GENERACIÓN DE EXPRESIONES REGULARESA PARTIR DEL AUTÓMATA FINITO.

Ing. Abel Pérez Martínez [1], M.Sc. Antonio Fernández Orquín [1]

1. Universidad de Matanzas "Camilo Cienfuegos", Vía Blanca Km.3, Matanzas, Cuba.

CD de Monografías 2009

Resumen.

Las expresiones regulares son una poderosa herramienta para detectar patrones en textos, muchas aplicaciones de Procesamiento de Lenguaje Natural los utilizan para su posterior análisis. La expresión regular es una representación complicada, pues los símbolos utilizados para su construcción generalmente no tienen un significado semejante al lenguaje natural. Para esto se ha elaborado un método que permite construir las expresiones, probarlas hasta ponerlas a punto para su correcta utilización y guardarlas en una biblioteca. Como elemento novedoso está la opción de etiquetar el texto, facilitando la posibilidad de buscar la cadena de macheo y sustituirla por cualquier elemento deseado e insertar etiquetas delante y detrás de la cadena encontrada. Se explica la modificación realizada al algoritmo de convertir a expresión regular. Como resultado más relevante está el hecho de haber obtenido un método y el consecuente recurso informático para la creación de expresiones regulares a partir de su autómata finito.

***Palabras claves**: Expresiones regulares, Autómata finito, Procesamiento de Lenguaje Natural.*

Introducción

Las expresiones regulares son una poderosa herramienta para detectar patrones en texto. Son muy útiles para el desarrollo del Procesamiento de Lenguaje Natural (PLN) porque en el núcleo de muchas aplicaciones de esta rama está la detección de patrones de texto para su posterior análisis y tratamiento. Además tienen múltiples ventajas, por ejemplo:

- Existe soporte para expresiones regulares en gran variedad de lenguajes de programación.
- La mayor parte de la sintaxis de las expresiones regulares trabaja igual en una amplia variedad de lenguajes de programación y herramientas.
- Las expresiones regulares pueden ayudar a escribir código corto y ahorran tiempo. Especialmente en *JavaScript*, donde mantener poco código es importante debido a que no todos tienen una conexión a Internet de alta velocidad. Según Friedl en (Friedl, 1998) incluso para quienes no dominan a la perfección la sintaxis, las expresiones regulares son generalmente la forma más rápida de hacer el trabajo, si se compara con el tiempo que tomaría hacerlo todo desde cero en el leguaje de programación.
- La mayoría de las expresiones regulares funcionan muy rápido en casi todos los casos. Aunque si se tienen en cuenta las bases de optimización de expresiones regulares se logran mejores resultados.
- Las expresiones regulares pueden encontrar prácticamente todo. Con el dominio de las expresiones regulares se pueden encontrar muchos usos apropiados donde los usuarios inexpertos pueden no pensar en usarlas.

Es recomendable para explotar a fondo el potencial de las expresiones regulares utilizar herramientas que permitan probar y corregir errores hasta poner a punto la expresión deseada.

La sintaxis para generar estos patrones es algo difícil de recordar, debido a que los símbolos utilizados para su construcción, generalmente no tienen un significado semejante al que pudieran tener en lenguaje natural o en lenguajes de programación.

Si se toma como ejemplo la siguiente expresión regular:

http(s)?://\w+\.\w+(:\d)?[\w\d\?+-&=]

Cuyo significado es una URL válida, es decir, comienza con *http*, la *(s)* es opcional ***(?)***, seguido de**:** *//*, los símbolos *\w+\.\w+* siguientes significan caracteres alfanuméricos, separados por punto. Opcionalmente *(:\d)?* para los dos puntos y el puerto (dígitos) y al final cualquier carácter para los parámetros de la URL. Se puede observar que es bastante complicada de recordar y generar si no se tienen vastos conocimientos acerca de la sintaxis de las expresiones regulares.

La expresión anterior podría detectar como validas cadenas como:

http://anubis.umcc.cu, *http://10.34.64.4*, así como *https://sigenu.umcc.cu:8080*

Y muchas más. Todo esto se logra con una sola línea de código, sin embargo programar el algoritmo para detectar el patrón equivalente a la expresión antes descrita no es menos complicado que recordar y generar la expresión, algunas ideas para esta solución pudieran ser:

- Preguntar si comienza con *http* ó *https*.
- Seguido de **://**
- Buscar caracteres alfanuméricos hasta encontrar un / ó :
    - Si aparece / ➔ buscar más caracteres alfanuméricos
    - Si aparece : ➔ buscar dígitos
        - Seguido de /
        - buscar más caracteres alfanuméricos

Hasta aquí se puede apreciar el arduo trabajo de codificación que supone hacerlo desde cero. Por lo que usar la expresión sigue siendo la mejor opción. No obstante las expresiones regulares pueden llegar a crecer hasta hacerse casi incomprensibles, por ejemplo: la expresión regular para validar una dirección IP con bastante precisión es:

(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)

Donde son válidas las cadenas que representan direcciones IP desde la 0.0.0.0 hasta la 255.255.255.255

Se han consultado numerosas herramientas que manipulan expresiones regulares y la mayoría no ofrece una interfaz cómoda para la creación expresiones regulares, sólo un área de texto para teclearla y probar las coincidencias, algunas ofrecen a través de un menú de opciones las diferentes partes de la expresión, para que el usuario las vaya armando y probando. Ninguna de las herramientas consultadas obtiene la expresión regular a partir de su representación gráfica, el autómata finito, elemento que sería de gran utilidad en la tarea de reducir los esfuerzo y mejora de la interpretación y creación de expresiones regulares.

La mayoría de las aplicaciones de PLN, como analizadores léxico, sintáctico o semántico, ofrecen módulos para tokenizado[1], etiquetado o separación en frases, etc. donde pueden ser aplicadas las expresiones regulares para agilizar estos procesos. Existen muchas herramientas que desempeñan estas tareas, pero las realizan todas juntas e internamente, es decir, sólo ofrecen el resultado final comportándose como una caja negra. Si para un fin determinado sólo se requiere uno de estos módulos, el investigador generalmente tiene que programar estas tareas.

También es común en trabajos de PLN, que parte de un patrón que se quiere encontrar en un texto, esté almacenado en un diccionario, es decir, en un fichero de texto plano donde en cada línea aparece una palabra, en estos casos hay que especificar todas las palabras del diccionario en el código de la expresión regular, esto hace que sea muy larga y por lo tanto difícil de crear y mantener.

La elaboración de una herramienta que asista en la creación y puesta a punto de expresiones regulares, podría contribuir a que un mayor número de personas hagan uso de ellas, además facilitaría el trabajo de aquellas que tienen dominio de ellas, pues se requerirá menor esfuerzo y dedicación para una mejor utilización.

El Objetivo general a cumplir durante el transcurso de la investigación ha sido: Desarrollar una herramienta informática con recursos para facilitar la creación, mantenimiento y prueba hasta puesta a punto de expresiones regulares.

En el epígrafe Materiales y métodos se pueden ver un resumen de los conceptos preliminares relacionados con la teoría de autómatas y lenguajes. Se resumen los elementos fundamentales acerca de los autómatas, expresiones regulares y algoritmo de conversión. En el apartado resultados y experimentación se recogen los requisitos que debe reunir una aplicación de este tipo, se describe las funcionalidades y a través de un esquema general se explica el funcionamiento de la aplicación desarrollada, así como la modificación realizada a la teoría para convertir de autómata a expresión regular. En la sección Pruebas y experimentos se describen los experimentos realizados. Finalmente en Análisis de los resultados se comentan como los experimentos realizados demuestran la hipótesis departida.

[1] Tokenizar se le llama a la acción de convertir un texto en una lista de palabras en las que cada una de ellas constituye un token o átomo. De esta forma se tendrá una palabra por línea.

Trabajos relacionados

Obtener y probar expresiones regulares es una tarea difícil, muchos investigadores se han planteado resolver esto mediante herramientas que faciliten el trabajo con las expresiones regulares, algunos sólo se limitan a exponer una interfaz gráfica donde se introduce la expresión regular y un texto de prueba, y se observan resaltadas con colores las coincidencias; otros van más allá y permiten almacenar dichas expresiones en bibliotecas para tenerlas disponibles. Mediante una búsqueda bibliográfica se han encontrado varias de estas herramientas que se exponen a continuación. Para el análisis se ha dividido el estudio en dos grupos: aplicaciones de escritorio y aplicaciones Web.

Aplicaciones de escritorio:

Visual RegExp v3.1(Riesterer, 2006) Funciona con el motor de expresiones regulares Perl y compatibles. No implementa opción de exportar e importar desde librerías, sólo el texto de la expresión regular desde un fichero de texto. No existe interfaz para diseñar expresiones regulares, hay que teclearla. Se distribuye bajo licencia GPL, es *open source* (código abierto, se refiere a que se distribuye también su código fuente) y está disponible para *Windows* y *Linux*.

Expresso v3.0(inc., 2007) Funciona con el motor de expresiones regulares de .NET. Implementa la opción de exportar e importar desde librerías, en dicha librería no se agrupan según ningún criterio, solo se almacenan las expresiones regulares y se pueden adicionar las expresiones que se están probando. Presenta interfaz para diseñar expresiones regulares mediante un menú con los componentes léxicos de una expresión regular, se escogen a conveniencia del usuario y se insertan en el área de edición de la expresión regular. Es software privativo, se realizó el análisis con una versión de evaluación que expiró su período gratis.

RegexDesigner v1.0.2727 (Sells, y otros, 2002) Utiliza el motor de expresiones regulares de.NET. No implementa opción de exportar e importar desde librerías, sólo guarda en proyecto (*.rep). No existe interfaz para diseñar expresiones regulares, hay que teclearla. Presenta Interfaz para probar expresiones regulares donde se pueden ver coloreadas las coincidencias. Incluye interfaz para remplazar y cortar el texto pero no para etiquetarlo. Genera ensamblados compilados .NET con la expresión encapsulada para ser utilizada por el programador. Además genera código C# de cómo utilizar la expresión que se está probando, listo para utilizarlo. Es gratis y *open source*.

RegexBuddy3 (Goyvaerts, 2007) Funciona con múltiples motores de expresiones regulares, incluso uno propio (JGsoft). Posee opción para exportar e importar desde librerías, cada entrada de la librería se almacena código, descripción y ejemplos de coincidencias. Para diseñar expresiones regulares utiliza un *TreeView* para la estructura jerárquica, se escogen las diferentes partes de las expresiones regulares y se colocan en su debido lugar, se puede arrastrar y colocar para reordenar su posición, al mismo tiempo se va observando la expresión generada. Es software privativo, se realizó el análisis con una versión de evaluación que tiene funcionalidades limitadas.

Aplicaciones sobre Web:

Regextester (2007) Utiliza el motor de expresiones regulares de .NET para procesar del lado servidor y el de *JavaScript* para trabajar en el cliente. No existe opción de exportar e importar desde librerías ni interfaz para diseñar expresiones regulares. Se pueden probar expresiones regulares, no así para remplazar y cortar el texto. Se puede cargar el texto desde fichero en el cliente.

Tester de Expresiones Regulares (metriplica.com, 2008) Utiliza el motor de expresiones regulares de *JavaScript*. No tiene opción de exportar e importar desde librerías. No se pueden diseñar expresiones regulares. Se pueden probar expresiones regulares y remplazar el texto.

RegexPal (Levithan, 2008) Hecho para el motor de expresiones regulares de *JavaScript*. No implementa la opción de exportar e importar desde librerías. No existe interfaz para diseñar expresiones regulares. Se pueden probar expresiones regulares. No posee interfaz para remplazar y cortar el texto.

re*Work* (2006) Hecho completamente en *JavaScript*, por lo que usa el motor de expresiones regulares de dicho lenguaje. No existe la opción de exportar e importar desde librerías. No se pueden diseñar expresiones regulares. Posee una interfaz para probar expresiones regulares. Se pueden remplazar y cortar el texto y además se pueden ver el grafo de la expresión y las coincidencias paso a paso. Adicionalmente tiene una opción para ver el código necesario para usar expresiones regulares en los lenguajes: *JavaScript*, *PHP*, *Python* y *Ruby*.

De las herramientas analizadas anteriormente se puede decir que no se ha encontrado, por el autor de esta investigación, evidencia de la existencia de software que genere expresiones regulares a partir de su representación gráfica, el autómata finito. Se han encontrado otros tipos de representación para obtener la expresión regular en algunos, entre los que se puede destacar el uso de un menú para escoger los componentes léxicos de la expresión regular e incluirlos en el área de trabajo y el uso de un *TreeView* para darle estructura jerárquica y armar por partes la expresión regular.

Materiales y métodos

Para resolver el problema planteado ha sido necesario basarse en las definiciones matemáticas de autómata finito y expresión regular, pero antes es necesario dejar claras otras definiciones que se han considerado necesarias para una mejor compresión como son: símbolo, alfabeto, palabra, lenguaje, lenguaje regular y máquina de estados finitos. Todas estas definiciones han sido extraídas el libro Autómatas y Lenguajes (Brena, 2003).

Conceptos preliminares

La noción más primitiva es la de símbolo, que es una representación distinguible de cualquier información. Los símbolos pueden ser cualesquiera, como w, 9, #, etc. Un símbolo es una entidad indivisible.

Un alfabeto es un conjunto no vacío de símbolos. Así, el alfabeto del idioma español, E = {a, b, c, . . . , z}, es sólo uno de tantos alfabetos posibles. Generalmente se utiliza la notación $\sum$ para representar un alfabeto.

Con los símbolos de un alfabeto es posible formar secuencias o cadenas de caracteres, tales como mxzxptlk, balks, r, etc. Las cadenas de caracteres son conocidas también como palabras. Un caso particular de cadena es la palabra vacía, ε, la cual no tiene ninguna letra.

La longitud de una palabra es la cantidad de letras que la conforma, incluyendo las repeticiones, se denota por |w| para una palabra w. Por ejemplo, |perro| es 5.

La concatenación de palabras da origen a nuevas palabras cuya longitud es la suma de las longitudes de las palabras originales Por ejemplo, si w = abra y v = cada, entonces wvbra es la palabra abracadabra. Esta operación es asociativa, pero no conmutativa.

Se dice que una palabra v es subcadena de otra w cuando existen cadenas x, y -posiblemente vacías- tales que xvy = w. Por ejemplo, "abra" es subcadena de "abracadabra", y ε es subcadena de toda palabra.

Un lenguaje es simplemente un conjunto de palabras. Así, {abracadabra} es un lenguaje (de una sola palabra), {ali, baba, y, sus, cuarenta, ladrones} es otro, etc. Dado que los lenguajes son conjuntos, se pueden efectuar con ellos todas las operaciones de los conjuntos (unión, intersección, diferencia) además la operación de concatenación de lenguajes, escrita como L1 • L2, como una extensión de la concatenación de palabras: L1 • L2 = {w|w = xy, x $\in$ L1, y $\in$ L2}.

Otra operación es la llamada "estrella de Kleene" o "cerradura de Kleene", en honor al matemático norteamericano S. C. Kleene, quien la propuso: Si L es un lenguaje, L*, llamado "estrella de Kleene" de L, es el más pequeño conjunto que contiene:

- La palabra vacía, ε
- El conjunto L
- Todas las palabras formadas por la concatenación de miembros de L*.

Los lenguajes regulares deben su nombre a que sus palabras contienen "regularidades" o repeticiones de los mismos símbolos y palabras.

Definición**:** Un lenguaje L es regular si y sólo si se cumple al menos una de las condiciones siguientes:

- L es finito.
- L es la unión o la concatenación de otros lenguajes regulares R1 y R2, L = R1 U R2 ó L = R1R2 respectivamente.
- L es la estrella de Kleene de algún lenguaje regular, L = R*.

Teorema de Kleene: Un lenguaje es regular si y sólo si es aceptado por algún autómata finito.(Brena, 2003)

Autómatas finitos.

Una máquina de estados finitos puede ser visualizada como un dispositivo con los siguientes componentes:

- Una cinta de entrada
- Una cabeza de lectura
- Un control

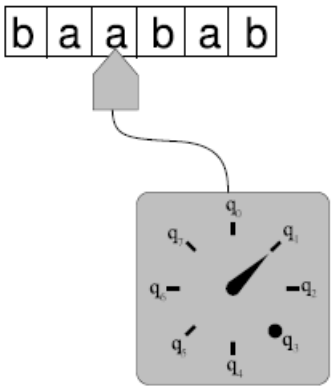

Figura 1 Representación general de una máquina de estados finitos.

La cabeza lectora se coloca en los segmentos de cinta que contienen los caracteres que componen la palabra de entrada, y al colocarse sobre un carácter lo "lee" (apunta al carácter que se encuentra inmediatamente a su derecha) y manda esta información al control (indicado por una carátula de reloj en la Figura 1), donde se determina cuál es la nueva posición de la "aguja". Se supone que existe forma de saber cuándo se acaba la entrada (por ejemplo, al llegar a una posición vacía en la cinta de entrada). La "aguja" del control puede estar cambiando de posición, y hay algunas posiciones llamadas finales (como la indicada por un punto, q3) que son consideradas especiales, porque permiten determinar si una palabra es aceptada o rechazada.(Brena, 2003)

Según Brena en (Brena, 2003), un autómata finito es un modelo matemático de un sistema que recibe una cadena constituida por símbolos de un alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce.

La definición formal que aparece a continuación es la que da Hopcroft en (Hopcroft, 2001), que ha sido traducida del inglés.

Definición: Un autómata finito A es un quíntuplo $(Q, \sum, \delta, s, F)$, donde:

- Q es un conjunto de estados.
- $\sum$ es el alfabeto de entrada.
- $s \in K$ es el estado inicial.
- $F \subseteq Q$ es un conjunto de estados finales.
- $\delta: K \times \sum \rightarrow Q$ es la función de transición, que a partir de un estado y un símbolo del alfabeto obtiene un nuevo estado.

La función de transición indica a qué estado se va a pasar sabiendo cuál es el estado actual y el símbolo que se está leyendo.

Si partiendo de un estado y un símbolo del alfabeto hay un y sólo un estado siguiente, entonces el autómata es determinista (AFD).

Si de cada estado parten varias transiciones con el mismo símbolo del alfabeto, es decir, que para un símbolo hay más de una transición posible al siguiente estado, entonces el autómata es no determinista (AFN).

Si existen transiciones vacías (o transiciones ε), es decir, que se permite cambiar de estado sin procesar ningún símbolo de la entrada, entonces el autómata es no determinista con transiciones vacías (AFN-ε). Cuando el autómata llega a un estado, se encuentra en ese estado y en los estados a los que apunte éste mediante una transición ε.

Dado que los AFN tienen menos restricciones que los AFD, resulta que los AFD son un caso particular de los AFN, por lo que todo AFD es de hecho un AFN. Para todo AFN-ε existe un AFN equivalente y para todo AFN existe un AFD equivalente. Existen algoritmos para transformar un autómata en otro. Los AFD son los más sencillos de construir, por tanto, puede ser útil diseñar un autómata complejo como AFN-ε o AFN para luego transformarlo en AFD para su implementación.

Formas de representar un autómata finito.

Además de notar un autómata finito a través de su definición formal es posible representarlo a través de otras notaciones que resultan más cómodas. Las más usuales son:

Las Tablas de Transiciones: donde se muestran por las filas los estados del autómata finito y por las columnas los símbolos del alfabeto y en las intersecciones de fila y columna se muestra el próximo estado, la Tabla 1 muestra un ejemplo de tabla de transiciones de un autómata finito determinista con dos estados (S1 y S2) y dos símbolos de entrada (0 y 1), donde se puede observar que a partir de cada estado y símbolo del alfabeto se puede pasar a un próximo estado. Tomado de Wikepedia (2008).

Tabla 1 Ejemplo de una tabla de transiciones de un autómata finito determinista.

	0	**1**

S_1	S_2	S_1
S_2	S_1	S_2

Los Diagramas de Transiciones: puede decirse que es la representación más comprensible para los humanos, dado que es una forma completamente visual donde se pueden observar los estados como círculos, el estado inicial se muestra marcado con una flecha que entra, el estado final se distingue por un doble círculo, pueden coincidir estados inicial y final como el de la Figura 2, que muestra el diagrama equivalente a la Tabla 1. Las transiciones se representan por flechas del estado de donde parten al estado que llegan acompañadas de una etiqueta que indica que símbolo del alfabeto es necesario para que se realice dicha transición. Figura tomada de Wikepedia (2008).

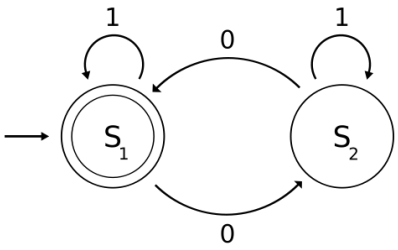

Figura 2 Ejemplo de un diagrama de transición de estados de un autómata finito determinista.

Las Expresiones Regulares como representación del autómata finito: esta representación es mediante una palabra que define el lenguaje regular aceptado por dicho autómata. Se demuestra que dado un autómata de estados finitos, existe una expresión regular que lo representa.

Expresiones Regulares

La notación de conjuntos permite describir los lenguajes regulares, pero se quisiera una notación en la que las representaciones de los lenguajes fueran simplemente texto (cadenas de caracteres). Así las representaciones de los lenguajes regulares serían palabras de un lenguaje (el de las representaciones correctamente formadas).

Las siguientes definiciones son tomadas del libro Autómatas y Lenguajes (Brena, 2003), donde se ha modificado la terminología para una mejor comprensión. El símbolo λ (expresión regular vacía) en la bibliografía aparece como ^ y este último tiene un significado especial en la mayoría de los motores de expresiones regulares.

Definición: Sea $\sum$ un alfabeto. El conjunto ER de las expresiones regulares sobre $\sum$ contiene las cadenas en el alfabeto $\sum$ U {λ, |, •, *, (,), Ø} que cumplen con lo siguiente:

- λ y Ø ER
- Si σ ∈ $\sum$, entonces σ ∈ $\sum$ ER.
- Si E1, E2 ∈ ER, entonces (E1 | E2) ∈ER, (E1 • E2) ∈ER, (E1)* ∈ER.

Definición: El significado de una expresión regular es una función L: ER→2^{Σ^*} (una función que toma como entrada una expresión regular y entrega como salida un lenguaje), definida de la manera siguiente:

$L(\emptyset) = \emptyset$ (el conjunto vacío)

$L(\lambda) = \{ \varepsilon \}$ (la palabra vacía)

$L(\sigma) = \{ \sigma \}, \sigma \in \sum$

$L((R \bullet S)) = L(R)L(S), \ R,S \in ER$

$L((R \mid S)) = L(R) \cup L(S), R, \ S \in ER$

$L((R)^*) = L(R)^*, \ R \in ER$

Según Brena en (Brena, 2003) a la hora de buscar una expresión regular que satisfaga un lenguaje regular particular es necesario tener en cuenta que debe cumplir dos características:

Corrección: La expresión regular propuesta debe representar solamente las palabras que satisfagan la condición, es decir, no deben representar palabras que no pertenezcan al lenguaje regular.

Completez: La expresión regular propuesta debe representar todas las palabras que satisfagan la condición, es decir, no debe haber palabras que pertenezcan al lenguaje que no sean representadas por dicha expresión regular.

Algoritmo de conversión de autómata finito a expresión regular.

El algoritmo que se describe a continuación es resultado de una compilación entre el descrito por Brena en (Brena, 2003) y la traducción al español del que se explica en (Hopcroft, 2001) por Hopcroft. Se ha modificado la terminología unificando el término de estado porque en algunos casos se referían a este como nodo.

Para transformar un autómata finito en una expresión regular equivalente, un procedimiento para hacerlo consiste en ir eliminando gradualmente estados del autómata, hasta que únicamente queden un estado inicial y un estado final. Comprende los siguientes pasos:

El primer paso en este procedimiento consiste en añadir dos estados nuevos al autómata: un nuevo estado inicial i de modo que el antiguo estado inicial q0 deje de ser inicial y un nuevo estado final f de manera que los antiguos estados finales $q_i \in F$ dejen de ser finales; además se añade una transición vacía del nuevo estado inicial al antiguo, $(i, \varepsilon, q0)$, y varias transiciones de los antiguos estados finales al nuevo: $\{(q_i, \varepsilon, f) | q_i \in F\}$. Esta transformación tiene por objeto que haya un estado inicial al que no llegue ninguna transición, y un solo estado final, del que no salga ninguna transición. Esta condición se requiere para llevar a cabo el siguiente paso.

El segundo paso consiste en eliminar estados intermedios, se llama estado intermedio a aquel que se encuentra en una trayectoria entre el estado inicial y el final. El procedimiento de eliminación de estados intermedios es directo. El objetivo es que al suprimir el estado en cuestión, no se alteren las cadenas que hay que consumir para pasar de uno a otro de los estados vecinos, es decir, al suprimir dicho estado, se deben reemplazar las transiciones que antes tomaban ese estado como punto intermedio para ir de un estado vecino a otro, por otras transiciones que vayan del estado vecino origen al estado vecino destino, pero ahora sin pasar por el estado eliminado. En la Figura 3 se observa como para pasar de p1 a qm y de pn a q1 se toma como estado intermedio a q. Así al eliminar el estado q se verán afectados los caminos p1-q-q1, p1-q-qm, pn-q-q1, pn-1-qm. En la Figura 4 se muestran los caminos ya sin el estado q donde por ejemplo: para pasar de p1 a q1 se conserva $\alpha_1 (\beta_1+\ldots+\beta_k)^*\gamma_1$

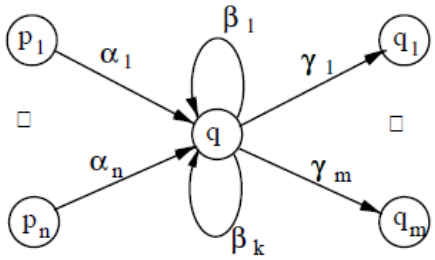

Figura 3 Fragmento de autómata antes de eliminar el estado q.

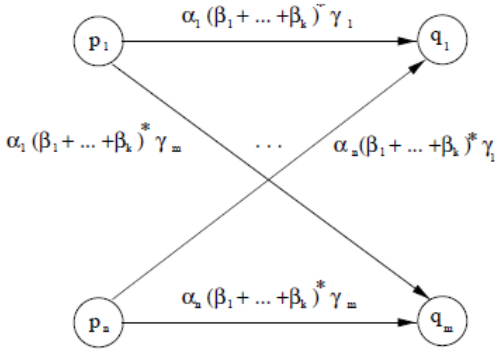

Figura 4 Fragmento de autómata finito después de haber eliminado el estado q.

Resultados y experimentación

Requisitos a implementar.

En todo proceso de construcción de software el paso primordial es el levantamiento de los requisitos dado que estos darán origen a las funcionalidades principales a ser implementadas en el software. En el caso de ERAFPLN (nombre que viene dado por **E**xpresión **R**egular **A**utómata **F**inito y **PLN**) dichos requisitos son los siguientes:

- Almacenar de forma clasificada por categorías las expresiones regulares utilizadas en anteriormente.

- Permitir introducir nuevas expresiones regulares.
- Ofrecer una interfaz para probar expresiones regulares.
- Obtener la expresión regular a partir de la representación del autómata finito.
- Ofrecer opciones específicas con recursos para PLN.

Descripción de la aplicación

ERAFPLN es un software para asistir la creación y pruebas de expresiones regulares. Permite almacenar expresiones regulares en una base de datos interna del sistema (en lo adelante se llamará biblioteca), separadas por categorías, para ayudar a su organización ya que para un mismo propósito puede haber múltiples expresiones. Para cada expresión regular se almacena además una descripción y un texto de prueba. Esto contribuye a la reutilización de expresiones regulares que hayan sido utilizadas en otras aplicaciones porque al tenerlas juntas se puede antes de intentar crear una nueva, verificar si ya existe en la biblioteca. El usuario es libre de insertar y eliminar expresiones y categorías de dicha biblioteca, además se puede exportar la biblioteca para ser utilizada en otro lugar que exista el sistema si se desea. Se pueden probar las expresiones regulares mostrando las subcadenas del texto de prueba que coinciden con la expresión actual en el área de prueba, donde dichas subcadenas toman color de fondo amarillo y cian alternadamente; si la expresión contiene errores de sintaxis se informa debidamente con opción para ver detalles del error.

La herramienta cuenta con un editor gráfico para dibujar autómatas finitos de forma muy sencilla e intuitiva. Mediante una barra de herramientas se escogen los elementos que componen dicho autómata (estados y enlaces), se cuenta con gran variedad de enlaces predeterminados como para el caso de alfanuméricos, dígitos, separadores, etc. Dichos elementos se pueden mover libremente en el área de dibujo así como ser eliminados o cambiar su valor. Se cuenta con un tipo de enlace que es muy útil para los usuarios que desarrollan PLN, dado que con frecuencia se accede a diccionarios (ficheros de texto donde generalmente en cada línea aparece una palabra), en lugar de tener que teclear todas las opciones posibles se escoge el diccionario y el sistema construye el enlace con todas las palabras que contiene.

Una vez encontrada la expresión que satisface la necesidad del usuario, se pueden hacer remplazos sobre las subcadenas que coinciden con la expresión, esto es muy útil también para PLN puesto que frecuentemente se realizan limpiezas a determinados textos, dicha limpieza consiste en eliminar toda coincidencia de la expresión regular dada; con la utilidad de remplazar de la herramienta, dejando en blanco el cuadro de texto donde se escribe el texto a remplazar se logra esta limpieza. Otra tarea muy habitual en el PLN es el etiquetado, ERAFPLN cuenta una utilidad de etiquetado donde se entran las etiquetas de apertura y cierre que envolverán a las subcadenas encontradas y se realiza el etiquetado teniendo en cuenta que si un texto ya lo está no volver a hacerlo.

Esquema general de la aplicación.

En correspondencia con lo anterior ahora se verá como está compuesta la aplicación. La Figura 5 muestra un esquema donde se pueden observar las diferentes partes que la componen. El lugar más importante lo ocupa la obtención de la expresión regular, para ello se ha implementado un editor gráfico que permite dibujar el autómata finito, que además es el encargado de almacenar dicho autómata en una estructura comprensible para el sistema y así poder aplicar los pasos del algoritmo de conversión hacia la expresión regular. Una vez obtenida la expresión se pude pasar a la zona de pruebas, donde se visualizan los resultados de coincidencias sobre el texto de prueba y es ahí donde se pueden aplicar las opciones relacionadas con PLN: remplazo y etiquetado. El otro componente es la biblioteca donde se almacena la información de las expresiones regulares

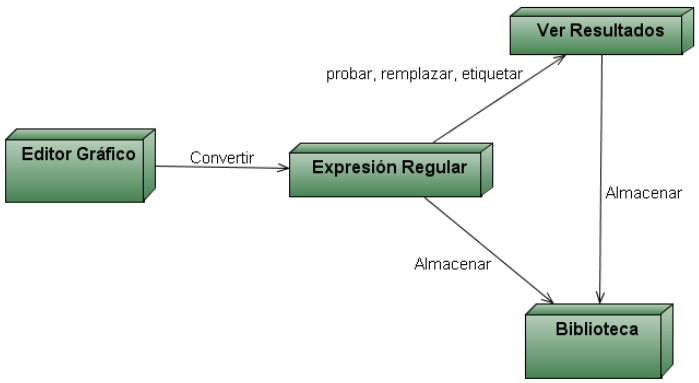

Figura 5 Esquema general de la aplicación.

Modificación a la teoría de convertir de autómata a expresión regular

En el capítulo 1 se explica cómo convertir de autómata finito a expresión regular, la expresión regular resultante contiene los símbolos del alfabeto, los operadores asterisco **(*)** y barra vertical **(|)** y los paréntesis para agrupar subexpresiones. El operador de concatenación está implícito cuando los elementos léxicos aparecen uno a continuación del otro. En la práctica los motores de expresiones regulares aceptan muchos más operadores, que pueden variar en dependencia de la librería que se esté usando. En este caso se ha considerado que la expresión regular como resultado de aplicar la conversión desde el autómata tenga además de los citados operadores, el operador +. Su significado es invariante entre las diferentes implementaciones y es el cuantificador de repetición de una o más veces, por lo que el fragmento de expresión regular afectado por el +, debe aparecer al menos una vez en el texto. Esto contribuiría a disminuir la longitud de la cadena que da como resultado, siendo más compacta.

Con el operador + se cumple que:

(X)+ = (X)(X)* = (X)*(X)

La Figura 7 muestra el algoritmo de conversión de autómata finito a expresión regular por el método de eliminación de estados, donde se ha desglosado en partes el paso de eliminar cada estado. Entre los pasos que implica eliminar un estado se ha destacado con color rojo en el que se han hecho modificaciones para obtener el operador +. Cuando se da un caso como el de la

Figura 6, es decir, que hay un enlace y un autoenlace consecutivos, se puede observar que si se va a eliminar el estado q1, los enlaces etiquetados como dígitos cumplen esta condición. Teniendo en cuenta esto, en el mencionado paso, se ha programado se la siguiente manera:

Para cada enlace e1 que llega y e2 que parte del estado a eliminar e, hacer:

- Si existe un autoenlace sobre e
- y
- El final de la etiqueta de e1 coincide con la etiqueta del autoenlace, entonces hacer:
    - Temporal e3 = la parte que no coincide de e1 concatenado con la etiqueta del autoenlace y el signo +.
    - Nuevo enlace e4 = e3 concatenado con e2

De esta forma en el ejemplo de la

Figura 6 se obtiene como resultado (dígitos)+(letras minúsculas) considerando que los motores de expresiones regulares abrevian dígitos como \d y las letras minúsculas comprenden el intervalo [a-z] la expresión regular que representa este autómata quedaría (\d)+[a-z].

Como los motores de expresiones regulares utilizan para evaluar internamente un AFN, cuando aparece una unión **(|),** el evaluador trata de tomar la vía más corta para llegar a un estado final. Como resultado, si una de las subexpresiones que separa la barra es subcadena de la otra, entonces la subexpresión más larga debe quedar la izquierda del operador de unión. De no ser así, el evaluador no detectará las variantes más cortas, siendo esto un gran problema, porque no se cumpliría la completez de la solución, como se plantea en anteriormente Por este motivo se ha modificado el proceso de generación teniendo en cuenta la longitud de ambos operandos, para así colocar siempre el más largo a la izquierda

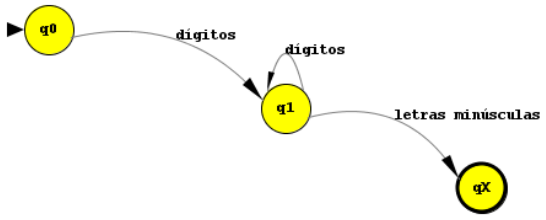

Figura 6 Autómata de ejemplo para obtener un operador +.

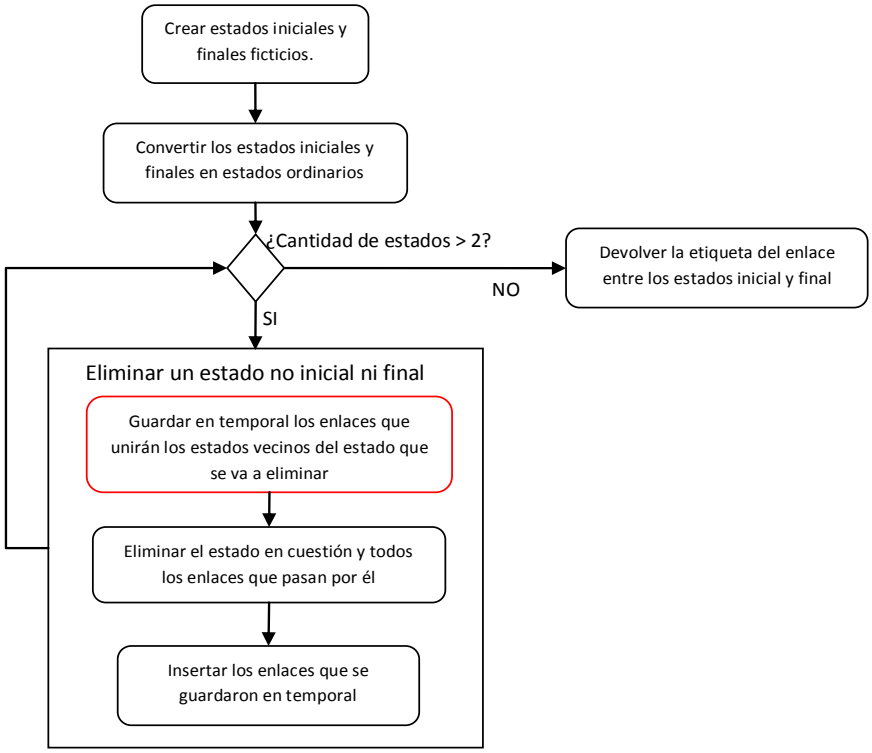

Figura 7 Diagrama de bloques del algoritmo de conversión de autómata finito a expresión regular.

Especialización de las funcionalidades para uso de PLN

Como se mencionaba al inicio del capítulo, se ha implementado un tipo de enlace especial que toma valores de un diccionario. La Figura 8 muestra el diálogo que aparece cuando se selecciona esta opción de enlace. Se le especifica un valor para etiqueta visible que se mostrará en el editor gráfico como etiqueta para dicho enlace. Con el botón […] se escoge mediante un diálogo de seleccionar fichero y se muestra más abajo el contenido. Con esto se evita tener que entrar una por una las opciones posibles en caso de alternativa múltiple, como por ejemplo: artículos, días de la semana, meses del año, entre otras. En trabajos de PLN, el uso de estos diccionarios es bastante frecuente, por lo que esta opción contribuye a esta rama de la investigación.

La aplicación contiene un menú de utilidades, que se ha dejado abierto el diseño para futuras opciones más especializadas en el tratamiento de textos. Se han implementado dos opciones, la primera es el remplazo, que es fundamental para el trabajo con expresiones regulares. La segunda como un caso particular del remplazo, es el etiquetado de textos, hay un diálogo en el que se le pide al usuario el texto antes y después de las coincidencias (ver Figura 9).

Figura 8 Diálogo para escoger el diccionario que dará valores a un enlace que representan los meses.

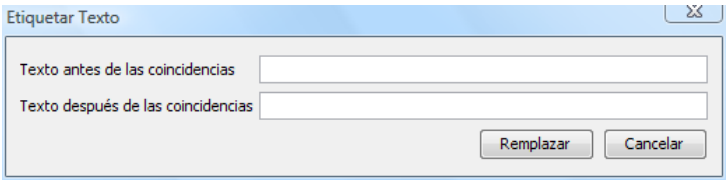

Figura 9 Diálogo de la utilidad de etiquetar texto.

Se considera etiquetado al proceso de insertar etiquetas (cadenas de texto que muchos casos toman la estructura de *tags* de XML (*Extensible Markup Language*), ejemplo: <etiqueta_antes>texto encontrado</etiqueta_después>) antes y después de cada coincidencia de un patrón dado. Esto puede resolverse con un simple *replace* que aporta cualquier motor de expresiones regulares, pero trae como consecuencia que si se aplica más de una vez sobre el mismo texto, se multiplican las etiquetas. Para resolver se ha diseñado el siguiente algoritmo:

1. Aplicar la expresión regular actual al texto.
2. Inicializar resultado como ε (cadena vacía) y ultima_cuenta en 0.
3. Para cada una de las coincidencias m, hacer:
    - Temporal_anterior = cadena_comprendida_entre ultima_cuenta y posición_comienzo_patrón
    - Temporal_despues = cadena_comprendida_entre posición_fin_patrón y fin_del_texto
    - Considerar ya etiquetado si:
        - Temporal_anterior termina con etiqueta_antes
        - y
        - Temporal_despues termina con etiqueta_después
    - Si no está etiquetado, entonces:
        - Añadir al resultado etiqueta_antes + patron_encontrado + etiqueta_después
    - Si ya lo está, entonces:
        - Añadir al resultado solo patron_encontrado
    - Hacer ultima_cuenta = posición_fin_patrón
4. Remplazar el texto anterior por lo que quedó acumulado en resultado.

Pruebas y experimentos

Como parte del inexorable proceso de pruebas por el que debe pasar todo software en sus fases finales antes de ser liberado como versión definitiva, ERAFPLN fue sometido a ser usado por dos grupos de estudiantes de tercer año de la facultad de informática, de 30 estudiantes cada uno. Se les pidió que durante el transcurso de una actividad de laboratorio resolvieran, mediante el software, tres ejercicios propuestos.

La resolución de los ejercicios consistió en el planteamiento de un problema donde había que encontrar el autómata finito determinista y la expresión regular que representara el lenguaje regular descrito en el texto del problema. Con la ayuda del software podrían comprobar, en tiempo real, si realmente su propuesta satisfacía o no las condiciones del ejercicio.

Pruebas con estudiantes

Los ejercicios que se tomaron para la prueba pedían hacer los siguientes patrones:

Experimento 1

1. Dirección de correo electrónico (sencilla, sin tener en cuenta la validez de los nombres de dominio ni la cantidad de estos).

   Solución: *\w+@\w+(\.\w+)+*

2. Cantidad numérica que representa precios de artículos (con el signo $ incluido y la coma como separador de cifras decimales).

   Solución: *((\$\d(\d)*[.,]\d(\d)*)|(\$\d(\d)*))*

3. Fecha con el formato día/mes/año (donde el mes puede ser expresado con su nombre o con número).

Solución:

((\d\d/)|(\d/))((((\d\d/\d)|(\d/\d))\d\d\d|((\d\d/\d)|(\d/\d))\d)|(enero|febrero|marzo|abril|mayo|junio|julio|agosto|septiembre|octubre|noviembre|diciembre)((/\d\d\d\d)|(/\d\d)))

A través de la observación por el investigador se midieron tres parámetros: desenvolvimiento con la interfaz gráfica, habilidad de creación utilizando la herramienta para la confección del autómata finito, así como pruebas y puesta a punto de expresiones regulares. No se enfocó en la creación de expresiones regulares porque el sistema es capaz de obtenerlas a partir de su autómata, sólo en ajustar algún detalle o modificación después de ser generada por el sistema.

Los resultados arrojados fueron los siguientes:

- En el desenvolvimiento con la interfaz gráfica en general.

- El 92% logró moverse con gran facilidad.
- El 5% logró moverse realizando algunas preguntas.
- El restante 3% tuvo dificultades para moverse.

En la habilidad de creación utilizando la herramienta para la confección del autómata finito. A continuación la Tabla 2 resume los resultados de este parámetro.

Tabla 2 Resultados del uso de la aplicación en los 3 ejercicios

	Trabajaron sin problemas	Tuvieron algunos problemas	No pudieron resolver el ejercicio
Ejercicio 1	83%	11%	6%
Ejercicio 2	79%	7%	4%
Ejercicio 3	72%	8%	9%

En las pruebas y puesta a punto de expresiones regulares.

- El 93% logró moverse con gran facilidad.
- El 3% logró moverse realizando algunas preguntas.
- El restante 4% tuvo dificultades para moverse.

De las cifras anteriores y en particular de la Tabla 2 (donde se muestran los resultados más importantes obtenidos del experimento anterior), se puede considerar que aunque la mayoría de los estudiantes pudieron utilizar a plenitud la herramienta, hubo algunos estudiantes que tuvieron problemas a la hora de utilizarla.

Se observaron las dificultades que presentaron a la hora de manipular el editor gráfico y de estas surgieron modificaciones que contribuyeron a mejorar la interacción con la aplicación. También se detectaron algunos *bugs*[2] en la generación de la expresión regular, que fueron reparados.

Ejemplo de modificaciones que se hicieron es que al seleccionar un enlace muchos estudiantes daban clic encima de la etiqueta del enlace, en lugar de dar clic encima del enlace. Elemento que se tomó en cuanta para que en la versión final al dar clic encima de la etiqueta o del enlace, este quedara seleccionado.

[2] Errores de programación

También se observó que algunos estudiantes no encontraban con facilidad algunas opciones, esto se debía a que la versión de pruebas que se les presentó no incluía los íconos que finalmente formarían parte del sistema.

En cuanto a *bugs*, fue mediante esta prueba que se detectó que cuando hay un operador de unión (|) y un operando es subcadena del otro, entonces hay que colocar el de mayor longitud a la izquierda, como se explica en el epígrafe **Error! Reference source not found.**.

Experimento con 5 estudiantes de segundo año

Retomando la hipótesis planteada: la representación del autómata finito permitirá la creación de expresiones regulares sin necesidad de poseer amplios conocimientos sobre su estructura y sintaxis. Para comprobarla se ha realizado un experimento con 5 estudiantes de segundo año de la carrera de informática, donde se les pidió que resolvieran ejercicios relacionados con la obtención de patrones de texto.

El experimento puede ser clasificado por el tiempo de duración como breve, pues duró menos de dos horas. Por las funciones de los métodos empíricos de investigación como verificador porque se orienta hacia la comprobación de una hipótesis. En función de las condiciones para la realización se clasifica como de laboratorio, debido a que los participantes fueron escogidos teniendo en cuenta si conocían o no acerca de los autómatas y expresiones regulares.

Se escogieron precisamente de ese curso porque no han recibido la asignatura Programación IV, donde se abordan los contenidos relacionados con las expresiones regulares y autómatas finitos. Antes de comenzar se les enseñó los elementos básicos de expresiones regulares y de autómatas finitos. En el caso de autómatas finitos consistió en describir que es y cómo funciona y un ejemplo paso a paso. En el caso de las expresiones regulares se les explicaron los elementos que las conforman con un ejemplo para cada parte, dejando escrito en la pizarra los caracteres reservados y un resumen de lo planteado. Esta información proporcionada a los estudiantes es lo que se entiende como conocimientos mínimos en el marco de esta investigación, es decir, que el único conocimiento que tienen acerca del tema es lo que se les enseñó brevemente, minutos antes.

La muestra utilizada no es muy representativa, fueron solo 5 estudiantes, no se escogió una muestra mayor porque en el momento de hacer el experimento los estudiantes se encontraban en pruebas finales y era muy difícil reunir la cantidad necesaria. No obstante se disidió que si al menos un estudiante podía resolver los ejercicios más rápido por la vía del autómata que por la expresión regular, con los escasos conocimientos sobre ambos métodos, ya se estaría obteniendo un resultado relevante, que corroboraría la validez de la hipótesis.

Una vez seleccionados los estudiantes, se les pidió que resolvieran tres ejercicios que consistieron en detectar patrones de texto, mediante el autómata finito y la expresión regular, directamente utilizando la herramienta para comprobar sus resultados en ambos casos. Las tareas les fueron puestas en orden de menor a mayor complejidad, para que

fueran ganando en práctica y así poder enfrentar problemas más complejos. Los ejercicios propuestos fueron los siguientes:

Experimento 2

1. Obtener el autómata finito y la expresión regular para detectar cifras numéricas correspondientes a cantidad de dinero, ejemplos de cadenas válidas: $1.00, $0.50, destacando que el símbolo de pesos es obligatorio al inicio de la cadena.

2. Obtener el autómata finito y la expresión regular para detectar la hora en el formato Horas : Minutos AM ó PM, sin tener en cuenta que fuera una hora válida, es decir, las cadenas 25:99am y 19:85PM serían aceptadas como válidas, como también lo son 1:1am, 01:01am, 11:58PM. Lo importante en este ejercicio era que fueran capaces de detectar dígitos separados por dos puntos y al final las constantes am y pm en sus variantes mayúsculas y minúsculas.

3. Obtener el autómata finito y la expresión regular para detectar la hora en formato militar, es decir, Horas de 0 a 23, dos puntos y minutos de 0 a 59. Esta vez sí era necesario que validaran que fuera correcta la hora, en este caso cadenas como 25:88 no son consideradas como válidas, sólo desde las 00:00 hasta las 23:59.

En cada caso se midieron los tiempos que tardaron en resolver cada uno de los ejercicios propuestos, si la solución era incorrecta se les notificaba y seguían intentando hasta dar con la respuesta correcta. En la Tabla 3 se muestran los valores de los tiempos de cada estudiante en cada ejercicio, así como el promedio por cada ejercicio.

Tabla 3 Tiempos de duración en resolver los ejercicios propuestos para el experimento 2.

Estudiantes	AF 1	ER 1	ER 2	AF 2	AF 3	ER 4
Estudiante #1	3	4	13	9	13	11
Estudiante #2	4	5	10	9	9	15
Estudiante #3	4	6	11	8	7	12
Estudiante #4	4	6	4	5	8	10
Estudiante #5	4	4	6	6	9	8
Tiempo promedio	3,8	5	8,8	7,4	9,2	11,2

Leyenda: AF – Autómata Finito, ER – Expresión Regular

También se pueden apreciar las diferencias de tiempo promedio entre la resolución de cada ejercicio mediante el autómata (columnas AF) respecto a la resolución utilizando directamente las expresiones regulares (columnas ER), en todos los casos fueron menores utilizando el autómata finito. Al aumentar la complejidad del ejercicio aumenta también la diferencia en tiempo entre ambos métodos. Además se les alteró el orden entre ambas formas de resolución para evitar favorecer alguna en específico.

Los errores más frecuentes fueron vistos en el caso de las expresiones regulares, se pueden citar algunos que aparecieron de forma reiterada, por ejemplo:

- Omitir el carácter de escape (\) delante del punto para interpretarlo como punto y no como cualquier carácter, que es su significado en expresión regular.
- No agrupar correctamente los fragmentos de expresión regular para aplicar cuantificadores como + y ?.
- No considerar exactamente la cantidad de repeticiones para un carácter determinado.

A continuación se muestran las soluciones a los ejercicios propuestos, cada uno con su autómata finito y la expresión regular. Es importante destacar que las soluciones que se proponen no son las únicas, cada problema tiene múltiples soluciones, y sólo quien posea amplios conocimientos podrá dar las soluciones óptimas.

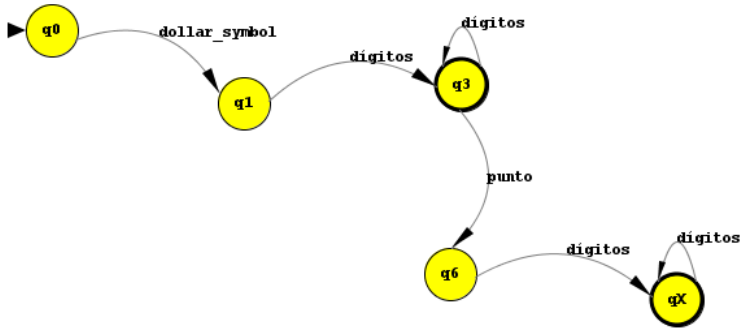

Figura 10 Autómata finito que da solución al ejercicio 1.

Una expresión regular que da solución al ejercicio 1 es:
((\$(\d)+[.,](\d)+)|(\$(\d)+))

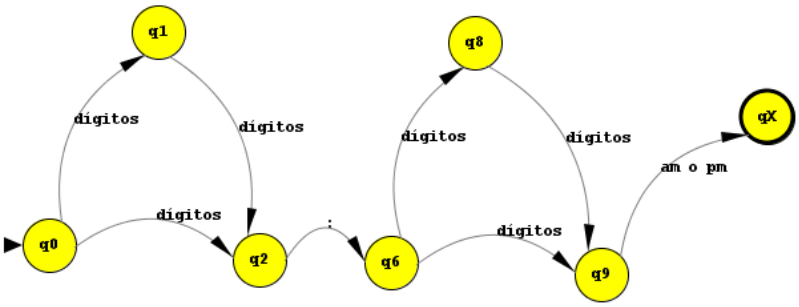

Figura 11 Autómata finito que da solución al ejercicio 2.

Una expresión regular que da solución al ejercicio 2 es:
(((\d\d)|(\d)):\d\d|((\d\d)|(\d)):\d)(AM|am|PM|pm|m|M)

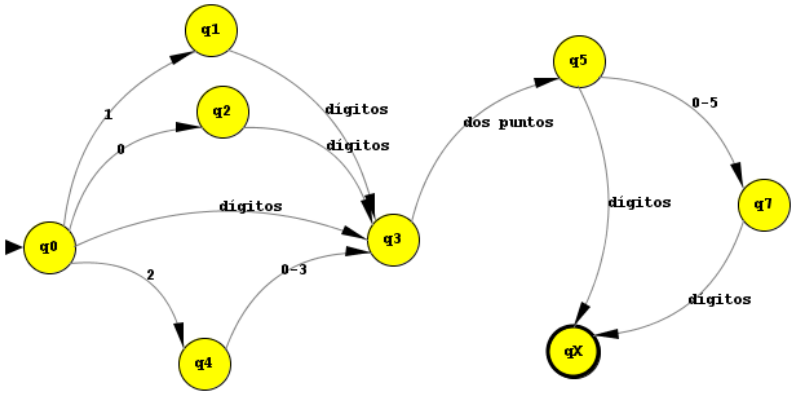

Figura 12 Autómata finito que da solución al ejercicio 2

Una expresión regular que da solución al ejercicio 3 es:

(((((1\d)|(\d))|(0\d)):|(2(0|1|2|3):))(0|1|2|3|4|5)\d|((((1\d)|(\d))|(0\d)):|(2(0|1|2|3):))\d)

Análisis de los resultados

Después de haber realizado las pruebas con los estudiantes de tercer año observando su desenvolvimiento con la herramienta, y haber desarrollado el experimento anterior, se puede afirmar que es más fácil obtener el patrón deseado a través del autómata finito que directamente con la expresión regular. Teniendo en cuenta además la diferencia evidente entre ambas vías de solución como se puede observar en los ejemplos mostrados anteriormente en la Figura 10, Figura 11 y Figura 12, se considera que queda demostrado no sólo que el autómata es la vía más rápida e intuitiva para resolver este tipo de problemas, sino también que es posible, a través de una herramienta informática, obtener la expresión regular a partir del autómata finito.

Conclusiones

Con el estudio del estado del arte se evidencia la necesidad del desarrollo de una herramienta informática en que resuelva la problemática existente, utilizando la obtención de la expresión regular a partir de su representación gráfica, el autómata finito. A través de elementos gráficos se pudo obtener una representación clara y sencilla del autómata finito en la computadora. Se obtuvo una modificación al algoritmo de conversión de un autómata en expresión regular, mejorando la expresión regular resultante. La implementación de opciones como el etiquetado e insertar un enlace desde un diccionario, facilitan el trabajo en aplicaciones de PLN. Las pruebas con estudiantes, en condiciones reales fuera de laboratorio, permitieron comprobar y validar el funcionamiento de la herramienta. Mediante el análisis de los resultados del experimento realizado se demostró que, con el apoyo de una herramienta informática, es posible la creación de expresiones regulares a partir del autómata finito, sin la necesidad de poseer amplios conocimientos sobre el tema.

Bibliografía

Brena, Ramón. 2003. *AUTOMATAS Y LENGUAJES*. Monterrey : s.n., 2003.

Cabon, Alain. 2007. JRegexpTester. *http://jregexptester.sourceforge.net/*. 2007.

Cuesta, Marcelino y Herrero, Fco.J. INTRODUCCION AL MUESTREO . *Dpto. Psicología. Universidad de Oviedo*. [En línea] [Citado el: 12 de 6 de 2009.] http://www.psico.uniovi.es/Dpto_Psicologia/metodos/tutor.7/p3.html.

El lenguaje de Programación Java™.

Fernández Orquín, Antonio, y otros. 2009. Sistema para el pre-procesamiento de textos para el Procesamiento del Lenguaje Natural. *COMAT*. 2009.

Friedl, Jeffrey E.F. 1998. *Mastering Regular Expressions Powerful Techniques for Perl and Other Tools*. s.l. : O'Reilly & Associates, 1998.

Gamma, Erich, y otros. 1995. *Design Patterns. Elements of Reusable Software*. s.l. : Addison-Wesley, 1995.

García Valiente, Iván L. 2009. RERAS: Reconocedor de entidades combinando expresiones regulares y aprendizaje supervisado. Universidad de Matanzas : s.n., 2009.

Goyvaerts, Jan. 2007. RegexBuddy3. *http://www.regexbuddy.com*. 2007.

Gunnerson, Eric. 2003. RegexWorkbench. *http://www.gotdotnet.com/*. 2003.

Hopcroft, John E. 2001. *Introducion to automata theory, languages, and computation*. s.l. : Addison-Wesley, 2001.

inc., Ultrapico. 2007. Expresso. *http://www.ultrapico.com*. 2007.

Juristo, Natalia, Moreno, Ana M. y Vegas, Sira. 2005. *TÉCNICAS DE EVALUACIÓN DE SOFTWARE*. 2005.

León Gil, Leysi, García Vasconcelos, Yaniseth y Miranda Santana, Lisandra. 2009. Extracción de Información en documentos docentes. *COMAT*. 2009.

Levithan, Steven. 2008. Regex Tester RegexPal. *http://regexpal.com/*. 2008.

MARTÍNEZ JUAN, FRANCISCO JAVIER. 1997. *GUÍA DE CONSTRUCCIÓN DE SOFTWARE EN*. OVIEDO : s.n., 1997.

metriplica.com, Grupo. 2008. ¡Prueba tus Expresiones Regulares! *http://www.metriplica.com/4_4_herramientas.asp*. 2008.

Miranda Santana, Lisandra. 2008. Virtual-Act 2008. *Virtual-Act 2008*. Universidad de Matanzas : s.n., 2008.

Moreno, Ana Mª. 2006. *Estimación de Proyectos Software*. 2006.

netbeans.org. 2009. NetBeans IDE - Features. [En línea] 2009. http://www.netbeans.org/about/.

Pérez Martínez, Abel y Fernández Orquín, Antonio. 2009. Generador de Expresiones Regulares como recurso para el procesamiento del lenguaje natural. *COMAT*. 2009.

2007. Regextester. *http://www.regexlib.com/regextester*. 2007.

2007. Regular Expression Editor. *http://www.waterproof.fr/*. 2007.

2006. reWork. Regular Expression Workbench. *http://osteele.com/tools/rework/*. 2006.

Riesterer, Laurent. 2006. Visual REGEXP. *http://laurent.riesterer.free.fr/regexp/*. 2006.

Sells, Chris y Weinhardt, Michael. 2002. RegexDesigner. *http://www.sellsbrothers.com/*. 2002.

Weitz, Dr. Edmund. 2006. The Regex Coach. *http://common-lisp.net/mailman/listinfo/regex-coach*. 2006.

2008. Wikipedia, la enciclopedia libre. *Expresión regular*. [En línea] 4 de 2008. http://es.wikipedia.org/wiki/Expresi%C3%B3n_regular.

2009. Wikipedia, la enciclopedia libre. *Christopher Alexander*. [En línea] 13 de 04 de 2009. [Citado el: 15 de 4 de 2009.] http://es.wikipedia.org/wiki/Christopher_Alexander.