Práctica 2

IPN Escuela Superior de Cómputo
Ing. En Sis. Computacionales

Compiladores 3CV5
M.C Saucedo Delgado Rafael Norman

2014090186
Elizalde Díaz Roberto Carlos

Introducción

Esta práctica está centrada en la implementación del algoritmo McNaughton-Yamada-Thompson para convertir una expresión regular en un AFN basado en el algoritmo 3.23 provisto por el libro Compiladores: principios técnicas y herramientas del escritor Aho.

Con la realización de esta práctica podremos comprender de mejor forma la transformación de una expresión regular a un autómata usando las plantillas ya preestablecidas por el algoritmo.

Para la realización de esta práctica se utilizó el lenguaje Java en su versión libre (openjdk) 11.0.9.1 y Netbeans 11 como IDE, corriendo en el sistema operativo Ubuntu 20.04.1 LTS.

Desarrollo

Para el diseño y la implementación de esta práctica se reutilizaron muchas de las clases que fueron creadas en prácticas pasadas, más precisamente en la práctica 1 y 2 como son:

- AFD
- AFN
- Automata
- AFReader
- Trasicion

Por tal motivo este documento se centrará en lo que le da objetivo a esta práctica que es el desarrollo de una clase para implementar el algoritmo de McNaughton-Yamada-Thompson.

Clase Thompson
Para poder solucionar el primer y mayor problema de esta práctica, que es la identificación de operadores y operandos para cada operación que se puede realizar en expresiones regulares, se maneja un concepto al que se le denomino bloques. Este concepto se utiliza para simular el árbol que se genera al derivar una una expresión regular.

Por ejemplo para la expresión regular (a|b)*ab:

Para el primer procesado la expresión regular misma (a|b)*ab es un bloque, pero a la vez está compuesta por más bloque, los cuales son delimitados por los operadores y la jerarquía de estos, así:

(a|b)*ab -> es un bloque

(a|b)* -> es un bloque

(a|b)-> es un bloque

a|b -> es un bloque

a -> es un bloque, b -> es un bloque

ab -> es un bloque

a -> es un bloque y b-> es un bloque

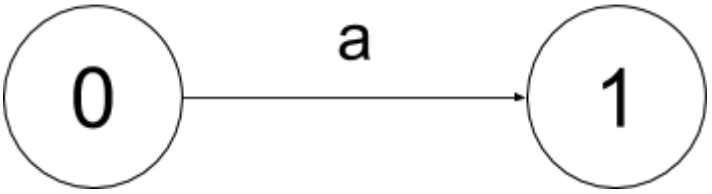
Cada bloque debe de ser procesado de manera individual y con su correspondiente operación y cuando se refiere a ser procesado no es más que obtener el autómata correspondiente a cada paquete.

Para realizar lo anterior tenemos en la clase el método recursivo:

private AFN procesar(String cadena)
Este método evalúa cada caracter que se encuentra en cadena la cual es la expresión regular que deseamos obtener su AFN correspondiente.

Esto lo hace en una serie de pasos:

1. Evalúa si el tamaño de la cadena, si es mayor a 1 la procesa normalmente, si es 1 entonces quiere decir que lo que se desea procesar es solo una letra [a-z]. Si este último es el caso, entonces se le crea el autómata correspondiente a ese caracter, por ejemplo para la cadena que sea un único caracter y el contador de estados este en 0, generará el siguiente autómata:



Este autómata será guardado en una lista de bloques (List<AFN>)

2. En caso de que cadena tenga un tamaño mayor a 1, entonces procederá a analizar caracter por caracter de cadena. Aquí puede derivar 4 casos:

   1- Que encuentre una letra [a-z], en cuyo caso mandará a procesar esa letra con procesar(cadena)
   2- Que encuentre un '(' en cuyo caso concatenará todos los caracteres hasta encontrar su ')' correspondiente de cierre y mandará a procesar(cadena) con la nueva cadena obtenida.
   3-Que encuentre un '*' el cual por definición se entenderá que lo anterior a '*' ya esta procesado y guardado en la lista bloques como el último elemento. Sabiendo lo anterior se llamará al método ***private AFN cerradura(AFN bloque)*** pasando como argumento a este último elemento de la lista bloques para que se le aplique la cerradura a ese autómata.
   4- Que encuentre un '|', y por la definición de este se sabe necesita algo procesado del lado izquierdo y algo procesado a la derecha para hacer la union entre estos, así que continuamos recorriendo la cadena hasta optener algo procesado por la derecha y así realizar la unión con el método private ***AFN union(AFN bloqueA, AFN bloqueB).***

3. Finalmente cuando la función recursiva hay procesado todos los bloques y guardado en la lista del mismo nombre solo nos quedará concatenar todos ellos con el método ***private AFN concatenacion(AFN bloqueA, AFN bloqueB).***

Pruebas

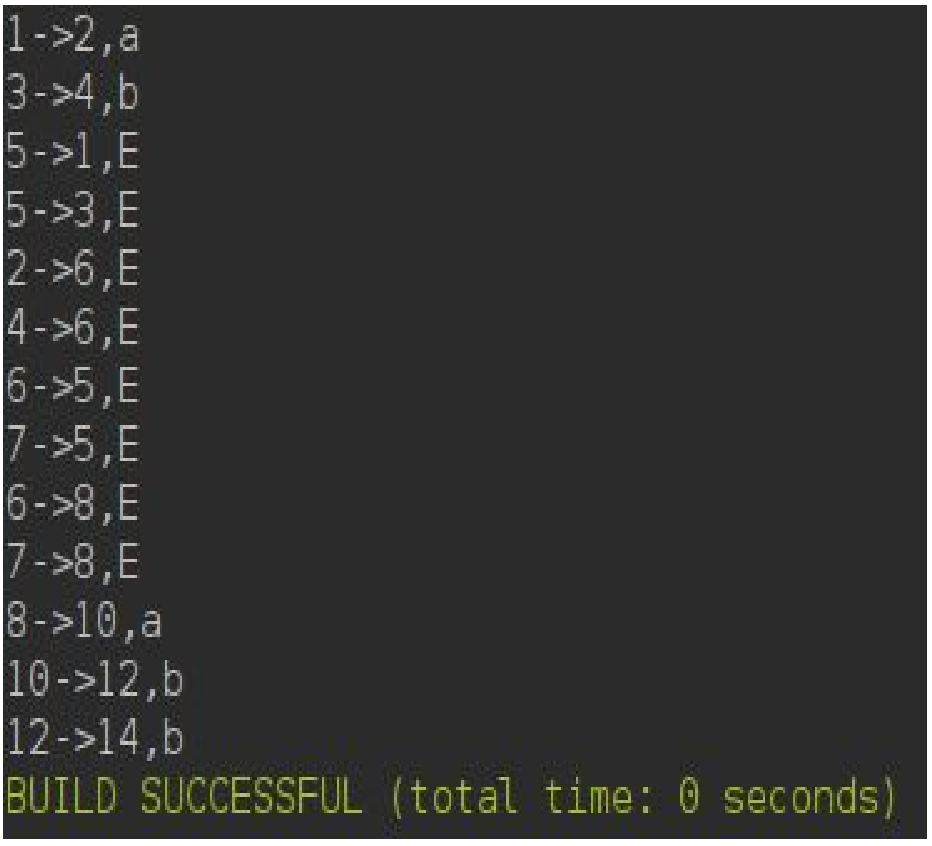
Para realizar las pruebas de esta práctica se utilizó como la expresión regular del Ejemplo 3.24 del libro de compiladores antes mencionado

Expresión regular: **(a|b)*abb**

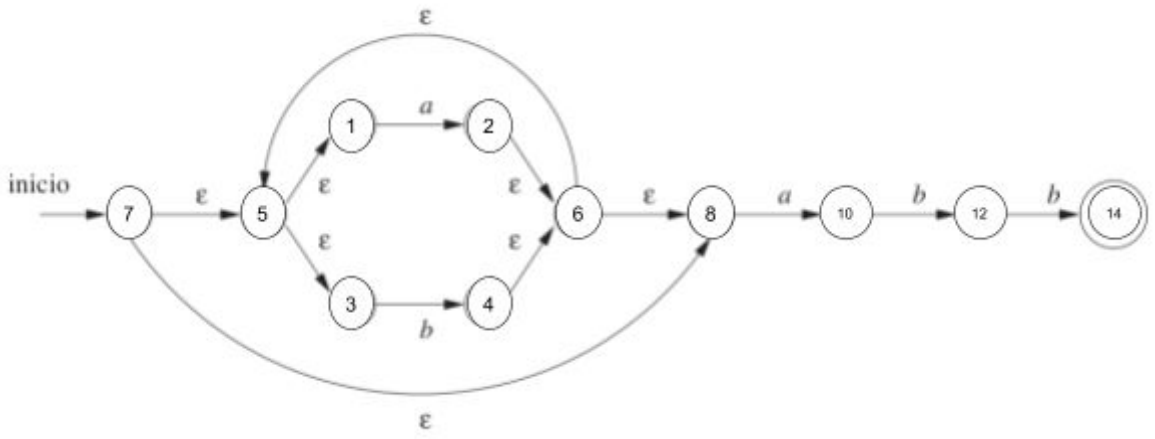
Para empezar la obtención de la creación del AFN instanciamos un objeto de tipo Thompson y llamamos al método convertir(cadena) pasando como argumento la expresión regular.

```
Thompson thompson = new Thompson();
AFN afntemp = thompson.convertir("(a|b)*abb");
```

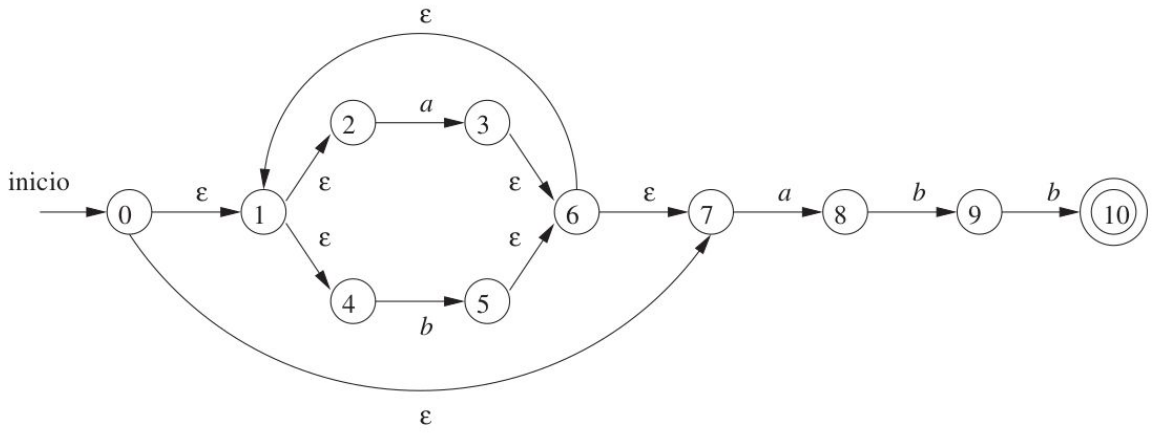
Imprimimos la transiciones obtenidas del objeto afntemp y tenemos lo siguiente en consola:



Utilizando estas transiciones podemos dibujar el siguiente autómata:



Comparado con el resultado del libro podemos apreciar que el resultado es el esperado:



Conclusiones

Esta es si lugar a dudas, la práctica más compleja y complicada que tuve que diseñar y desarrollar de todo el semestre, pues la parte de reconcer correctamente y como saber operar la gerarquia de operaciómes en la expresión regular se volvio completamente un tortura por la cual me sentia muy estresado cuando llegaba la hora de volver analizar el problema, plantear un algoritmo para solucionarlo y hacerlo de manera correcta. Llegué a considerar muchas veces el dejar esta práctica en el olvido y dedicarme a otras prácticas u otras materias. Sin embargo la semana de vacaciones llego y puede descansar una semana para que la siguiente, con mente fresca, me sentará a resolver este problema. La planeación de bloques que propuse no creo que sea muy óptima pero la encontré un tanto sencilla de entender y fácil de implementar.

Esta práctica me ayudó mucho a mis técnicas de diseño de planeación, ya que sentía una notoria diferencia al momento de programar las demás prácticas, pues los errores que tenían al terminar de programarlas no eran tantos del diseño ni de la planeación sino errores de dedo.

Referencias

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, Compiladores: principios, técnicas y herramientas, Segunda Edición.
- Teoria de Automatas, lenguajes y Computación, Hopcroft, Tercera Edición
- Videos de clase