Instituto Politécnico Nacional

ESCUELA SUPERIOR DE CÓMPUTO

MÁQUINAS DE BRUJIN Y EXPRESIONES REGULARES DE AUTÓMATAS CELULARES ELEMENTALES

Autor:

Juan Carlos Garcia Medina

PROFESOR:
Dr. Genaro Juárez Martínez

MATERIA:
Sistemas Complejos

, 6 de abril de 2020

Índice general

Capítulo 1

Diagramas de Bruijn

Los diagramas de Bruijn fueron originalmente propuestos en teoría de registro de desplazamiento [1] y son grafos cuyos nodos son secuencias de símbolos de algún alfabeto, por ejemplo el conjunto de estados de un autómata. Las secuencias tienen la misma longitud, comúnmente llamado número de etapas debido a una aplicación a la teoría de desplazamiento de registro.

Un diagrama de Bruijn también es representado por su matriz de conexión; Para estos fines es conveniente representar los estados de un automata celular ECA(k,r) por los dígitos $0,1,...,k-1$, sus vecindades parciales de números de $2r$ dígitos de base k.[2]

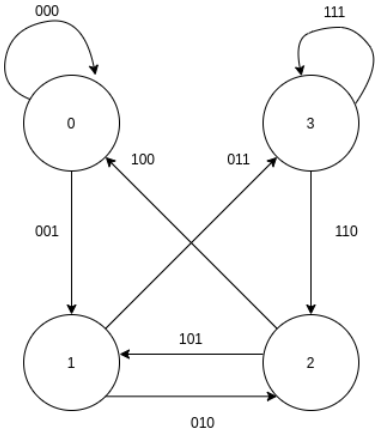

Figura 1.1: Diagrama de bruijn genérica para ECA(2, 1))

$$M_{R22} = \begin{bmatrix} 0 & 1 & . & . \\ . & . & 1 & 0 \\ 1 & 0 & . & . \\ . & . & 0 & 0 \end{bmatrix} \tag{1.1}$$

La matriz de conexión M correspondiente al diagrama de Bruijn de la Figura 1.1 se define como sigue:

$$M_{i,j} = \begin{cases} 1, & \text{if } j \in \{ki, ki+1, ..., ki+k-1 (\text{mod} k^{2r})\} \\ 0, & \text{en otro caso} \end{cases} \tag{1.2}$$

donde modulo k^{2r} representa el número de vértices y j toma valores de $\{ki, ki+1, ..., ki+k-1(\text{mod} k^{2r})\}$. Por lo tanto ECA$(2,1)$ es:

$$M_{i,j} = \begin{cases} 1, & \text{if } j \in \{2i, 2i+1 (\text{mod} 4)\} \\ 0, & \text{en otro caso} \end{cases} \tag{1.3}$$

Con la disposición de estas ecuaciones es posible crear un programa que permita generar la matriz de adyacencia utilizando específicamente 1.3.

```
#include <bits/stdc++.h>

using namespace std;
const int k = 2;

const int MOD = k*k; //k^(2*r)
bool condition(int i, int j)
{
    if( (j == k*i %MOD) || j == ((k*i +1) %MOD) )
        return true;
    return false;
}

int main()
{

```

```
    vector <vector <int>> M(4, vector <int>(4));
    cout << "Matrix adj" << endl;
    for(int i = 0; i < 4; i++)
    {
        for(int j = 0; j < 4; j++)
        {
            if(condition(i, j))
                M[i][j] = 1;
            else
                M[i][j] = 0;
        }
    }

    for(int i = 0; i < 4; i++)
    {
        for(int j = 0; j < 4; j++)
            cout << M[i][j] <<" ";
        cout << endl;
    }
```

Obteniendo como salida la matriz de adyacencia genérica del diagrama de Bruijn para ECA(2, 1).

```
Matrix adj
1 1 0 0
0 0 1 1
1 1 0 0
0 0 1 1
```

Para el caso de las etiquetas de cada arista del grafo también es posible realizar un programa y encontrar los traslapes y la conexión de vertices para cada arista.

```

    string binary [4]= {"00", "01", "10", "11"};
    cout << endl << "labels" << endl;

    for(int i = 0; i < 4; i++)
    {
```

```
        for(int j = 0; j < 4; j++)
        {
            if(binary[i][1] == binary[j][0])
            {
                cout << i << " " << j << endl;
                cout << binary[i] << " . " << binary[j] << " = " << binary[i
    ][0] << binary[i][1] << binary[j][1] << " = ";
            }
        }
    }
```

En la salida podemos encontrar que el vértice 0 hacia el vértice 0 tiene la etiqueta **000**, el vértice 0 hacia el vértice 1 tiene la etiqueta **001** y así sucesivamente.

```

labels
0 0
00 . 00 = 000
0 1
00 . 01 = 001
1 2
01 . 10 = 010
1 3
01 . 11 = 011
2 0
10 . 00 = 100
2 1
10 . 01 = 101
3 2
11 . 10 = 110
3 3
11 . 11 = 111
```

Estos resultados son de una máquina genérica de un ECA(2, 1), sin embargo con ligeras modificaciones podemos encontrar las conexiones específicas para las reglas propuestas:

```
    const int rules[6] = {15, 22, 30, 54, 90, 110};

    for(int rule: rules)
    {
        cout << "Regla: " << rule << endl;
        bitset<8> current(rule);
        //cout << current.to_string() << endl;
        for(int i = 0; i < 4; i++)
        {
            for(int j = 0; j < 4; j++)
            {
                if(binary[i][1] == binary[j][0])
                {
                    cout << i << " " << j << endl;
                    cout << binary[i] << " . " << binary[j] << " = " <<
    binary[i][0] << binary[i][1] << binary[j][1] << " = ";

                    int representation = (binary[j][1]-48) + (binary[i
    ][1]-48) * 2 + (binary[i][0]-48)*4;
                    //cout <<"rep: "<< representation << endl;
                    cout << current[representation] << endl;
                }
            }
        }
    }

    return 0;
}
```

Ver código y reporte

1.1. Regla 15

```
Regla: 15
0 0
00 . 00 = 000 = 1
0 1
00 . 01 = 001 = 1
1 2
01 . 10 = 010 = 1
1 3
01 . 11 = 011 = 1
2 0
10 . 00 = 100 = 0
2 1
10 . 01 = 101 = 0
3 2
11 . 10 = 110 = 0
3 3
11 . 11 = 111 = 0
```

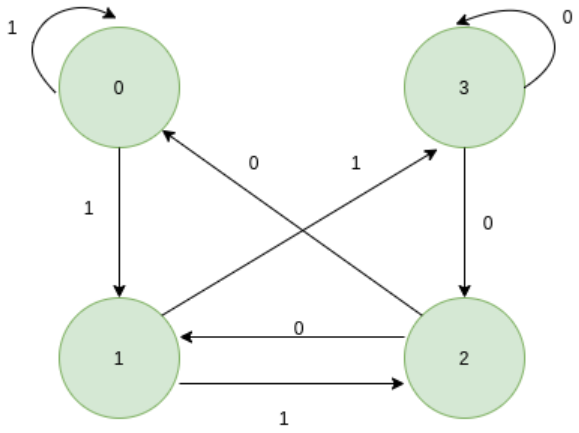

Figura 1.2: Diagrama de Bruijn regla 15

1.2. Regla 22

```
Regla: 22
0 0
00 . 00 = 000 = 0
0 1
00 . 01 = 001 = 1
1 2
01 . 10 = 010 = 1
1 3
01 . 11 = 011 = 0
2 0
10 . 00 = 100 = 1
2 1
10 . 01 = 101 = 0
3 2
11 . 10 = 110 = 0
3 3
11 . 11 = 111 = 0
```

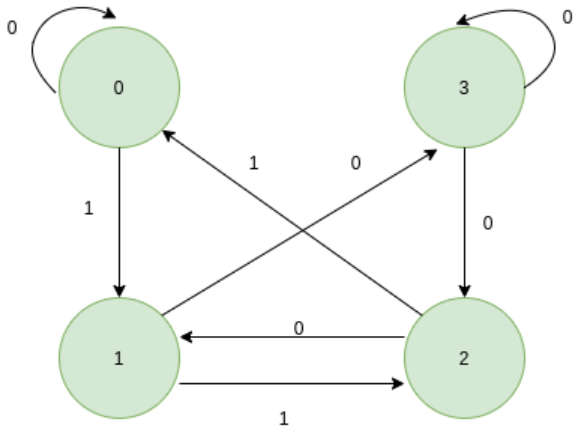

Figura 1.3: Diagrama de Bruijn regla 22

1.3. Regla 30

```
Regla: 30
0 0
00 . 00 = 000 = 0
0 1
00 . 01 = 001 = 1
1 2
01 . 10 = 010 = 1
1 3
01 . 11 = 011 = 1
2 0
10 . 00 = 100 = 1
2 1
10 . 01 = 101 = 0
3 2
11 . 10 = 110 = 0
3 3
11 . 11 = 111 = 0
```

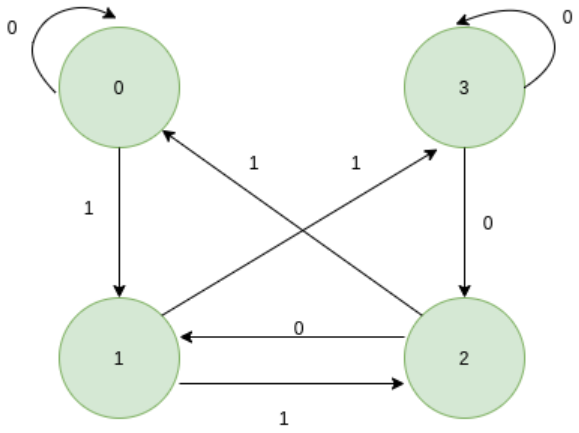

Figura 1.4: Diagrama de Bruijn regla 30

1.4. Regla 54

```
Regla: 54
0 0
00 . 00 = 000 = 0
0 1
00 . 01 = 001 = 1
1 2
01 . 10 = 010 = 1
1 3
01 . 11 = 011 = 0
2 0
10 . 00 = 100 = 1
2 1
10 . 01 = 101 = 1
3 2
11 . 10 = 110 = 0
3 3
11 . 11 = 111 = 0
```

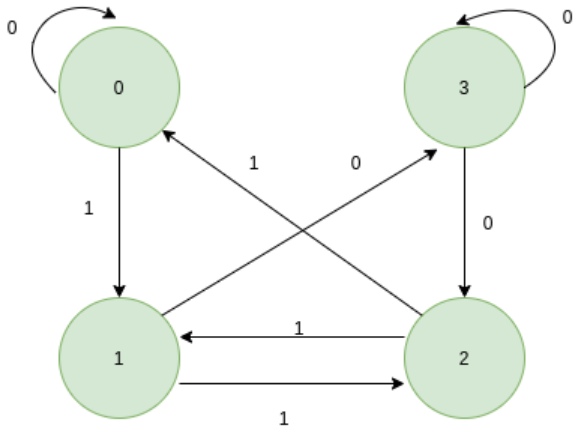

Figura 1.5: Diagrama de Bruijn regla 54

1.5. Regla 90

```
Regla: 90
0 0
00 . 00 = 000 = 0
0 1
00 . 01 = 001 = 1
1 2
01 . 10 = 010 = 0
1 3
01 . 11 = 011 = 1
2 0
10 . 00 = 100 = 1
2 1
10 . 01 = 101 = 0
3 2
11 . 10 = 110 = 1
3 3
11 . 11 = 111 = 0
```

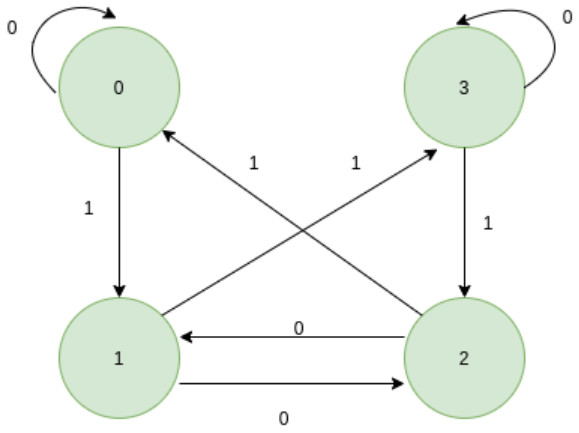

Figura 1.6: Diagrama de Bruijn regla 90

1.6. Regla 110

```
Regla: 110
0 0
00 . 00 = 000 = 0
0 1
00 . 01 = 001 = 1
1 2
01 . 10 = 010 = 1
1 3
01 . 11 = 011 = 1
2 0
10 . 00 = 100 = 0
2 1
10 . 01 = 101 = 1
3 2
11 . 10 = 110 = 1
3 3
11 . 11 = 111 = 0
```

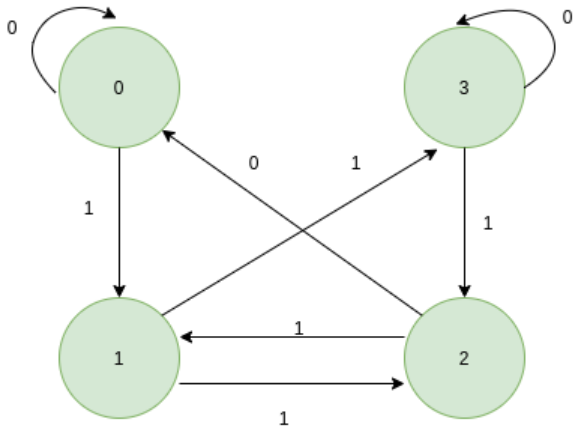

Figura 1.7: Diagrama de Bruijn regla 110

Capítulo 2

Ecuaciones Simbólicas

Para proceder al cálculo de la expresión regular es posible utilizar la recurrencia de la ecuación 2.1.

$$R_{i,j}^{k} = R_{i,j}^{k-1} + R_{i,k}^{k-1}(R_{k,k}^{k-1})^{*}R_{k,j}^{k-1} \tag{2.1}$$

donde i es el estado inicial y j es el estado final. El caso base cuando $k = 0$ es el camino directo a cada nodo.

2.1. Regla 22

Ecuación	Expresión
1	$(0+1)^*$
2	$(01)^*$
3	$(001)^*$
4	$11(01)^*00$
5	$(01)^*(0+1)$
6	$(000111)^*$
7	$(0+0(01)^*00)^*$
8	$(((01)^*+0)00^*1)^*$
9	$(0+1)+11(01)^*(0+1)$
10	$((01)^*00)(0+0(01)^*00)^*$
11	$(11(01)^*00)(0+0(01)^*00)^*$
12	$(0+0(01)^*00)^*0(01)^*(0+1)$
13	$(0+1(01)^*00)(0+0(01)^*00)^*$
14	$((0+1)+11(01)^*1)+$ $(11(01)^*00)(0+$ $0(01)^*00)^*(0(01)^*(0+1))$
15	$(0^*10^*)(10^*10+(10^*+10^*10)0^*)^*$

Expresiones regulares derivadas de la regla 22 calculadas con la recurrencia 2.1
Tabla consultada de [3]

Bibliografía

[1] H. V. McIntosh, "De Bruijn diagrams." http://delta.cs.cinvestav.mx/~mcintosh/oldweb/ra/node7.html, 2005. mcintosh@servidor.unam.mx. 1

[2] J. C. S. T. M. Genaro Juárez Martínez, Harold V. McIntosh and S. V. C. Vergara, "Determining a regular language by glider-based structures called phases f_i_1 in rule 110," *Journal of Cellular Automata*. 1

[3] R. H. D. D. I. Z. Genaro Juárez Martínez, Andrew Adamatzky, "On patterns and dynamics of rule 22 cellular automaton," *Complex Systems*, 2019. 13