
INSTITUTO POLITÉCNICO NACIONAL,
ESCUELA SUPERIOR DE CÓMPUTO

REPORTE: Diagramas de Bruijn y ecuaciones

SISTEMAS COMPLEJOS

Oscar Andrés Rosas Hernandez

2 de abril de 2020

Índice general

1. Diagramas de Bruijn	2
1.1. Regla 15	4
1.2. Regla 22	4
1.3. Regla 30	4
1.4. Regla 54	4
1.5. Regla 90	4
1.6. Regla 110	4
2. Ecuaciones simbólicas (expresiones regulares)	5

Capítulo 1

Diagramas de Bruijn

Los diagramas de Bruijn proporcionan una manera conveniente de describir las configuraciones de autómatas celulares (CA).

Sea Σ el alfabeto y $s \geq 1$ un número, y el grafo de Bruijn $B(s, \Sigma)$ como sigue: $B(s, \Sigma)$ tiene un conjunto de vértices Σ^s y vertices (ax, xb) para todo $a, b \in \Sigma, x \in \Sigma^s$.

Para calcular estos tuve que mostrar las reglas de una manera que me permitiera facilmente crear los diagramas, para eso cree este pequeño programa en C++ que me permite mostrarlas de una manera comoda ss:

```
#include <bitset>
#include <cstdint>
#include <iostream>
using namespace std;

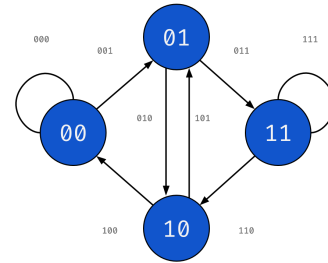
auto show_rules(const uint8_t rule_id) {
    const auto rule = bitset<8>{rule_id};

    cout << "Rule " << rule.to_ulong() << endl;
    for (auto i = 0; i < 8; ++i)
        cout << bitset<3>(i) << " -> " << rule[i] << endl;

    cout << endl;
}

auto main() -> int {
    const auto rules = {15, 22, 30, 54, 90, 110};
    for (const auto rule : rules)
        show_rules(rule);

    return 0;
}
```



Dando este resultado:

Rule 15	Rule 22	Rule 30	Rule 54	Rule 90	Rule 110
000 -> 1	000 -> 0	000 -> 0	000 -> 0	000 -> 0	000 -> 0
001 -> 1	001 -> 1	001 -> 1	001 -> 1	001 -> 1	001 -> 1
010 -> 1	010 -> 1	010 -> 1	010 -> 1	010 -> 0	010 -> 1
011 -> 1	011 -> 0	011 -> 1	011 -> 0	011 -> 1	011 -> 1
100 -> 0	100 -> 1	100 -> 1	100 -> 1	100 -> 1	100 -> 0
101 -> 0	101 -> 0	101 -> 0	101 -> 1	101 -> 0	101 -> 1
110 -> 0	110 -> 0	110 -> 0	110 -> 0	110 -> 1	110 -> 1
111 -> 0	111 -> 0	111 -> 0	111 -> 0	111 -> 0	111 -> 0

Podemos editarlo un poco para saber el valor de las conexiones del diagrama:

```
#include <bitset>
#include <cstdint>
#include <iostream>
using namespace std;

auto show_rules(const uint8_t rule_id) {
    const auto rule = bitset<8>{rule_id};

    cout << "Rule " << rule.to_ulong() << endl;
    for (auto i = 0; i < 8; ++i) {
        const auto total = bitset<3>(i).to_string();
        const auto start = total.substr(0, 2);
        const auto end = total.substr(1, 2);
        cout << start << " to " << end << " -> " << rule[i] << endl;
    }
    cout << endl;
}

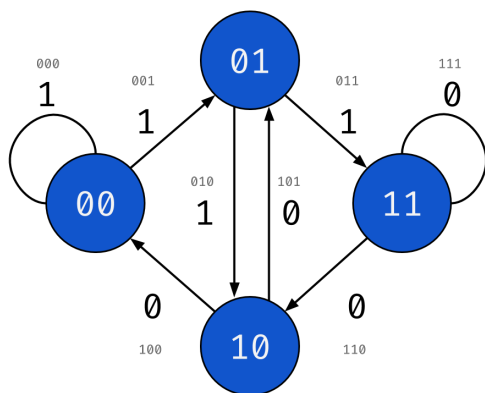
auto main() -> int {
    const auto rules = {15, 22, 30, 54, 90, 110};
    for (const auto rule : rules)
        show_rules(rule);

    return 0;
}
```

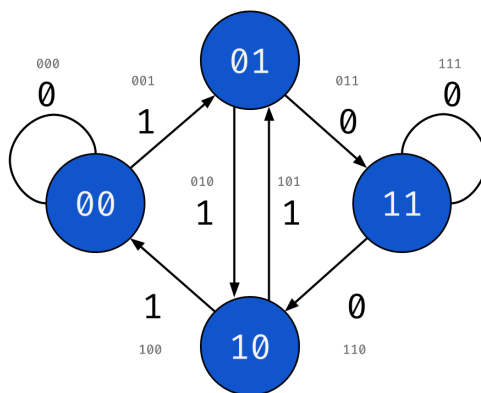
Rule 15 00 to 00 -> 1 00 to 01 -> 1 01 to 10 -> 1 01 to 11 -> 1 10 to 00 -> 0 10 to 01 -> 0 11 to 10 -> 0 11 to 11 -> 0	Rule 30 00 to 00 -> 0 00 to 01 -> 1 01 to 10 -> 1 01 to 11 -> 1 10 to 00 -> 1 10 to 01 -> 0 11 to 10 -> 0 11 to 11 -> 0	Rule 90 00 to 00 -> 0 00 to 01 -> 1 01 to 10 -> 0 01 to 11 -> 1 10 to 00 -> 1 10 to 01 -> 0 11 to 10 -> 1 11 to 11 -> 0
Rule 22 00 to 00 -> 0 00 to 01 -> 1 01 to 10 -> 1 01 to 11 -> 0 10 to 00 -> 1 10 to 01 -> 0 11 to 10 -> 0 11 to 11 -> 0	Rule 54 00 to 00 -> 0 00 to 01 -> 1 01 to 10 -> 1 01 to 11 -> 0 10 to 00 -> 1 10 to 01 -> 1 11 to 10 -> 0 11 to 11 -> 0	Rule 110 00 to 00 -> 0 00 to 01 -> 1 01 to 10 -> 1 01 to 11 -> 1 10 to 00 -> 0 10 to 01 -> 1 11 to 10 -> 1 11 to 11 -> 0

Con esto listo podemos calcular facilmente los diagramas de Bruijn.

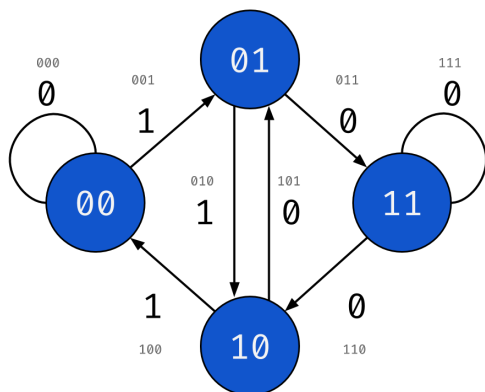
1.1. Regla 15



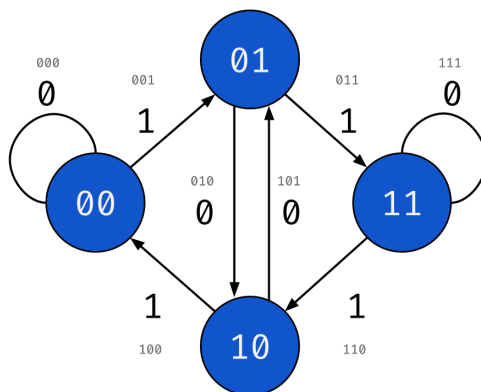
1.4. Regla 54



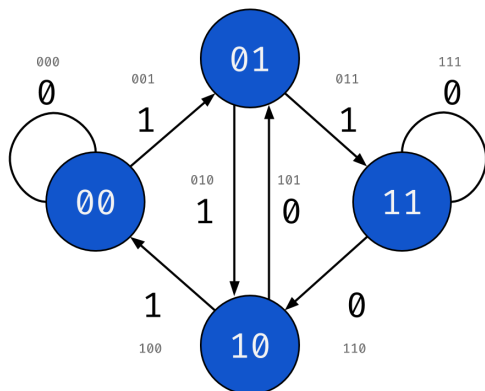
1.2. Regla 22



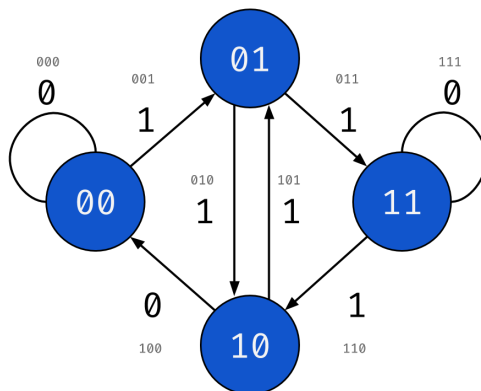
1.5. Regla 90



1.3. Regla 30



1.6. Regla 110



Capítulo 2

Ecuaciones simbólicas (expresiones regulares)

Una vez que construyeron los diagramas podemos calcular todas las ecuaciones simbólicas (expresiones regulares). Hasta calcular la ecuación que representa todo el autómata.

La expresión mas importante es:

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

Podemos crear un programa que se encargue de generar estas ecuaciones:

```
const nonEmpty = (x: string) => (x === "" ? "0" : x);

const parenthesis = (x: string) => {
  if (x.length < 2) return x;
  if (x[x.length - 1] === ")") return x;
  if (x.split("").every(e => e === "1" || e === "0")) return x;

  return `(${x})`;
};

const or = (x: Array<string>) => {
  const elements = x.filter(e => e).filter(e => e !== "");
  return [...new Set(elements)].join(" + ");
};

const and = (x: Array<string>) => {
  if (x.some(e => e === "")) return "";
  return x
    .filter(e => e)
    .filter(e => e !== "ε")
    .filter(e => e !== "ε^*")
    .map(e => parenthesis(e))
    .join("");
};

const kleeneClosure = (x: string) => {
```

```

const data = x.split(" + ");
const result = data.filter(e => e !== "ε").join(" + ");

if (!result) return "";

return `(${result}^*)`;
};

const path = [
  ["", "", "", ""],
  ["", "", "", ""],
  ["", "", "", ""],
  ["", "", "", ""]
];

path[0][0] = "0";
path[0][1] = "1";
path[1][2] = "1";
path[1][3] = "0";
path[2][0] = "1";
path[2][1] = "0";
path[3][2] = "0";
path[3][3] = "0";

path[1][1] = "ε";
path[2][2] = "ε";

path[0][2] = "∅";
path[0][3] = "∅";
path[1][0] = "∅";
path[2][3] = "∅";
path[3][0] = "∅";
path[3][1] = "∅";

path[0][0] = path[0][0] + " + ε";
path[3][3] = path[3][3] + " + ε";

function R(i: number, j: number, k: number): string {
  if (k === 0) return path[i][j];

  //console.log(`${a}+${b}(${c})^*${d}`);
  const a = R(i, j, k - 1);

  const b = parenthesis(R(i, k, k - 1));
  const c = R(k, k, k - 1);
  let d = parenthesis(R(k, j, k - 1));

  if (c === "ε" && d === "ε") d = "";

  const concat = and([b, kleeneClosure(c), d]);
  return nonEmpty(or([a, concat]));
}

for (let k = 0; k < 4; ++k) {
  for (let i = 0; i < 4; ++i) {
    for (let j = 0; j < 4; ++j) {
      console.log(`R(${i}, ${j}, ${k}) = ` + R(i, j, k));
    }
  }
  console.log();
}

```

Logrando estas ecuaciones para la regla 22:

```

R(0, 0, 0) = 0 + ε
R(0, 1, 0) = 1
R(0, 2, 0) = ∅
R(0, 3, 0) = ∅
R(1, 0, 0) = ∅
R(1, 1, 0) = ε
R(1, 2, 0) = 1
R(1, 3, 0) = 0
R(2, 0, 0) = 1
R(2, 1, 0) = 0
R(2, 2, 0) = ε
R(2, 3, 0) = ∅
R(3, 0, 0) = ∅
R(3, 1, 0) = ∅
R(3, 2, 0) = 0
R(3, 3, 0) = 0 + ε

R(0, 0, 1) = 0 + ε
R(0, 1, 1) = 1
R(0, 2, 1) = 11
R(0, 3, 1) = 10
R(1, 0, 1) = ∅
R(1, 1, 1) = ε
R(1, 2, 1) = 1
R(1, 3, 1) = 0
R(2, 0, 1) = 1
R(2, 1, 1) = 0
R(2, 2, 1) = ε + 01
R(2, 3, 1) = 00
R(3, 0, 1) = ∅
R(3, 1, 1) = ∅
R(3, 2, 1) = 0
R(3, 3, 1) = 0 + ε

R(0, 0, 2) = 0 + ε + 11(01~*)1
R(0, 1, 2) = 1 + 11(01~*)0
R(0, 2, 2) = 11 + 11(01~*)(ε + 01)
R(0, 3, 2) = 10 + 11(01~*)00
R(1, 0, 2) = 1(01~*)1
R(1, 1, 2) = ε + 1(01~*)0
R(1, 2, 2) = 1 + 1(01~*)(ε + 01)
R(1, 3, 2) = 0 + 1(01~*)00
R(2, 0, 2) = 1 + (ε + 01)(01~*)1
R(2, 1, 2) = 0 + (ε + 01)(01~*)0
R(2, 2, 2) = ε + 01 + (ε + 01)(01~*)(ε + 01)
R(2, 3, 2) = 00 + (ε + 01)(01~*)00
R(3, 0, 2) = 0(01~*)1
R(3, 1, 2) = 0(01~*)0
R(3, 2, 2) = 0 + 0(01~*)(ε + 01)
R(3, 3, 2) = 0 + ε + 0(01~*)00

R(0, 0, 3) = 0 + ε + 11(01~*)1 + (10 + 11(01~*)00)(0 + 0(01~*)00~*)(0(01~*)1)
R(0, 1, 3) = 1 + 11(01~*)0 + (10 + 11(01~*)00)(0 + 0(01~*)00~*)(0(01~*)0)
R(0, 2, 3) = 11 + 11(01~*)(ε + 01) + (10 + 11(01~*)00)(0 + 0(01~*)00~*)0 + 0(01~*)(ε + 01)
R(0, 3, 3) = 10 + 11(01~*)00 + (10 + 11(01~*)00)(0 + 0(01~*)00~*)(0 + ε + 0(01~*)00)
R(1, 0, 3) = 1(01~*)1 + (0 + 1(01~*)00)(0 + 0(01~*)00~*)(0(01~*)1)
R(1, 1, 3) = ε + 1(01~*)0 + (0 + 1(01~*)00)(0 + 0(01~*)00~*)(0(01~*)0)
R(1, 2, 3) = 1 + 1(01~*)(ε + 01) + (0 + 1(01~*)00)(0 + 0(01~*)00~*)0 + 0(01~*)(ε + 01)
R(1, 3, 3) = 0 + 1(01~*)00 + (0 + 1(01~*)00)(0 + 0(01~*)00~*)(0 + ε + 0(01~*)00)
R(2, 0, 3) = 1 + (ε + 01)(01~*)1 + (00 + (ε + 01)(01~*)00)(0 + 0(01~*)00~*)(0(01~*)1)
R(2, 1, 3) = 0 + (ε + 01)(01~*)0 + (00 + (ε + 01)(01~*)00)(0 + 0(01~*)00~*)(0(01~*)0)
R(2, 2, 3) = ε + 01 + (ε + 01)(01~*)(ε + 01) + (00 + (ε + 01)(01~*)00)(0 + 0(01~*)00~*)0 + 0(01~*)(ε + 01)
R(2, 3, 3) = 00 + (ε + 01)(01~*)00 + (00 + (ε + 01)(01~*)00)(0 + 0(01~*)00~*)(0 + ε +

```



```

0(01~*)00)
R(3, 0, 3) = 0(01~*)1 + (0 + ε + 0(01~*)00)(0 + 0(01~*)00~*)(0(01~*)1)
R(3, 1, 3) = 0(01~*)0 + (0 + ε + 0(01~*)00)(0 + 0(01~*)00~*)(0(01~*)0)
R(3, 2, 3) = 0 + 0(01~*)(ε + 01) + (0 + ε + 0(01~*)00)(0 + 0(01~*)00~*)0 + 0(01~*)(ε
+ 01)
R(3, 3, 3) = 0 + ε + 0(01~*)00 + (0 + ε + 0(01~*)00)(0 + 0(01~*)00~*)(0 + ε +
0(01~*)00)

```

[3]

Bibliografía

- [1] *Cellular Automata*. Jarkko Kari, Spring 2013
<https://www.cs.tau.ac.il/~nachumd/models/CA.pdf>
- [2] *A New Kind of Science*. Wolfram Stephen, 2002
- [3] *On Patterns and Dynamics of Rule 22 Cellular Automaton*. Genaro J. Martínez, Andrew Adamatzky, Rolf Hoffmann, Rennes, France, Ivan Zelinka
- [4] *Introduccion a los automatas celulares elementales*. Carlos Zacarias Reyes Martinez
Escuela Superior de Computo