PRÁCTICA 1: EXPRESIONES REGULARES

Blanca Calderón

Contenido

Pasos previos (apartado1):

Antes de nada, se ha llevado a cabo la traducción de la expresión regular al formato JFLAP (cambiando los caracteres [-] y + por sus respectivos en JFLAP) cuyo resultado ha sido a(!+a)*(bc*(m+n+o+p+q))(!+(bc*(m+n+o+p+q)))* (Nótese que se ha decidido no incluir el carácter ñ al no ser universal).

A continuación, se ha generado el autómata finito no determinista a través de la expresión (comprobando su validez con dos casos de ejemplos, siendo éstos la cadena válida "abcm" y la no válida "abc") a su vez se ha pasado éste a autómata finito determinista comprobando también que funcione correctamente usando los ejemplos del enunciado ("abmbo" la cual es válida y "aabcccmbqccp" que no lo es).

Finalmente se ha generado la matriz de transición de estados, para ello se ha analizado el autómata finito determinista observando a que nodo iríamos según que carácter recibimos y el estado en el que nos encontramos actualmente llevando a cabo la matriz poniendo los caracteres en la fila y los estados en la columna.

	a	b	c	m	n	o	p	q
Q0	Q1							
Q1	Q2	Q3						
Q2	Q2	Q3						
Q3			Q4	Q9	Q7	Q5	Q6	Q8
Q4			Q4	Q9	Q7	Q5	Q6	Q8
Q5		Q10						
Q6		Q10						
Q7		Q10						
Q8		Q10						
Q9		Q10						
Q10			Q11	Q16	Q14	Q12	Q13	Q15
Q11		Q10	Q11	Q16	Q14	Q12	Q13	Q15
Q12		Q10						
Q13		Q10						
Q14		Q10						
Q15		Q10						
Q16		Q10						

*Los estados finales son Q5, Q6, Q7, Q8, Q9, Q12, Q13, Q14, Q15 Y Q16.

Implementación del programa en Java

Antes que nada, debemos crear los archivos necesarios en Java, en este caso tendremos tres clases distintas: Main, AFD y MaquinaDeEstados:

- Archivo AFD → Este archivo representa al autómata finito determinista y tendrá los atributos alfabeto (Lista de caracteres), estados (lista de enteros), estado inicial (entero), estados finales (lista de enteros) y matriz (hashmap).

Además, contará con las funciones cargarAlfabeto(), cargarEstados(), establecerEstadoInicial(estado) el cual establece el estado inicial, establecerEstadoFinal,inicializarMatriz(),cargarMatriz(),getSiguienteEstado(estadoActual, caracter), isFinal(estado) que indica si el estado introducido es el final, getEstadoInicial(0), getAlfabeto(), getMatriz(), limpiarEstadoFin(), limpiarEstados(), limpiarAlfabeto() y otros getters y setters.

-Archivo MaquinaDeEstados → Sus atributos son estadoActual (entero), automata(AFD), correctas (lista de cadenas que contiene palabras correctas encontradas),tokens (lista de cadenas que contiene tokens generados por el autómata) y resultados (contiene valores de retorno de la función almacenarCaracter ()) y sus funciones dos constructores (uno vacio y otro que recibe el autómata correspondiente),inicializar(),getCorrectas(),getTokens(), inicializarEstadoActual(),limpiarCorrectas(),limpiarTokens(),limpiarResultados() aceptaCaracter(carácter,estado),isFinal(estado),comprobarCadenas(cadena,cadena), almacenarCorrecta (cadena, lista de cadenas, cadena, cadena), almacenarCaracter(entero, cadena,array de caracteres),importarMatriz(archivo), crearMatrizcadena(),unionConSIguiente(carácter, entero, entero) junto con funciones básicas get y set para cada atributo.

-Main → Contiene los atributos automata (AFD), maquina (MaquinaDeEstados), cad (cadena), opción (cadena), expresiones (lista de cadenas), expresiones2 (lista de cadenas), expresión (cadena), tipos(lista de cadenas), escaner (Scanner), nombres(lista de cadenas), nombres2(lista de cadenas), archivo (FileWriter) y continuar(entero).

En el main encontramos un bucle en el que se mostrará por pantalla las opciones a llevar a cabo eligiendo el usuario a través del teclado, esta selección se hará a través de una estructura switch que según opción elegida llamará a las funciones correspondientes de las otras clases y mostrará resultados por terminal.

También, vemos que hay dos funciones más, mostrarResultados la cual recibe la máquina de estados y la cadena a analizar mostrando por pantalla los resultados conseguidos tras analizar la cadena (tokens, cadenas correctas e incorrectas), la otra función es importarArchivo que recibe el nombre del archivo y la máquina de estados y que es utilizada para importar el archivo jff del autómata en la matriz. También tenemos recogerCadena (cadena, array de cadenas, array de cadenas, maquinaDeEstados), analizarUnAutomata (cadena, cadena, cadena, maquinaDeEstados) y analizarAutomatas (cadena, array de cadenas, array de cadenas, maquinaDeEstados)

Las opciones a elegir son 1, 2, 3 o -99 siendo el último usado para terminar ejecución.

APARTADO1: Expresión a(!+a)*(bc*(m+n+o+p+q))(!+(bc*(m+n+o+p+q)))*

***Usando automáta previamente creado en pasos previos**

1.1º Elegir ejercicio a llevar a cabo → Tras ejecutar programa se mostrará por pantalla un menú con las opciones a elegir (cada opción corresponde a un ejercicio), para realizar este apartado se deberá elegir la opción 1 lo que te llevará a la función recogerCadena que mostrará opción elegida y pedirá una cadena por teclado que recogerá para mandársela a la función analizarAutomatas y comenzar su análisis.

1.2º Importación y carga del archivo jff (autómata)→ Tras elegir opción a realizar se pasará a ejecutar la función analizarAutomatas que mientras la cadena no este vacía la analizará en cada autómata de la lista pasada de autómatas de ese apartado y llamará a función analizarUnAUtomata para analizar cadena en un autómata específico cargando en esta función el autómata finito de la expresión a(!+a)*(bc*(m+n+o+p+q))(!+(bc*(m+n+o+p+q)))* ya creado en JFlap cómo se ha explicado anteriormente a través de la función importarArchivo de la clase MaquinaDeEstados. En caso de que cadena no este vacía y halla sido analizado por todos los autómatas de la lista se quitará primer carácter para pasar a analizar el siguiente. Cuando la cadena esta vacía se mostrarán los resultados por el terminal.

Para cargar archivo jff se usará la función importarMatriz (inputFile) de la clase maquina (Máquina de Estados) a la que le pasaremos el archivo jff del autómata correspondiente excepto si el autómata es el de cadenas que se llamará a la función crearMatrizCadena, en esta función primero limpia valores del autómata anterior con limiparAnterior() luego se lee el archivo jff (en formato XML) introduciendo los estados encontrados en el archivo (diferenciado por el tag 'state' en el archivo) y las transiciones (su tag es 'transition') en dos NodeList .

Tras esto, se lleva a cabo un bucle que recorre la NodeList que contiene los estados creando un elemento para cada nodo (cada nodo es un estado) y cargando el atributo 'id' de estos en la estructura estados del autómata correspondiente a través de la función cargarEstados (). Además, identifica los estados finales y el inicial revisando el tag de los elementos (si tag es 'final' es estado final y si es 'initial' inicial) y se introducen en sus estructuras correspondientes (estadoIni y estadosFinales) a través de las funciones establecerEstadoFin y establecerEstadoIni.

Continua llamando a inicializarMatriz() que creará la estructura de la matriz (hashmap) según el número de nodos (estados) del archivo introduciéndolos en él. Posteriormente con otro bucle se introducen las transiciones usando las variables estadoBase (estado en el que nos encontramos y diferenciado por tag 'from' en el archivo XML), estadoSig (estado al que pasaremos y diferenciado por tag 'to' en el archivo) y c (contiene carácter de la transición cuyo tag es 'read'), se crea una vez más elemento esta vez usando las transiciones y se usa para asignar valor correspondiente a cada variable.

Por último, se usa función cargarMatriz (estadoBase, c, estadoSig) para cargar transición actual en la matriz (hashmap cuya primera clave es el estado actual y su valor es otro hashmap con carácter como clave y estadoSiguiente como valor) y se carga además carácter actual en el alfabeto del autómata, esto se repite hasta introducir todas las transiciones.

1.3º Análisis de la cadena introducida → Para analizar la cadena se llamará a la función comprobarCadenas(cadena, cadena) de la clase maquina la cual recibirá la cadena y la expresión a analizar como parámetros y devolverá la parte de la cadena que queda tras eliminar parte correcta.

En esta función primero dividimos cadena pasada como una lista de caracteres y damos valor a estado inicial (variables que usaremos dentro de la función), para realizar análisis llevaremos a cabo un bucle que recorrerá la lista de caracteres comprobando si es un carácter válido a través de la función aceptaCaracter (carácter, estado) la cual comprueba si el carácter es válido en el autómata. Para ello, primero comprueba si está en el alfabeto, si no lo está devuelve falso, si lo está comprueba en la matriz creada si con estado pasado y carácter podemos pasar a siguiente estado, es decir, si carácter cumple expresión para poder continuar.

Si esta condición se cumple devuelve verdadero y si no retorna falso.

Tras haber comprobado si carácter es correcto, si lo es se pasa a siguiente estado y se incluye el carácter en la palabra correcta que vamos formando, también guarda carácter validoe n posibleCorrecta, esto es necesario ya que podemos tener una palabra que sea diminutivo de otra que programa debe tener en cuenta (Ej: v y verdadero con expresión ((v+V)(e+E)(r+R)(d+D)(a+A)(d+D)(e+E)(r+R)(o+O))+(V+v)+((f+F)(a+A)(l+L)(s+S)(o+O))+(F+f), ambas son válidas y si introducimos cadena verbK en posibleCorrecta guardaríamos la v ya que con este carácter tendríamos estado final pero como siguiente carácter es correcto seguimos mirando hasta llegar a K ay que no es válido, vemos que no se ha llegado a estado final con verb por lo que cogemos como correcta la v al no haber una palabra correcta más larga).

Tras llevar a cabo el código de esta función se pasa a analizar el siguiente carácter (sigue bucle) en caso que se devuelve falso se saldrá del bucle mostrando <ERROR, carácter> y analizando si palabra formada es correcta o no a través de comprobarValidez que compruieba si estado es final y si lo es llama a almacenarCorrecta que guarda la palabra formada en lista de correctas al igual que el token conseguido y devuelve la cadena quitando la parte correcta. En comprobarValidez si el estado no es final también se comprueba si la cadena posibleCorrecta construida no esta vacía ya que si no lo está se debe tener en cuenta llamando una vez más almacenarCorrecta generando token correspondiente y quitando parte válida de la cadena.

Cuando se sale del bucle de comprobar caracteres (se llega a final de la cadena) se debe comprobar una vez más si tenemos palabraválida llamando a comprobarValidez y llevando a cabo lo señalado anteriormente.

Función comprobarCadena devolverá a la clase main la cadena quitando parte correcta si se ha encontrado y se continuará análisis con el resto de los autómatas correspondientes del apartado (en este caso solo hay un autómata) hasta que cadena este vacía y se muestren resultados obtenidos (tokens).

1.4º Seleccionar nueva opción o terminar ejecución →

Tras haber evaluado cadena pasada y mostrado resultado (llamando a función mostrarResultados) por pantalla se pregunta al usuario si quiere realizar una nueva operación o terminar ejecución introduciendo -99 por teclado.

En el caso que quiera realizar más operaciones si elige opción 1 vuelve a llevar a cabo pasos 1.1º, 1.2º y 1.3º.

APARTADO 2: Palabras reservadas del lenguaje PSEInt.

-Paso previo → Las palabras reservadas elegidas han sido función, algoritmo y repetir. Para cada palabra se ha llevado a cabo la creación de una expresión regular (no diferenciando entre mayúsculas y minúsculas) y de un autómata finito determinista en JFlap.

Las expresiones regulares son las siguientes:

1º Función →(f+F)(u+U)(n+N)(c+C)(i+I)(o+O)(n+N)

2º Algoritmo → (a+A)(l+L)(g+G)(o+O)(r+R)(i+I)(t+T)(m+M)(o+O)

3º Repetir →(R+r)(e+E)(p+P)(e+E)(t+T)(i+I)(r+R)

-2. 1ºElegir opción → Igual que en el paso 1. 1º.

-2. 2º Importación y carga del archivo → Igual que paso 1.2º pero esta vez tendremos tres autómatas en la lista de autómatas.

-2. 3º Analizar cadena introducida→ Igual que paso 1.3º pero carácter se analiza en tres autómatas antes de quitarlo para continuar análisis.

-2.4º Elegir nueva opción o terminar ejecución → Al terminar análisis de la cadena se muestra por pantalla palabras válidas encontradas y tokens generados que nos queda al final a través de la función mostrarResultados.

Además, nos da la opción de elegir una nueva opción o terminar ejecución. Si elige opción 2 vuelve a hacer pasos anteriores, opción 1 va a 1º y 3 a 3º.

APATADO 3: Reconocer tipos básicos de datos de PSEInt.

-Paso previo → Se han llevado a cabo las expresiones para cada tipo aceptado en PSEINT (entero, real, booleano y cadena) y el autómata determinista para cada una.

Las expresiones regulares son:

-Entero → (!+-)(0+1+2+3+4+5+6+7+8+9)(0+1+2+3+4+5+6+7+8+9)*

-Real→ (!+-))(!+0+1+2+3+4+5+6+7+8+9*(.)(0+1+2+3+4+5+6+7+8+9)(0+1+2+3+4+5+6+7+8+9)*

Booleanos→((v+V)(e+E)(r+R)(d+D)(a+A)(d+D)(e+E)(r+R)(o+O))+(V+v)+((f+F)(a+A)(l+L)(s+S)(o+O))+(F+f)

-Cadenas→
("+')(a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+x+y+z+A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z+#+$+%+&+.+/+,+0+1+2+3+4+5+6+7+8+9+:+;+<+~+>+?+@+_+[+]+\+{+}+ü+Ç+é+â+ä+à+å+ç+ê+ë+è+ï+î+ì+Ä+Å+É+æ+Æ++ö+ò+û+ù+ÿ+Ö+Ü+á+í+ó+ú+ñ+Ñ

+ª+º+¿+⌐+¬+ø+£+Ø+×+ƒ+á+í+ó+ú+ª+º+¿+®+½+¼+¡+«+»+|+┤+Á+Â+À+©+Đ+Ê+Ë+È+ı+Í+Î+Ï+┘+┌
+Ó+ß+Ô+Ò+õ+Õ+µ+þ+Þ+Ú+Û+Ù+ý+Ý+¯+´+-+±+‗+¾+¶+§+÷+¸+°+¨+·+¹+³+²+)*('+")

Caracteres especiales de JFlap +,*,!,(,) se introducirán el autómata al crear el AFND añadiendo dos nodos para cada uno ya que no los podemos incluir en la expresión regular.

Aunque finalmente esta expresión no se ha utilizada ya que el tiempo de carga del autómata en el programa era muy alto por lo que en vez de esto se ha creado una nueva función en clase MaquinaDeEstados llamada crearMatrizCadena() la cual crea la matriz de transición del autómata de cadenas manualmente. En esta función primero se cargan los estados a través de un bucle y llamando a cargarEstados del autómata (habrá tantos estados como caracteres ASCII). Tras esto se han establecido los estados finales que serían el 257 y 258 (estados a los que se pasa con " y ') y el estado inicial (el 0). También se carga el alfabeto ASCII en el alfabeto del autómata.

Seguimos cargando las transiciones en la matriz, primero añadiendo la transición correspondiente a las " y ' que aparecen al principio de la cadena y después el resto usando dos bucles anidados (el primero recorre los estados base y el segundo por los siguientes) , dentro de este bucle se unirá los dos estados a los que pasamos con " y ' con todos los estados correspondientes a cada carácter controlando que no sean los estados finales ni los correspondientes a los caracteres de " y ' (37 y 99) sustituyendo estadoSiguiente por 37 cuando es estadoSiguiente es 257 y 99 cuando es 258, en el caso de que estadoBase sea 257 se sustituirá por 37 y por 99 si es 258, esta sustitución se hace llamando a la función unionConSiguiente al que le pasamos carácter, estadoBase y estado siguiente especifico que queremos (37 y 99 en nuestro caso), en esta función une nodos 1, 2 y estado base con el estado siguiente especificado. Cuando no es ninguno de estos casos se unen los nodos entre ellos de forma normal también usando función unionConSiguiente.

-3.1º Elegir opción →

Igual que en el paso 1.1º pero esta vez elegiríamos opción 3 e introduciríamos una cadena.

-3.2º Importación y carga del archivo →

Igual que en el paso 2.3º pero introduciendo los archivos correspondientes a cada autómata excepto en el caso de las cadenas que se lleva a cabo la creación de la matriz de forma manual con la función matrizCadenas().

-3.3º Analizar cadena introducida →

Este paso se llevaría a cabo igual que en el paso 2.3º

-3.4º Elegir nueva opción o terminar ejecución →

Al terminar análisis de la cadena se muestra por pantalla palabras válidas encontradas, cadena incorrecta que nos queda al final y tokens al llamar a la función mostrarResultados(). Además, nos da la opción de elegir una nueva opción o terminar ejecución. Si elige opción 2 vuelve a hacer pasos anteriores, opción 1 va a 1º y 2 a 2º.