Práctica 1

IPN Escuela Superior de Cómputo
Ing. En Sis. Computacionales

Compiladores 3CV5
M.C Saucedo Delgado Rafael Norman

2014090186
Elizalde Díaz Roberto Carlos

Introducción

En esta práctica se va a diseñar e implementar clases AFN y AFD y algunos métodos para la manipulación y uso de estos dos tipos de autómatas finitos usando la metodología orientada a objetos.

Los clases de los autómatas vendrán acompañadas con algunas otras clases con funciones variadas como lo es la lectura de un autómata a partir de un archivo con un el formato .af especificado por el profesor de la clase.

El formato af (autómata finito) se describe a continuación:

- Los símbolos del alfabeto que pueden usar para el autómata son: a,c,d,...,x,y,z. (sin acentos, sin la ñ), 26 símbolos en total.
- Se usa E para detonar a épsilon.
- Las etiquetas de los estados son números enteros positivos: 1,2,3,4,...(no hay estado cero)
- Cada línea define una transición del autómata, en el formato, estado->estado,simbolo
- La primera línea indica al estado inicial: inicial:estado
- La segunda línea indica a los estados finales: finales:estado,estado,estado,estado
- Las lineas siguientes, cada una, indican una transición.

La forma final de representar autómata por ejemplo, sería la siguiente:

```
inicial:1
finales:11
1->2,E
1->8,E
2->3,E
2->5,E
3->4,a
5->6,b
6->7,E
4->7,E
7->2,E
7->8,E
8->9,a
9->10,b
11->11,b
```

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 desarrollar esta práctica ser diseñaron e implementaron las siguientes clases:

- Automata: Superclase de los autómatas AFN y AFD que contiene los métodos y comportamientos generales de un autómata.
- AFN: Subclase de Automata que representa a un autómata finito no determinista.
- AFD: Subclase de Automata que representa a un autómata finito determinista.
- AFReader: Clase que nos permite leer un archivo .af para obtener un autómata
- Transición: Clase que modela transiciones entre estados de un autómata

Clase Automata
Esta clase es la implementación principal de los autómatas, ya sea AFN o AFD. Esta clase es abstracta con la finalidad de implementar dos métodos abstractos los cuales será implementados por separado por las clases AFN y AFD. Esta clase tendrá como miembros a las características principales de un autómata como lo son las transiciones (List<Transicion>), un estado inicial(int) y un conjunto de estados finales(List<Integer>).

Los métodos de la clase Automata son los siguientes:

- ***public void cargar_desde(String ruta)***: Método que cargará un autómata desde un archivo .af. Como argumento se le pasará la ruta de donde se cuentra el archivo.
- ***public void agregar_transicion(int inicio, int fin, char simbolo):*** Método que nos permite agregar transiciones.
- ***public void eliminar_transicion(int inicio, int fin, char simbolo):*** Método que nos permite eliminar una transición.
- ***public int obtener_inicial():*** Método para obtener el estado inicial del autómata
- ***public List<Integer> obtener_finales():*** Método que obtiene todos los estados finales del autómata.
- ***public void establecer_inicial(int estado):*** Método que establece un estado denotado por ‘estado’ del autómata como inicial.
- ***public void establecer_final(int estado, boolean unico):*** Método que establece un estado como final. Si lo único es verdadero significa que él estado pasado por argumento será ahora el único estado final, de lo contrario se agregará el estado a los estados finales ya existentes.

- ***public boolean esAFD():*** Método que determina si el autómata almacenado es un AFD, basándose en la premisa que partiendo de un estado, sólo es posible pasar a otro único estado mediante un símbolo.
- ***public boolean esAFN():*** Método que determina si el autómata almacenado es un AFN. Llama a es AFD(), si esAFD() retorna true, entonces esAFN() retorna false, de otra forma retorna true.
- ***public List<Transicion> getTrans():*** Método que obtiene todas las transiciones del autómata.
- ***public abstract boolean acepta(String cadena):*** Método abstracto para ser implementado individualmente para determinar si una cadena es aceptada por el autómata
- ***public abstract String generarCadena():*** Método abstracto para ser implementado individualmente para que el autómata genere una cadena.

Clases AFN y AFD

Estas clases se implementan a la superclase Automata por lo que los métodos abstractos acepta(String) y generarCadena() son implementados para ser programados individualmente.

Clase AFReader

Esta clase implementa las funciones necesarias para leer un archivo .af y generar así un autómata.

Esta clase contiene miembros que donde se irá almacenando datos relacionados al autómata como los son las transiciones, los estados finales y el estado inicial. También cuenta con una serie de constantes que se utilizan en los diferentes algoritmos como por ejemplo los separadores “->”, “,” y “:”.

En el constructor se utilizará la ruta del archivo proporcionada como argumento para abrir dicho archivo y leer linea por linea para obtener tanto las transiciones, el estado inicial y los estados finales con las siguientes funciones:

- ***private int obtenerEstadoInicialDelArchivo(String linea):*** A este método se le pasa como argumento la primera línea del archivo af, pues por definición esta contendrá los estados iniciales. Separará de la cadena linea dos mediante el separado “:” y quedará del lada izquierdo la palabra ‘inicial‘ y del lado derecho el estado inicial.
- ***private List<Integer> obtenerEstosFinalesDelArchivo(String linea):*** Este método realizará una función similar con la única diferencia que este realizará

este proceso de separación pues puede encontrar varios estados finales separados por comas.

- ***private List<Transicion> obtenerTransicionesDelArchivo(List<String> lineas):*** Este último método empezará a leer desde la tercera línea del archivo que es en donde empiezan a representar las transiciones. Este método separará en dos cada linea mediante el separado '->' dejando de lado izquierdo el estado de inicio y del otro lado el estado fin concatenado con una coma y el símbolo de la transición. Ahora solo se separará por el separador ',' y así podremos obtener todos los datos necesarios para agregar la transición correspondiente a esa línea del archivo,

Clase Transicion

Esta clase nos sirve para mapear la forma general de una transición en un autómata. Esta cuenta con los miembros para almacenar el estado inicio, el estado de fin y el símbolo con el cual se puede pasar entre ellos.

Pruebas

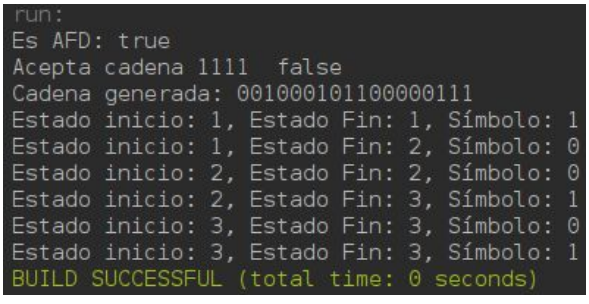
Para realizar las pruebas con estas clases se creó el archivo test2.af con la siguiente descripción:

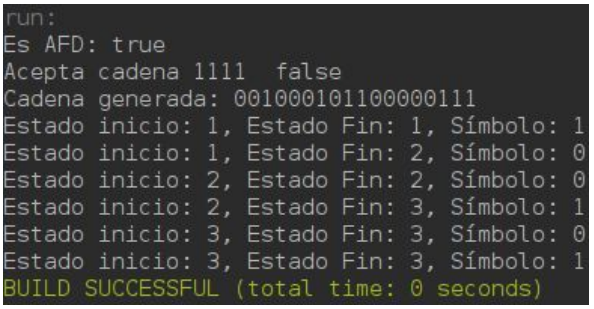
```
AFD afn = new AFD();
afn.cargar_desde("/ruta/ruta/ruta/test2.af");
```

Archivo test2.af:
inicial:1
finales:3
1->1,1
1->2,0
2->2,0
2->3,1
3->3,0
3->3,1

Este autómata fue obtenido del libro de Teoría de Autómatas del autor Hopcroft. Este autómata permite cadenas que contengan 0s y 1s y que terminen con 01.

Una vez cargado el autómata probamos las funciones esAFD(), le probamos si acepta la cadena '1111' , hacemos que genere una cadena e imprimimos las transiciones para corroborar que fueron leídas y guardadas correctamente, teniendo el siguiente resultado en consola dos veces para corroborar la generación aleatoria de la cadena:





Conclusiones

Puedo concluir con el desarrollo de esta práctica que la programación de autómatas puede llegar a ser algo compleja y no muy clara al principio sin contar con ciertas ocasiones en las que puede llegar a ser repetitivo, sin embargo al ya haber tenido cierto tiempo de haber cursado la unidad de aprendizaje Teoría de Autómatas, me vino bastante bien poder refrescar mis recuerdos sobre esa gran variedad de temas y mejorar mis prácticas de programación.

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