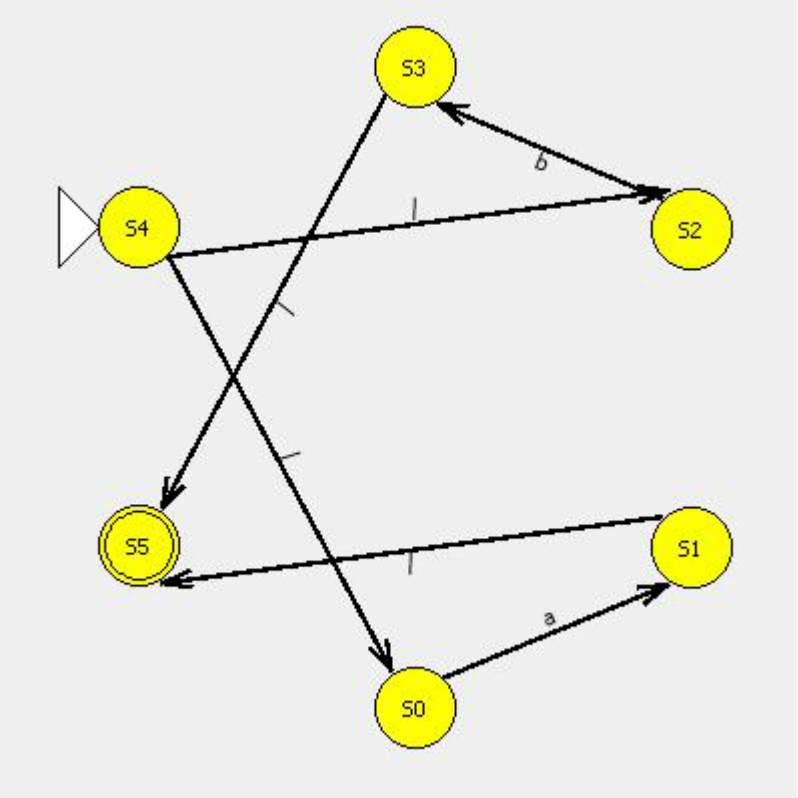


Si ejecutamos el algoritmo el autómata de salida es:



Si ejecutamos el ejemplo en TALFi se obtiene como resultado:

Entrada:



Tomamos como ejemplo el estado S4 del que salen transiciones lambda.

El cierre lambda del estado S4, es decir los estados que pueden llegar usando la lambda, serían: S4, S2, S0.

Una vez que tenemos el cierre lambda, hacemos la delta extendida del resultado del cierre lambda para cada letra del alfabeto. Vemos que con “a” vamos de S0 a S1 (y a S5 con una lambda) y que con “b” vamos de S2 a S3 (y a S5 con una lambda).

Por lo tanto, el resultado final de S4 serán dos transiciones por letra, de S4 a S1 y a S5 con “a” y de S4 a S3 y a S5 con “b”.

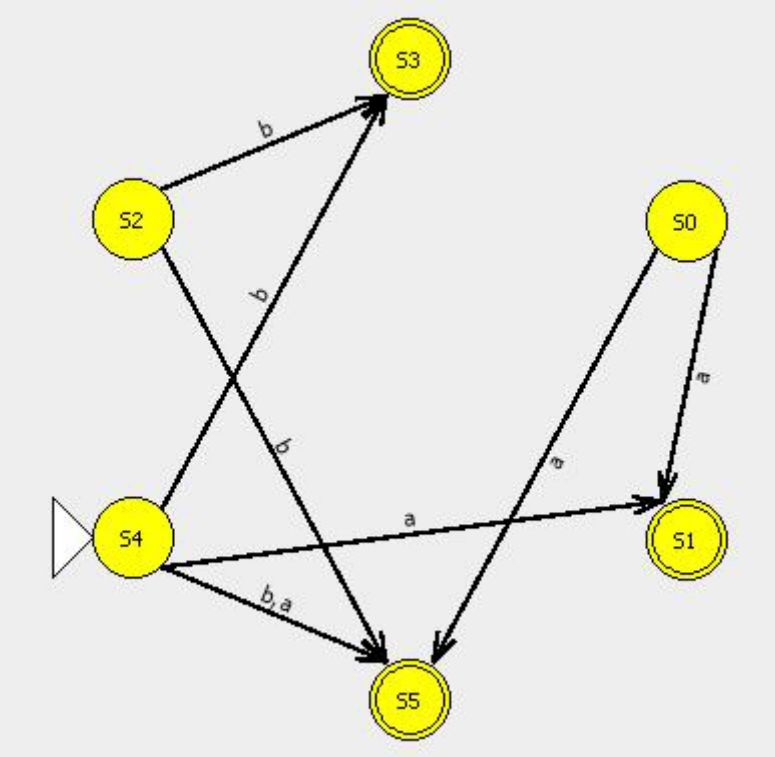
Para el resto de estados se sigue el mismo método, dando lugar a la siguiente tabla de salida, donde 0 significa que no hay transición:

Pasos

	a	b
S5	0	0
S1	0	0
S0	[S5, S1]	0
S3	0	0
S2	0	[S5, S3]
S4	[S5, S1]	[S5, S3]

El autómata de salida será el siguiente:

Salida:



6. Ejercicio de transformación de ER a AFND-λ

El algoritmo desarrollado para resolver este ejercicio consiste en lo siguiente:

Entrada: Expresión regular.
Salida: Autómata finito no determinista con transiciones lambda.

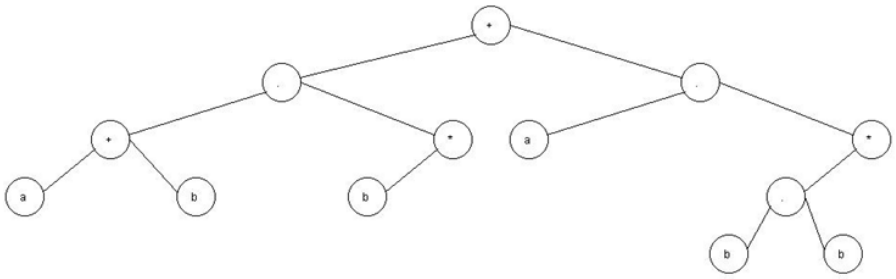
- Paso 1: Generar el árbol binario resultado del análisis sintáctico de la expresión regular.
- Paso 2: El algoritmo genera autómatas para cada "región" del árbol. Ejemplo: si encuentra a, entonces genera un autómata que acepta una sola letra: a.
- Paso 3: Une todos los autómatas generados en uno solo mediante transiciones lambda.

Dada la siguiente expresión regular: (a+b)b*+a(bb)* sobre el alfabeto A={a,b} obtener el autómata finito no determinista con transiciones lambda que la reconoce.

En la aplicación, una vez seleccionado el algoritmo correspondiente, lo primero que debemos hacer es meter al alfabeto, en esta caso meteremos “a, b”; para a posteriori meter la expresión tal cual la encontramos aquí: “(a+b)b*+a(bb)*”.

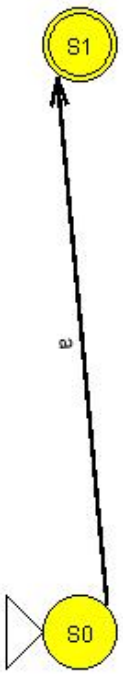
Una vez realizados estos pasos ejecutamos el algoritmo con la opción de pasos marcados y obtenemos el HTML con los autómatas dibujados en cada paso de ejecución.

Primero mostraremos el árbol sintáctico que se genera a partir de la expresión:



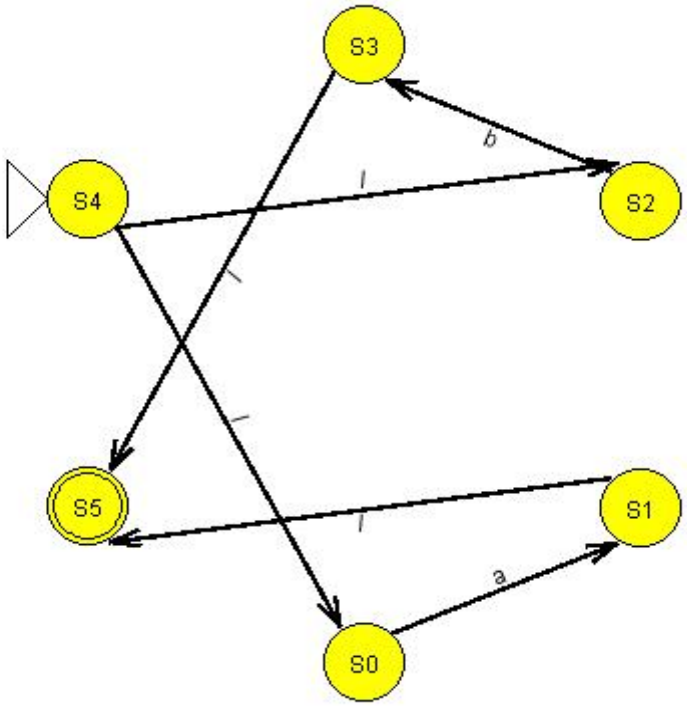
Éstas son las imágenes que muestra la página HTML de “pasos” y que se corresponden con la creación del autómata según se va recorriendo el árbol sintáctico (ver imagen superior) que se ha generado. Cada uno de los autómatas tiene como título el número de paso en que se genera y la operación que lo desencadena (como aclaración para el usuario):

PASO NUMERO 1: AUTOMATA QUE RECONOCE UNA SOLA LETRA



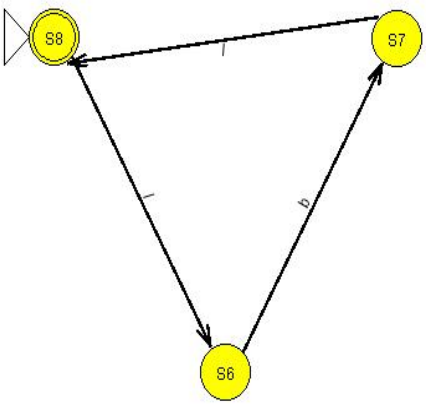
PASO NUMERO 2: AUTOMATA QUE RECONOCE UNA SOLA LETRA

PASO NUMERO 3: UNION DE DOS AUTOMATAS

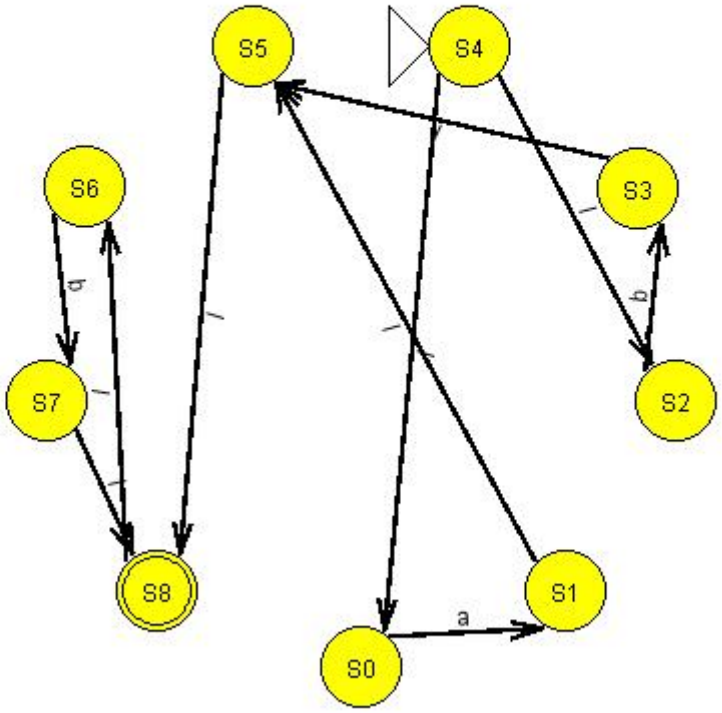


PASO NUMERO 4: AUTOMATA QUE RECONOCE UNA SOLA LETRA

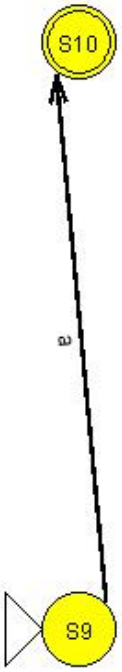
PASO NUMERO 5: AUTOMATA QUE RECONOCE LA REPETICION CERO O MAS VECES DEL ANTERIOR



PASO NUMERO 6: CONCATENACION DE DOS AUTOMATAS



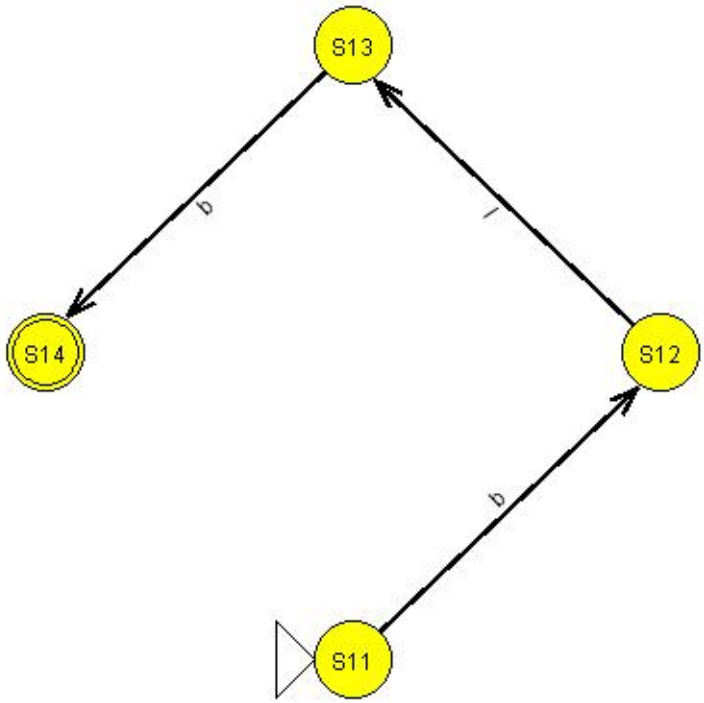
PASO NUMERO 7: AUTOMATA QUE RECONOCE UNA SOLA LETRA



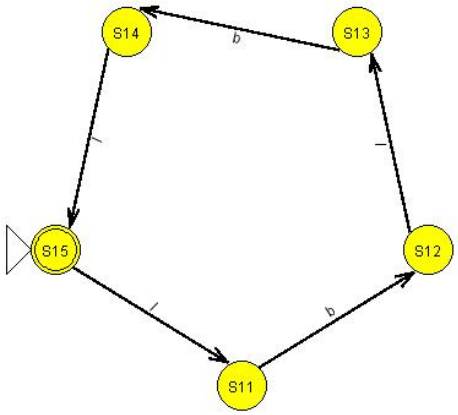
PASO NUMERO 8: AUTOMATA QUE RECONOCE UNA SOLA LETRA

PASO NUMERO 9: AUTOMATA QUE RECONOCE UNA SOLA LETRA

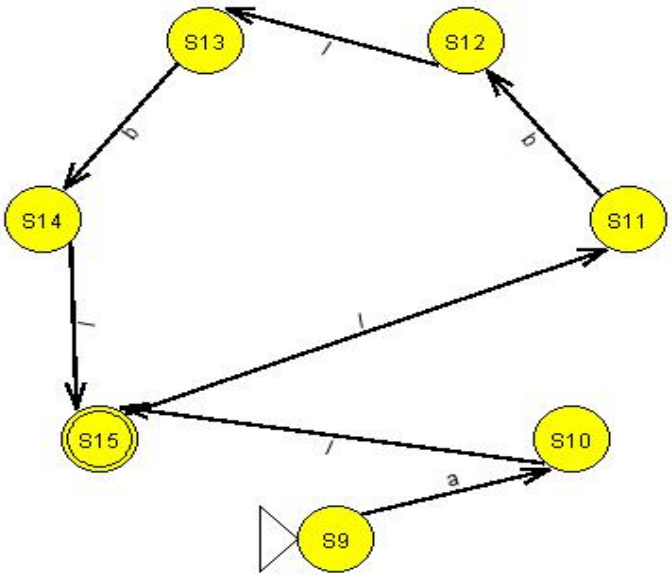
PASO NUMERO 10: CONCATENACION DE DOS AUTOMATAS



PASO NUMERO 11: AUTOMATA QUE RECONOCE LA REPETICION CERO O MAS VECES DEL ANTERIOR



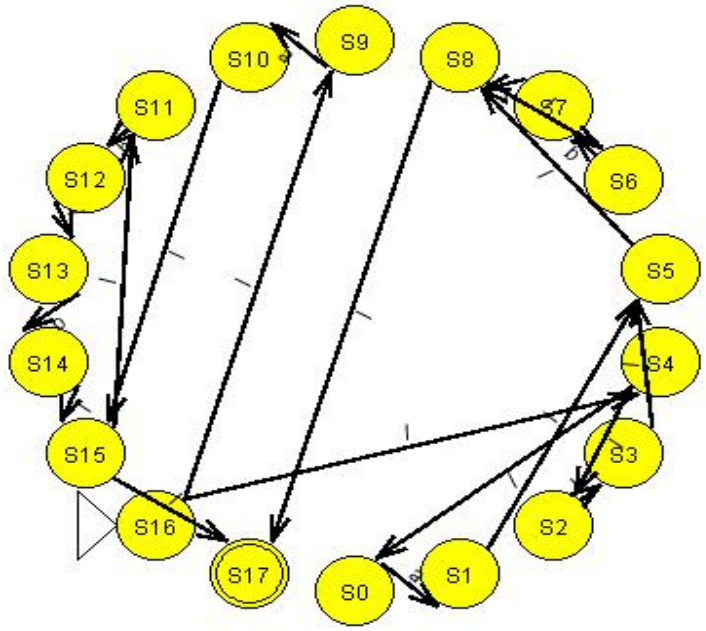
PASO NUMERO 12: CONCATENACION DE DOS AUTOMATAS



Y el autómata final, que es el del último paso del algoritmo, unión de los anteriores dependiendo de la operación que sea:

PASO NUMERO 13: UNION DE DOS AUTOMATAS

AUTOMATA FINAL



7. **Equivalencia entre ER y AF.**

No hemos implementado, como tal, un algoritmo que permita la equivalencia entre expresiones regulares y autómatas finitos. Pero se puede obtener concatenando la ejecución de los algoritmos anteriores.

Como se explicó anteriormente, se puede transformar de ER a AF ejecutando paso a paso el resto de algoritmos, una vez obtenido el autómata equivalente se puede calcular la equivalencia. Todo ello se puede gestionar mediante la interfaz gráfica de la aplicación y obtener el resultado deseado.

3.4.2 Ejercicios no automáticos

Este conjunto de ejercicios no son computables de manera automática, por tanto no existe un algoritmo que los resuelva. Sin embargo, si tenemos una solución correcta se puede comprobar si la solución propuesta por el usuario es equivalente.

1. Ejercicio de obtención de la expresión regular que reconoce un lenguaje expresado en modo textual.

Hemos desarrollado un algoritmo utilizando los algoritmos automáticos que hemos implementado. Dichos algoritmos comprueban la equivalencia entre la expresión regular almacenada en la base de datos como solución con la expresión regular proporcionada por el usuario.

2. Ejercicio de generación de un lenguaje asociado a un autómata.

No es posible automatizar la comprobación de lenguajes en forma textual.

3. Ejercicio de generación de un autómata que acepta palabras de un lenguaje.

Hemos utilizado el algoritmo de equivalencia para poder comprobar la distinguibilidad entre el autómata propuesto por el usuario con el autómata almacenado como solución.

3.5 Requisitos tecnológicos

Los requisitos de la aplicación son los siguientes:

- Ordenador con JRE 5.0 o superior.

- Es aconsejable que el usuario sea el administrador del sistema, ya que es probable que tenga que dar algún tipo de permisos de entrada/salida.

3.6 Entorno de desarrollo

TALFi es una aplicación Java. Hemos usado Eclipse como herramienta de programación, y el JDK 6.0 de Java. En los siguientes enlaces se puede descargar gratuitamente ambas herramientas:

http://www.eclipse.org/

http://java.sun.com/javase/downloads/index.jsp

3.7 Bibliotecas usadas

El sistema TALFi no requiere de la instalación de ninguna librería puesto que para su programación se han usado única y exclusivamente elementos contenidos dentro de la librería estándar de java, tanto para la lógica de la aplicación como para la programación de la interfaz gráfica.

Para la interfaz se utilizan las clases que proporciona la biblioteca **Swing**, incluida en el estándar java, cómo las ventana de tipo JFrame, con paneles JPanel y botones JButton, así como otros componentes más específicos.

Inicialmente se valoró la idea de utilizar unas librerías específicas de creación de grafos que facilitaban la labor de representación y lógica de los algoritmos sobre dichos grafos, estas librerías son: **JGraph** y **JGraphT**. Sin embargo finalmente se rechazó su uso, debido fundamentalmente a los problemas encontrados al implementar los grafos que incluían como autómatas, ya que su principal uso era como digrafos valorados para

algoritmos de recorrido mínimo o cálculo de árboles de recubrimiento, lo cual implicaba que el nombre de las aristas fuera un número y no cualquier tipo de carácter como permite TALFi.

Cómo hemos especificado en los requerimientos la librería mínima necesaria es la versión 5.0 del estándar Java.

3.8 Arquitectura de la aplicación

La aplicación está dividida en varias carpetas.

- La carpeta **bin**: en esta carpeta guardamos toda la programación de la aplicación, tanto de la interfaz gráfica como de las clases necesarias para implementar los autómatas etc.
- La carpeta **doc**: esta carpeta contiene toda la documentación de la aplicación, es decir, contiene el JavaDoc. Para mayor información sobre JavaDoc remitimos al lector al apéndice número tres al final de la memoria.
- La carpeta **HTML**: esta carpeta guarda los HTML de salida creados por la aplicación. Es decir, las páginas HTML que la aplicación genera al ejecutar cada uno de los algoritmos, y donde se muestran los pasos que se han realizado junto a imágenes con texto explicativo.
- La carpeta **imagenes**: guarda las imágenes que usa la aplicación en su interfaz gráfica.
- La carpeta **XML**: guarda los XML de entrada (ejercicios, ejemplos) y los XML temporales que la aplicación crea constantemente durante la ejecución del programa. La aplicación utiliza un formato predeterminado de ficheros XML para cada uno de los algoritmos de los que se puede extraer tanto autómatas como expresiones regulares, según se necesiten. Para mayor información consúltese el anexo número cuatro al final del documento.

3.9 Pruebas del Sistema

Las principales pruebas a las que se ha sometido TALFi han sido pruebas para testear la robustez de los algoritmos implementados. Se han probado cerca de 100 ejercicios de la asignatura de Teoría de Autómatas y Lenguajes Formales, con pruebas satisfactorias. A su vez, nuestro tutor del proyecto ha probado con sus propios ejemplos la aplicación.

Hemos querido que nuestro tutor sea nuestro “tester” de la aplicación, ya que es la persona idónea: conoce la teoría de autómatas perfectamente y no ha participado en la programación de la aplicación.

3.10 Implementación y cierre del proyecto

Ésta herramienta ha sido programada en lenguaje Java.

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

La decisión de programar la aplicación sobre el lenguaje Java es debida sobre todo a nuestro conocimiento de este leguaje, adquirido en gran medida durante nuestros años de formación universitaria. Con Java es sencillo crear interfaces gráficas gracias su librería **Swing** y la predecesora de esta, **AWT**, que proporcionan todos los componentes gráficos necesarios para presentar un interfaz amigable y sencilla a la vez.

Java es un estándar hoy en día con el que cuentan casi todos lo PCs de mundo y que cualquier usuario tiene instalado, lo cual nos permite llegar a un número elevado público sin que estos tengan que realizar complicadas labores de instalación gracias a la difusión del software Java.

Así mismo es de destacar que, al ser Java un lenguaje multiplataforma, la interfaz gráfica adopta la apariencia del sistema operativo sobre el que se esté ejecutando de forma dinámica sin tener que realizar ninguna modificación sobre el código ni la ejecución.

El proyecto queda cerrado con las siguientes características, o algoritmos implementados:

- Desarrollo de los modelos principales para el objetivo acordado:

Hemos implementado todas las aplicaciones necesarias para hacer un buen tratamiento de éstos tres modelos que hemos considerado básicos para el correcto funcionamiento de la aplicación.

Existen en el código funciones implementadas de las que no se hace uso en nuestra aplicación que podrían ser utilizadas en posteriores mejoras de la misma.

- Desarrollo de la interfaz:

La interfaz desarrollada en Java es sencilla de manejar y comprender para cualquier usuario a la que va destinada la aplicación, con mensajes de ayuda y aclaraciones para posibles dudas a cerca del funcionamiento de algún componente específico. Los métodos de entrada y salida de datos están presentes tanto en forma interna (XML internos dentro de la propia aplicación), como externa mediante el uso de archivos HTML.

3.11 Posible trabajo futuro

La aplicación TALFi se centra en la parte de autómatas deterministas y no deterministas con o sin transiciones lambda que reconocen leguajes expresables mediante la expresiones regulares, sin embargo, más allá hay una gran amplitud de lenguajes que se pueden reconocer mediante otros mecanismos de los que hablaremos en este punto.

La razón por la cual la aplicación que se ha codificado se centra en estas cuestiones teóricas es, fundamentalmente, que todos los mecanismos que se han implementado son automáticos; es decir, que existen algoritmos para cada uno de ellos que, no necesitan la participación humana para realizarse.

Como futuras ampliaciones de la aplicación se pueden realizar mecanismos de reconocimiento de lenguajes que no pueden ser descritos mediante expresiones regulares y se necesita un concepto más amplio para su descripción, tales como los lenguajes independientes de contexto y los lenguajes recursivamente enumerables.

Para llevar a cabo el reconocimiento de lenguajes independientes del contexto se añade a los autómatas un mecanismo que aumenta la memoria del mismo, y no es otro que una **pila**. Con esta nueva mejora aparecen los nuevos autómatas que llamaremos **autómatas**

de pila, capaces de reconocer lenguajes mucho más potentes debido a la memoria disponib1e en la pila.

Representar estos autómatas de pila y crear los algoritmos capaces de generarlos a partir del lenguaje que reconocen no es una tarea sencilla, ni siquiera es automática. Tampoco lo es generar la gramática incontextual de un lenguaje dado. Sin embargo sí es posible realizar de forma automática la transformación desde una gramática incontextual a un autómata de pila equivalente y viceversa.

En esta categoría de autómatas muchas otras tareas no son automáticas, ya que por ejemplo tampoco lo es comprobar la equivalencia entre dos autómatas de pila, sin embargo aplicando tareas de **testing** es posible llegar a comprobar que dos autómatas son equivalentes en un número de palabras ya que generan la misma respuesta ante palabras iguales.

El proceso de testing se podría realizar de manera manual eligiendo una serie de palabras divididas en dos tipo: las que queremos que el autómata de pila reconozca y las que no, una ver realizada la selección se pasan todas y cada una de las palabras por las dos máquinas que queremos comprobar su equivalencia. De esta forma a través de este proceso podemos decir si dos máquinas son equivalentes o no para el conjunto de palabras seleccionadas. Si no lo son para ese conjunto es evidente que en general no lo serán luego podemos dar una respuesta negativa con toda seguridad, en caso contrario única y exclusivamente podemos afirmar que ambas máquinas son equivalentes para el grupo de palabras propuesto pero no podemos extrapolar dicho resultado a una afirmación general, pues podrían encontrarse palabra no seleccionadas que no recibieran la misma respuesta de ambas máquinas. Como hemos dicho antes el proceso de comprobación de equivalencia entre autómatas de pila ni es sencillo ni es automático.

Tenemos que tener en cuenta que los autómatas de pila no cubren todos los lenguajes posibles. Es decir, podemos encontrar lenguajes que no puedan ser reconocidos por estos autómatas. Ellos son capaces de reconocer por ejemplo el lenguaje que tiene las mismas bs que as, sin embargo no pueden reconocer una extensión de este que tiene también el mismo número de letras cs; para estos lenguajes existen otros autómatas todavía más potentes que se denominan **máquinas de Turing**.

Si ya era complicado obtener algoritmos automáticos para autómatas de pila, es todavía más complejo obtenerlos para máquinas de Turing, pues al tener mayor potencia reconocedora también se complica su análisis automático.

Las máquinas de Turing tiene la particularidad de poseer una memoria infinita (al menos teóricamente) en forma de cinta y la capacidad de moverse sobre la cinta mediante un cursor en las dos direcciones, hacia la izquierda y hacia la derecha, gracias a esto es por lo que pueden reconocer lenguajes con 3 o más letras que aparecen el mismo número de veces.

Los mecanismos sobre máquinas de Turing no son automáticos tampoco y es donde se hace todavía más necesario un proceso de testing para comprobar que la máquina que se ha creado reconoce o no al menos las palabras involucradas en el proceso siempre y cuando se tenga otra con la que comparar las respuestas dadas.

Queda por tanto como trabajo futuro la posibilidad de añadir a la aplicación la capacidad de creación de dichos autómatas y su comprobación de equivalencia.

Otros trabajos automáticos que se podrían incluir en la aplicación, y que son mucho más sencillos que los de máquinas de Turing o autómatas de pila, son las transformaciones entre gramáticas.

En esta línea existe un algoritmo sencillo y de fácil aplicación para la obtención de la **Forma Normal de Chomsky** de una gramática incontextual. Nuestro proyecto está centrado más hacia los autómatas, es por ello que no se planteó la implementación de dicho algoritmo, sin embargo es otra posible tarea futura que se podría realizar de cara a obtener una aplicación que cerrara todos los campos de la asignatura de TALF.

Es posible así mismo realizar mejoras en las capacidades de la aplicación sobre las propias cuestiones que implementa. Como por ejemplo mayor claridad a la hora de representar los autómatas gráficamente con más puntos de anclaje en las aristas, o la capacidad de pintarlas de forma que el recorrido de la línea se el mínimo posible.

Por otro lado, también se podrían realizar mejoras en los algoritmos, sobre todo orientadas a la claridad en la especificación de los pasos de minimización, incluyendo el diagrama de transiciones inversas o la razón por la que dos estados son indistinguibles y en que momento de le ejecución sucede, todo con el fin de que los alumnos puedan aprender más claramente donde y por qué se realiza cada paso en dicho algoritmo.

Otra posible mejora en la aplicación consiste en la detección de los estados no deterministas. De esta forma estos estados podrían señalarse en el interfaz gráfico

Por último queremos indicar que esta aplicación, TALFi, podría utilizarse para realizar un curso de corrección de ejercicios que se podría colgar en un servidor con el fin de que los alumnos los resolvieran de cara a la preparación de la asignatura sin más que permitir el dibujo de autómatas como respuesta a los ejercicios y corregirlos usando la lógica implementada en TALFi para los algoritmos.

En definitiva, hay muchas más posibles variaciones en las que trabajar para mejorar la aplicación en el futuro.

Capítulo 4.

Descripción de la aplicación

Tras ésta introducción de los aspectos externos de la aplicación que hemos considerado más importantes, vamos a ver una descripción más exhaustiva de la aplicación en sí. La aplicación está dividida en dos partes: la aplicación por consola y el interfaz gráfico que funciona sobre la aplicación de consola.

La aplicación por consola es plenamente funcional. Ha sido usada para hacer pruebas internas y probar que los algoritmos implementados funcionan correctamente.

El interfaz gráfico extiende las posibilidades de la aplicación por consola. Ofrece un entorno visual más "amigable" e intuitivo. Creemos que introducir y mostrar los resultados de una forma gráfica es indispensable en una aplicación de éstas características.

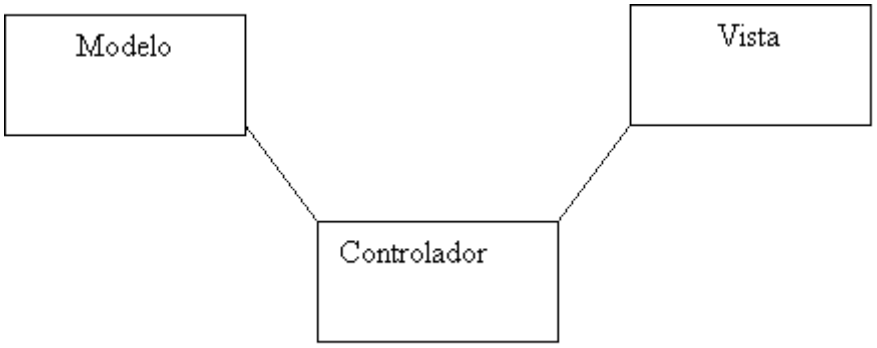
Iremos explicando ambas formas de entender la aplicación a lo largo de éste apartado. Para ello consideramos que la mejor forma es ir desgranándola desde la base, el inicio del proyecto, a los apartados finales.

4.1. Patrones de diseño

Los patrones de diseño que hemos usado para el desarrollo y diseño de la aplicación son los siguientes:

a) Modelo-Vista-Controlador:

Está presente en toda la aplicación, es el patrón que conforma la aplicación en sí, lo hemos desarrollado de la siguiente manera:



La vista está formada por todas las clases relacionadas con la interfaz gráfica.

El modelo está formado por todas las clases relacionadas con la lógica de la aplicación, es decir: algoritmos, autómatas, alfabetos, expresiones regulares.

El cometido del controlador es hacerles interactuar entre ambos, siempre que el controlador lo permita.

Es decir, la vista nunca accede al modelo sin pasar por el controlador y viceversa
(ver diagramas de interacción en las páginas 46 a 48).

b) Singleton:

El patrón Singleton se aplica cuando nos interesa que un objeto sea ú*nico* en la aplicación, y que el acceso a ella sea por un *único* sitio.

Nuestra aplicación contiene varios objetos que exigen ese tratamiento:

- Tenemos un único traductor HTML.
- Un único mensajero.
- Un único controlador.
- Un único parser de xml.
- Una única clase par guardar el autómata copiado o cortado para pegarlo.

c) Application Service:

El Controlador, realiza las funciones relacionadas con servicio de aplicación.

Es decir: su función es la de puerta a la lógica de programa desde la interfaz gráfica o vista.

4.2 Visión teórica de la Aplicación

Usaremos el Modelo-Vista-Controlador como base para explicar el corazón de la aplicación.

- **Vista**: la interfaz gráfica de la aplicación creemos que está suficientemente detallada en el manual de usuario de la aplicación, que adjuntamos en el capítulo cinco de éste escrito.
- **Controlador**.
- **Modelo**: autómatas, algoritmos y expresiones regulares que realmente hacen funcionar la aplicación como tal. Veamos cómo funcionan uno a uno los algoritmos implementados.

Pasaremos por tanto a explicar el modelo, dividido en tres grandes bloques:

- **Autómatas**: donde se encuentran las clases que guardan la información de los autómatas y realizan sus funciones específicas.
- **Expresión regular**: donde se encuentran las clases para la definición de expresiones regulares y su transformación a árboles sintácticos.
- **Algoritmos**: donde se desarrolla la lógica de todos los algoritmos implementados en TALFi.

A. Autómatas

El paquete de autómatas incluye las siguientes clases e interfaces, que pasaremos a explicar a continuación una por una:

- Alfabeto y Alfabeto_imp.
- Automata.
- AutomataFD.
- AutomataFND.
- AutomataFNDL

Alfabeto y Alfabeto_Imp

Alfabeto es una interfaz que define las funciones de los alfabetos de la aplicación. Alfabeto_Imp implementa éstas funciones. Están incluidas las funciones que consideramos básicas como pueden ser devolver listas de letras, devolver el alfabeto entero, ver si está una letra en el alfabeto etc.

Autómata

Interfaz que define las funcionalidades básicas para cualquier autómata, sea del tipo que sea.

AutomataFD

Clase que implementa todas las funciones de Autómata. Funciones que permiten entre otras cosas mostrar las aristas de un vértice en concreto, insertar aristas, mostrar los estados que componen el autómata, eliminar vértices etc.

AutomataFND y AutomataFNDLambda

Clases que extienden AutomataFD, ya que además de las funciones definidas en Autómata necesitan de otras funciones específicas para resolver otras cuestiones propias del autómata en concreto. Funciones como cierre lambda, que calcula el cierre lambda de un estado, o delta extendida que devuelve la lista de estados a los que podemos ir usando una letra del alfabeto nos permiten tratar éstos autómatas correctamente.

Coordenadas

Clase que almacena las coordenadas de los estados. Se usa únicamente en la parte gráfica de la aplicación en el caso de que se guardara el autómata incluyendo las coordenadas que se generan tras la modificación.

B. Expresión regular

El paquete de Expresión Regular contiene las siguientes clases e interfaces que explicaremos a continuación:

- ArbolER y ArbolER_Imp.
- ExpresionRegular y ExpresionRegular_Imp.

ArbolER y ArbolER_Imp

ArbolER es una interfaz que define la funcionalidad de los árboles sintácticos para expresiones regulares. Incluye funciones básicas para el tratamiento de árboles binarios, como getHijoIz que devuelve el hijo izquierdo de un nodo, getRaiz que devuelve la raíz del árbol, o setHijoDr que modifica el hijo derecho de un nodo. La clase que implementa a la interfaz es ArbolER_Imp.

ExpresionRegular y ExpresionRegular_Imp

ExpresionRegular es una interfaz que define los métodos que debe implementar las expresiones regulares. Funciones como getExpresionRegular, o getArbolER son imprescindibles para el tratamiento de expresiones regulares. La clase que implementa ésta interfaz es ExpresionRegular_Imp.

C. Algoritmos

El paquete de Algoritmos contiene las siguientes clases e interfaces que explicaremos a continuación:

- Algoritmo.
- ERtoAFNDLambda.
- AFD_TO_ER.
- Minimización.
- Automatas_Equivalentes.
- AFN_TO_AFD.
- AFNDLambda_to_AFND.

Algoritmo

Interfaz que define todos los métodos que deberán tener como mínimo cada uno de los algoritmos que se quieran crear en la aplicación, es una interfaz sencilla, con únicamente tres métodos, a saber: ejecutar que recibe como parámetro si es por pasos, registrarControlador que identifica el controlador que llevará acabo la ejecución, en caso de que haya más de uno y finalmente el método que devuelve el xml getXML de pasos en caso de que se haya ejecutado con esa opción.

ERtoAFNDLambda

Algoritmo que implementa la interfaz antes descrita y que añade a estos tres métodos varios privados que realizan en conjunto la implementación del algoritmo que obtiene el autómata finito no determinista con transiciones lambda equivalente a la expresión regular que hace las veces de entrada al algoritmo. Para ello simplemente se transforma a un árbol sintáctico que se va recorriendo, por cada nodo del árbol recorrido, se obtiene el autómata finito no determinista con transiciones lambda equivalente hasta el momento. Por cada paso se genera un comentario predefinido dependiendo de la operación que se aplique que posteriormente se añadirá al HTML que se muestra, como aclaración para los alumnos.

AFD_TO_ER

Algoritmo que implementa la interfaz y define nuevos métodos que ayudan en su ejecución. En este caso se enfoca el algoritmo mediante la eliminación de estados no finales, con la excepción del inicial, modificando las transiciones entre los que no se eliminan de forma que el autómata tras la eliminación reconozca el mismo lenguaje. Una vez termina este proceso para cada estado no final se calcula su expresión regular, partiendo del estado inicial que como se ha comentado se mantiene en el autómata, para posteriormente concatenando todas las expresiones obtenidas para generar una única expresión final resultado. Hay varios factores que implican un cambio en el desarrollo del algoritmo, uno de ellos y quizá el más importante es el comprobar si el estado inicial del autómata es también final, pues dependiendo de esto la expresión resultante deberá tener unas características u otras.

Minimización

Algoritmo de minimización de autómatas finitos deterministas, cuenta con los tres métodos que debe implementar de la interfaz Algoritmo y con numerosos métodos privados debido a la complejidad del algoritmo. Los métodos privados llevan el verdadero cómputo que realiza y los métodos implementados hacen la labor de interfaz con el resto de la aplicación.

Se genera un nuevo autómata basado en el de entrada, colapsando aquellos estados que son indistinguibles.

Automatas_equivalentes

Algoritmo de equivalencia de autómatas, de nuevo cuenta con los tres métodos que debe implementar y otros privados y públicos, los primeros son los que se encargan de la implementación propiamente dicha del algoritmo y los segundos son los que llevan a cabo la inicialización de los dos autómatas que intervienen en el algoritmo y la preparación previa del mismo.

Enfocamos el algoritmo de una manera práctica, inicializamos una única “tabla” que identifica a los autómatas con una mezcla de ambos (renombrando estados si es necesario) y minimizamos a fin de encontrar distinguibilidad entre los estados de los dos autómatas. La respuesta se calcula en función del resultado de esta distinguibilidad obtenida en la tabla única.

AFN_TO_AFD

Algoritmo de transformación de autómata finito no determinista a autómata finito determinista, como el resto de algoritmos cuenta con los tres métodos que debe implementar y otros privados.

El algoritmo genera un nuevo autómata a partir del no determinista, obteniendo nuevos estados a partir de las aristas no deterministas.

AFNDLambda_to_AFND

Algoritmo que implementa la interfaz Algoritmo, extendiéndola con nuevos métodos propios para la ejecución del mismo.

Dado un autómata con transiciones lambda, lo transformaremos en uno equivalente sin ellas. Para ello nos guiaremos de una serie de pasos, que se ejecutarán para cada estado del autómata, y para cada letra del alfabeto del mismo. Los pasos son:

- Cierre lambda del estado
- Delta extendida de los estados del cierre lambda, usando todas las letras del alfabeto.
- Los estados cuya delta extendida lleven a un estado serán una arista entre éste estado y el estado inicial del que se ha hecho el cierre lambda.

Ejecutando éstos pasos para todos los estados del autómata, tendremos como resultado un autómata finito no determinista sin transiciones lambda.

D. Controlador

El paquete de Algoritmos contiene las siguientes clases e interfaces que explicaremos a continuación:

- Controlador.
- Controlador_imp.

Controlador

Interfaz que define los métodos necesarios que deberán implementar las clases que pretendan ejercer de controlador, en este caso el controlador dentro de la aplicación se encarga de recibir la query (ver apéndice uno a final de la memoria para más información) que prepara la vista gráfica según los deseos del usuario y pasar los parámetros correctos al algoritmo correspondiente que se selecciona mediante las opciones especificadas en la query.

Los métodos son: registrarVista, getSalida, obtenerXML, tratarMensaje y setIdioma.

Controlador_imp

Esta es la clase que ejerce el papel de controlador en la aplicación, como es evidente, implementan todos los métodos de la interfaz anterior y algunos más que son de régimen privado y que realizan varias labores necesarias en la aplicación.

Es labor del controlador obtener los archivos xml de pasos, para delegando en otras clases, convertirlos a un formato html, del cual se pasará la ruta a la vista para que lo muestre. También deberá obtener del algoritmo correspondiente el resultado, ya sea un autómata o una expresión regular, que de nuevo mandará a la vista para que se encargue de mostrarla.

Así mismo, también tiene la capacidad, delegando en otras clases, de extraer tanto autómatas como expresiones regulares de archivos xml, para obtener la entrada que el usuario haya introducido gráficamente.

Una vez descritos los paquetes de código más importantes de la aplicación, pasaremos a explicar el comportamiento de las clases y su interacción a la hora de la ejecución de las diferentes opciones que tiene la aplicación TALFi con la ayuda de los diagramas UML.

4.3. Diagramas UML

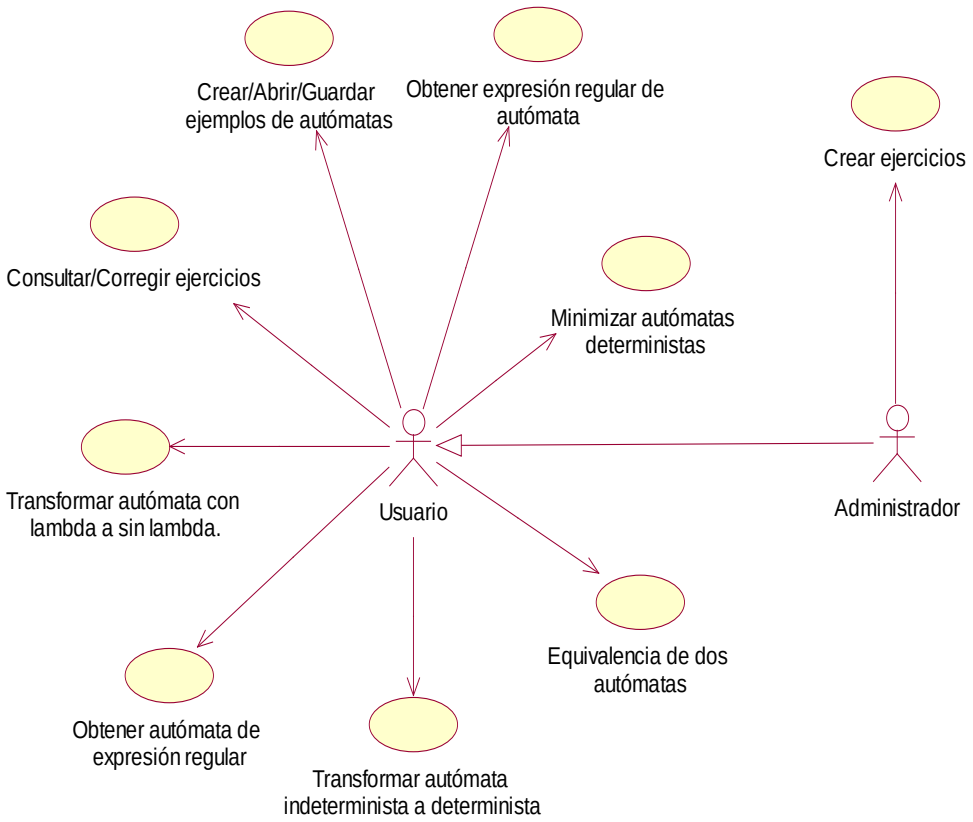
En esta sección vamos a mostrar algunos diagramas UML de la interacción de las clases de la aplicación y algunos casos de uso para la misma.

Un diseño UML completo de la aplicación podría ocupar una memoria nueva y completa, por tanto hemos optado por hacer un resumen de los paquetes y clases más importantes, de cara a futuros desarrolladores, siéndole la siguiente documentación útil para la ampliación y desarrollo futuro de la aplicación.

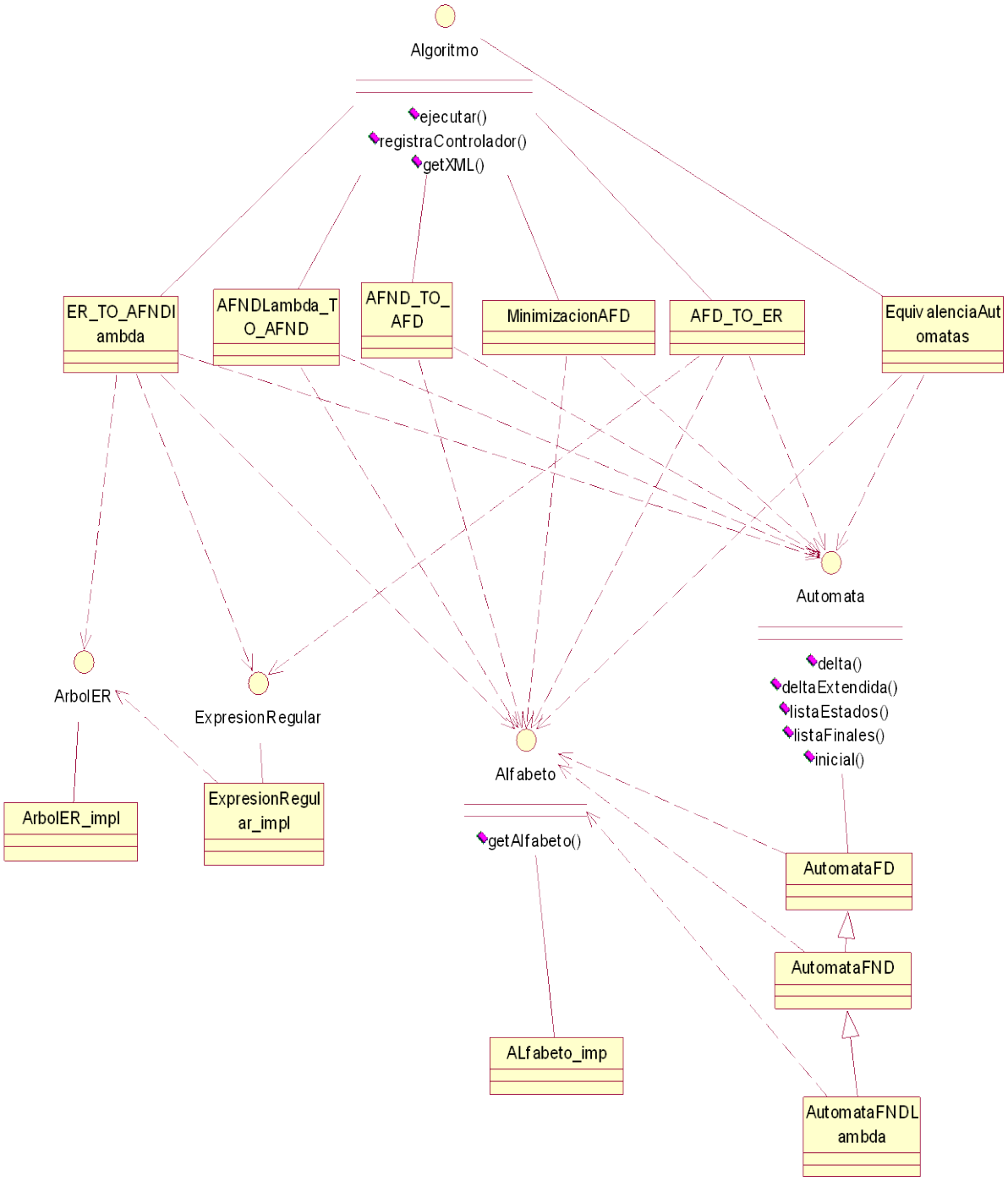
Es decir:

- Diagrama de casos de uso de la aplicación.
- Diagramas de clases de los paquetes: controlador y modelo (hemos decidido que la vista no es relevante en cuanto al diseño).
- Diagrama de interacción de un algoritmo concreto.

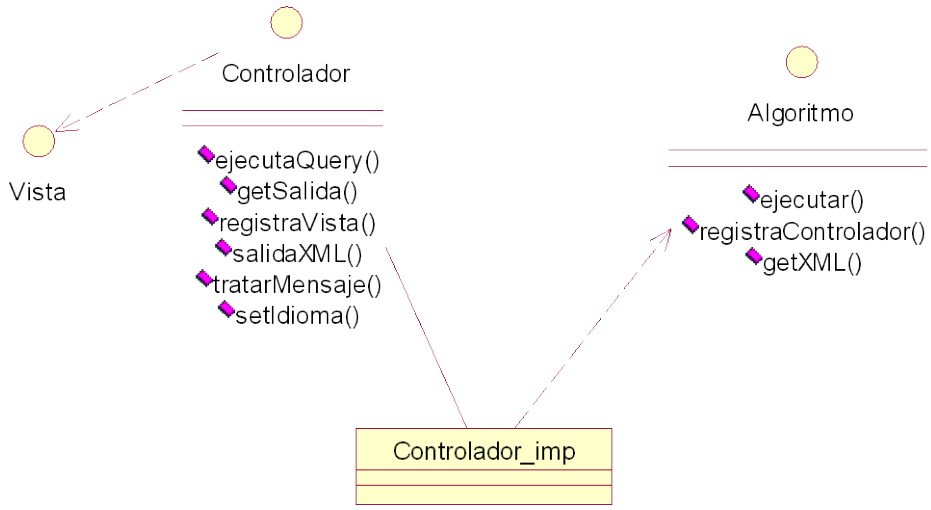
→ Diagrama de casos de uso



→ **Diagrama de clases (paquete modelo)**



→ **Diagrama de clases (paquete controlador)**



A continuación exponemos algunos diagramas de interacción:

→ Diagrama de interacción para el algoritmo de equivalencia.

→ Diagrama de interacción para el algoritmo de paso de expresión regular a autómata no determinista con lambdas.

→ Diagrama de interacción para el algoritmo de paso de autómata determinista a expresión regular.

Diagrama de interacción para el algoritmo de equivalencia.

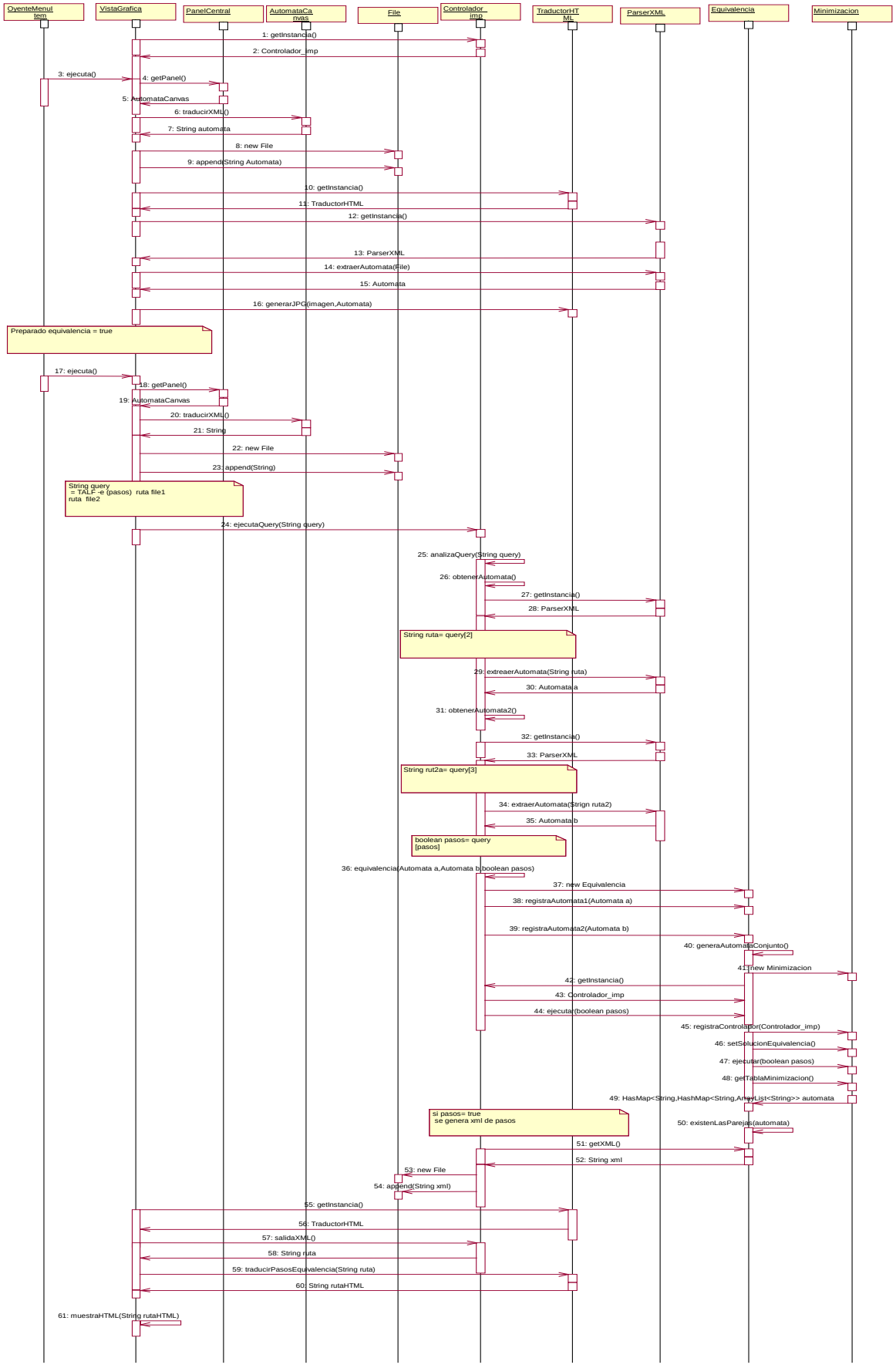

OyenteMenuItem
VistaGrafica
PanelCentral
AutomataCanvas
File
Controlador_imp
TraductorHTML
ParserXML
Equivalencia
Minimizacion
1: getInstancia()
2: Controlador_imp
3: ejecuta()
4: getPanel()
5: AutomataCanvas
6: traducirXML()
7: String automata
8: new File
9: append(String Automata)
10: getInstancia()
11: TraductorHTML
12: getInstancia()
13: ParserXML
14: extraerAutomata(File)
15: Automata
16: generarJPG(imagen,Automata)
Preparado equivalencia = true
17: ejecuta()
18: getPanel()
19: AutomataCanvas
20: traducirXML()
21: String
22: new File
23: append(String)
String query
= TALF -e (pasos) ruta file1
ruta file2
24: ejecutaQuery(String query)
25: analizaQuery(String query)
26: obtenerAutomata()
27: getInstancia()
28: ParserXML
String ruta= query[2]
29: extreaerAutomata(String ruta)
30: Automata a
31: obtenerAutomata2()
32: getInstancia()
33: ParserXML
String rut2a= query[3]
34: extraerAutomata(Strign ruta2)
35: Automata b
boolean pasos= query
[pasos]
36: equivalencia(Automata a,Automata b,boolean pasos)
37: new Equivalencia
38: registraAutomata1(Automata a)
39: registraAutomata2(Automata b)
40: generaAutomataConjunto()
41: new Minimizacion
42: getInstancia()
43: Controlador_imp
44: ejecutar(boolean pasos)
45: registraControlador(Controlador_imp)
46: setSolucionEquivalencia()
47: ejecutar(boolean pasos)
48: getTablaMinimizacion()
49: HasMap<String,HashMap<String,ArrayList<String>> automata
si pasos= true
se genera xml de pasos
50: existenLasParejas(automata)
51: getXML()
52: String xml
53: new File
54: append(String xml)
55: getInstancia()
56: TraductorHTML
57: salidaXML()
58: String ruta
59: traducirPasosEquivalencia(String ruta)
60: String rutaHTML
61: muestraHTML(String rutaHTML)

Diagrama de interacción para el algoritmo de paso de expresión regular a autómata no determinista con lambdas.

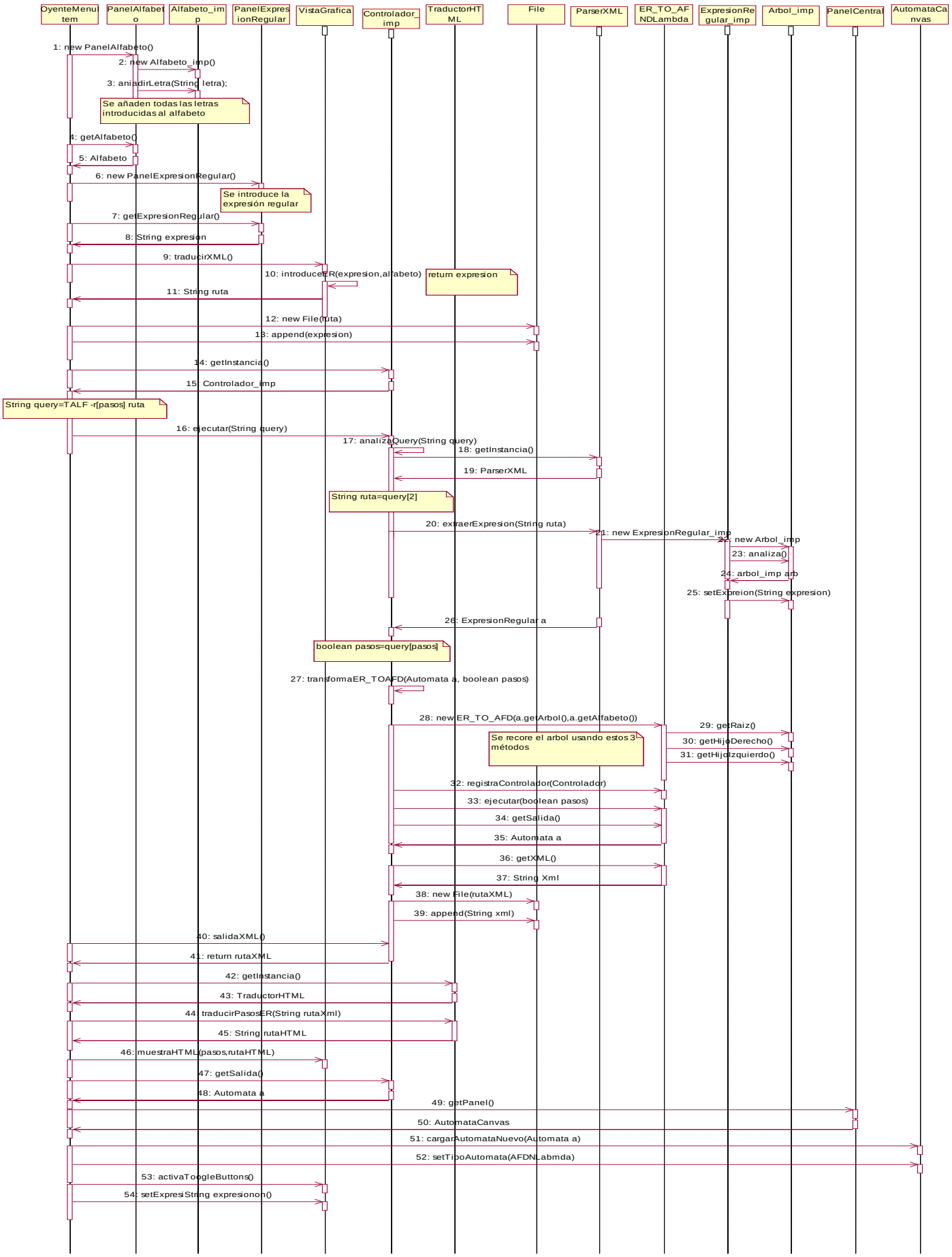
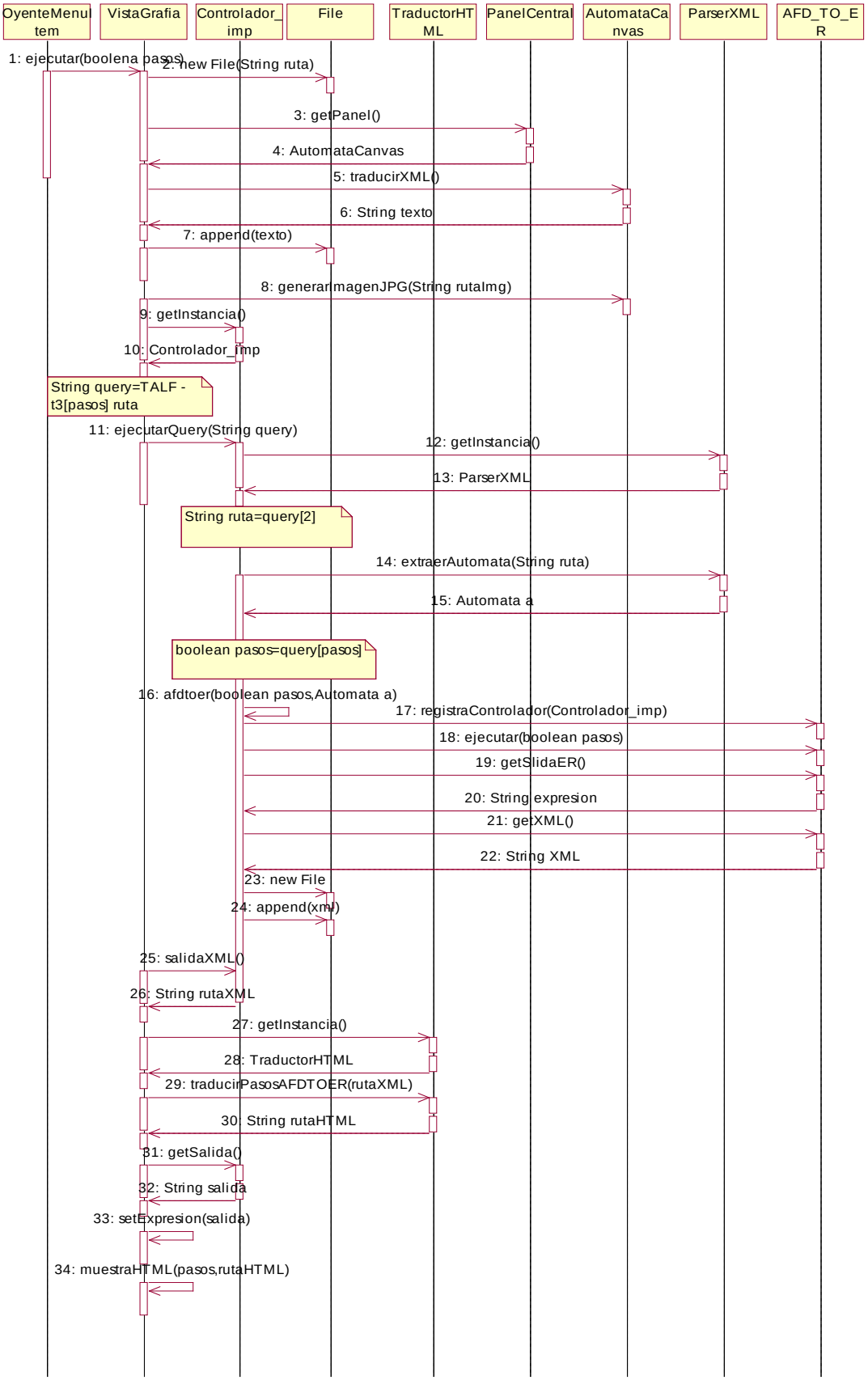

OyenteMenuItem
PanelAlfabeto
Alfabeto_imp
PanelExpresionRegular
VistaGrafica
Controlador_imp
TraductorHTML
File
ParserXML
ER_TO_AFNDLambda
ExpresionRegular_imp
Arbol_imp
PanelCentral
AutomataCanvas
1: new PanelAlfabeto()
2: new Alfabeto_imp()
3: aniadirLetra(String letra);
Se añaden todas las letras introducidas al alfabeto
4: getAlfabeto()
5: Alfabeto
6: new PanelExpresionRegular()
Se introduce la expresión regular
7: getExpresionRegular()
8: String expresion
9: traducirXML()
10: introduceER(expresion,alfabeto)
return expresion
11: String ruta
12: new File(ruta)
13: append(expresion)
14: getInstancia()
15: Controlador_imp
String query=TALF -r[pasos] ruta
16: ejecutar(String query)
17: analizaQuery(String query)
18: getInstancia()
19: ParserXML
String ruta=query[2]
20: extraerExpresion(String ruta)
21: new ExpresionRegular_imp
22: new Arbol_imp
23: analiza()
24: arbol_imp arb
25: setExpreion(String expresion)
26: ExpresionRegular a
boolean pasos=query[pasos]
27: transformaER_TOAFD(Automata a, boolean pasos)
28: new ER_TO_AFD(a.getArbol(),a.getAlfabeto())
29: getRaiz()
Se recore el arbol usando estos 3 métodos
30: getHijoDerecho()
31: getHijoIzquierdo()
32: registraControlador(Controlador)
33: ejecutar(boolean pasos)
34: getSalida()
35: Automata a
36: getXML()
37: String Xml
38: new File(rutaXML)
39: append(String xml)
40: salidaXML()
41: return rutaXML
42: getInstancia()
43: TraductorHTML
44: traducirPasosER(String rutaXml)
45: String rutaHTML
46: muestraHTML(pasos,rutaHTML)
47: getSalida()
48: Automata a
49: getPanel()
50: AutomataCanvas
51: cargarAutomataNuevo(Automata a)
52: setTipoAutomata(AFDNLabmda)
53: activaToogleButtons()
54: setExpresiString expresionon()

Diagrama de interacción para el algoritmo de paso de autómata determinista a expresión regular.



4.4 Restricciones principales

Como toda aplicación, la herramienta TALFi tiene una serie de restricciones que pasamos a detallar:

- Java 5.0:
  Hemos comprobado que la aplicación puede no ejecutarse correctamente en ordenadores con una versión de java inferior a la 5.0. Es totalmente necesario tener instalada la máquina virtual de Java en el ordenador donde se va a ejecutar la aplicación. Sería aconsejable no tener una política de privacidad muy fuerte en java, ya que es posible que si es muy restrictiva fallen ciertos componentes de la aplicación.
- Navegador WEB:
  No es estrictamente necesario que el usuario tenga instalado un tipo de navegador WEB específico pues la aplicación detecta varios tipos de navegador. Es válido cualquier programa que sea capaz de leer archivos HTML, ya que es la forma de visualizar el resultado final por pasos. Mozilla Firefox, Internet Explorer, Opera…cualquier navegador web de la actualidad puede mostrar los resultados de la aplicación ya que los resultados se muestran en archivos HTML sencillos, pues no son más que imágenes acompañadas de texto explicativo.
- Permisos del equipo:
  Puede ser que debido a ciertas restricciones del equipo donde se ejecute TALFi, ésta pueda no ejecutarse correctamente. Es posible que éstas medidas de seguridad afecten de alguna manera a Java, o a la seguridad de los navegadores, impidiéndonos visualizar correctamente los resultados por ejemplo. Estos permisos son fundamentales a la hora de la ejecución:
    - Lectura de ficheros sobre la carpeta en la que se guardan, tanto el proyecto como los archivos que el usuario elija almacenar.
    - Escritura de ficheros, en las mismas circunstancias que se ha explicado con la lectura.

4.5 Suposiciones y dependencias

El componente humano es fundamental a la hora del manejo de la aplicación.

Es recomendable que los usuarios de la herramienta TALFi lean el manual de uso de la aplicación antes de usar la herramienta. Es un manual muy detallado donde se explica con detalle la misma y que puede solucionar muchos posibles errores de uso de la misma. El manual se incluye en la aplicación como herramienta de ayuda y está contenido en el capítulo cinco en esta memoria.

Para acceder al manual de usuario, no hay más que acceder al menú de navegación de la aplicación y pulsar el botón de Manual de usuario.

Recordamos que ésta aplicación tiene un fin educativo, luego el usuario de la misma ha debido de estudiar (o está estudiando) temas referentes al estudio de los autómatas. El uso de la aplicación sin una base teórica previa no se recomienda pues puede resultar una ardua tarea.

4.6 Bases de datos

La aplicación permite la gestión de usuarios mediante una base de datos en formato XML, así como la inclusión de ejercicios y ejemplos que se pueden cargar de forma directa desde al panel situado a la izquierda en la vista gráfica que tiene forma de árbol de directorios.

4.6.1 Base de datos de usuarios

La base de datos de usuarios consta de un único fichero xml con toda la información de los usuarios que se hayan dado de alta en la aplicación mediante la interfaz gráfica. Para poder utilizar nuestra aplicación es estrictamente necesario haberse registrado como usuario de la misma; para ello se requiere que el usuario elija un nombre de usuario y una contraseña, así como que introduzca su dni.

El programa da validación de datos comprueba que el DNI del usuario tenga los 8 dígitos de que consta un DNI en España, así como que el nombre que haya elegido no se encuentre ya registrado en la base de datos. Se podría comprobar que el usuario que desea registrarse es alumno de un determinado curso comprobado si su DNI está dentro de la lista de matriculados, en nuestro caso al no tener dicha lista no se realiza comprobación alguna respecto de si el DNI es válido. Más allá de la citada anteriormente, que el DNI debe tener 8 dígitos, extender dicha comprobación es una tarea sencilla que se podría realizar como objetivo futuro. No se permite introducir la letra en el DNI, únicamente los dígitos.

Una vez dado de alta el usuario, éste puede acceder en cualquier momento a la aplicación sin más que teclear su nombre de usuario y contraseña correctamente, él mismo puede cambiar cualquiera de sus datos y actualizarlos mediante la opción correspondiente del menú de usuario. La aplicación dispone de un panel de información de usuario en la esquina inferior izquierda, justo debajo del árbol de ejemplos y ejercicios en el que se muestra el nombre de usuario y el número de ejercicios que éste ha resuelto correctamente, erróneamente y los consultados. Estos datos se actualizan automáticamente al llevar a cabo la consulta o corrección de cualquier ejercicio. La base de datos registra la ruta del ejercicio consultado o resuelto de forma correcta o incorrecta, esta información se puede consultar mediante la opción del menú de usuarios de consultar.

El usuario administrador, tiene como privilegios la creación de ejercicios en la aplicación y la consulta de la información sobre el perfil de cualquier otro usuario registrado mediante la opción de buscar usuario del menú correspondiente, sin más que introducir el DNI de la persona que se desea consultar la información, este rol está exclusivamente reservado para el administrador, con el objetivo de que el profesor pueda acceder a la información del los ejercicios que los alumnos resuelven habitualmente y de forma correcta o incorrecta.

Bajo ninguna circunstancia se permite que el administrador altere ninguno de los datos que los usuarios introdujeron al registrarse y tampoco existe ninguna forma de que los propios usuarios modifiquen sus listas de ejercicios tanto consulados como resueltos bien o mal.

La Base de datos se almacena en un fichero.xml con la información de los usuarios y administrador encriptada, para evitar que se pueda obtener cualquier información de usuario al revisar el archivo de la base de datos.

4.6.2 Base de datos de ejemplos y ejercicios

La base de datos de ejemplos y ejercicios de la aplicación se carga en el panel izquierdo de la vista gráfica que tiene forma de árbol de directorios, con el fin de que se puedan cargar cualquiera de los ejemplos o ejercicios de forma directa sin más que hacer doble click sobre el ejemplo que se desee.

Cualquier usuario puede guardar un ejemplo en un archivo xml en cualquier parte de su pc para posteriormente cargarlo de nuevo, sin embargo y por comodidad cada vez que se guarde un ejemplo la aplicación pregunta al usuario si desea incluir dicho ejemplo en la base de datos, si contesta de forma afirmativa, el ejemplo queda cargado en el árbol de directorios automáticamente, y éste se refresca instantáneamente, de forma que a partir de ese momento el usuario ya dispone del ejemplo para cargarlos sobre el panel de dibujo haciendo doble click sobre él. El ejemplo se guardará en la carpeta del árbol según corresponda al tipo de autómata que se ha guardado. Esta característica es totalmente independiente del guardado del archivo.xml que el usuario haya decidido y donde lo considere oportuno, es decir, tanto si el usuario responde afirmativamente a guardar el ejemplo en la base de datos como si no, se guarda el archivo xml en la carpeta que se decidió anteriormente.

Es posible también guardar los ejercicios en la base de datos y se realiza de la misma forma que con los ejemplos, sin embargo esta acción está restringida al usuario administrador, ya que es necesario especificar la solución en algunos casos como en los ejercicios de calcular la expresión regular de un leguaje, en otros sin embargo el profesor (o administrador) sólo ha de dibujar el autómata que desea que el alumno vea y al guardar el ejercicio; la aplicación calcula la solución y la almacena automáticamente. Los ejercicios al ser guardados también refrescan el árbol de ejemplos/ejercicios automáticamente.

Al actualizar un ejercicio o ejemplo nuevo modificando el archivo xml del que se carga dinámicamente el árbol de la vista gráfica, se modifica el árbol de directorios de forma dinámica también.

Capítulo 5.

Manual de usuario

5.1 Perfiles de usuario

La aplicación soporta básicamente dos tipos de usuario:

1. Usuario administrador del sistema.
2. Usuario normal.

El usuario normal puede realizar todas las opciones que ofrece la aplicación, ejecutar algoritmos sobre ejemplos, crear ejemplos, y evaluarse a sí mismo probando los ejercicios.

El usuario administrador puede además crear nuevos ejercicios para añadirlos a la base de datos de ejercicios.

5.2 Ejecución de algoritmos

Esta parte de la aplicación permite realizar las siguientes operaciones:

1. Traducir una expresión regular a autómata finito con transiciones lambda.
2. Eliminar las transiciones lambda de un autómata finito con transiciones lambda, obteniendo un autómata finito no determinista.
3. Traducir un autómata finito no determinista a un autómata finito determinista.
4. Minimizar un autómata finito determinista.
5. Comprobar la equivalencia de dos autómatas finitos deterministas.
6. Traducir un autómata finito determinista a expresión regular.

Para realizar las distintas operaciones, podemos dibujar nuestro propio autómata, o escribir nuestra propia expresión regular, o bien podemos cargar uno de los ejemplos de autómatas disponibles en el árbol de directorios de la izquierda de la ventana.

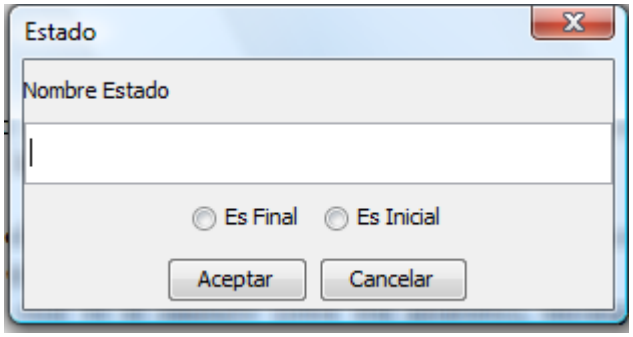
a) ¿Cómo dibujar un autómata en la interfaz?

Para dibujar un nuevo autómata, del tipo que sea, en la interfaz gráfica debemos hacer click sobre el siguiente botón:

Ahora podemos comenzar a dibujar el autómata, mediante los siguientes botones:

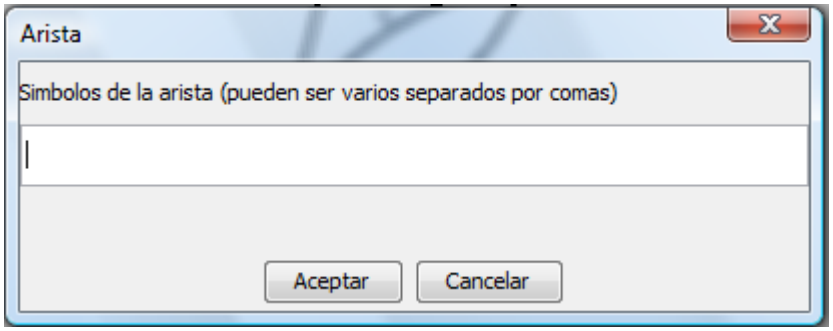
La **flecha** indica que estás activando el modo de editar, es decir puedes seleccionar los estados y aristas y moverlos por la ventana.

El **círculo** es el símbolo de Nuevo Estado, al pulsarlo, el sistema nos solicita el nombre y su condición de final, inicial o estado normal.



La **barra** es el símbolo de nueva arista, una vez pulsado, debemos seleccionar un estado y llevar la arista que se dibuja a otro estado, o al propio estado.

El sistema nos solicita cual es el símbolo (letra del alfabeto), encargado de hacer la transición.



La calavera permite eliminar estados y aristas, una vez que se ha pulsado, haciendo click con el ratón sobre un estado o una arista, desaparecerá.

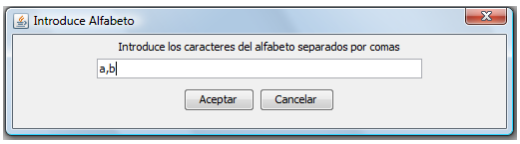
En cualquier momento podemos editar un estado o una arista, debemos pulsar con el botón derecho del ratón sobre un estado o una arista y el sistema nos permitirá modificar sus características.

Hay que tener en cuenta que en los autómatas con transiciones lambda el símbolo / es la lambda, si usamos ese símbolo al etiquetar una transición mientras dibujamos un autómata, dicho autómata se considerará autómata con transiciones lambda.

b) ¿Cómo cargar una expresión regular?

Para cargar una expresión regular en el sistema debemos utilizar el siguiente botón:

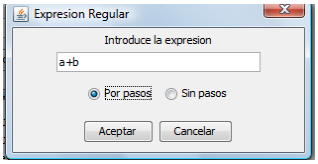
Acto seguido el sistema nos solicita el alfabeto correspondiente a la expresión regular, en el ejemplo hemos elegido el alfabeto a, b.



Después el sistema nos solicita cual es la expresión regular, en el ejemplo a+b.

Cuando el sistema recibe la expresión regular calcula automáticamente el autómata con transiciones lambda que reconoce el lenguaje expresado por la expresión regular.

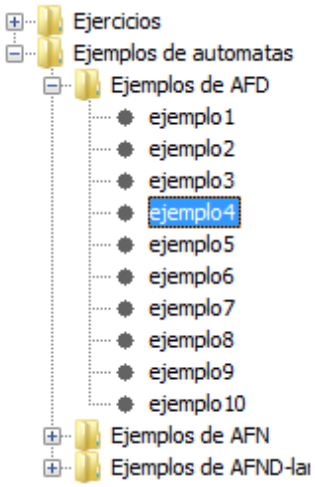
El sistema nos da a elegir si queremos que se ejecute por pasos, o sin pasos.



Si elegimos la primera opción, se abrirá una página web html, con toda la información de ejecución del algoritmo y se cargará el autómata con transiciones lambda en la ventana principal.

Si por el contrario decidimos ejecutarlo sin pasos, el sistema cargará el autómata con transiciones lambda en la ventana principal, sin mostrar la página HTML con los pasos.

c) ¿Cómo cargar un ejemplo de la base de datos de ejemplos?

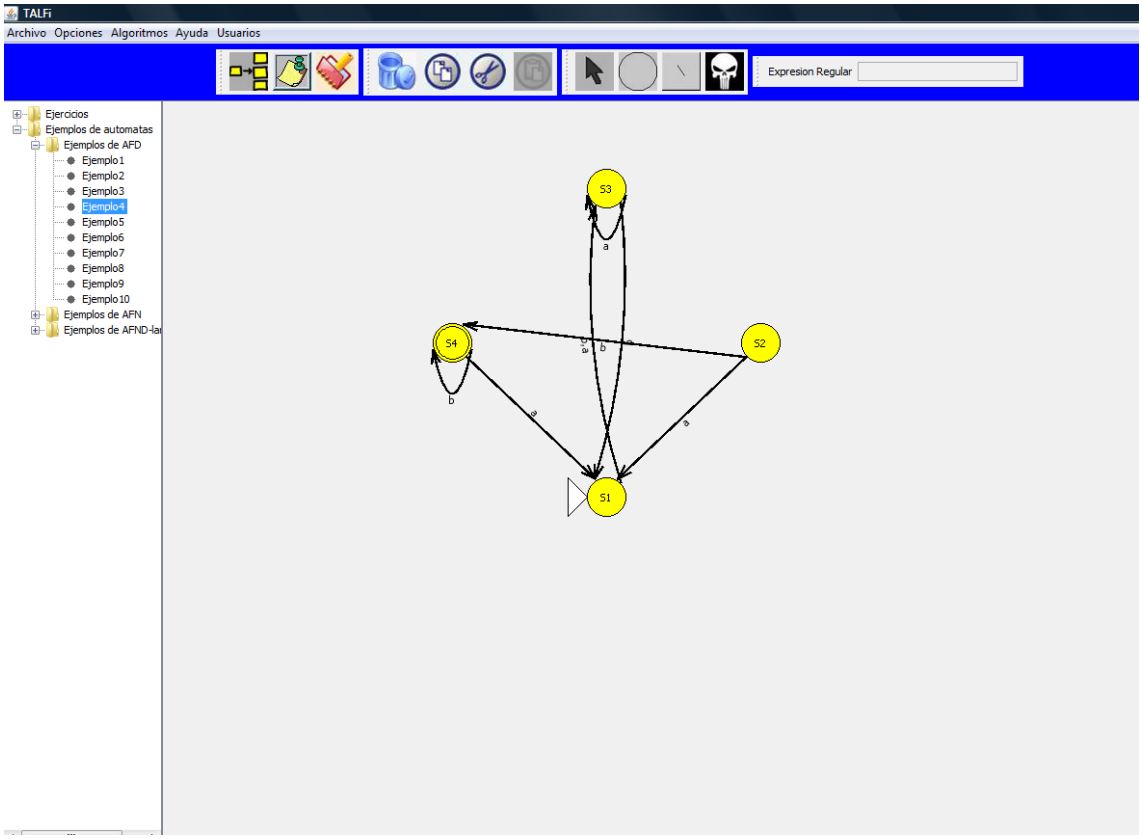


Los ejemplos se dividen en los tres tipos de autómatas con los que trabaja la aplicación, si como en la imagen seleccionamos el ejemplo 4, se está seleccionando un autómata finito determinista, esto hay que tenerlo en cuenta para posteriormente seleccionar el algoritmo adecuado.

En este caso, los algoritmos posibles serían: minimización, equivalencia de autómatas y traducción de autómata finito determinista a expresión regular.

Una vez cargado el ejemplo, podremos modificarlo a nuestro gusto, con los botones de edición, al igual que ocurre con los autómatas dibujados por nosotros.

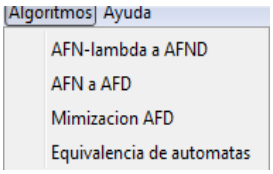
Haciendo doble click con el ratón sobre cualquiera de los ejemplos, cargaremos la información del ejemplo sobre la ventana principal, como se ve en la siguiente imagen.



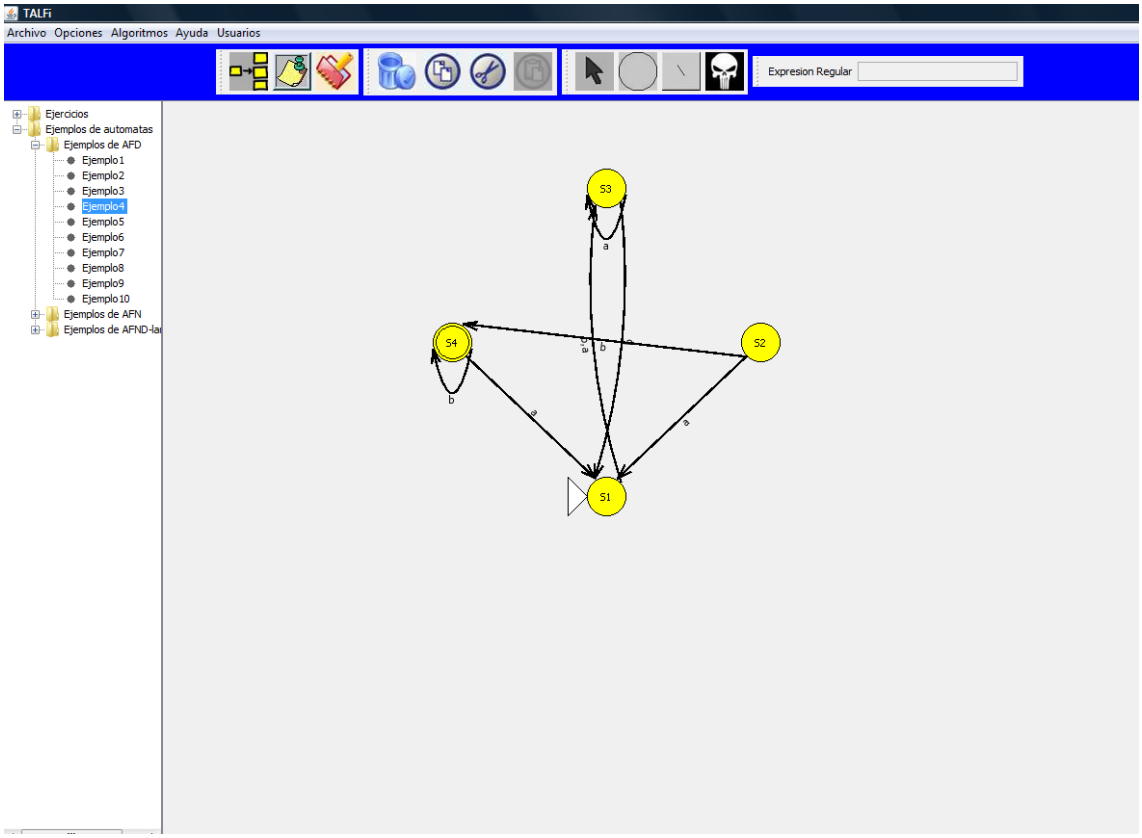
d) ¿Cómo ejecutar un algoritmo?

Para ejecutar un algoritmo debemos seleccionar en el menú de navegación, la pestaña algoritmos y seleccionar el algoritmo que queramos ejecutar.

Hay que tener en cuenta que el algoritmo debe ser adecuado para el tipo de autómata que tenemos dibujado en la ventana.

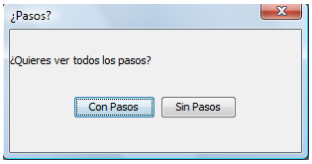


Por ejemplo si tenemos cargado en la ventana un autómata finito determinista, como se ve en la imagen:



Podemos ejecutar por ejemplo el algoritmo de minimización de autómatas finitos deterministas.

El sistema solicitará si queremos ver los pasos de ejecución o no, mediante la siguiente ventana:



Si decidimos ver los pasos, el sistema ejecutará un explorador de internet y cargará una página web html con la información de ejecución y los pasos de ejecución.

En el ejemplo mostrará: el autómata de entrada, la tabla de minimización y el autómata de salida.

Además el sistema cargará en la interfaz gráfica el nuevo autómata generado, en nuestro ejemplo, el autómata minimizado, con el cual podemos operar y aplicar nuevos algoritmos.

Otros algoritmos tienen un tratamiento distinto, pero es sencillo seguir las instrucciones que marca la aplicación para ejecutarlos.

5.3 Sistema de proposición y evaluación de ejercicios

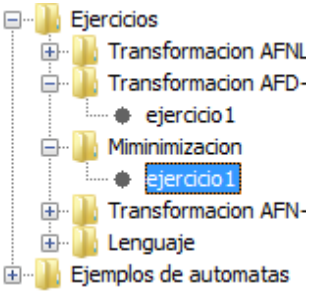
La aplicación cuenta con un sistema de proposición de ejercicios y corrección de las soluciones enviadas por el usuario.

Los ejercicios se engloban en 5 grandes grupos:

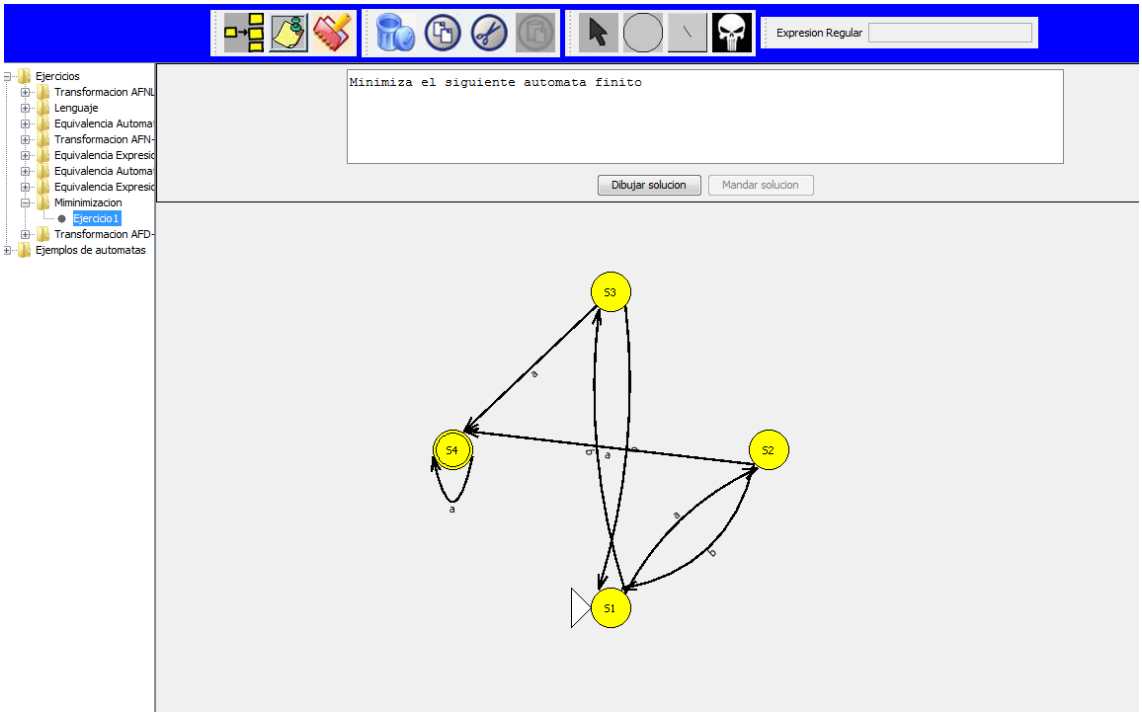
1. Ejercicios de Lenguaje/Expresiones regulares.

2. Ejercicios de transformación AFNLambda a AFN.
3. Ejercicios de transformación AFN a AFD.
4. Ejercicios de minimización de AFD.
5. Ejercicios de transformación de AFD a ER.

Para cargar un ejercicio nuevo, que queremos intentar resolver, debemos ir al menú de la aplicación, y elegir un ejercicio de la batería de ejemplos:



Si hacemos doble click sobre cualquiera de los ejemplos, el sistema cargará en la interfaz gráfica, el enunciado y el contenido del ejercicio. Tal y como se observa en la siguiente imagen:



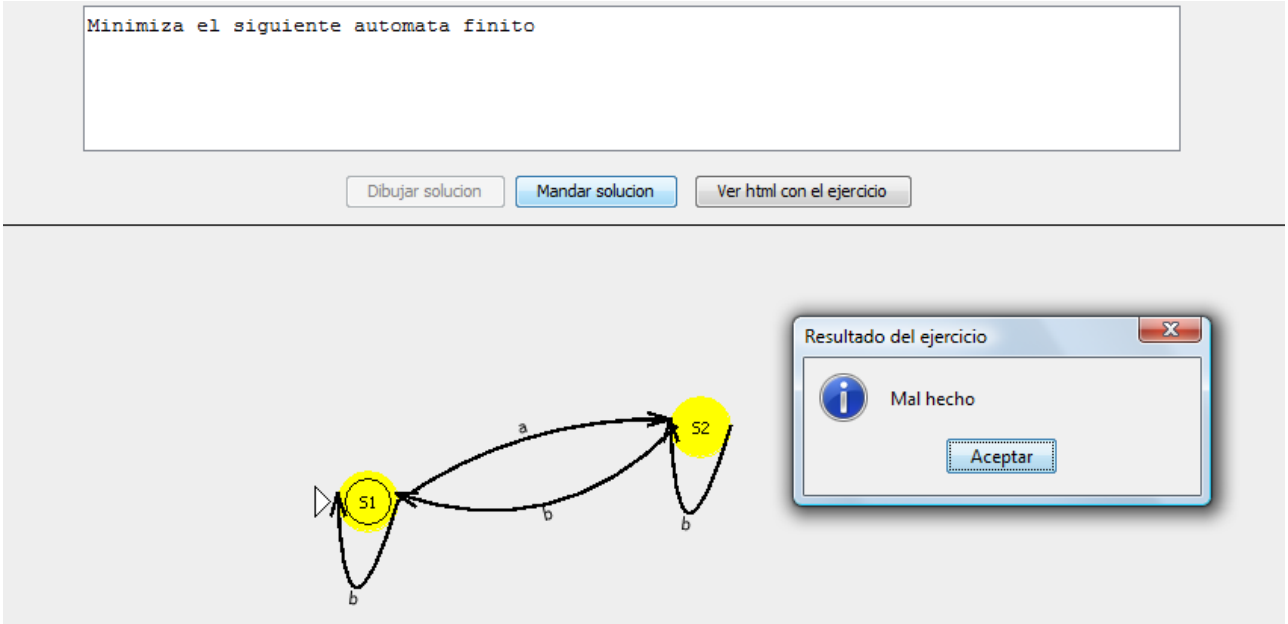
En la imagen se observa, que hemos seleccionado un ejercicio de minimización y el sistema nos muestra el enunciado y el autómata a minimizar.

Una vez resuelto el ejercicio en papel, debemos enviar nuestra solución al sistema.

Si hacemos click sobre el botón: **Dibujar solución,** el sistema solicitará que tipo de autómata queremos dibujar, en este caso autómata finito determinista, y acto seguido podemos dibujarlo sobre la ventana de la aplicación.

Si una vez que le hemos dado a Dibujar solución hemos olvidado, que autómata queríamos minimizar no tenemos más que dar al botón: **ver html con el ejercicio** y se abrirá un explorador web mostrándonos el enunciado del ejercicio, con toda la información.

Una vez terminado el dibujo, debemos enviarlo a corregir, pulsando sobre el botón **Mandar solución.** El sistema nos indicará cual es el resultado de la corrección, como se ve en la siguiente imagen:



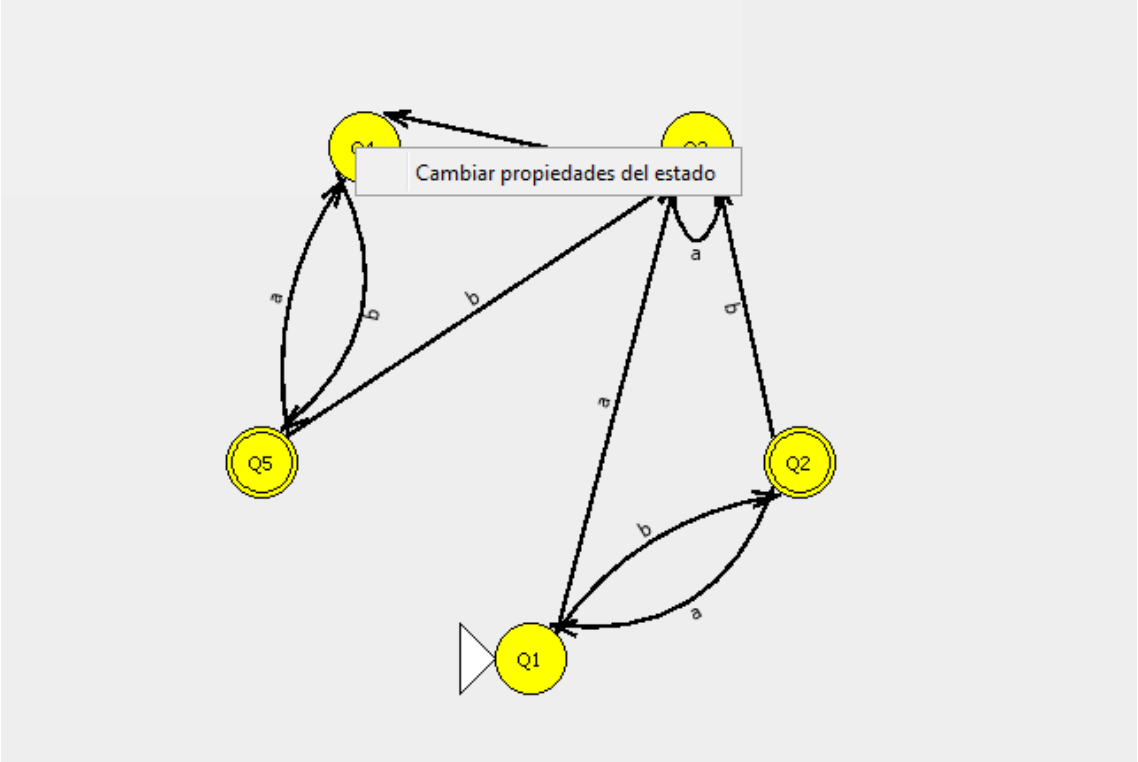
El tratamiento de ejercicios con expresiones regulares (lenguaje o traducción de AFD a expresión regular) es el mismo, salvo que en este caso debemos escribir la expresión regular resultado.

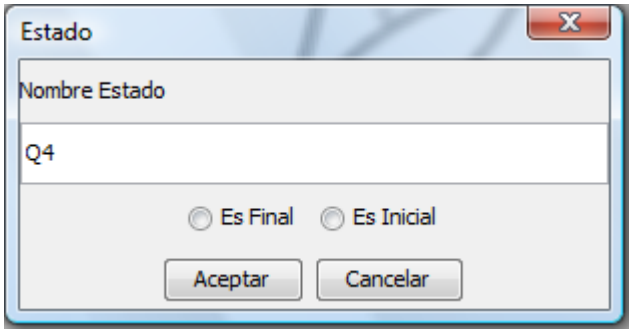
5.4 Cambiar propiedades de los estados y aristas

Una vez tenemos en nuestro panel dibujado el autómata que hemos cargado del árbol de ejemplos de la BD, o hemos dibujado nosotros mismos a mano, o incluso recuperado de nuestro disco duro, podemos modificar cada uno de los estados del mismo o de sus aristas, la información que podemos modificar depende evidentemente del objeto sobre el que hayamos pulsado.

Para modificar un objeto del autómata hemos de pulsar con el botón derecho del ratón sobre el objeto que deseemos modificar. El propio panel se encarga automáticamente de identificar si es un estado, una arista o nada, sobre lo que hemos pulsado, en cualquiera de los tres casos nos aparecerá sobre el panel un menú contextual con las opciones de modificar estado, arista o varis de copiar, cortar y pegar que explicaremos después. Por ahora vamos ha centrarnos en modificar aristas y estados.

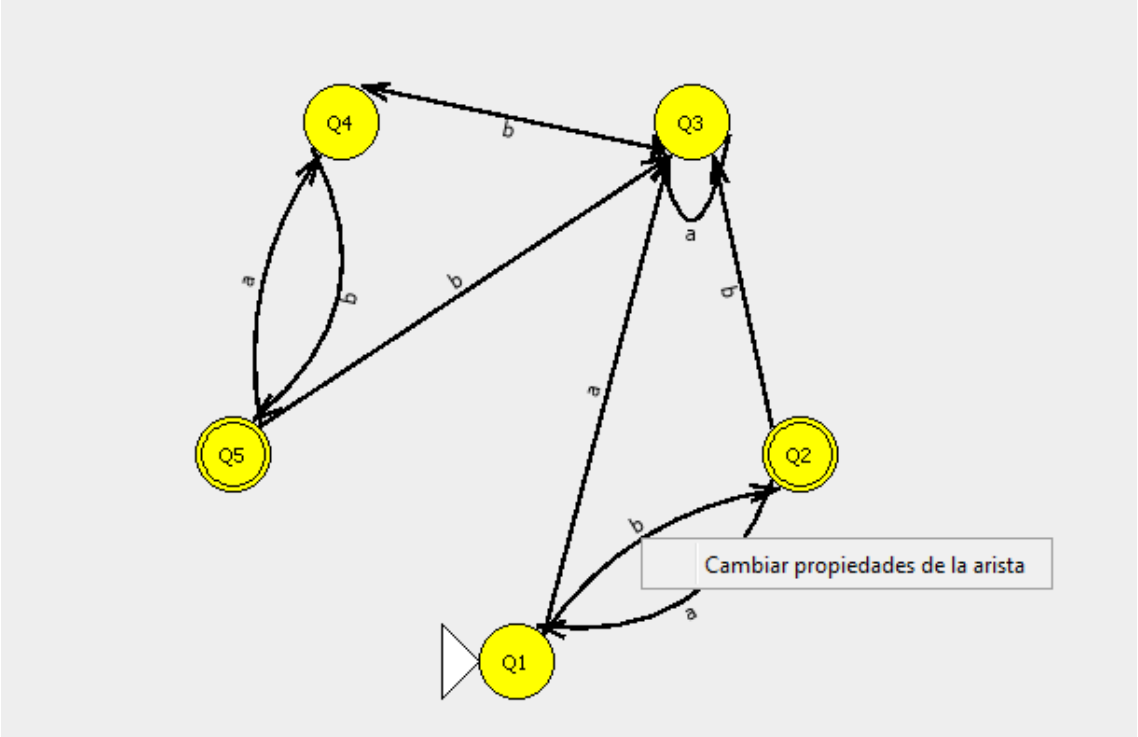
En el caso de la imagen que se muestra a continuación hemos pulsado con el botón derecho sobre el estado Q4, si nos ponemos sobre el menú contextual y pulsamos para cambiar las propiedades del estado, nos aparecerá una ventana exactamente igual a la que nos aparece al añadir estados, con los campos de información rellenos según las características que tenga el estado seleccionado. De esta forma podemos cambiar a nuestro gusto las propiedades del estado y una vez las aceptemos se verán reflejadas en el panel de dibujos inmediatamente.

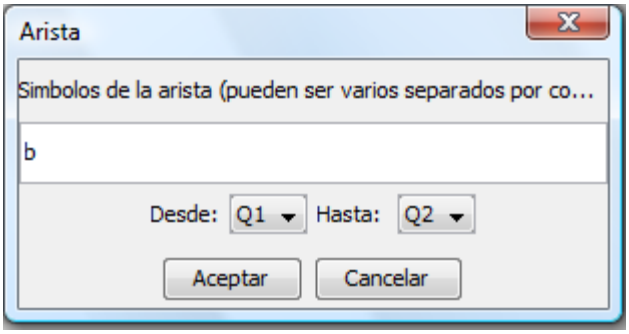




De igual forma se procede con las aristas, solo que en este caso la ventana de cambio de propiedades es distinta e incluye un campo de texto (relleno con las letras de la transición) para poner las letras que desencadenarán el paso por la arista y que se pueden modificar o añadir nuevas y dos menús donde están los estados entre los que está dibujada la arista y que podemos cambiar a nuestro gusto sin más que seleccionar los estados entre los que deseemos que se redibuje la arista de cualquiera de las dos listas con todos los estados del autómata. De nuevo al aceptar los cambios son visibles de forma inmediata.

En la imagen a continuación expuesta se ve que hemos pinchado con el derecho sobre la arista que une los estados Q1 y Q2 y que tiene como letra una “b”, la siguiente imagen muestra el cuadro de texto explicado del cambio de propiedades de las aristas.

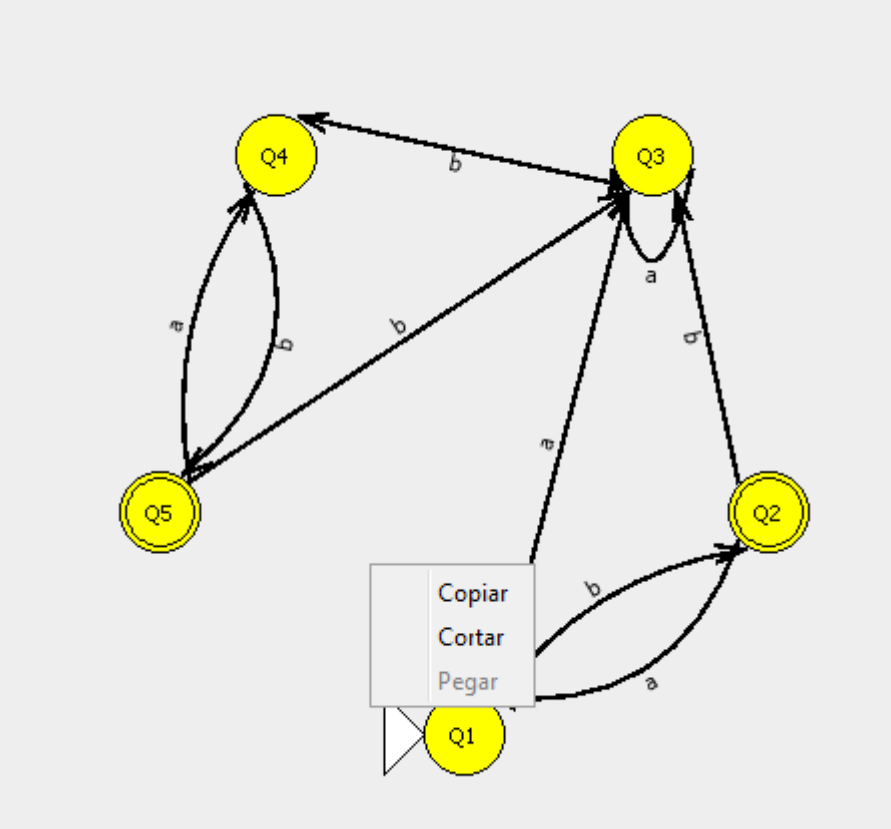




5.5 Copiar, cortar y pegar

En el apartado anterior hemos visto cómo al pulsar con el botón derecho sobre un estado o una arista salían dos menús diferentes, pasando por alto el que salía al pulsar sobre el panel sin ningún dibujo debajo, pues bien es el momento de explicar las opciones que salen, las conocidas por todos copiar, cortar y pegar, sólo que en nuestra aplicación tienen un comportamiento un poco distinto, veámoslo.

Antes de nada resaltar que estas opciones de copiar, cortar y pegar se pueden realizar así mismo mediante los botones correspondientes de la barra de herramientas sobre el panel de dibujo o incluso con los eventos de pulsación de teclas a los que estamos acostumbrados: Control + C para copiar, Control + V para pegar y Control + X para cortar.



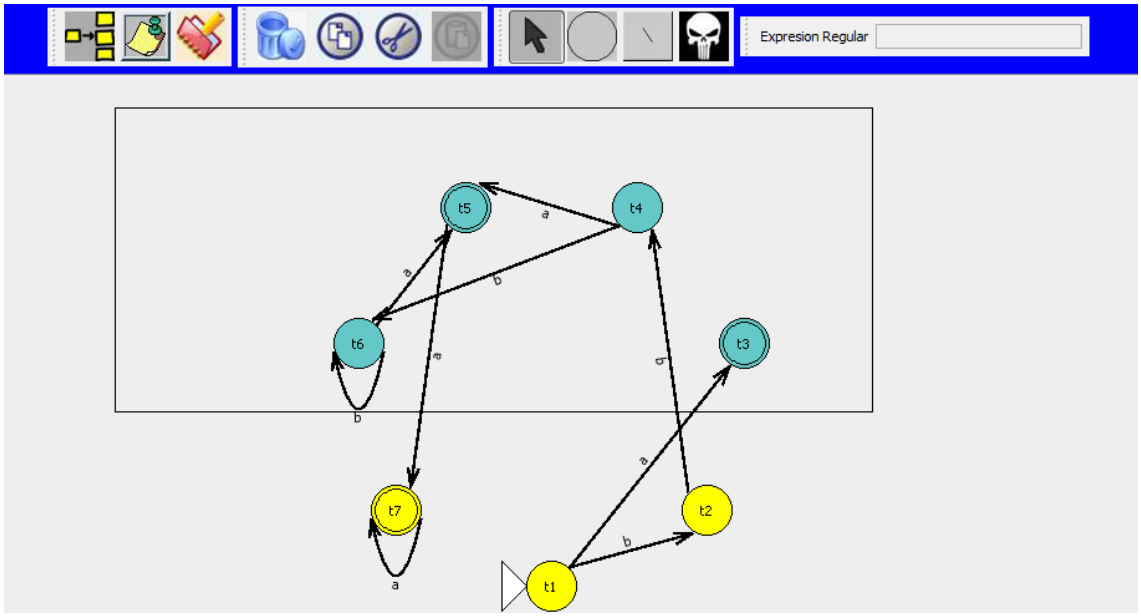
Como podemos ver en las imágenes los botones de pegar están desactivados, ¿por qué? ¿Es que no se puede pegar? Pues no, hasta que no hayamos copado o cortado alguna vez no es posible pegar nada, una vez hayamos hecho cualquiera de las dos cosas el botón de pegar se activará y ya podremos pegar todas las veces que deseemos, siempre se pegará el último autómata cortado.

Los eventos de copiar, cortar y pegar en la aplicación realizan la acción sobre el contenido completo del panel, es decir que copian, cortan y pegan autómatas completos, en ningún caso se puede seleccionar parte de un autómata y copiarlo, cortarlo o pegarlo, cosa que sí se puede hacer a la hora de suprimir parte de un autómata como veremos ahora.

Veamos la imagen del botón de pegar activado. 

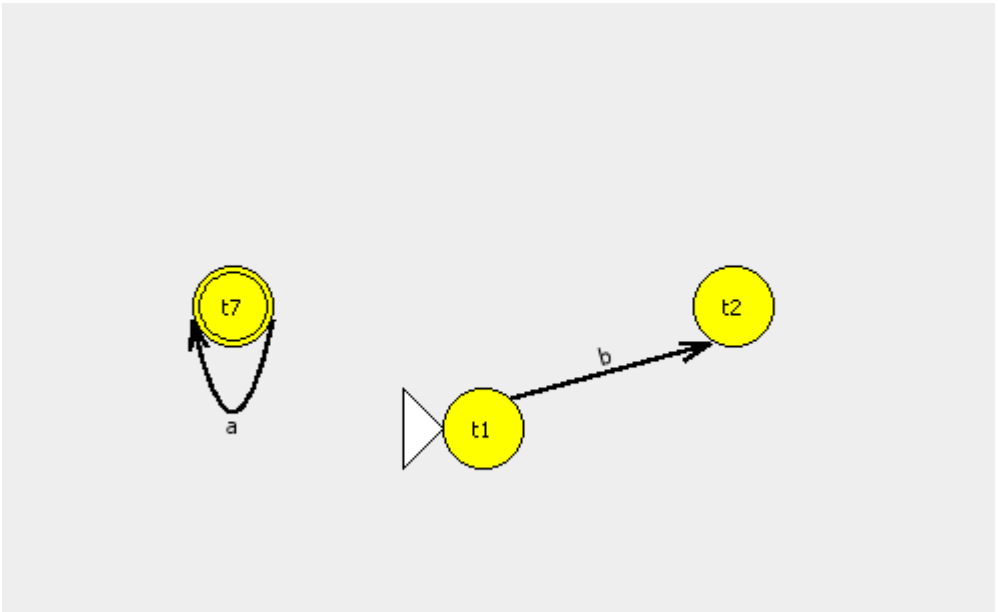
5.6 Seleccionar parte de un autómata. Interacción

Es posible suprimir parte de una autómata dibujado en el panel, por cualquiera de los métodos que facilita la aplicación, para ellos el primer e imprescindible paso es tener seleccionada la herramienta de editar (la “flechita” de la herramientas de la barra superior), a continuación hemos de pinchar con el botón izquierdo del ratón sobre el panel y sin soltarlo arrastrar sobre él para abrir un cuadro de selección, a medida que vayamos englobando estados, estos cambiarán de color con el fin de confirmar gráficamente que los hemos seleccionado. Cuando hayamos alcanzado los estados deseados y queramos finalizar el dibujo del cuadro de selección sólo tenemos que soltar el botón, el cuadro desaparecerá pero los estados seleccionados se mantendrán.



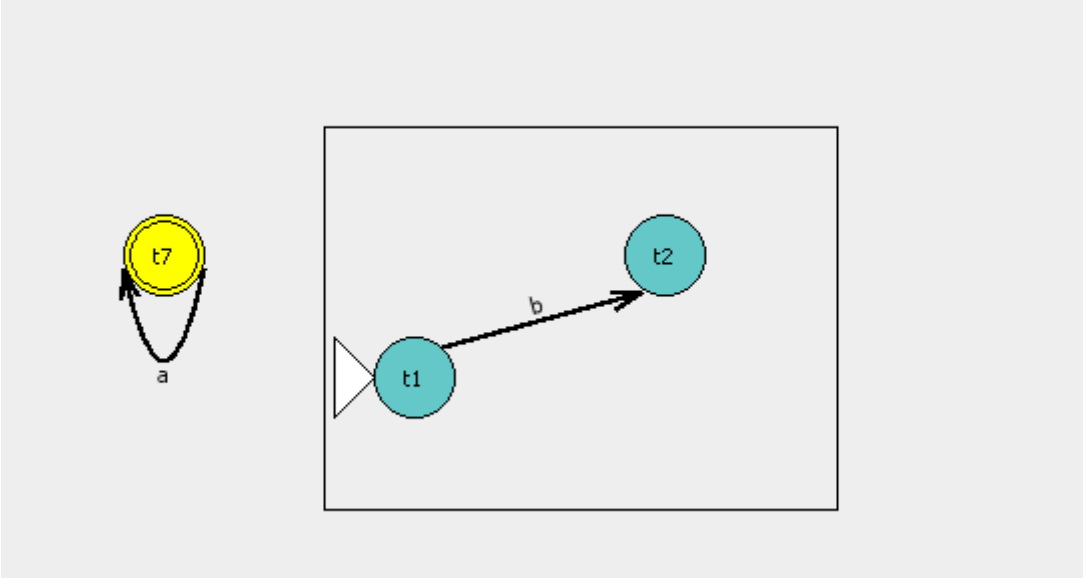
Una vez seleccionada la parte deseada, podemos o bien variar su posición dentro del panel, o suprimir los estados seleccionados, para realizar esta última tarea sólo hemos de pulsar el botón suprimir y se eliminarán los estados que están seleccionados y con sus respectivas aristas tanto de entrada como de salida a los mismos.

En esta imagen hemos borrado los estados que seleccionamos anteriormente, nada más que pulsando suprimir.

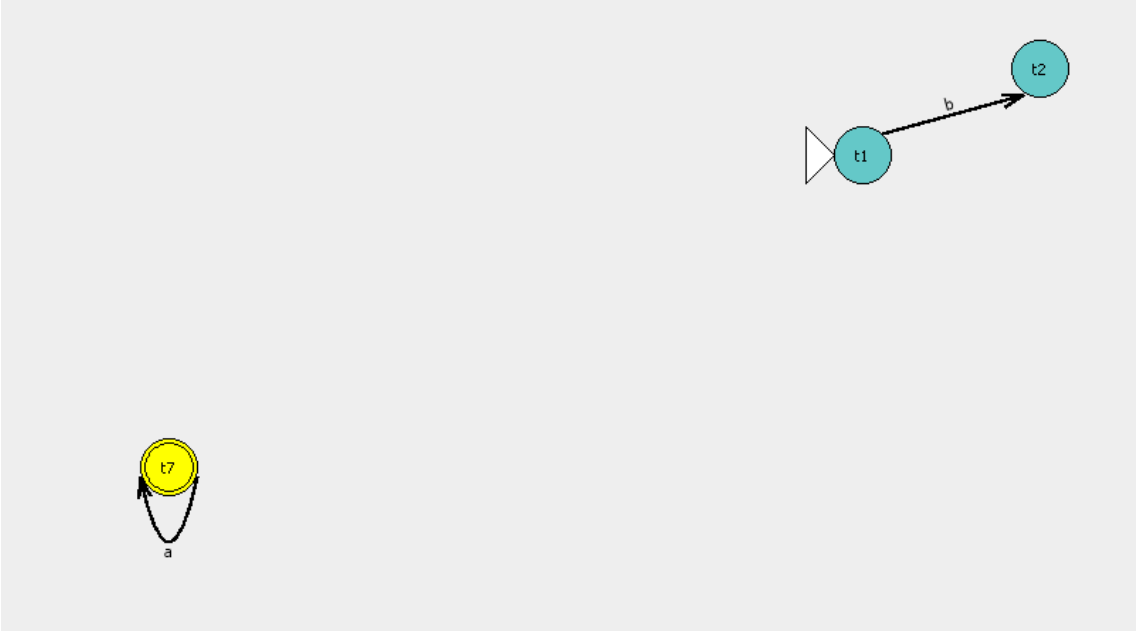


Si quisiéramos mover la parte seleccionada del autómata, solamente habríamos de pulsar u arrastrar uno de los estados seleccionados mediante el cuadro de selección que habríamos abierto anteriormente. Así de esta forma movemos la selección entera que hayamos hecho.

Seleccionemos de los estados que quedan los dos estados t1 y t2:



Y los movemos a la esquina derecha pulsando y arrastrando t2:

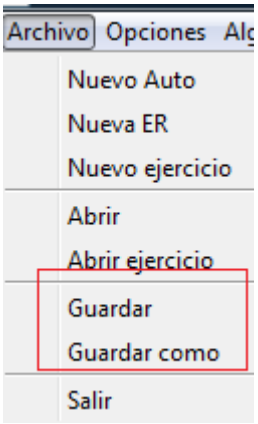


5.7 Almacenar y recuperar autómatas y ejercicios. Adjuntarlos a la BD

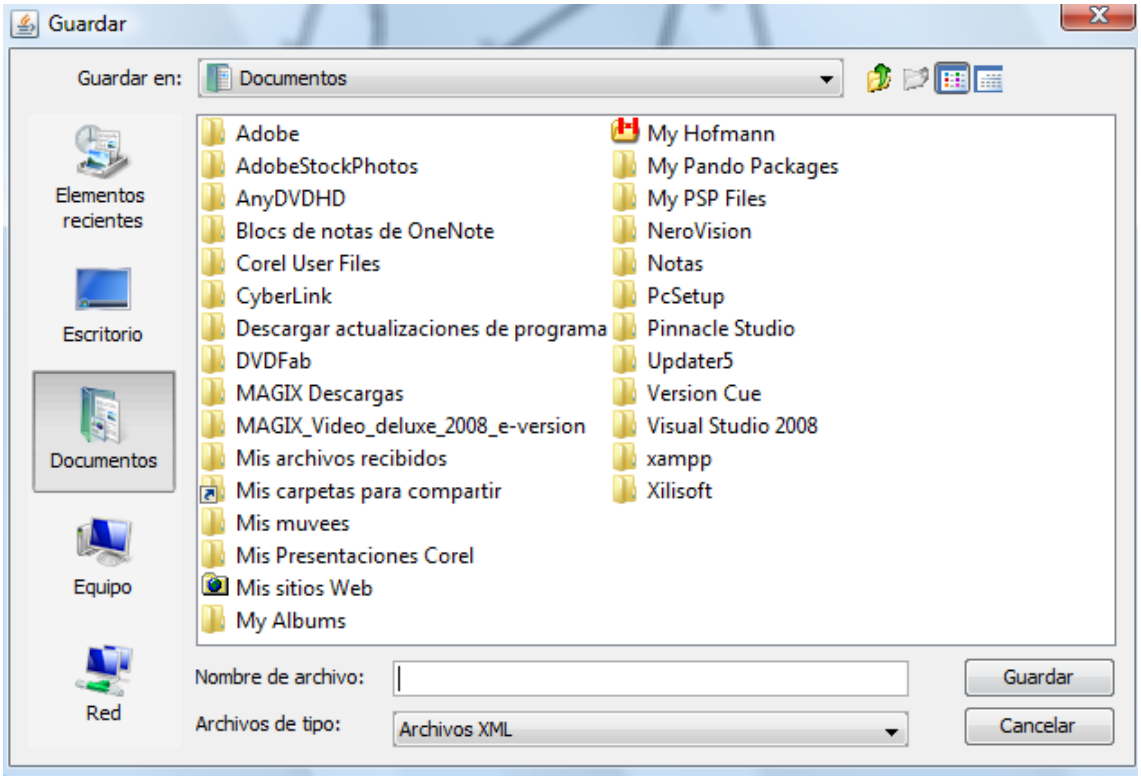
5.7.1 Guardar/Recuperar autómatas dibujados

Con el fin de poder salvaguardar los dibujos de autómatas que los usuarios realicen en sus sesiones, se ha incluido una opción en al menú de archivo de guardar y guardar como, de las cuales explicaremos las diferencias después. Primero nos centraremos en guardar como.

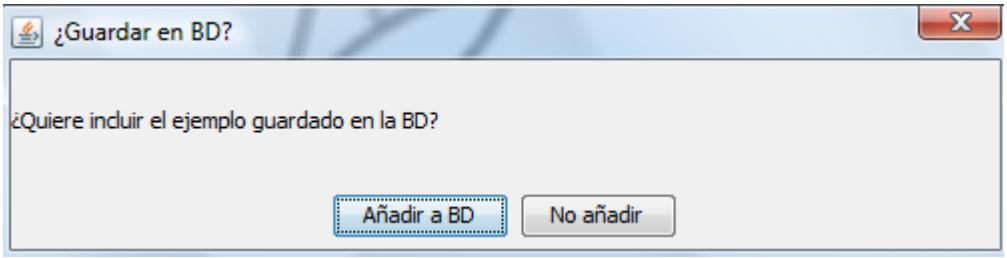
Una vez tenemos el autómata deseado en el panel, ya sea dibujándolo a mano o cargándolo desde los ejemplos, podemos guardarlos pulsando sobre la opción de menú guardar como, entonces nos aparecerá una ventana de guardado donde podremos seleccionar la carpeta de destino así como el nombre que queramos darle al archivo, la extensión será siempre ".xml" aunque no la pongamos explícitamente. Una vez de acuerdo con los datos pulsaremos guardar.



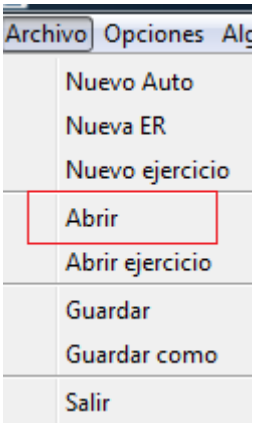
La diferencia con guardar a secas, es que si ya habíamos cargado el autómata desde un archivo guardado en nuestro PC, la aplicación memoriza dicha ruta y al pulsar guardar, se sobrescribe. Si pulsamos guardar sobre un autómata que no partía de un archivo guardado en el PC, se produce el mismo comportamiento que con el botón de menú de guardar como.



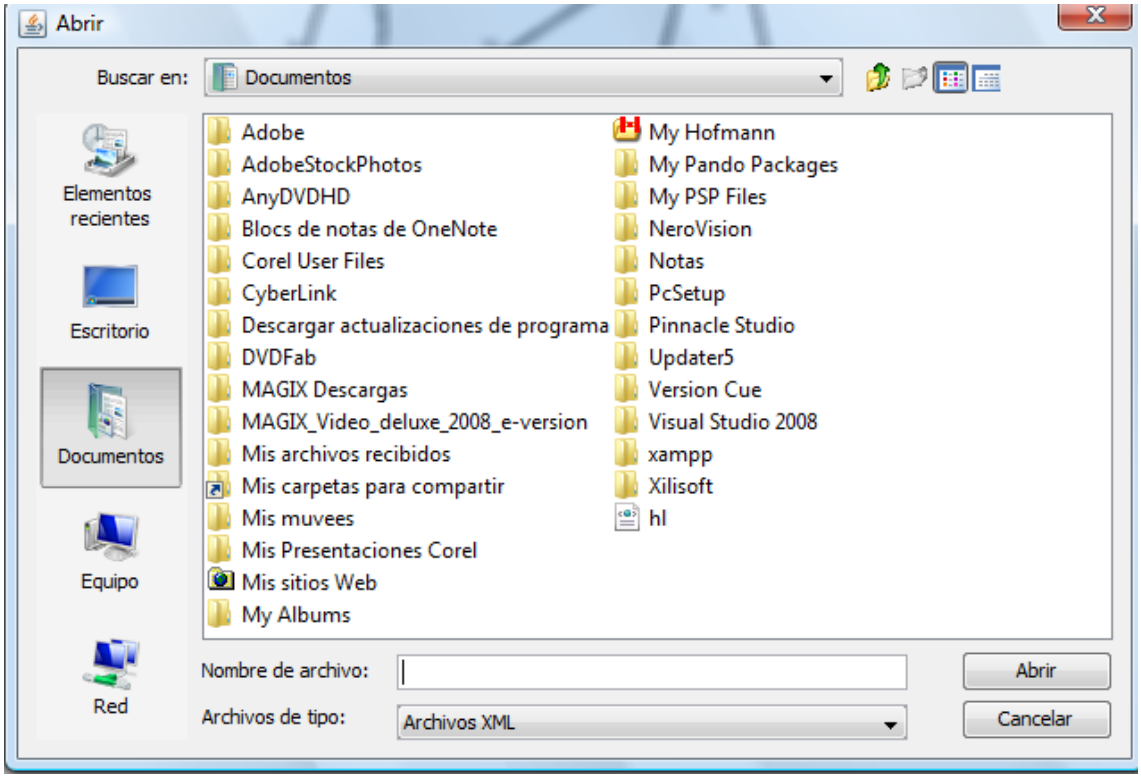
La opción de guardar una vez almacenado el ejemplo en el fichero correspondiente nos permite introducir dicho autómata en el árbol de ejemplos de la parte izquierda de la aplicación. Para ello tras confirmar el guardado del autómata, se inicia una ventana en la que se nos pregunta si deseamos guardar al autómata en la base de datos de ejemplos, si contestamos afirmativamente, de inmediato se realiza la acción y el panel de ejemplos en forma de árbol se recarga dinámicamente y ya es posible ejecutar el ejemplo que acabamos de guardar. Si hubiéramos contestado que no deseamos guardarlo, la ventana desaparece y el autómata no se guarda en los ejemplo, pero sí queda registrado en nuestro disco duro tal y como hicimos anteriormente.



Para abrir un autómata previamente guardado con el método anterior se dispone de nuevo en el menú con un botón específico para tal uso su nombre: abrir.



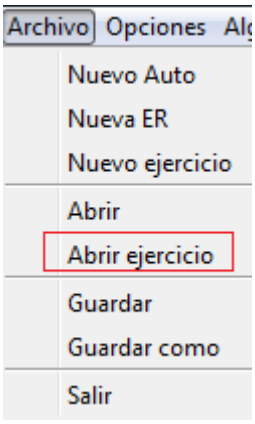
Al pulsar sobre él, nos aparece una ventana de apertura de archivos muy similar a la de guardar, donde debemos de buscar la carpeta que contenga el archivo que queremos abrir. En el panel central de la vista de abrir, nos aparecerán los archivos que contiene la carpeta en la que estemos, sin embargo única y exclusivamente aparecerán las carpeta, que podemos abrir para entrar en un nuevo directorio, y los archivos con extensión .xml ya que son los únicos reconocidos por la aplicación. Nada más hemos de selecciona un archivo válido de un autómata y hacer doble click sobre él, o pulsar aceptar para que se dibuje en el panel, manteniendo las coordenadas que los estados tenían cuando se guardó.



5.7.2 Recuperar ejercicios

Ya hemos explicado a lo largo de la memoria que el único que puede crear y guardar ejercicios es el profesor en su rol como administrador, sin embargo la aplicación sí que incluye la capacidad de abrir los ejercicios que el profesor les pueda hacer llegar a los alumnos, eso sí, siempre en un formato .xml como el resto de archivos aceptados por el sistema.

Para abrir un ejercicio se realiza como si fuera un autómata, sólo que para esta función se ha incluido un botón diferente en el menú, es el de abrir ejercicio.



Al pulsar sobre el botón aparece una ventana de diálogo igual a la que se utilizó para abrir autómatas, y con las mismas características que aquella, con lo que remitimos al lector a su explicación para comprender esta.

Antes de concluir este apartado de guardar y recuperar autómatas y ejercicios, queremos indicar otra optimización de la aplicación, pues una vez se ha guardado o abierto un autómata o ejercicio la siguiente vez que se intente realizar una de dichas acciones, la aplicación abrirá la ventana de guardar o abrir respectivamente, con la última carpeta

donde se guardó o abrió el último archivo, es decir que la aplicación guarda información sobre la última carpeta en la que se efectuó una acción de este tipo.

5.8 Aclaraciones finales

Para finalizar el manual de usuario, queremos indicar que estas explicaciones se han realizado con la apariencia de la aplicación sobre un sistema operativo Windows, más concretamente en su versión Vista. Otro usuarios que la ejecuten sobre otros sistemas operativos apreciarán solamente cambios de apariencia en las ventanas, pues hemos incorporado la opción de que java tenga la apariencia del sistema operativo en que se ejecute.

De igual forma sucede con el idioma, cada una de las ventanas de información y diálogo se muestran con el texto en español, pero es posible cambiarlo o si se encuentra que java se ejecuta con un idioma distinto al español, se ejecutará por defecto en inglés. Todas las ventanas al ser ejecutadas con el idioma inglés tienen los textos de indicación, botones y título en dicho idioma.

Capítulo 6.

Planificación temporal del proyecto

En este apartado queremos exponer de manera resumida pero concisa como hemos organizado el trabajo durante los meses que hemos estado desarrollando el proyecto.

La planificación completa se encuentra en el CD adjunto en formato mpp de Microsoft Project para más detalles, el archivo se llama planificación.mpp.

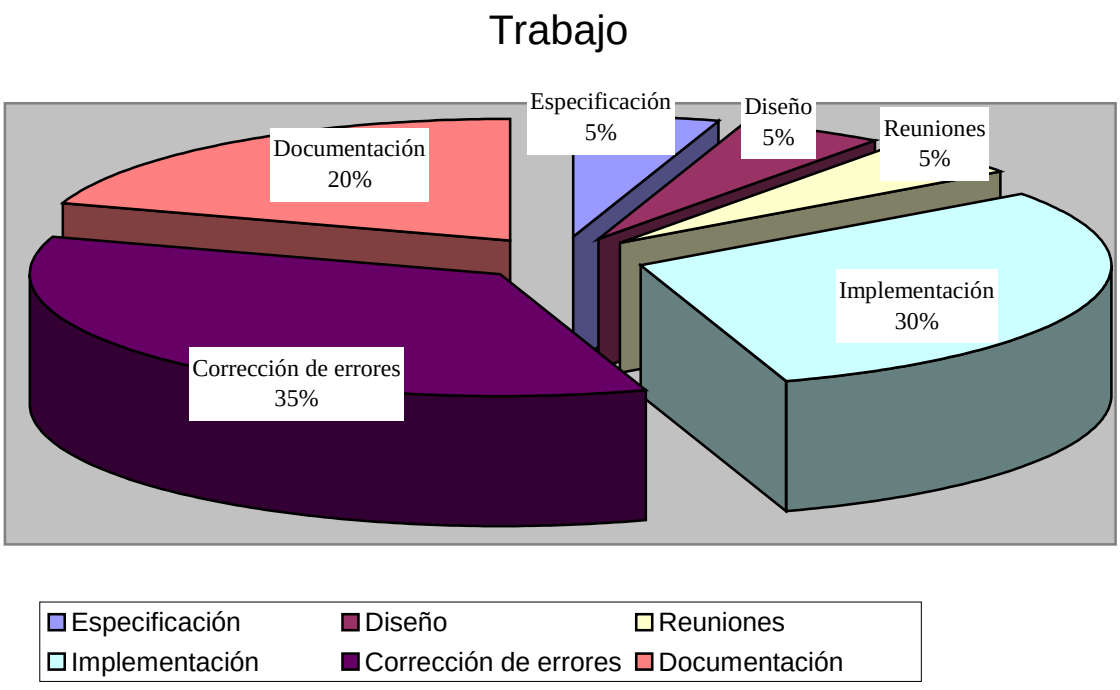
6.1. Diagrama de Gantt

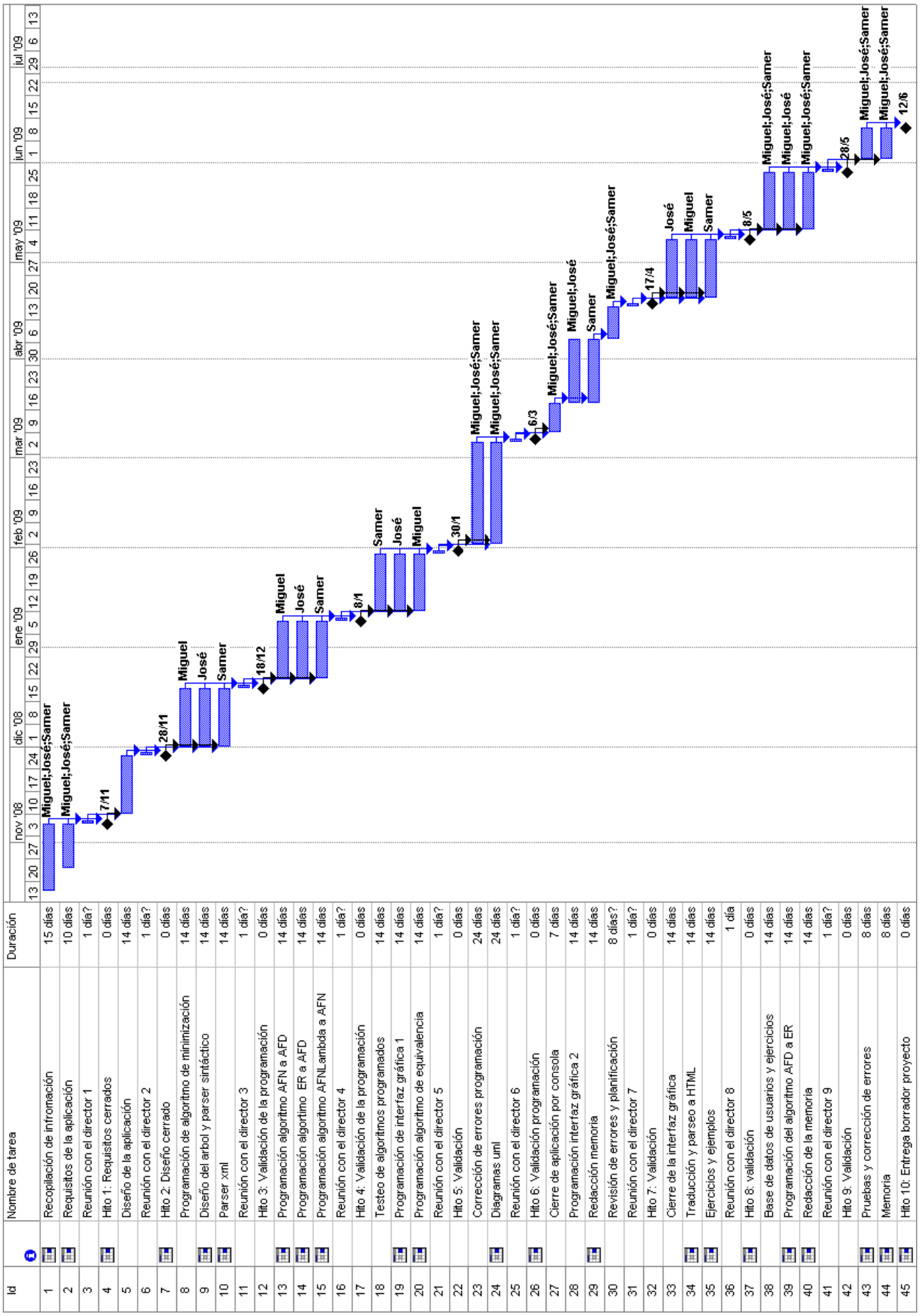
En el diagrama de Gantt se observa el trabajo repartido entre los 3 miembros del grupo de trabajo durante los meses de desarrollo.

El diagrama de Gantt tiene como objetivo mostrar la planificación, es decir, el trabajo previsto. Nosotros lo hemos usado para ello y para finalmente mostrar el trabajo real, que es el diagrama que adjuntamos.

6.2. Resumen de la planificación temporal

Si se observa detenidamente el diagrama se puede realizar un informe sobre la cantidad de trabajo que se ha dedicado a cada tarea, todo ello se puede resumir en el esquema que se presenta a continuación. En el cual se han agrupado las tareas en los siguientes grupos: especificación, diseño, implementación, corrección de errores, documentación y reuniones.




Id
Nombre de tarea
Duración
nov '08
dic '08
ene '09
feb '09
mar '09
abr '09
may '09
jun '09
jul '09
1 Recopilación de infromación 15 días
2 Requisitos de la aplicación 10 días
3 Reunión con el director 1 1 día?
4 Hito 1: Requisitos cerrados 0 días
5 Diseño de la aplicación 14 días
6 Reunión con el director 2 1 día?
7 Hito 2: Diseño cerrado 0 días
8 Programación de algoritmo de minimización 14 días
9 Diseño del arbol y parser sintáctico 14 días
10 Parser xml 14 días
11 Reunión con el director 3 1 día?
12 Hito 3: Validación de la programación 0 días
13 Programación algoritmo AFN a AFD 14 días
14 Programación algoritmo ER a AFD 14 días
15 Programación algoritmo AFNLamboda a AFN 14 días
16 Reunión con el director 4 1 día?
17 Hito 4: Validación de la programación 0 días
18 Testeo de algoritmos programados 14 días
19 Programación de interfaz gráfica 1 14 días
20 Programación algoritmo de equivalencia 14 días
21 Reunión con el director 5 1 día?
22 Hito 5: Validación 0 días
23 Corrección de errores programación 24 días
24 Diagramas uml 24 días
25 Reunión con el director 6 1 día?
26 Hito 6: Validación programación 0 días
27 Cierre de aplicación por consola 7 días
28 Programación interfaz gráfica 2 14 días
29 Redacción memoria 14 días
30 Revisión de errores y planificación 8 días?
31 Reunión con el director 7 1 día?
32 Hito 7: Validación 0 días
33 Cierre de la interfaz gráfica 14 días
34 Traducción y parseo a HTML 14 días
35 Ejercicios y ejemplos 14 días
36 Reunión con el director 8 1 día
37 Hito 8: validación 0 días
38 Base de datos de usuarios y ejercicios 14 días
39 Programación del algoritmo AFD a ER 14 días
40 Redacción de la memoria 14 días
41 Reunión con el director 9 1 día?
42 Hito 9: Validación 0 días
43 Pruebas y corrección de errores 8 días
44 Memoria 8 días
45 Hito 10: Entrega borrador proyecto 0 días
7/11
28/11
18/12
8/1
30/1
6/3
17/4
8/5
12/6
Miguel;José;Samer
Miguel
José
Samer

Capítulo 7.

Glosario de términos

- **Autómata:**

Un autómata es un modelo matemático para una máquina de estado finita (FSM sus siglas en inglés). Una FSM es una máquina que, dada una entrada de símbolos, "salta" a través de una serie de estados de acuerdo a una función de transición (que puede ser expresada como una tabla). En la variedad común "Mealy" de FSMs, esta función de transición dice al autómata a que estado cambiar dados uno determinado estado y símbolo.

– **TALF:**

Siglas de Teoría de autómatas y lenguajes formales, asignatura del segundo curso de Ingeniería Informática en la universidad complutense de Madrid. Trata lo referente al estudio de autómatas.

– **Alfabeto:**

El alfabeto, es el conjunto ordenado de las letras de un idioma. Referente a los autómatas, el alfabeto es el conjunto de letras que el autómata es capaz de comprender.

– **Lenguaje:**

Un lenguaje formal es un conjunto de palabras (cadenas de caracteres) de longitud finita en los casos más simples o expresiones válidas (formuladas por palabras) formadas a partir de un alfabeto (conjunto de caracteres) finito. El nombre lenguaje se justifica porque las estructuras que con este se forman tienen reglas de buena formación (gramática) e interpretación semántica (significado) en una forma muy similar a los lenguajes hablados o naturales.

– **Expresión regular:**

Una expresión regular es una expresión que describe un conjunto de cadenas sin enumerar sus elementos. Los elementos estarán formados por símbolos del alfabeto que describa la expresión regular.

– **Lambda:**

Lambda (Λ λ) es la undécima letra del alfabeto griego. En el estudio de autómatas normalmente se usa para nombrar a la palabra “vacía”.

– **Algoritmo:**

Un algoritmo es una lista bien definida, ordenada y finita de operaciones que permite hallar la solución a un problema. Dado un estado inicial y una entrada, a través de pasos sucesivos y bien definidos se llega a un estado final, obteniendo una solución.

– **Computabilidad:**

La Teoría de la computabilidad es la parte de la computación que estudia los problemas de decisión que pueden ser resueltos con un algoritmo o equivalentemente con una máquina de Turing.

Capítulo 8.

Palabras clave

Autómata, TALF, alfabeto, lenguaje, expresión regular, lambda, algoritmo, computabilidad.

Capítulo 9.

Conclusión

Desarrollar éste proyecto ha sido un auténtico reto desde el principio hasta el final. Ya que los algoritmos no solo debían funcionar y dar el resultado correcto sino que además debían aportar una explicación de cómo se ha llegado al resultado final, mediante unas tablas con los pasos que se han seguido para llegar a la solución. Pasos que pueden servir a los alumnos como guía y les ayudan a comprender la asignatura en su totalidad.

La lógica de la aplicación se centra en la codificación de todos los algoritmos automáticos de la parte teórica de la asignatura relacionada con expresiones regulares y autómatas finitos, cada uno de ellos con una salida explicativa en forma de los pasos que se han de dar para obtener el resultado correcto.

La aplicación permite convertir entre los diferentes tipos de autómatas y expresiones regulares a otro cualquiera de estas materias, ya que se cierra el ciclo completo de transformación con todos los algoritmos desarrollados. Este era nuestro principal objetivo, la realización de una aplicación de autómatas y expresiones cerrada, de ayuda a los alumnos.

Una vez que se cumple el objetivo de desarrollar una herramienta didáctica a través de los algoritmos con la muestra de los pasos, hemos desarrollado una interfaz gráfica sobre la aplicación por consola y la propia lógica de los algoritmos, funcionando de manera independiente. La interfaz gráfica aporta facilidad de uso y funcionalidad a los usuarios que prefieran este enfoque más visual.

Es posible utilizar la aplicación si necesidad de la interfaz gráfica gracias a la vista por consola con comandos, muy al estilo de UNIX, y gracias a la creación y edición de archivos XML con un formato determinado (ver apéndice cuatro) que representan los parámetros de entrada a los algoritmos.

La interfaz gráfica es traducible a cualquier idioma pensando en posibles usos para alumnos de cualquier nacionalidad. La versión entregada puede ser utilizada en inglés y español. Se puede traducir de manera sencilla a cualquier otro idioma ya que cuenta con ficheros XML de lenguajes.

En la interfaz gráfica se incluye la capacidad de creación de ejercicios por parte del profesor. Estos ejercicios pueden ser enviados a los alumnos para que los intenten realizar en sus ordenadores y comprueben el resultado con el que ha introducido el profesor como solución en el propio ejercicio, es decir, una vez el profesor los ha enviado el ejercicio éste se corrige automáticamente si la necesidad del reenvío de la solución. De esta forma el alumno puede autoevaluarse y aprender de sus errores.

Los resultados de los ejercicios bien o mal realizados se guardan en la base de datos del usuario de forma que el profesor puede controlar si se han dado las respuestas correctamente o no y en cuáles de ellos.

En la aplicación hemos reunido muchos conocimientos aprendidos durante la carrera en multitud de asignaturas que hemos estudiado, en nuestro caso, las pertenecientes a la rama de software, que son:

- → *Ingeniería del Software*: para la planificación de un proyecto de envergadura, diseño de la aplicación, etc.

- → *Estructuras de Datos y de la Información*: tratamiento de las estructuras de datos, en nuestro caso, tablas hash, árboles binarios de búsqueda, listas, etc.
- → *Informática Gráfica*: colocación de los autómatas mediante algoritmos trigonométricos sobre la ventana de la interfaz gráfica.
- → *Laboratorios de programación.*
- → *Programación Orientada a Objetos*: el proyecto está implementado en Java, lenguaje mundialmente conocido como orientado a objetos.
- → *Accesibilidad en la web*: código HTML que hemos aprendido durante la asignatura.
- → *Bases de datos y Sistemas de Información*: todo lo aprendido en la asignatura sobre tratamiento de ficheros (bases de datos) XML.
- → *Teoría de Autómatas y Lenguajes Formales*: todo el conocimiento teórico sobre el que se basa la aplicación se explica en esta asignatura.

TALFi es una aplicación versátil y útil para el alumnado y profesorado de las asignaturas de Teoría de Autómatas en general. Facilitará el estudio de los contenidos y ayudará a las personas interesadas en la comprensión de forma agradable gracias a la interfaz diseñada y la ejecución por pasos de los algoritmos.

Capítulo 10.

Bibliografía

Nuestro director de proyecto es profesor en la temática que trata el proyecto, y la mayor fuente de información procede de las reuniones con él y sus propias notas.
A pesar de ello hemos tenido que consultar alguna bibliografía complementaria para implementar alguno de los algoritmos, de la que dejamos aquí constancia:

- → 1- Hopcraft, Motwani, Uliman.
  "Introducción a la teoría de autómatas. Lenguajes y computación"
  Addison Wesley 2002.
- → 2- Dean Kelley:
  "Teoría de Autómatas y Lenguajes Formales".
  Prentice Hall, 1997.
- → 3- J. G. Brookshear,
  Teoría de la Computación: Lenguajes Formales, Autómatas y Complejidad, Addison-Wesley Iberoamericana, 1993.

Anexos

Anexo 1.

Listado de órdenes de la aplicación por consola

Paralelamente a la aplicación con interfaz gráfica existe una aplicación no gráfica que es la que lleva acabo toda la ejecución de los algoritmos implementados durante el proyecto. Por tanto, debemos dejar constancia de que es posible prescindir de la aplicación por interfaz gráfica y obtener una aplicación no visual con la misma potencia que la gráfica.

La clase que se encarga de tokenizar estas órdenes y obtener la información de ella es el controlador, que recibe las órdenes de la vista gráfica.

Hemos querido que el conjunto de órdenes que se envían a la parte de lógica de la aplicación tengan un formato similar a las órdenes del Shell de Unix, dicho esto, pasamos a dar una breve descripción de los comandos.

Todas las órdenes tienen el siguiente formato: Talfi –orden [-p] rutaxml1 [rutaxml2]

Donde –p indica ejecución por pasos y rutaxml la ruta donde se almacena el autómata(o expresión regular) sobre el que queremos ejecutar el algoritmo.

Los archivos resultantes de cada ejecución se almacenan en un fichero temporal del que el usuario no tiene conocimiento en la aplicación con interfaz gráfica, pero no así en la aplicación por consola, el directorio es: XML/pruebas .

- **Algoritmo de minimización: Talfi –m [-p] rutaxml**

Ejecuta el algoritmo de minimización sobre el autómata almacenado en el archivo cuya ruta sea rutaxml, devuelve el autómata minimizado, si se ha decidido ejecución por pasos:-p, se obtiene además una tabla de equivalencia con toda la información correspondiente.

- **Algoritmo de equivalencia AFD: Talfi –e [-p] rutaxml1 rutaxml2**

Ejecuta el algoritmo de equivalencia sobre los autómatas almacenados en los archivos cuya ruta sea rutaxml1 y rutaxml2, devuelve el resultado de la equivalencia, si se ha decidido ejecución por pasos:-p, se obtiene además una tabla de equivalencia con toda la información correspondiente.

- **Algoritmo de transformación AFN->AFD: Talfi –t2 [-p] rutaxml**

Ejecuta el algoritmo de transformación de AFN->AFD sobre el autómata almacenado en el archivo cuya ruta sea rutaxml, devuelve el autómata transformado, si se ha decidido ejecución por pasos:-p, se obtiene además una tabla de pasos con toda la información correspondiente.

- **Algoritmo de transformación AFNLambda->AFN: Talfi –t1 [-p] rutaxml**

Ejecuta el algoritmo de transformación de AFNLambda->AFN sobre el autómata almacenado en el archivo cuya ruta sea rutaxml, devuelve el autómata transformado, si se ha decidido ejecución por pasos:-p, se obtiene además una tabla de pasos con toda la información correspondiente.

- **Algoritmo de transformación AFD->ER : Talfi –t3 [-p] rutaxml**

Ejecuta el algoritmo de transformación de AFD->ER sobre el autómata almacenado en el archivo cuya ruta sea rutaxml, devuelve la expresión regular resultante de la

transformación, si se ha decidido ejecución por pasos:-p, se obtiene además una tabla de pasos con toda la información correspondiente.

- **Algoritmo de transformación ER->AFNLambda: Talfi –r [-p] rutaxml**

Ejecuta el algoritmo de transformación de ER->AFNLambda sobre la expresión regular almacenada en el archivo cuya ruta sea rutaxml, devuelve el autómata resultante de la transformación, si se ha decidido ejecución por pasos:-p, se obtiene además una tabla de pasos con toda la información correspondiente.

Anexo 2.

WEB Applet de la aplicación

Aprovechando la funcionalidad del lenguaje **Java** y sus posibilidades de ejecución sobre páginas web, gracias la tecnología de la **Máquina Virtual** y los llamados **Applets**, hemos creado la aplicación de forma que mediante este Applet incrustado en una web accesible a través de Internet cualquier alumno puede ejecutar la aplicación sin necesidad de tener el archivo **.jar** en su PC.

Este enfoque mejora varios aspectos que hemos tenido en cuenta para decidirnos a realizar este paso. El primero de ellos es el control que el administrador (como pudiera ser el profesor de la asignatura) podría llevar un control exhaustivo de los perfiles de los usuarios registrados en la aplicación y que, suponemos, están matriculados con dicho profesor en la asignatura. Como hemos reiterado a lo largo de la descripción de la memoria en ningún caso los alumnos podrían acceder a otros perfiles de usuarios, ni modificar los ejercicios que has resulto mal ni bien, es por ello que este aspecto ayuda al profesor a ejercer un control sobre todos los alumnos que puede repercutir en el mejor entendimiento de los problemas a la hora de resolver los ejercicios. Todo ellos va en beneficio de los propios alumnos ya que de esta manera el profesor puede preparar clases de ejercicios o simplemente ejercicios genéricos que ayuden a los alumnos.

El profesor puede añadir también ejercicios a la base de datos de los mismos, así como ejemplos también, de forma que la próxima vez que el alumno entre en la aplicación, se muestren en el árbol de ejemplos y puedan cargarlos sin más que hacer click sobre ellos mediante el ratón.

Construir este Applet no supone demasiado esfuerzo comparado con las ventajas que añade tanto para los usuarios como para el profesor, sin embargo antes de poder ejecutar la aplicación la página nos indica que debemos dar permisos al Applet para que pueda acceder a determinadas características de nuestro Java instalado en el PC. La aplicación en ningún caso modificará ninguna de las propiedades del nuestro Java sino que identificará aspectos como el idioma o el sistema operativo sobre el que se está ejecutando para la correcta ejecución de la aplicación y sin las cuales no podría llevarse a cabo ninguna de las tareas para las que está diseñada.

Anexo 3.

JavaDoc

Todo el código fuente de la aplicación cuenta con comentarios explicativos, aparte de ello hemos realizado toda la documentación JavaDoc que permitirá a futuros desarrolladores reutilizar el código y comprenderlo mejor, ya que adjuntamos con él toda una API de funcionalidad sobre cada una de las clases, métodos y funciones que contiene el código fuente.

Índice del JavaDoc de la aplicación: se encuentra en el CD adjunto a la memoria en la carpeta JavaDoc. El índice se encuentra en la página: index.html.

Anexo 4.

Estructura archivos XML

La aplicación cuenta con numerosos archivos XML que sirven tanto para almacenar los menús, como para almacenar la información de entrada y salida de la aplicación, como plantilla para almacenar los mensajes que devuelve la aplicación, etc. Por regla general existe un fichero XML por cada idioma en el que se puede ejecutar la aplicación.

Todos los archivos XML cumplen el estándar ISO-8859.

Este anexo está orientado a futuros desarrolladores de la aplicación que quieran entender de forma rápida el formato y funcionalidad de los archivos XML.

1.-**Archivos XML relacionados con el menú de la aplicación.**

Estos archivos almacenados en la aplicación como menu_[idioma].xml, en concreto, menu_esp.xml y menu_eng.xml, respectivamente en español e inglés tienen como estructura principal las cadenas de caracteres del menú de la aplicación.

Como por ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<menu>
	<archivo>
		<texto>Archivo</texto>
		<nuevo>Nuevo Auto</nuevo>
		<nueva>Nueva ER</nueva>
		<abrir>Abrir</abrir>
		<guardar>Guardar</guardar>
		<guardarc>Guardar como</guardarc>
		<guardare>Abrir ejercicio</guardare>
		<imprimir>Imprimir</imprimir>
		<salir>Salir</salir>
		<ejercicio>Nuevo ejercicio</ejercicio>
	</archivo>

	....
</menu>
```

2.-**Archivos XML relacionados con los mensajes por pantalla.**

Estos archivos almacenados en la aplicación como mensajes.xml y messages.xml, respectivamente en español e inglés tienen como estructura principal las cadenas de caracteres que devuelve como mensajes de alerta, error e información de la aplicación.

Como por ejemplo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<mensajes>

	…
	<pasos>
		<pregunta>Steps?</pregunta>
		<preguntaG>Do you want to see all the steps?</preguntaG>
		<con>With steps</con>
		<sin>Without steps</sin>

	</pasos>
```

```
      …
</mensajes>
```

3.-**Archivos XML relacionados con los mensajes de las páginas HTML.**

Estos archivos almacenados en la aplicación como trhtml_esp.xml y trhtml_eng.xml, respectivamente en español e inglés tienen como estructura principal las cadenas de caracteres que se usan como mensajes en las páginas HTML generadas automáticamente en la aplicación.

Como por ejemplo:

```
<traductorhtml>
      <minimizacion>
            <title>Minimization algorithm result</title>
            <input>Input</input>
            <output>Output</output>
            <steps>Steps-Minimization table</steps>
      </minimizacion>
      …
</traductorhtml>
```

4.-**Archivos XML relacionados con los autómatas.**

Estos archivos almacenados en la carpeta de ejemplos, aunque pudieran aparecer en cualquier otro lugar, se generan automáticamente mediante la interfaz gráfica y contiene toda la información relacionada con los autómatas finitos, es decir, estados, transiciones, alfabeto y tipo de autómata.

Como por ejemplo:

```
<authomata>
      <type>
            <item>AutomataFD</item>
      </type>
      <alphabet>
            <item>1</item>
            <item>0</item>
      </alphabet>
      <states>
            <state>S0</state>
            <state>S1</state>
      </states>
      <init>
            <state>S0</state>
      </init>
      <finals>
            <state>S0</state>
      </finals>
      <arrows>
            <arrow>
            <state>S0</state>
            <state>S1</state>
            <item>1</item>
      </arrow>
            <arrow>
            <state>S1</state>
            <state>S0</state>
            <item>0</item>
```

```
      </arrow>
            <arrow>
            <state>S0</state>
            <state>S0</state>
            <item>0</item>
      </arrow>
            <arrow>
            <state>S1</state>
            <state>S1</state>
            <item>1</item>
      </arrow>
      </arrows>
</authomata>
```

5.-**Archivos XML relacionados con los ejercicios**

Estos archivos almacenados en la carpeta de ejercicios, aunque pudieran aparecer en cualquier otro lugar, se pueden generan automáticamente mediante la interfaz gráfica y contiene toda la información relacionada con los ejercicios a resolver por los usuarios, es decir, enunciado, entrada y resultado.

Como por ejemplo, el siguiente ejemplo de minimización de autómatas:

```
<ejercicio>
      <tipo>Minimizacion</tipo>
      <enunciado>Minimiza el siguiente automata finito</enunciado>
      <input>
            <authomata>
                  <type>
                        <item>AutomataFD</item>
                  </type>
                  <alphabet>
                        <item>a</item>
                        <item>b</item>
                  </alphabet>
                  <states>
                        <state>S1</state>
                        <state>S2</state>
                        <state>S3</state>
                        <state>S4</state>
                  </states>
                  <init>
                        <state>S1</state>
                  </init>
                  <finals>
                        <state>S4</state>
                  </finals>
                  <arrows>
                        <arrow>
                              <state>S1</state>
                              <state>S2</state>
                              <item>a</item>
                        </arrow>
                        …
                  </arrows>
            </authomata>
      </input>
      <output>
            <authomata>
                  <type>
                        <item>AutomataFD</item>
                  </type>
                  <alphabet>
```

```
                        <item>a</item>
                        <item>b</item>
                    </alphabet>
                    <states>
                        <state>S10</state>
                        <state>S43</state>
                        <state>[S21,S32]</state>
                    </states>
                    <init>
                        <state>S10</state>
                    </init>
                    <finals>
                        <state>S43</state>
                    </finals>
                    <arrows>
                        <arrow>
                            <state>S10</state>
                            <state>[S21,S32]</state>
                            <item>b</item>
                        </arrow>
                        …
                    </arrows>
                </authomata>
            </output>
</ejercicio>
```

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y / o el prototipo desarrollado.

Fdo:

Miguel Ballesteros Martínez:

José Antonio Blanes García:

Samer Nabhan Rodrigo: