



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO



UNIDAD DE APRENDIZAJE

TEORÍA COMPUTACIONAL

TAREA

PRÁCTICA 7

ALUMNO

BARRERA ESTRELLA EMANUEL

PROFESOR

JUÁREZ MARTÍNEZ GENARO

GRUPO

2CM1

Introducción

En los autómatas, se tiene muchas veces el conflicto de la intersección de dos o más autómatas, con este proceso se puede tener un diferente cálculo de posibles estados finales cuando se tiene en cuenta el factor de la combinación de varios eventos, como muchos otros autómatas pueden ser más complejos. En la teoría computacional se pueden hacer uso de varios elementos para facilitar los procesos con un comportamiento menos estable como un autómata finito no determinístico. En las ciencias pueden ser de utilidad en la intersección de lenguajes.

La intersección de varios lenguajes regulares es otro lenguaje regular. Se utiliza la operación de intersección de conjuntos; así, para el alfabeto $S = \{x, y\}$ si $L1 = \{x, xy, yy\}$, $L2 = \{yz, yy\}$ y $L3 = \{y, yy\}$ entonces su intersección será $L1 \cap L2 \cap L3 = \{yy\}$.

Para diseñar el autómata que reconoce este lenguaje, vamos a considerar los autómatas finitos $L(M_n)$. El nuevo autómata estará definido por la quintupla (S', S', d', i', F') , donde:

S' será el producto cartesiano de todos los conjuntos de estados originales $S' = S1 \times S2 \times S3 \times \dots \times Sn$. El alfabeto tiene que ser el mismo para todos los autómatas. $S' = S$. La transición desde el estado $p = (p1, p2, p3, \dots, p_n)$ al estado $q = (q1, q2, q3, \dots, q_n)$ de S' (todas las transiciones posibles), para un símbolo del alfabeto, si y solo si existe la transición desde p_i a q_i para todo $i \in n$. El estado inicial será aquel que está formado por los estados iniciales originales: $i' = (i1, i2, i3, \dots, in)$. Los estados de aceptación serán aquellos que están formados por estados de aceptación originales. $F' = (F1, F2, F3, \dots, Fn)$.

Desarrollo

En esta práctica se deberá de realizar una autómeta que siga un tablero de ajedrez, el cual, consista de un reconocimiento de estados y pueda saber si ganó la pieza a lo largo del tablero, la ficha empezando desde una determinada casilla. El usuario ingresara la cadena con los colores de movimientos que puedan hacer que la pieza se mueva, en cuyo caso exitan cadenas que logren obtener la casilla donde se denomine la posición de ganar, la salida del programa mostrará un historial de estados en un archivo de texto.

Con base a las pautas anteriormente establecidas, se procederá a dar un ligero bosquejo de cómo deberá comportarse el tablero del autómeta. En caso de que se pueda conseguir un buen aprovechamiento, el autómeta como está elaborada, podrá ser modificado y se requerirá que el usuario ingrese una cadena para dos fichas diferentes de las cuales pueda mostrar cómo dos autómetas funcionan simultáneamente y que la problemática principal pueda complicarse mucho más con sólo una pieza más en el tablero.

Sin más preámbulo, de la siguiente forma está estructurado el tablero del ajedrez, enumerando las casillas de forma que sea evidente conocer la trayectoria de las piezas de juego:

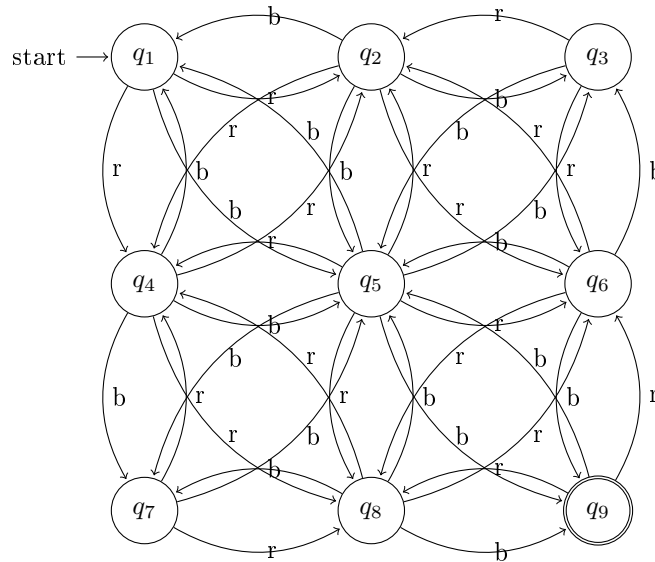
1	2	3
4	5	6
7	6	9

Con lo anterior establecido, se podrá notar que existen muchas dificultades y que se trata de un autómeta finito no determinístico. Con la técnica de las propiedades de los autómetas no determinísticos,

se puede convertir este problema en algo mucho más sencillo a cuenta de no perder su funcionamiento esencial. Siguiendo pues, el mapeo de cada casilla alcanzable por la ficha, el objetivo es que dada una cadena de un lenguaje que consta de dos elementos : (B, R), se pueda formar un criterio y buscar las alternativas de trabajar con múltiples subprocesos. A continuación la simbología de la aplicación.

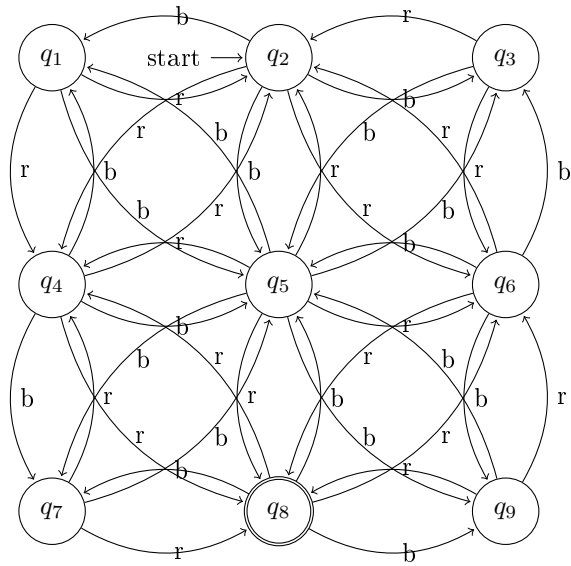
Símbolo	Color
B	Negro
R	Rojo

Así pues, si hablamos de una cadena que pertenece a un universo sigma Σ con todas sus cadenas contenidas en él, contienen todo un conjunto de caminos para la ficha que pueda tomar. Por ejemplo, dado el caso que conozcamos la cadena BBR, nos referimos a una cadena del lenguaje Σ donde la ficha podrá viajar en el orden sucesivo de una casilla de color Negro, seguido nuevamente de otra casilla de color negro y finalmente llegando a una casilla de color rojo, no importa cuáles casillas específicamente, simplemente que cumplan con esa regla. A continuación se puede mejorar la explicación con el grafo para una ficha que comienza desde la casilla 1 y se mueve hasta la casilla 9.



	R	B
\emptyset	\emptyset	\emptyset
$\rightarrow q1$	$\{q2, q4\}$	$\{q5\}$
q2	$\{q4, q6\}$	$\{q1, q3, q5\}$
q3	$\{q2, q6\}$	$\{q5\}$
q4	$\{q2, q8\}$	$\{q1, q5, q7\}$
q5	$\{q2, q4, q6, q8\}$	$\{q1, q3, q7, q9\}$
q6	$\{q2, q8\}$	$\{q3, q5, q9\}$
q7	$\{q4, q8\}$	$\{q5\}$
q8	$\{q4, q6\}$	$\{q5, q7, q9\}$
q9	$\{q6, q8\}$	$\{q5\}$

De lo anterior, podemos idear múltiples de rutas que la ficha pueda recorrer sin importar cuántas veces pase por ellas, en este caso podemos observar como nuevamente se fija la ruta que sea de una casilla que inicie desde la posición 2 hasta la posición 8.



	R	B
\emptyset	\emptyset	\emptyset
q1	{q2,q4}	{q5}
\rightarrow q2	{q4,q6}	{q1,q3,q5}
q3	{q2,q6}	{q5}
q4	{q2,q8}	{q1,q5,q7}
q5	{q2,q4,q6,q8}	{q1,q3,q7,q9}
q6	{q2,q8}	{q3,q5,q9}
q7	{q4,q8}	{q5}
q8	{q4,q6}	{q5,q7,q9}
q9	{q6,q8}	{q5}

Con esto mejor explicado, podemos proceder a la implementación del autómata, para empezar se aclara que se usa el lenguaje Java versión 1.8. El programa deberá reconocer los patrones que puedan tomar las cadenas cuando se inserten en el autómata y éste decidirá si es una ficha ganadora o no.

```

public class Automata7 {

    private ArrayList<ArrayList<Integer>> matriz;
    private ArrayList<ArrayList<Integer>> matriz2;
    public ArrayList<Integer> jugadas [];

    public Automata7() {
        jugadas = new ArrayList [2];
        matriz = new ArrayList<>();
        matriz2 = new ArrayList<>();
    }
}

```

Para el reconocimiento de cadenas, es bueno que la clase tenga definidas variables de instancia. Lo que se presenta es una estructura de datos lo más parecida a una matriz que pueda acumular cantidades mínimas de información para desplegarlas cuando sea requerido.

```

public void analizar(char c) {
    ArrayList<ArrayList<Integer>> backup;
    backup = new ArrayList<>();
    ArrayList<Integer> estados;
    if (matriz.isEmpty()) {
        switch (c) {
            case 'R':
            case 'r':
                estados = new ArrayList<>();
                estados.add(2);
                matriz.add(estados);
                estados = new ArrayList<>();
                estados.add(4);
                matriz.add(estados);
                break;
            case 'B':
            case 'b':
                estados = new ArrayList<>();
                estados.add(5);
                matriz.add(estados);
                break;
        }
    }
    switch (c) {

```

Cuando el autómata pide la cadena, analiza caracter por caracter qué estados va recorriendo y ampliar la matriz de forma que se quede el registro de cuáles estados ha pasado, seguido de esto lo que queda es acumular todos los estados posibles y que se guarden. Después de revisar todos los caracteres, el programa hace un conjunto de sólo las cadenas que son ganadoras:


```

public int finalizar() {
    ArrayList<ArrayList<Integer>> backup;
    backup = new ArrayList<>();
    ArrayList<Integer> estados;
    for (int i = 0; i < matriz.size(); i++) {
        if (matriz.get(i).get(matriz.get(i).size() - 2) == 9) {
            estados = (ArrayList<Integer>) matriz.get(i).clone();
            backup.add(estados);
        }
    }
    matriz.clear();
    matriz = (ArrayList<ArrayList<Integer>>) backup.clone();
    System.out.println("Total de cadenas ganadoras para la ficha 1: "
        + matriz.size());
}

```

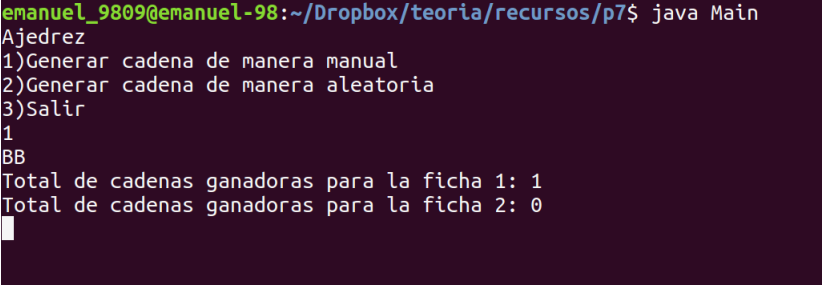
Con esto hecho, se denomina si la cadena ingresada pertenece al lenguaje de todas las cadenas que son ganadoras, con lo anterior, es posible que se puedan recuperar todos los registros de estados gracias a la estructura de datos matriz que se uso previamente.

```

comand("rm_ganadoras1.txt");
comand("echo_>>_ganadoras1.txt");
if (!matriz.isEmpty()) {
    PrintWriter writer;
    try {
        writer = new PrintWriter("ganadoras1.txt", "UTF-8");
        for (int i = 0; i < matriz.size(); i++) {
            for (int j = 0; j < matriz.get(i).size(); j++) {
                writer.print(matriz.get(i).get(j) + "_");
            }
            writer.println("");
        }
        writer.close();
    } catch (FileNotFoundException ex) {
        Logger.getLogger(Automata7.class.getName()).log(Level.
            SEVERE, null, ex);
    } catch (UnsupportedEncodingException ex) {
        Logger.getLogger(Automata7.class.getName()).log(Level.
            SEVERE, null, ex);
    }
}
}

```

Lo único que queda es escribir la salida del programa, simplemente usamos un fichero de un archivo de texto en el que escribimos el registro de todos los estados por los que pasó el autómata. A continuación la ejecución del programa:

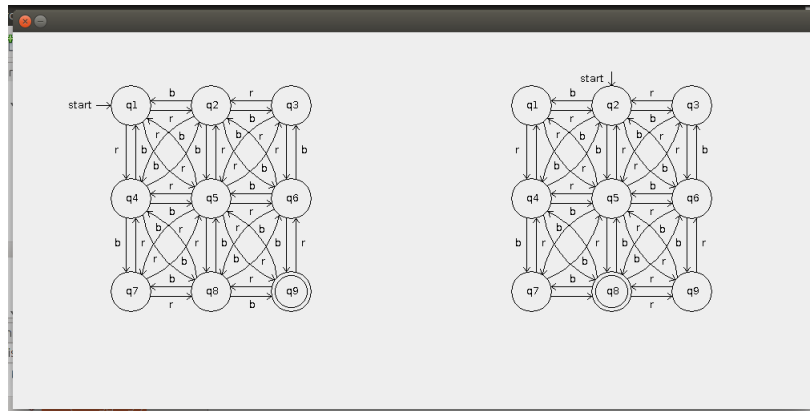


```

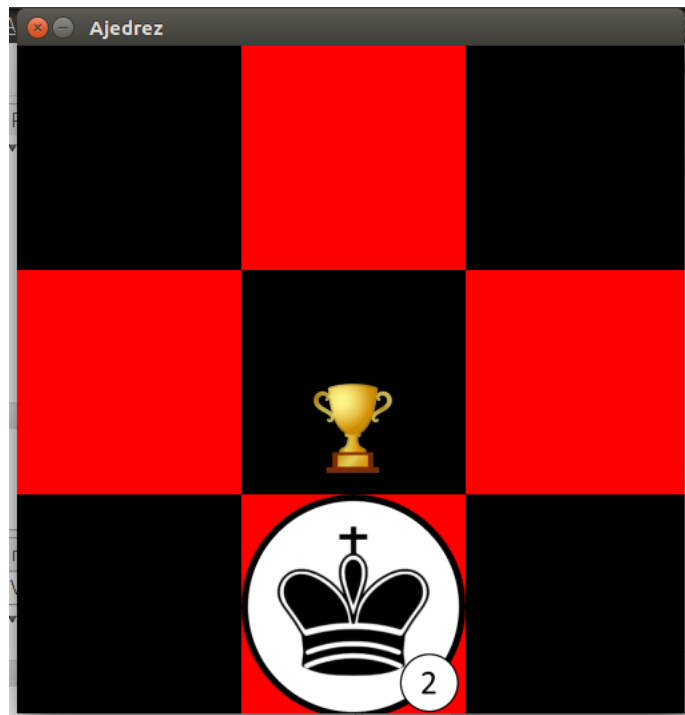
emanuel_9809@emanuel-98:~/Dropbox/teoria/recursos/p7$ java Main
Ajedrez
1)Generar cadena de manera manual
2)Generar cadena de manera aleatoria
3)Salir
1
BB
Total de cadenas ganadoras para la ficha 1: 1
Total de cadenas ganadoras para la ficha 2: 0

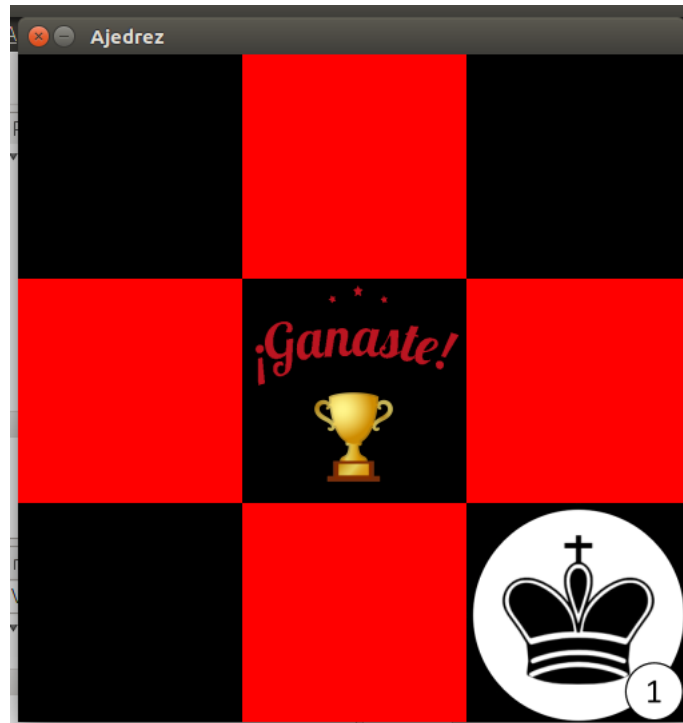
```

Después de esto, se muestra el grafo hecho en Graphics de Java que muestra el comportamiento de los autómatas:



Y finalmente, hecho en Java Swing, una ligera demostración del comportamiento del programa con las imágenes, dándole una buena presentación y una interfaz amigable. Es bueno aclarar que cada autómata puede intersectarse en el partido de la llegada hasta la casilla ganadora, pero como se muestra a continuación es evidente que llega a ser muy complicado:





Referencias

Hopcroft, John E, 1939- Introduction to automata theory, languages, and computation / John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. - 2nd ed.

http://aconute.es/computacion/automatasFinitos/ejemplo_intersec.html