INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

U.A: Teoría Computacional

“Reporte práctica 03”

Grupo: 2CV13

Profesor: De la O Torres Saul

Alumno: Sánchez Becerra Ernesto Daniel

INTRODUCCIÓN:

¿Qué es una expresión regular?

Las regex (en inglés, regular expressions) son las unidades de descripción de los lenguajes regulares**,** que se incluyen en los denominados lenguajes formales. Son un instrumento clave de la informática teórica, la cual, entre otras cosas, establece las bases para el desarrollo y la ejecución de programas informáticos, así como para la construcción del compilador necesario para ello. Es por esto que las expresiones regulares, también denominadas regex y basadas en reglas sintácticas claramente definidas, se utilizan principalmente en el ámbito del desarrollo de software.

Para cada regex existe un denominado autómata finito (también conocido como máquina de estado finito) que acepta el lenguaje especificado por la expresión y que, con ayuda de la construcción de Thompson, se desarrolla a partir de una expresión regular. Por otro lado, para cada autómata finito también hay una expresión regular que describe el lenguaje aceptado por el autómata. Este puede generarse bien con el algoritmo de Kleene o bien con la eliminación de estados.

¿Qué es un alfabeto?

Un alfabeto es un conjunto finito no vacío cuyos elementos se llaman símbolos. Denotamos un alfabeto arbitrario con la letra Σ. Una cadena o palabra sobre un alfabeto Σ es cualquier sucesión finita de elementos de Σ.

¿Qué es un autómata finito no determinista?

Un autómata finito no determinista (abreviado AFND) es un autómata finito que, a diferencia de los autómatas finitos deterministas (AFD), posee al menos un estado $q \in Q$, tal que para un símbolo $a \in \Sigma$ del alfabeto, existe más de una transición $\delta(q,a)$ posible. Lo que se descubre tras que alguien haga mal muchos autómatas finitos y lo arreglan con un nombre bonito. Todo autómata finito no determinista puede ser convertido a autómata finito determinista. Pero no al contrario.

En un AFND puede darse cualquiera de estos dos casos:

- Que existan transiciones del tipo $\delta(q,a)=q_1$ y $\delta(q,a)=q_2$, siendo $q_1 \neq q_2$;
- Que existan transiciones del tipo $\delta(q, \varepsilon)$, siendo q un estado no-final, o bien un estado final pero con transiciones hacia otros estados.

Cuando se cumple el segundo caso, se dice que el autómata es un autómata finito no determinista con transiciones vacías o transiciones ε (abreviado AFND-ε). Estas transiciones permiten al autómata cambiar de estado sin procesar ningún símbolo de entrada. Considérese una modificación al modelo del autómata finito para permitirle ninguna, una o más transiciones de un estado sobre el mismo símbolo de entrada.

Objetivo:

El alumno tendrá que realizar un programa que para una expresión regular determinada, se calcule su AFN correspondiente.

Desarrollo:

En esta práctica se llevo a cabo la conversión de una expresión regular a un Autómata Finito No Determinista utilizando el algoritmo de Thompson pues con este proceso vamos a poder realizar la conversión de una expresión regular a un autómata siguiendo las siguientes reglas:

Dadas las reglas que definen las expresiones regulares se pueden escribir como AFND-e:

- Φ es una expresión regular que describe el lenguaje vacío: en este caso se construye un AFND-e de dos estados, uno inicial y otro final, que no tienen transiciones, por lo cual no están conectados. De esta manera el autómata reconoce el lenguaje vacío.

- ε es una expresión regular que describe el lenguaje {ε}, que es un lenguaje que únicamente contiene la cadena vacía: el autómata que reconoce este lenguaje es aquel que el estado inicial también es final.

- si "a" está en el alfabeto, "a" (sin comillas) es una expresión regular que describe el lenguaje {a}: el autómata que reconoce este lenguaje tiene definida una transición desde el estado inicial hacia un estado final.

- si existen r y s expresiones regulares r es una expresión regular que describe L(r) y s es una expresión regular que describe L(s)
    - r+s describe L(r) U L(s) (lenguaje generado por r unión lenguaje generado por s)
    - r.s describe L(r). L(s) (lenguaje generado por r concatenado lenguaje generado por s)
    - r* describe L(r)* (lenguaje generado por r clausura)

Las precedencias de operador son *,., +.

Además, cabe recalcar que esta práctica fue elaborada utilizando el entorno de programación “NetBeans” por lo que se utilizó lenguaje de programación Java.

Una vez entendido esto, se elaboró el programa de la siguiente manera; Se tienen 4 paquetes los cuales son:

1. análisis: este paquete las siguientes clases que básicamente nos van a ayudar a hacer el análisis de la expresión que se manda como argumento al analizador sintáctico.
    a. Alfabeto: esta clase es la que representa el alfabeto en el cual se construye la expresión regular.
    b. AnalizadorLexico: Esta clase implementa el analizador léxico y el método para que el analizador sintáctico pueda consumir tokens. Es importante mencionar que los lexemas de la entrada siempre estarán formados por un

solo carácter, por lo que no se requiere utilizar autómatas finitos para construirlos.
   c. AnalizadorSintactico: Clase que implementa un analizador sintáctico predictivo para una expresión regular realizando una traducción de la misma a su correspondiente AFN.
   d. Token: Clase que representa un token de una expresión regular. El token tiene 2 atributos que serán el identificador y el valor.
   e. TokenExprReg: Clase donde se especifica que tipo de operando se está utilizando en la expresión regular ya sea el + que representa una cerradura, un asterisco que representa la cerradura de kleene, un paréntesis ya se derecho o izquierdo y una barra vertical.

2. estructuras: este paquete contiene las clases que nos permitirán construir el AFN después de que la expresión regular pasó por el proceso de análisis:
   a. AFN: Clase que representa la abstracción para un AFN. El AFN es construido a partir de una expresión regular a través de las reglas del algoritmo de Thompson.
   b. Automata: Clase que representa la abstracción para un Autómata Finito ya se AFD o AFN. Inicialmente, un AFN es construido a partir de una expresión regular a través de las construcciones de Thompson.
   c. Conjunto: Esta clase representa un conjunto genérico. La misma es utilizada para los conjuntos de estados y transiciones, correspondientes a un autómata finito y a un determinado estado destino de los estados de este.
   d. Estado: esta clase representa un estado para el autómata.
   e. Transición: Implementa la transformación de un autómata, representada por el símbolo y el estado destino. El estado inicial está dado por el estado en el que está contenida esta transición.

3. algoritmos: en este paquete solo se cuenta con una clase la cual lleva el proceso de ejecución del algoritmo de Thompson.
   a. Thompson: Esta clase implementa los algoritmos de Thompson para cada uno de los operadores de una expresión regular. Cabe destacar que las operaciones implementadas en esta clase tienen de efectos secundarios (side-effects) sobre los AFN recibidos como argumentos ya que en ciertos casos estos resultan alterados. Sin embargo, a efectos prácticos esto no es un problema, ya que en la traducción de una expresión regular a AFN siempre será significativo solo el último AFN construido. La solución que podría aplicarse para corregir este efecto es implementar una función de copia para los AFN.

4. Clase principal: en este paquete se tiene la clase principal y es la que ejecuta las otras clases y muestra el resultado tabulado de nuestro Autómata que se construyó.

Conclusión

Con la elaboración de esta práctica se comprendió mejor el algoritmo de Thompson el cual es una alternativa para hacer la traducción de una expresión regular a un Autómata Finito

No Determinista pues es importante saber hacer esta conversión pues de gran manera se comprende mejor cuál podría ser una forma de comunicación de los lenguajes que se utilizan, en este caso, podrían ser lenguajes que nosotros utilizamos cotidianamente y que queremos establecer la comunicación con algún sistema como podría ser una computadora.

Además, se refuerzan algunos términos y temas vistos en la clase como lo son los Autómatas, los lenguajes, los alfabetos, la cerradura abierta y cerrada, la cerradura de kleene, entre otras cosas.

Nota: el código que da solución a este problema se encuentra adjunto a la carpeta donde esta ubicado este reporte de práctica.