Autómata Celular - Reglas y Generador de Atractores

Frías Mercado Carlos Elliot

November 7, 2017

3CM6

Computing Selected Topics

Prof: Genaro Juárez Martínez

Contents

1 AC - Reglas

1.1 Descripción del programa

Este programa permite simular las 256 reglas de producción de los autómatas celulares, permitiendo a su vez guardar o cargar archivos, cambiar colores, la velocidad y tamaño del anillo a evaluar por generación por cada iteración.

1.2 Pruebas del funcionamiento

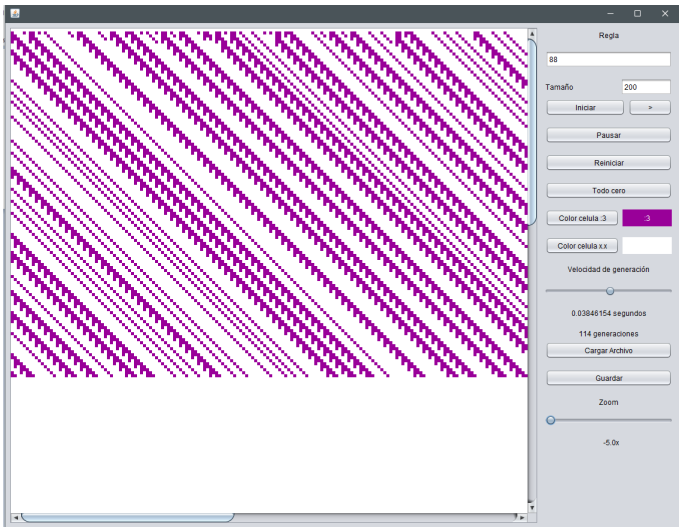


Figura 1: Inicialización del simulador con valores aleatorios

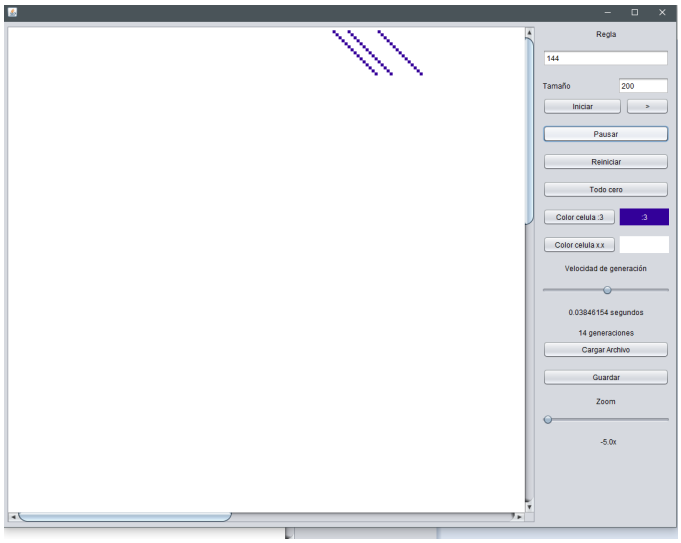


Figura 2: Cambio de colores del programa y patrón introducido por el usuario

Figura 3: Aspecto del Archivo Guardado

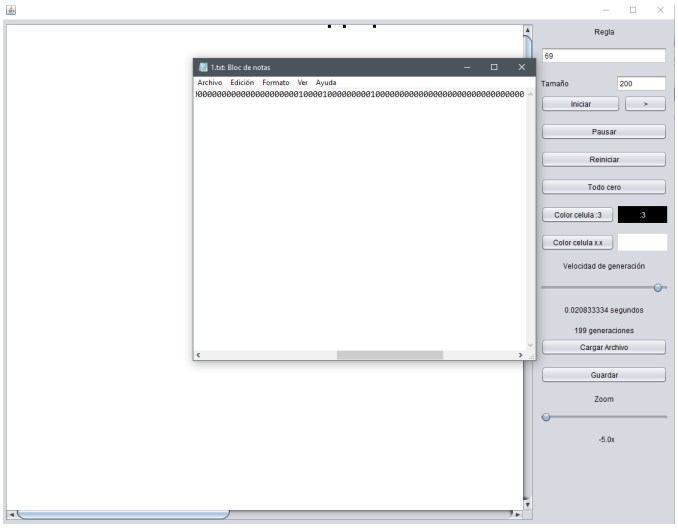


Figura 4: Archivo Cargado

1.3 Codigo fuente

1.3.1 Clase Universo

```
/*  * To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.  */
package gol;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseWheelEvent;
import java.awt.event.MouseWheelListener;
import javax.swing.JPanel;
import logica.Dios;
import logica.Vida;
/**  *  * @author Maku  */
class Universo extends JPanel  {
 int[][] m;
   int tam, click, tamCelula = 10, tiempo; //tam = num de celulas
   private Graphics2D g2d;
 Color viva,  muerta;
```

```
int[] regla = new int[4];
Dios gg = new Dios();
private int zoom = 0;
private static final double ZOOM_AMOUNT = 1.1;
    Universo ()             {                               }
Universo(int[] regla, int tam, int tiempo, Color vi, Color mu)
{
  this.regla = regla;
     this.tam = tam;
          setPreferredSize(new Dimension(tam*tamCelula,tam*tamCelula)); //x, y
          m = gg.creaVida(tam);
     this.tiempo = tiempo;
 viva = vi;
  muerta = mu;
  this .addMouseListener(new MouseListener()
{
  @Override
                       public void mouseClicked(MouseEvent e)
                         {
                       int x, y;
Point p = e.getPoint();
y = (int) Math.floor((float)p.getX()/tamCelula);
x = (int) Math.floor((float)p.getY()/tamCelula);
if(x < tam && y < tam)
{
if(m[x][y] == 1)
{
m[x][y] = 0;
}
else
{
m[x][y] = 1;
}
click = 1;
repaint();
}
}

@Override
public void mousePressed(MouseEvent e) {
//throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.
}

@Override
public void mouseReleased(MouseEvent e) {
```

```
// throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.
}

@Override
public void mouseEntered(MouseEvent e) {
//  throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.
}

@Override
public void mouseExited(MouseEvent e) {
//  throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.
}                                                    });

Universo(int[] regla, int tam, int tiempo, int cero, Color vi, Color mu)
{
this.regla = regla;
this.tam = tam;
setPreferredSize(new Dimension(tam*tamCelula,tam*tamCelula)); //x, y
m = gg.destruyeVida(tam);
this.tiempo = tiempo;
viva = vi;
muerta = mu;
this .addMouseListener(new MouseListener() {
@Override
public void mouseClicked(MouseEvent e)
{
int x, y;
Point p = e.getPoint();
y = (int) Math.floor((float)p.getX()/tamCelula);
x = (int) Math.floor((float)p.getY()/tamCelula);
 if(x < tam && y < tam)
{
if(m[x][y] == 1)
{
m[x][y] = 0;
}
else
{
m[x][y] = 1;
}
click = 1;
repaint();
}
```

```
}

@Override
public void mousePressed(MouseEvent e) {
//throw new UnsupportedOperationException("Not supported yet."); //To change body of ge

@Override
public void mouseReleased(MouseEvent e) {
// throw new UnsupportedOperationException("Not supported yet."); //To change body of g

@Override
public void mouseEntered(MouseEvent e) {
//  throw new UnsupportedOperationException("Not supported yet."); //To change body of

@Override
public void mouseExited(MouseEvent e) {
throw new UnsupportedOperationException("Not supported yet."); //To change body of gene
}

Universo(int[] regla, int tam, int tiempo, int[][] car, Color vi, Color mu)
{
this.regla = regla;
this.tam = tam-2;

System.out.println(this.tam);
setPreferredSize(new Dimension(tam*tamCelula,tam*tamCelula)); //x, y
m = car;
this.tiempo = tiempo;
viva = vi;
click = 1;
muerta = mu;
this .addMouseListener(new MouseListener() {
@Override
public void mouseClicked(MouseEvent e)
{
int x, y;
Point p = e.getPoint();
y = (int) Math.floor((float)p.getX()/tamCelula);
x = (int) Math.floor((float)p.getY()/tamCelula);
if(x < tam && y < tam)
{
if(m[x][y] == 1)
{
m[x][y] = 0;
}
else
```

```
{

m[x][y] = 1;
}
click = 1;
repaint();
}
}

@Override
public void mousePressed(MouseEvent e) {
//throw new UnsupportedOperationException("Not supported yet."); //To change body of ge

@Override
public void mouseReleased(MouseEvent e) {
// throw new UnsupportedOperationException("Not supported yet."); //To change body of g

@Override
public void mouseEntered(MouseEvent e) {
//  throw new UnsupportedOperationException("Not supported yet."); //To change body of

@Override
public void mouseExited(MouseEvent e) {
//  throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.

});
}
@Override
public void paintComponent(Graphics g)
{             super.paintComponent(g);
if(click == 1)
{
click = 0;
// repaint();
}
else
      {
m = analiza(m, regla);
}
g2d = (Graphics2D) g;
for(int i = 1; i <= tam; i++)
{
for(int j = 1; j <= tam; j++)
if(m[i][j] == 1)
{
```

```
g2d.setColor(viva);
g2d.fillRect(tamCelula*j, tamCelula*i, tamCelula, tamCelula); //x, y, tamx, tamy
g2d.drawRect(tamCelula*j, tamCelula*i, tamCelula, tamCelula); //x, y, tamx, tamy //x, y
}                              else
{
g2d.setColor(muerta);
g2d.fillRect(tamCelula*j, tamCelula*i, tamCelula, tamCelula); //x, y, tamx, tamy
g2d.drawRect(tamCelula*j, tamCelula*i, tamCelula, tamCelula); //x, y, tamx, tamy //x, y
}
}
}                         // repaint();
try
{
//System.out.println("tiempo " +tiempo);
Thread.sleep(tiempo);
 }
catch (Exception e) {}
     } //paint component
public int[][] analiza(int[][]m, int[] regla)
        {
Vida v = new Vida();
//System.out.println("hay"+regla[0]);
return v.existe(m,regla, tam);
}
 public int[][] getMatriz()
{
return m;
}
}//class
```

1.3.2 Clase FrameTodo

```
private void BtnStepActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_BtnStepActionPerformed
// TODO add your handling code here:
if(autoriza() == 1)
{
if(gens == 0 && arshivo == 0)
     {
        ScrollM.revalidate();
        tam = Integer.parseInt(txtTamUniv.getText()) ;
                un=newUniverso(regla,tam,getTiempo(), cViva, cMuerta);
                ScrollM.getViewport().add(un);
                ScrollM.setVisible(true);
                repaint();
                gens++;
```

```
     }
    else
     {
        arshivo = 0;
         repaint();
         gens++;
          }
 un.setTiempo(tiempo);
 un.setZoom(zoom);
 lbGens.setText(gens + " generaciones");
 BtnPausar.setText("Reanudar");
}
}
//GEN-LAST:event_BtnStepActionPerformed
private void BtnReiniciarActionPerformed(java.awt.event.ActionEvent evt)
{
//GEN-FIRST:event_BtnReiniciarActionPerformed
// TODO add your handling code here:
//ScrollM.removeAll();
if(autoriza() == 1)
{
 ScrollM.revalidate();
//   ScrollM.repaint();
gens = 0;
lbGens.setText("0 generaciones");
tam = Integer.parseInt(txtTamUniv.getText()) ;
un = new Universo(regla, tam,getTiempo(), cViva, cMuerta);
ScrollM.getViewport().add(un);
    ScrollM.setVisible(true);
BtnPausar.setText("Pausar");
 timer.stop();
     repaint();
un.setTiempo(tiempo);
un.setZoom(zoom);
}
}//GEN-LAST:event_BtnReiniciarActionPerformed
private void VelocidadSliderStateChanged(javax.swing.event.ChangeEvent e
{
//GEN-FIRST:event_VelocidadSliderStateChanged
// TODO add your handling code here:
jLabelTiempo.setText(Float.toString((float) (1.0 / VelocidadSlider.getVa
tiempo = 1000/VelocidadSlider.getValue();
setTiempo(tiempo);
}
//GEN-LAST:event_VelocidadSliderStateChanged
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
```

```
{//GEN-FIRST:event_jButton1ActionPerformed

//INICIAR
jButton1.setText("Iniciar");
if(autoriza() == 1)
 {
            if(timer.isRunning())
                     {
                         System.out.println(":3");
                     }
                    else
                {
//
ScrollM.removeAll();
                    ScrollM.revalidate();
                    gens = 0;
                    lbGens.setText("0 generaciones");
                    tam = Integer.parseInt(txtTamUniv.getText()) ;
        un = new Universo(regla, tam,getTiempo(), cViva, cMuerta);

                                          ScrollM.getViewport().add(un);
                    ScrollM.setVisible(true);
                    timer.start();
                    repaint();
                 }
}
//autoriza
}//GEN-LAST:event_jButton1ActionPerformed
    private void BtnPausarActionPerformed(java.awt.event.ActionEvent evt
{//GEN-FIRST:event_BtnPausarActionPerformed
// TODO add your handling code here:
if(timer.isRunning())
{
    timer.stop();
   BtnPausar.setText("Reanudar");
}
else
{
   if(BtnPausar.getText().equals("Reanudar"))
        {
            timer.restart();
            BtnPausar.setText("Pausar");
        }
}
}
```

2 Generador de Atractores

2.1 Descripción del programa

Este programa se encarga de generar el código del programa Mathematica para poder garficar el atractor generado por la regla dada del autómata celular.

2.2 Pruebas del funcionamiento

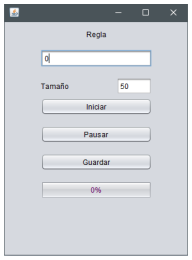


Figura 5: Inicio del simulador y analizador aleatoriamente.

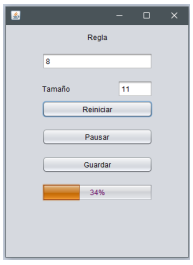


Figura 6: Ejecución del analizador

Figura 7: Análisis Terminado

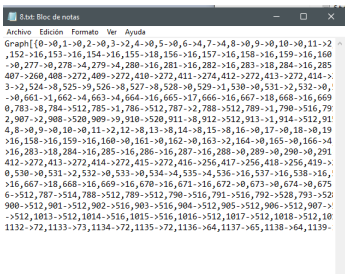


Figura 8: Archivo generado.

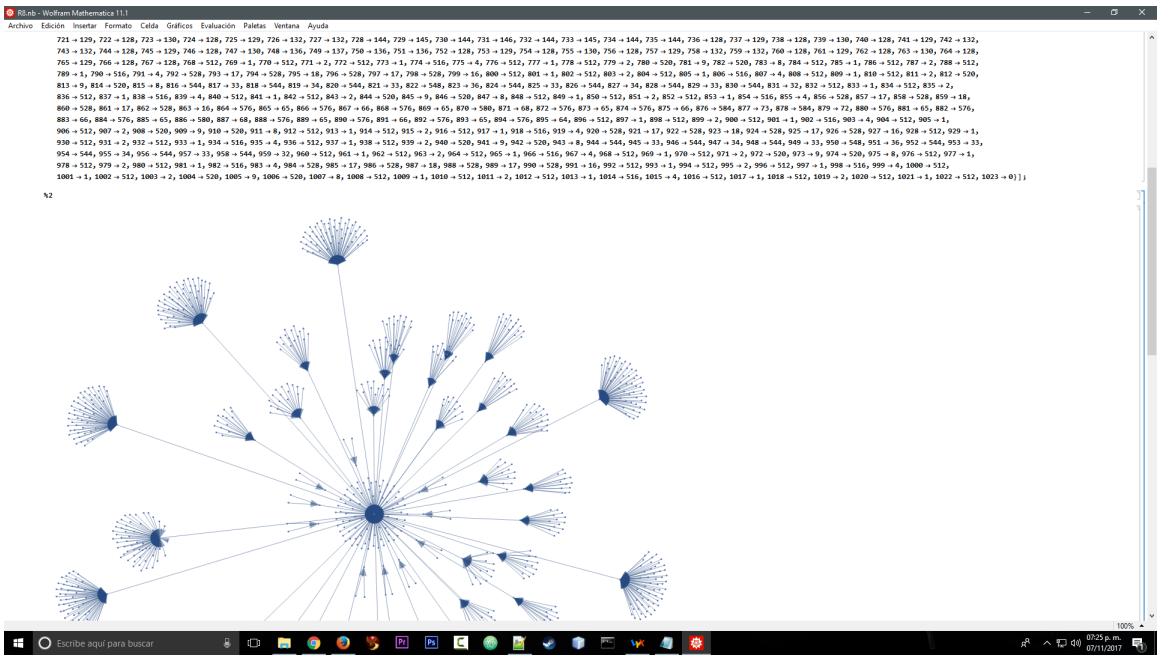


Figura 9: Graficación del atractor

2.3 Codigo fuente

2.3.1 Clase Vida

```
Public int[] binario(int r, int tam)  //entero
    a binario
{
int[] b = new int[tam];
String bin = Integer.toBinaryString(r);
while(bin.length() < tam)
    {
        bin = "0" + bin;
    }
//  System.out.println("s" + bin);
for(int i = 0; i < tam; i++)
    {
        b[i] = Character.getNumericValue(bin.
            charAt(i));
    }
return b;
}
```

```
public int[] existe (int[] m, int regla, int
    tam) //analizador del automata y
    sustituidor
{
     int[] aux = new int[tam], sust = new int
         [8]; //aqui va el sustituidor
     String cadenita;
     int[] bin = binario(regla, 8);
     for(int i = 0; i < tam; i++) //aqui divido
          el arreglo en pedacitos de 3 para
         analizarlo
          {
if(i == 0) //uno el inicio con el final
{
cadenita = Integer.toString(m[tam - 1]);
cadenita += Integer.toString(m[i]) + Integer.
    toString(m[i + 1]);
}
else
{
if(i == (tam - 1))
{
cadenita = m[i - 1] + "" + m[i] + "" + m[0] +
    ""; //uno el final al inicio
}
else
{
cadenita = m[i - 1] + "" +m[i] + "" +m[i + 1]
    + "";
}
}
          //    System.out.println(cadenita);
switch(cadenita)
     {
          case "000":
              aux[i] = bin[7];
          break;
          case "001":
              aux[i] = bin[6];
          break;
          case "010":
              aux[i] = bin[5];
          break;
          case "011":
              aux[i] = bin[4];
          break;
```

```
            case "100":
                aux[i] = bin[3];
            break;
            case "101":
                aux[i] = bin[2];
            break;
            case "110":
                aux[i] = bin[1];
            break;
            case "111":
                aux[i] = bin[0];
            break;
            }
    }
        return aux;
    }//main
    public int decimal(int[] m)
    {
    String bin = "";
    int dec;
    for(int i = 0; i < m.length; i++)
    {
        bin += Integer.toString(m[i]);
    }
    dec = Integer.parseInt(bin, 2);
     //convierto binario a decimal
    return dec;
    }
```

2.3.2 Clase FrameTodo

```
if (src == timer)
{
Prog.setValue(nodoActual);
if(nodoActual == (int)Math.pow(2,tam)) //ya terminé
    {
        timer.stop();
        esc.escribe(regla, "}];");
        JOptionPane.showMessageDialog(null, "Análisis Terminado. Se ha creado un
    }
else
    {
        int[] mat = new int[tam]; //actual
        int[] mat2 = new int[tam]; //evaluado
        int nodoSiguiente; //evaluado en decimal
```

```
        String coma = ",";
        mat = v.binario(nodoActual, tam); //obtengo mi nodo actual en binario
        mat2 = v.existe(mat, regla, tam);
        nodoSiguiente = v.decimal(mat2);
        //System.out.println(nodoActual + " -> " + nodoSiguiente);
        if(nodoActual == (int)Math.pow(2,tam) - 1)
            {
                coma = "";
            }
        String txt = nodoActual + "->" + nodoSiguiente+coma;
        esc.escribe(regla, txt);
        nodoActual++;
    }
}
```

2.3.3 Clase Escritor

```
        public int escribe(int regla, String text)
            {
                    try (BufferedWriter bw = new BufferedWriter(new FileWriter(r
                            {
                               bw.write(text);
                               bw.close();
                               return 1;
                            }
                    catch (IOException e)
                            { e.printStackTrace(); return 0;}
           }
```

2.4 Clasificación de los atractores

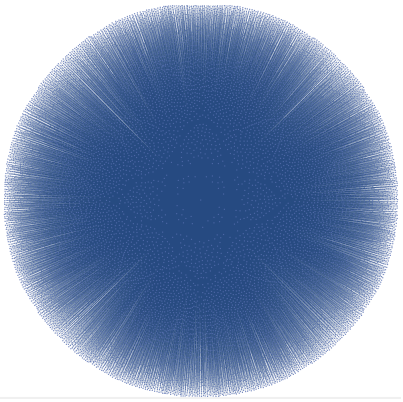
2.4.1 Clase I - Estáticos

1. Descripción

   (a) Éstos autómatas celulares presentan un comportamiento estático y homogéneo donde su atractor siempre se dirige a un punto y se mantiene estacionado ahí de manera infinita.

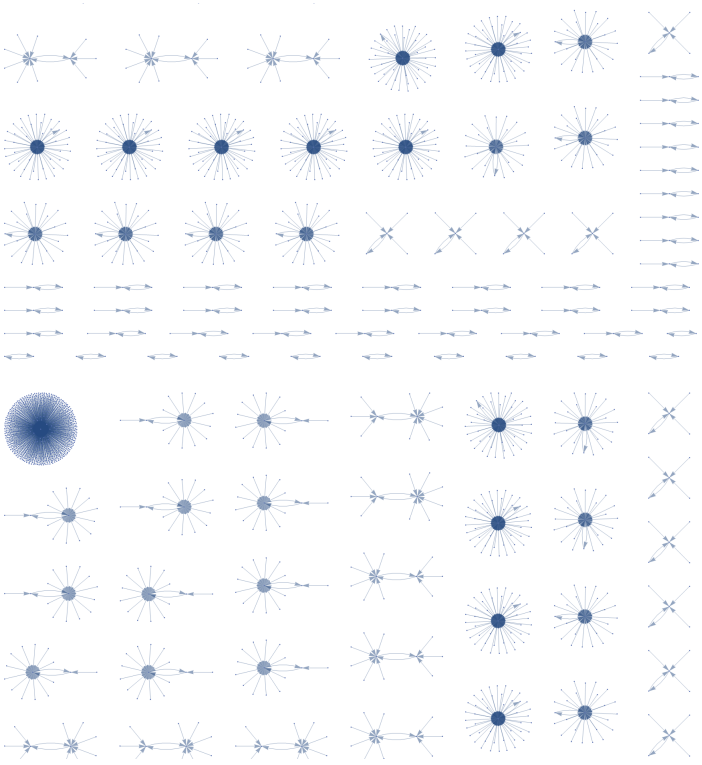
2. Atractores de automatas clase I

   (a) Regla 0



i.

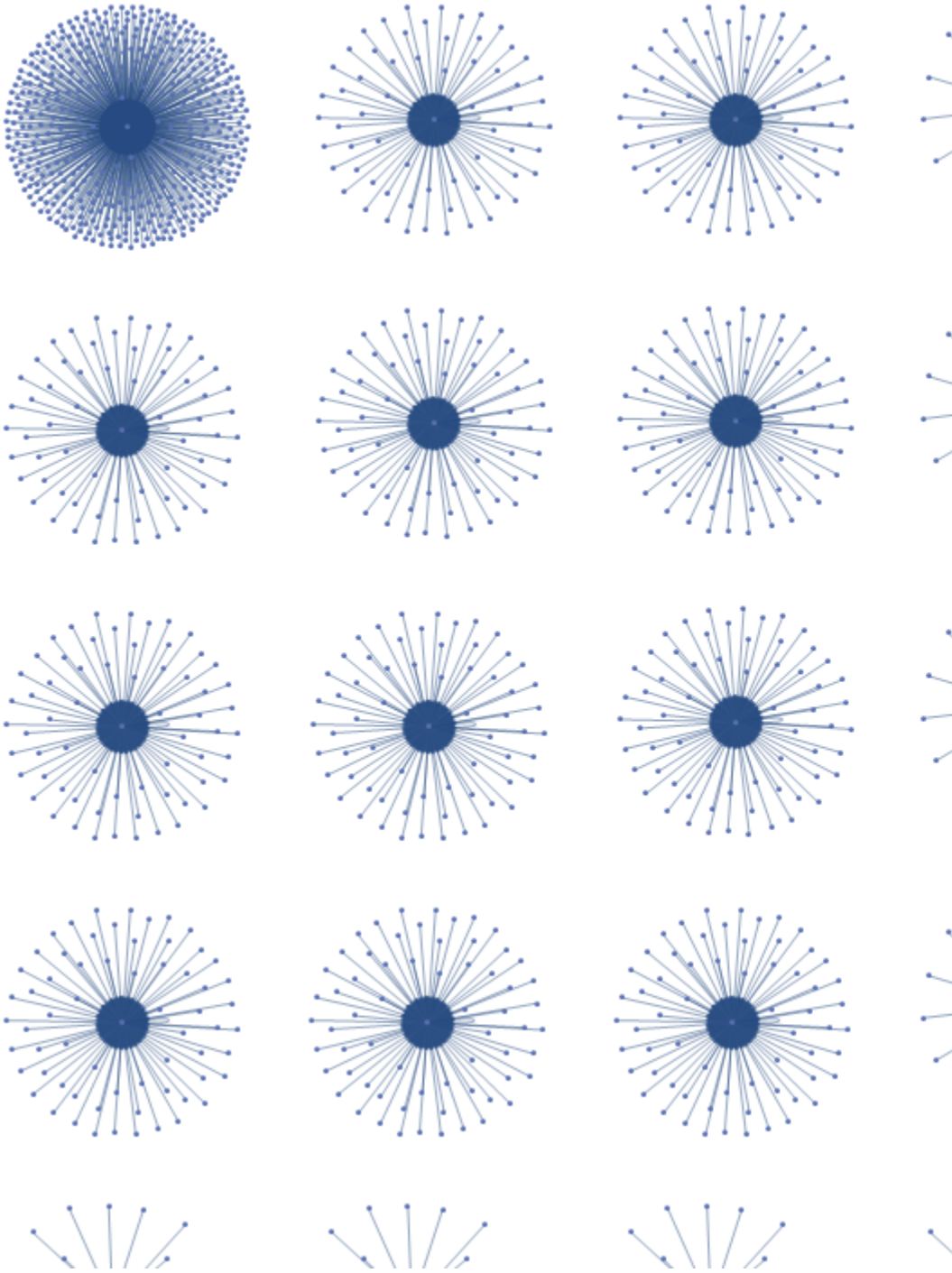
(b) Regla 1



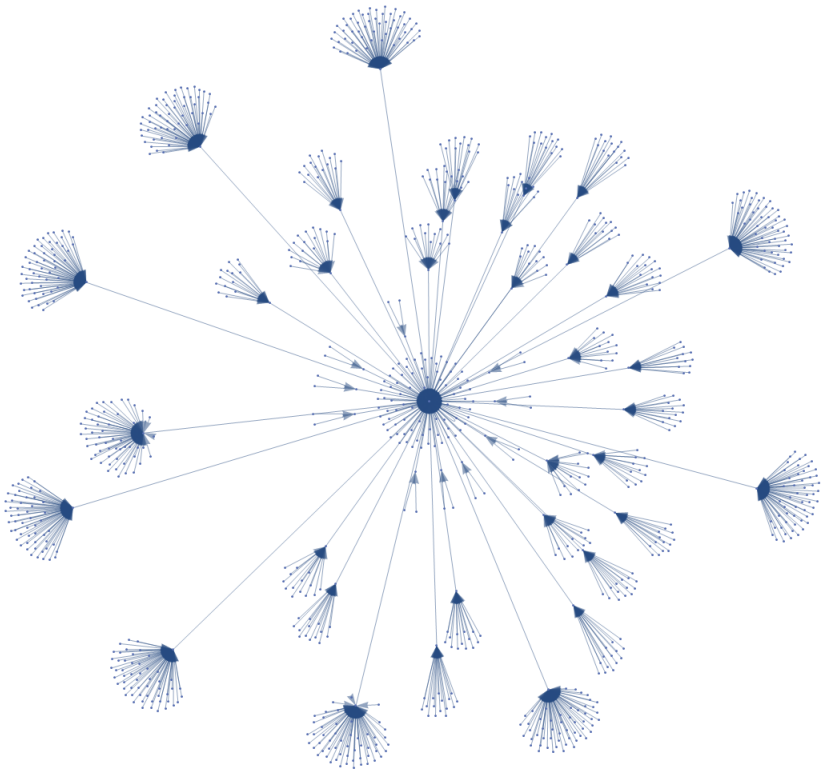
i.

ii.

(c) Regla 4

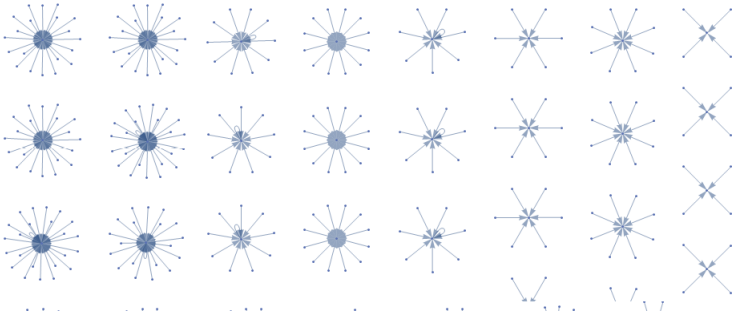
i. 

(d) Regla 8



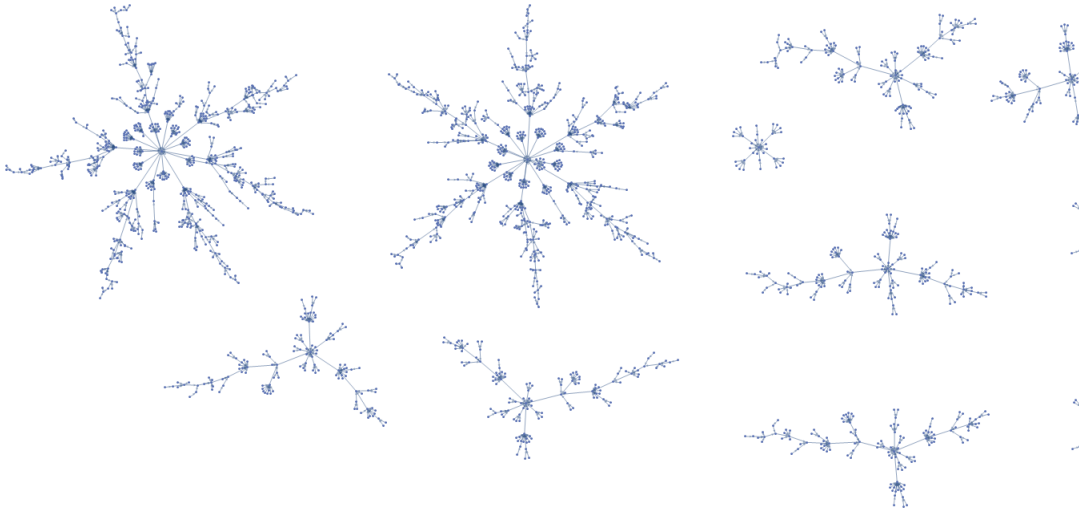
i.

(e) Regla 12



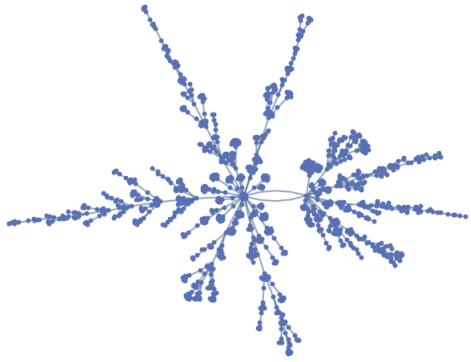
i.

(f) Regla 13



i.

(g) Regla 28

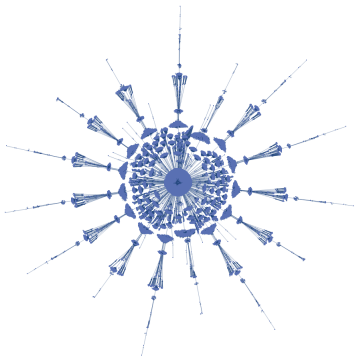


i.

(h) Regla 29

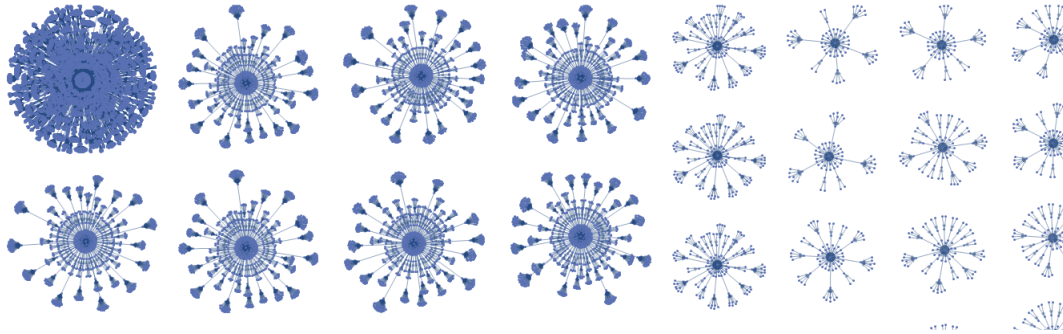
i.

(i) Regla 32



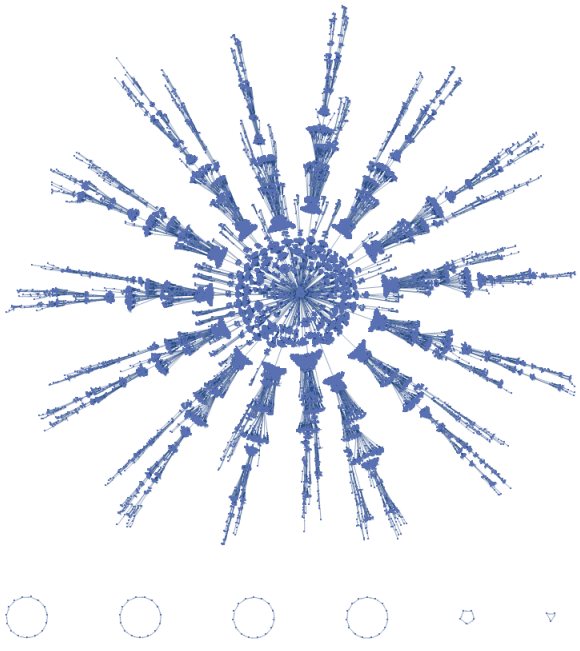
i.

(j) Regla 36



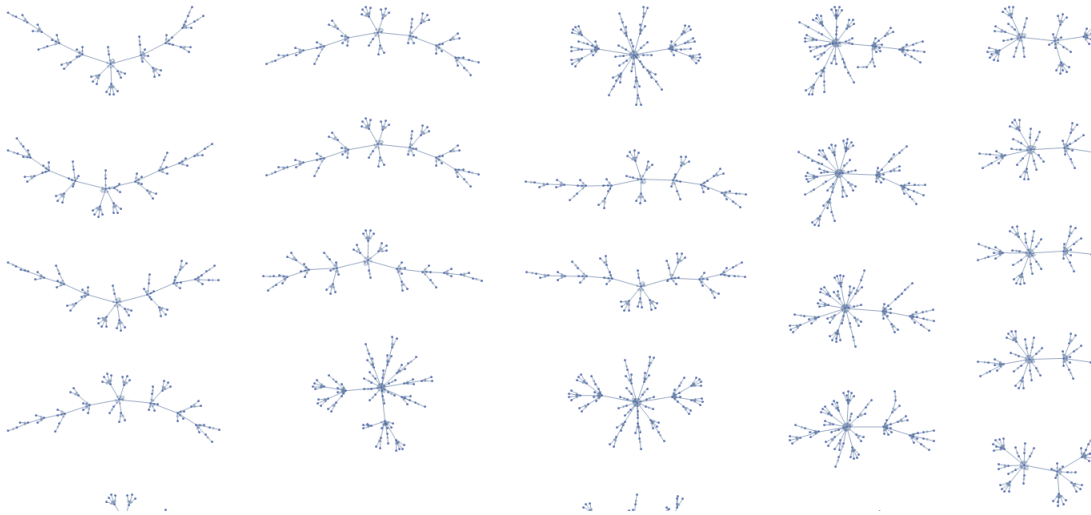
i.

(k) regla 40



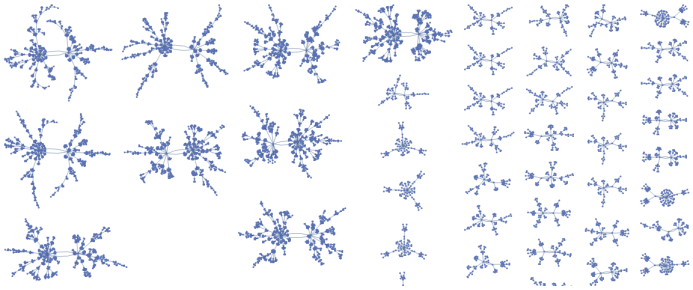
i.

(l) Regla 44



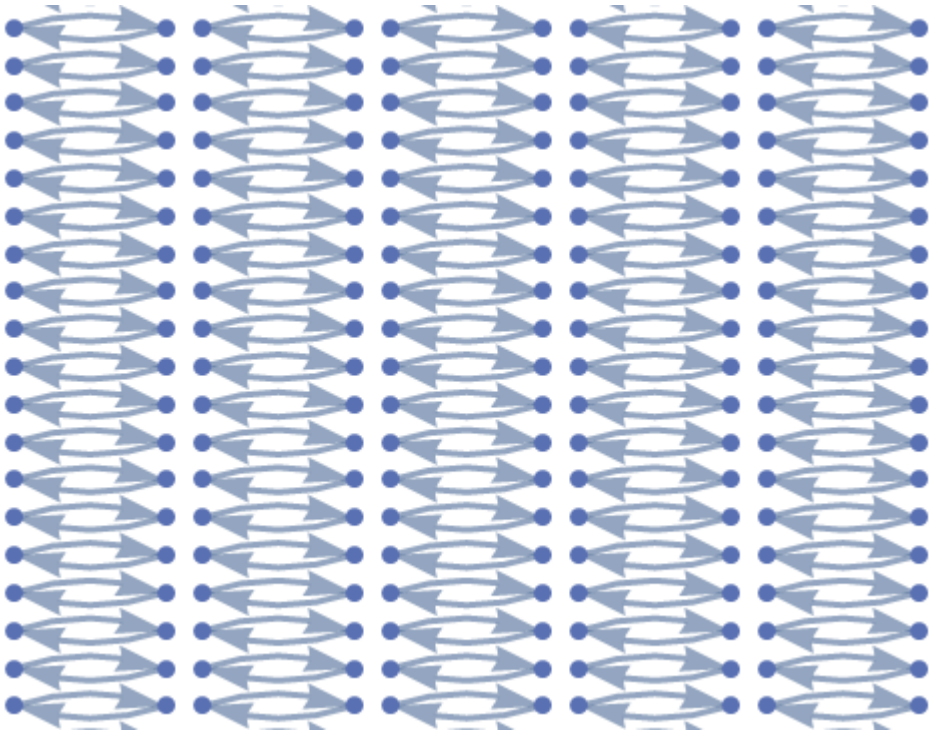
i.

(m) Regla 50

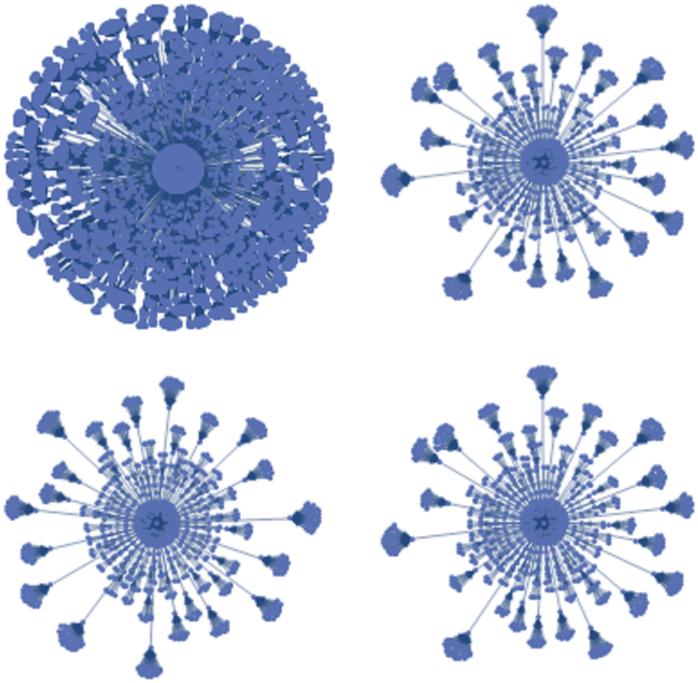


i.

(n) Regla 51

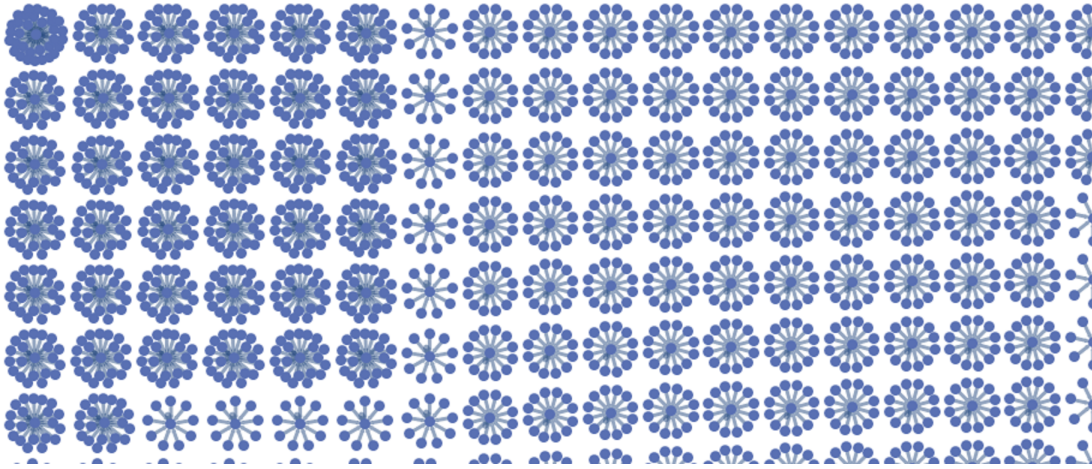
i. 

(o) Regla 72



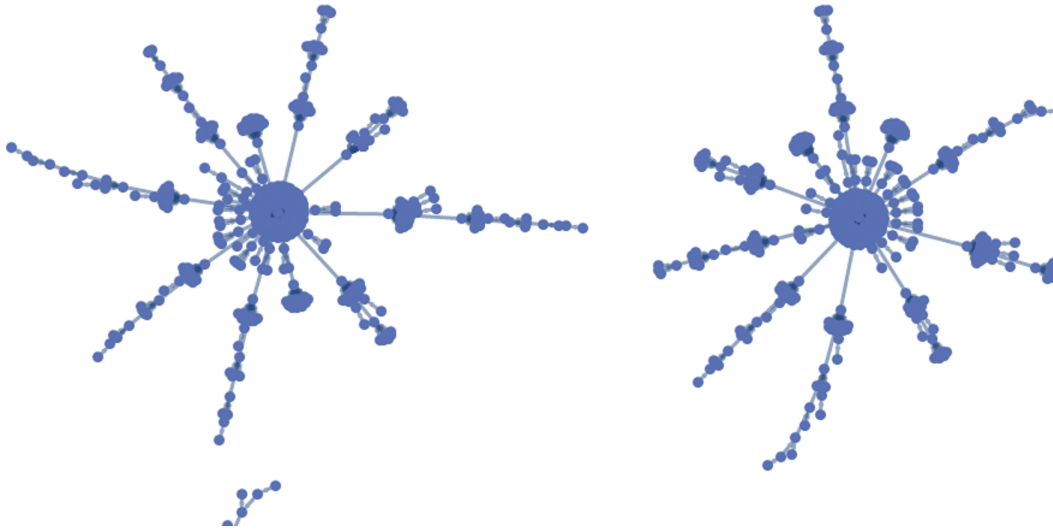
i.

(p) Regla 76



i.

(q) Regla 77

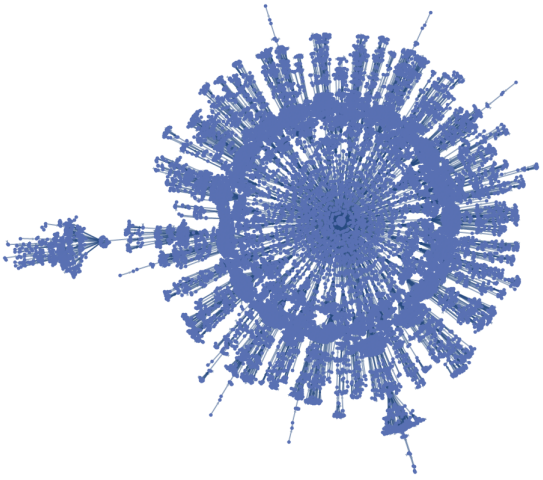


i.

(r) Regla 78

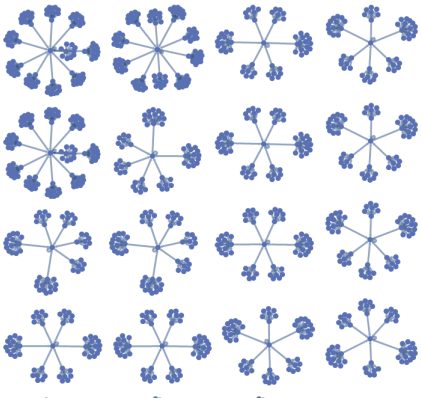
i.

(s) Regla 104



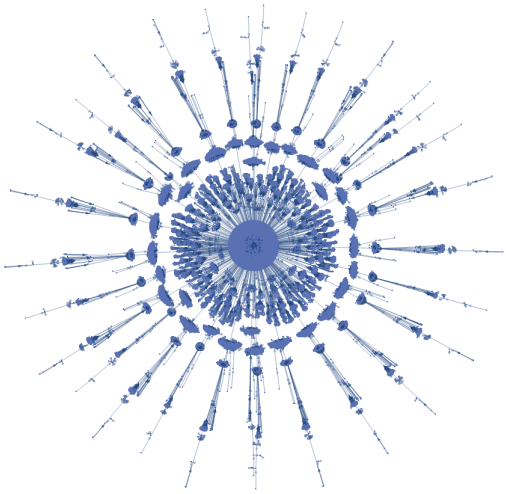
i.

(t) Regla 108



i.

(u) Regla 128

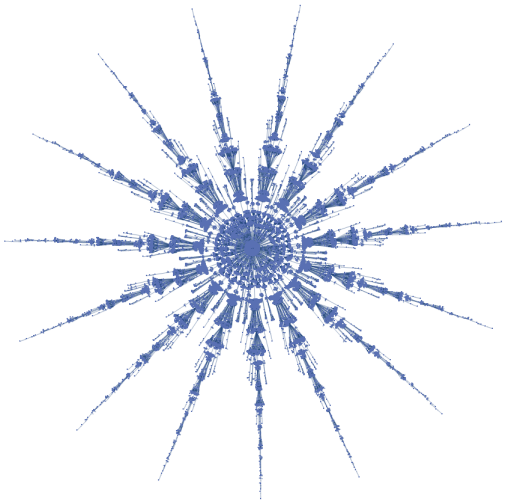


i.

(v) Regla 132

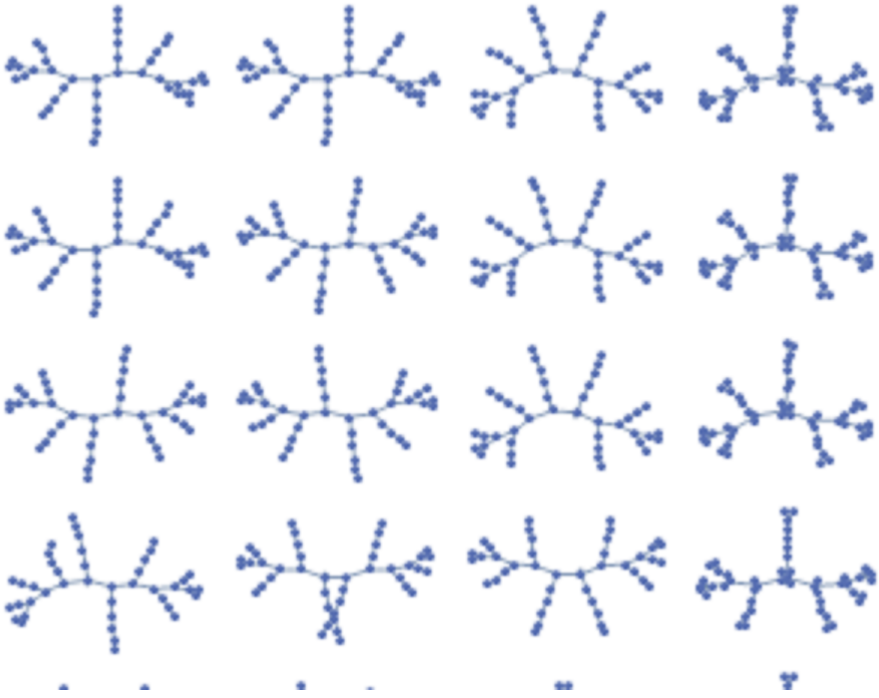
i.

(w) Regla 136



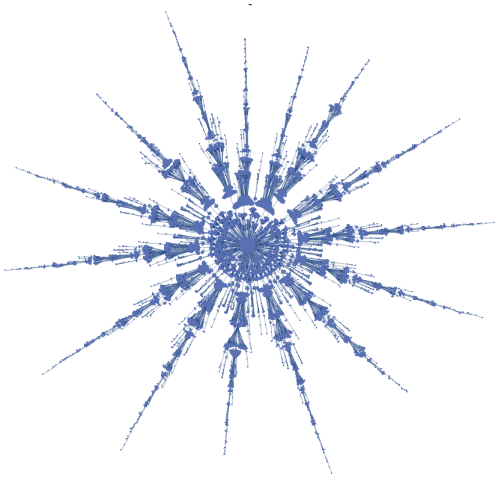
i.

(x) Regla 140



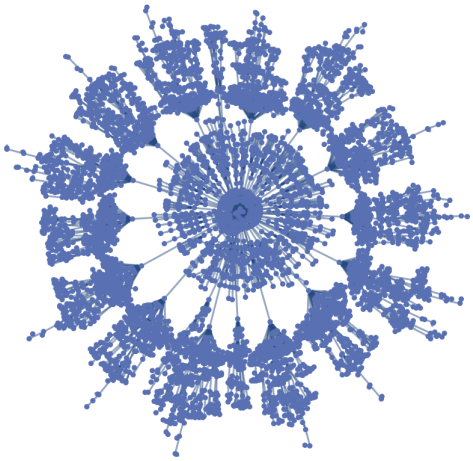
i.

(y) Regla 160



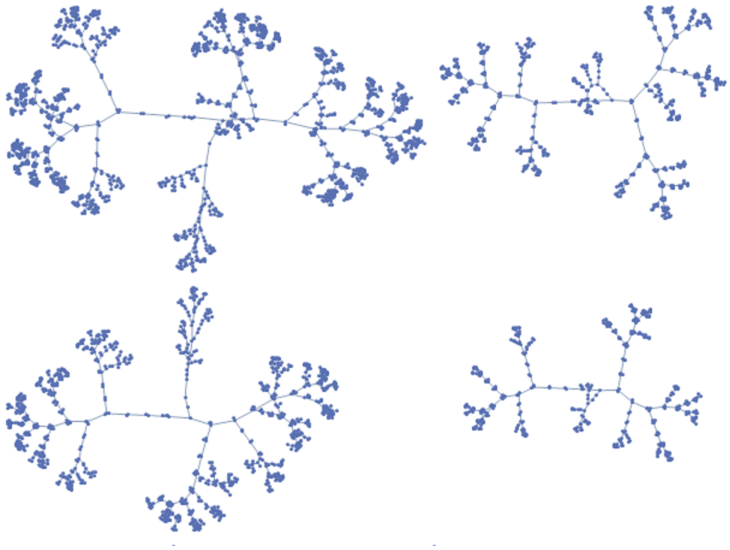
i.

(z) Regla 164

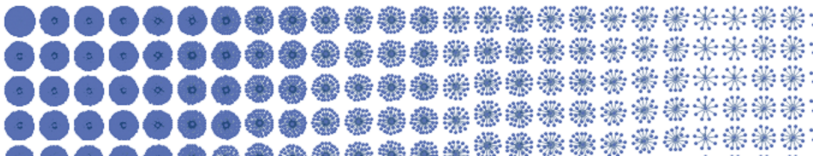


i.

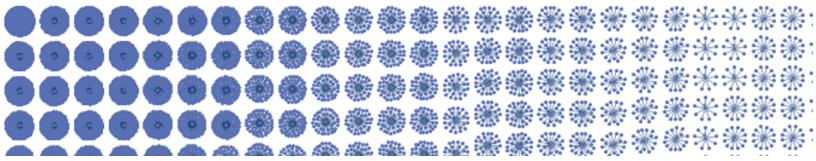
() Regla 172

i. 

() Regla 200

i. 

() Regla 204

i. 

() Regla 232

i. 

2.4.2 Clase II - Periódicos

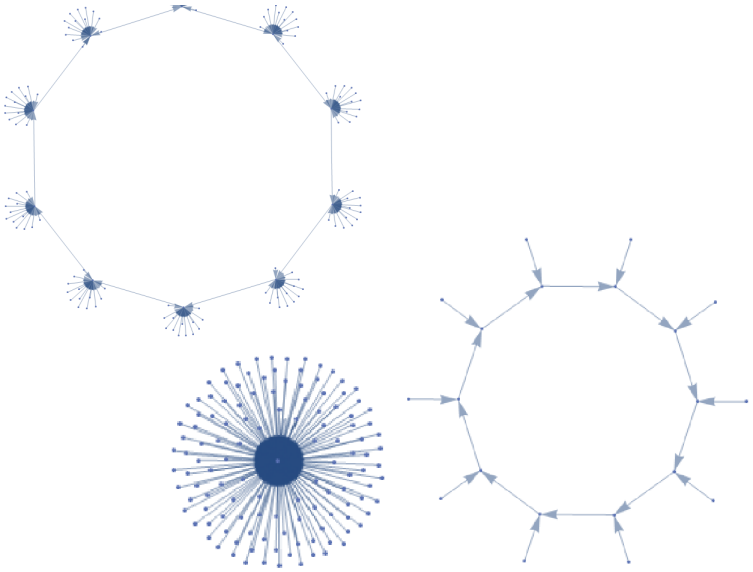
1. Descripción

   (a) El atractor o los atractores de éstas reglas siempre terminan en n estados cíclicos que se repiten uno tras otro indefindiamente, por más ramas y divisiones que tenga siempre caen en los estados loop.

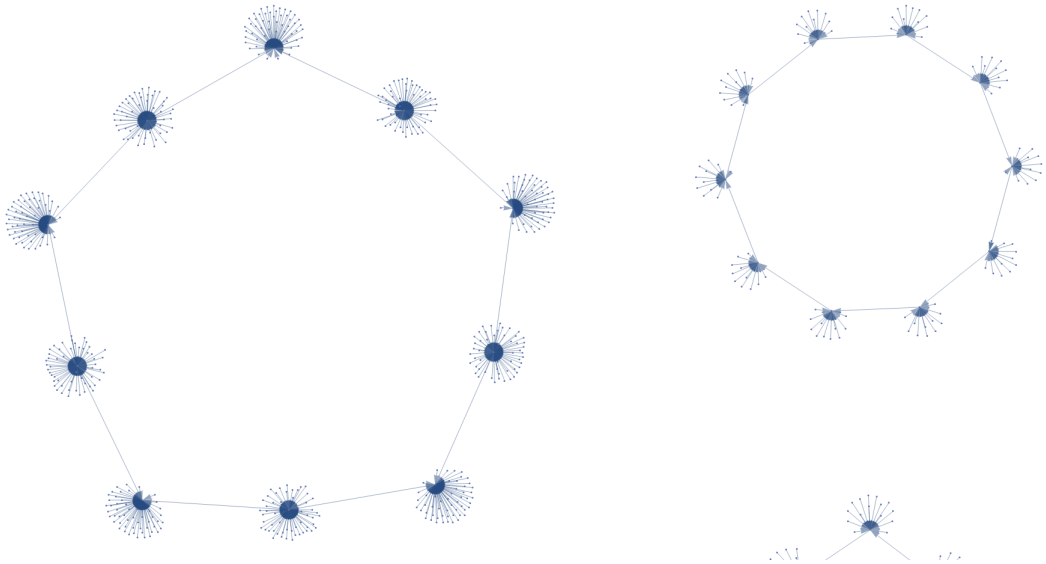
2. Reglas

   (a) Regla 2

i.

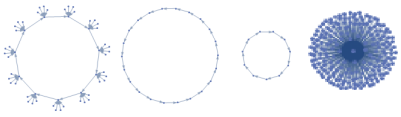


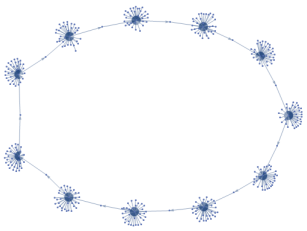
ii.



(b) Regla 3

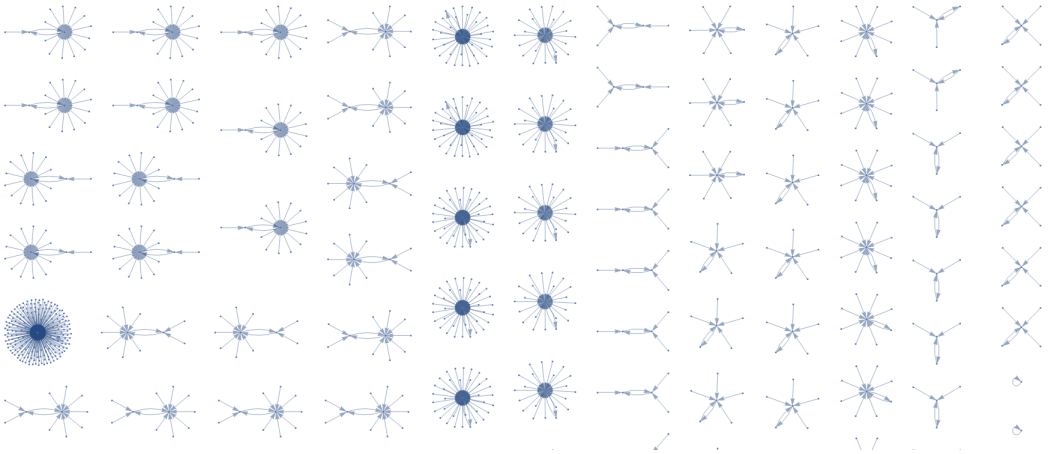
i.





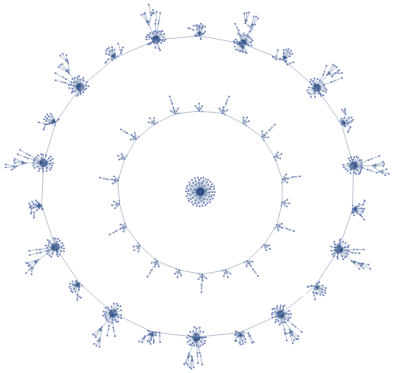
ii.

(c) Regla 5

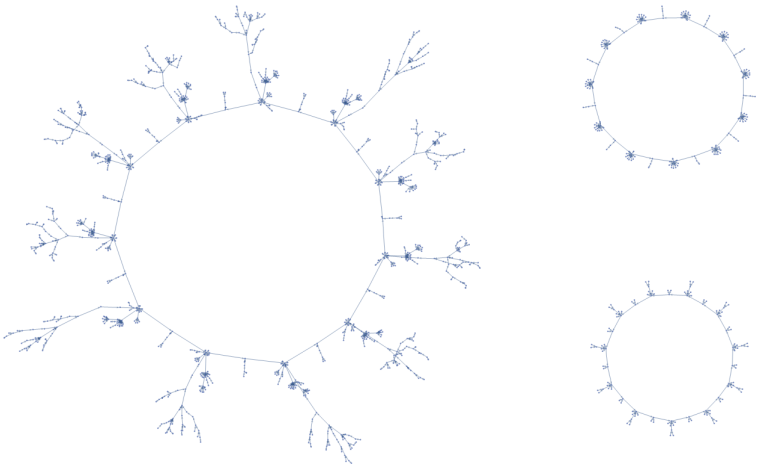


i.

(d) Regla 6

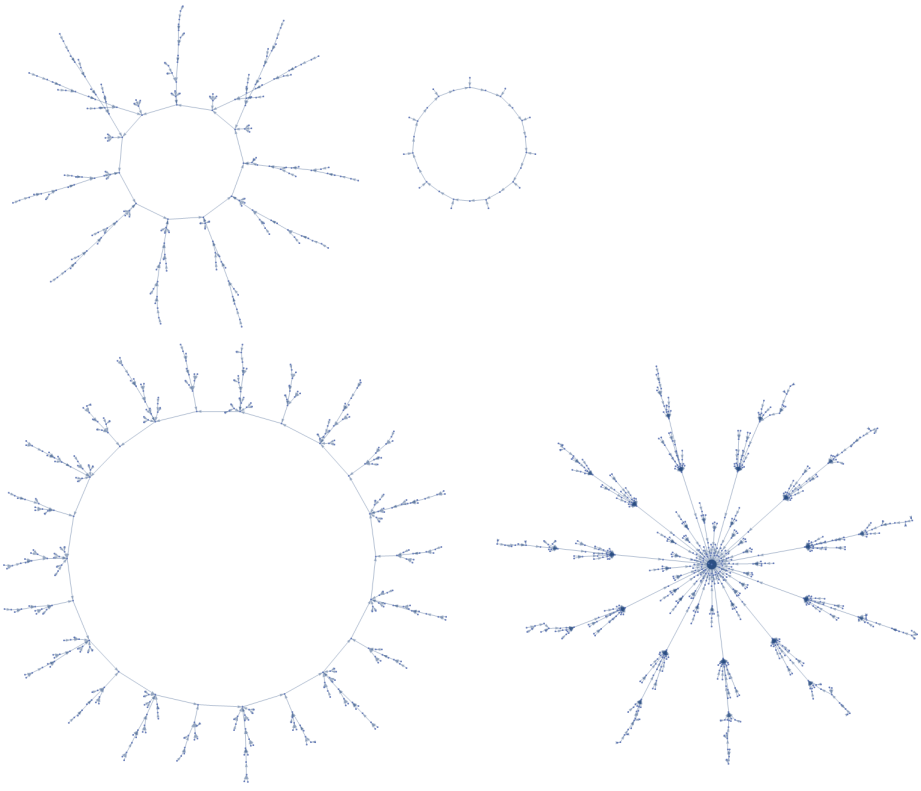


i.



ii.

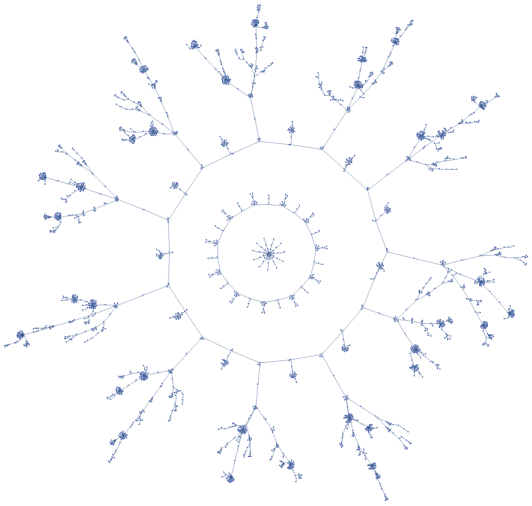
(e) Regla 7



i.

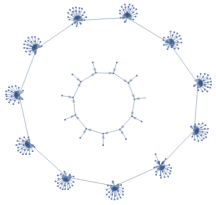
ii.

(f) Regla 9



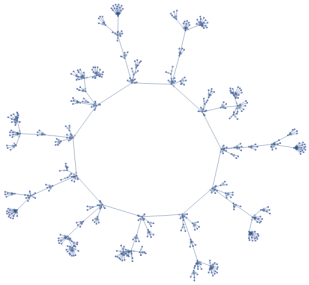
i.

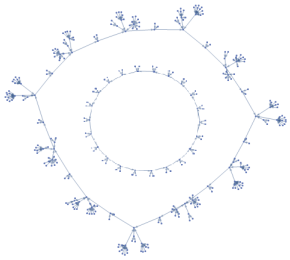
(g) Regla 10

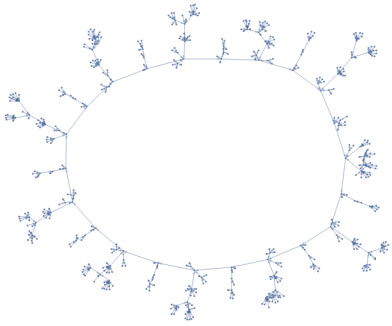


i.

(h) Regla 11

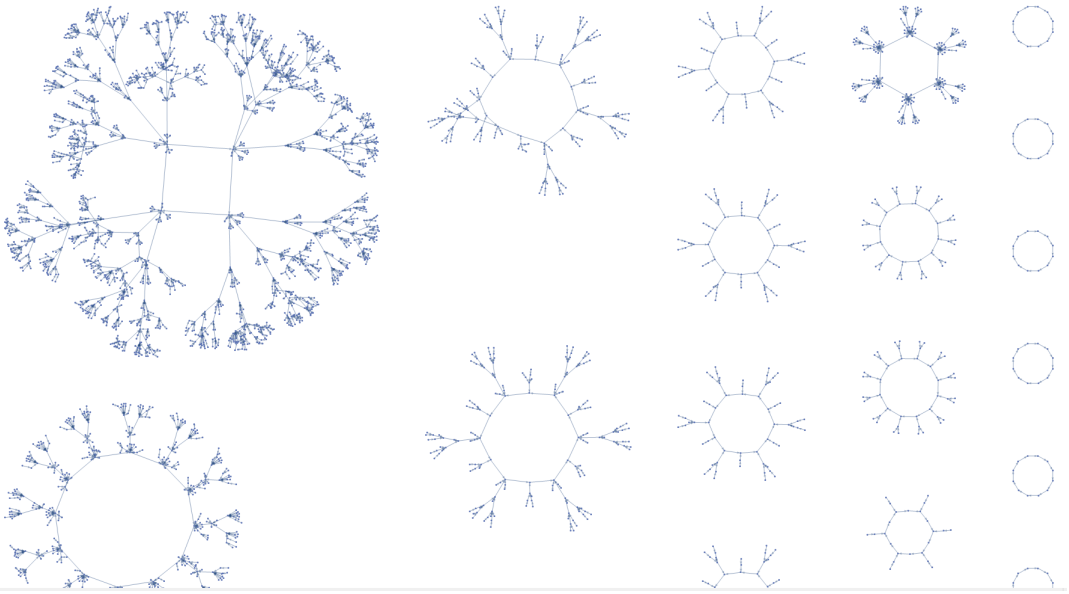




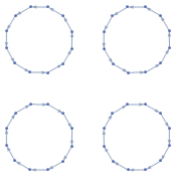


i.

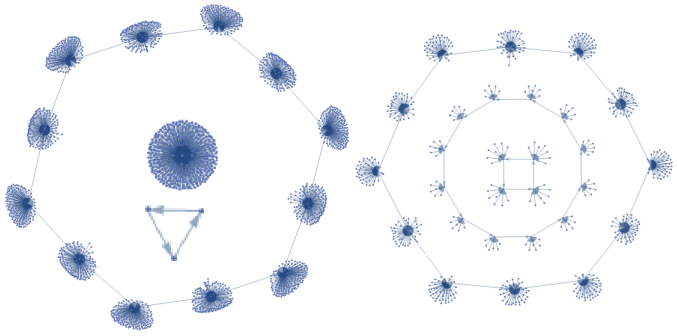
(i) Regla 14

i. 

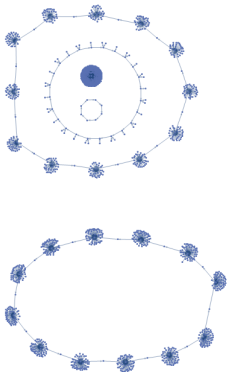
(j) Regla 15

i. 

(k) Regla 16

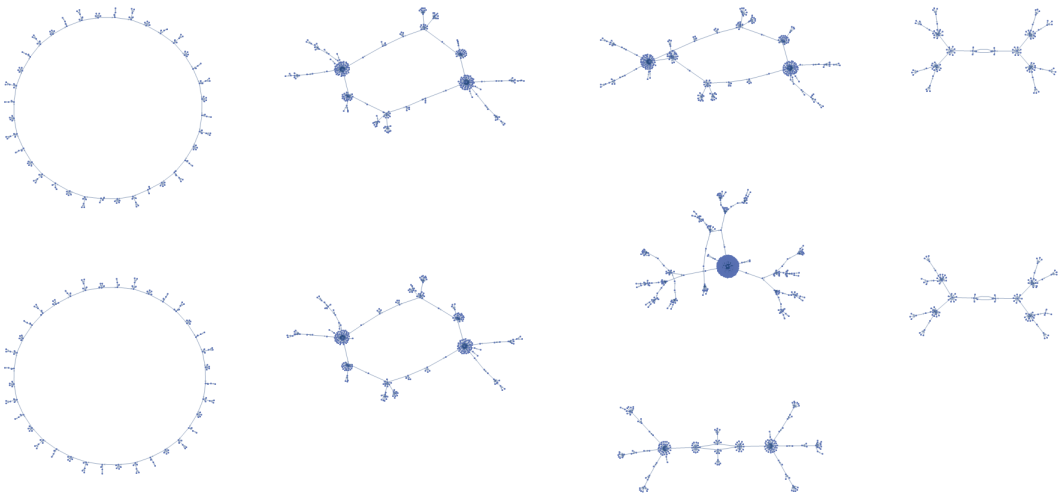
i. 

(l) Regla 17



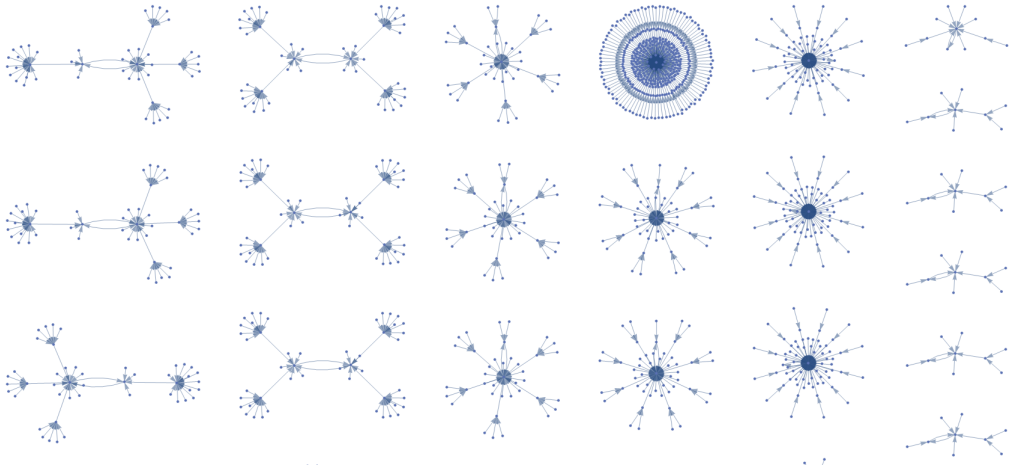
i.

(m) Regla 18



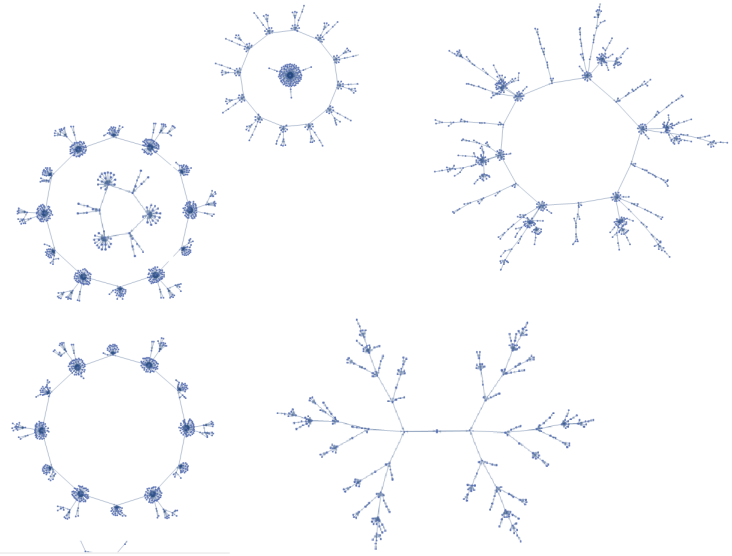
i.

(n) Regla 19



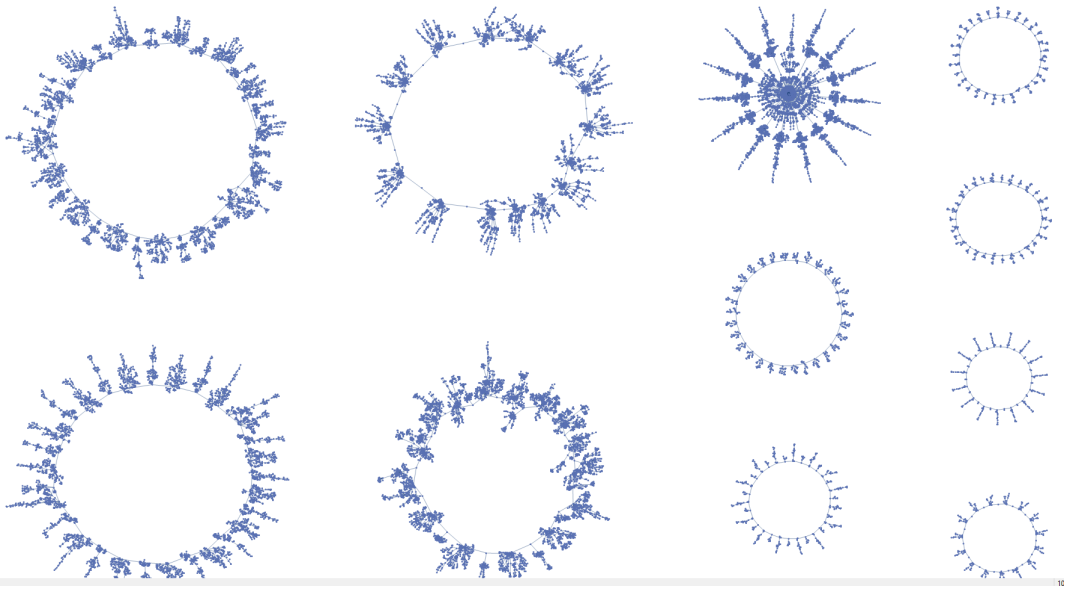
i.

(o) Regla 20



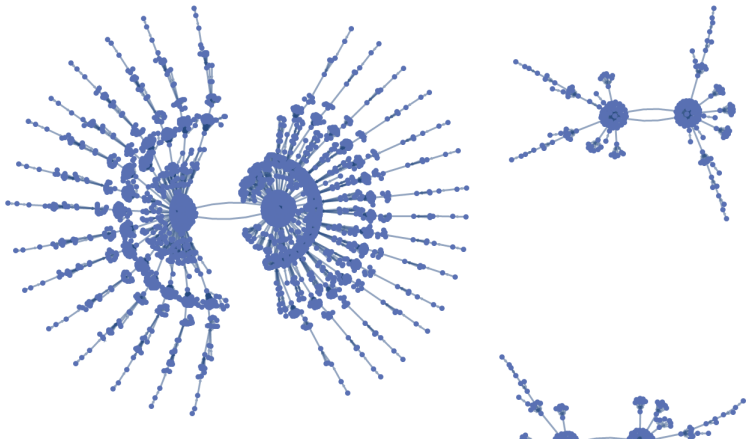
i.

(p) Regla 21



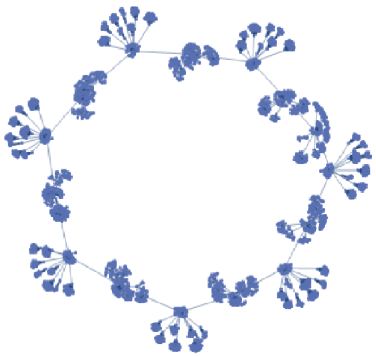
i.

(q) Regla 23



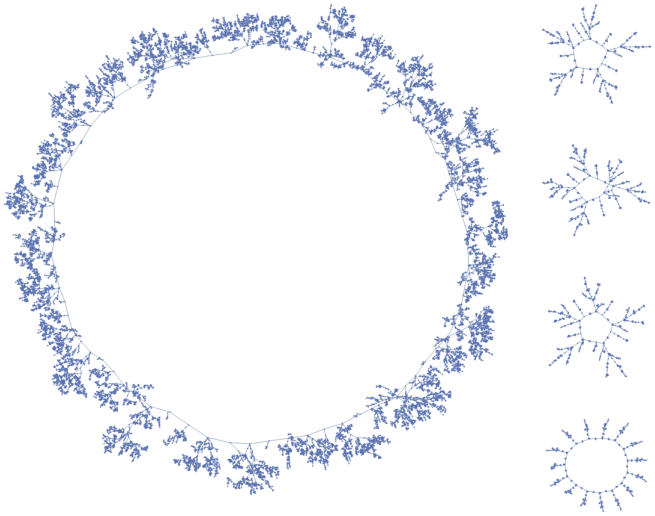
i.

(r) Regla 24



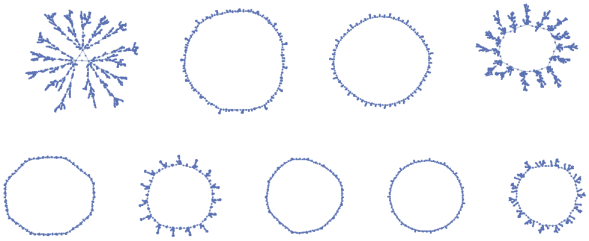
i.

(s) Regla 25



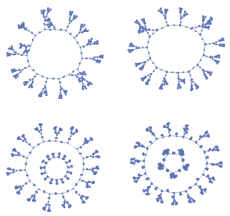
i.

(t) Regla 26



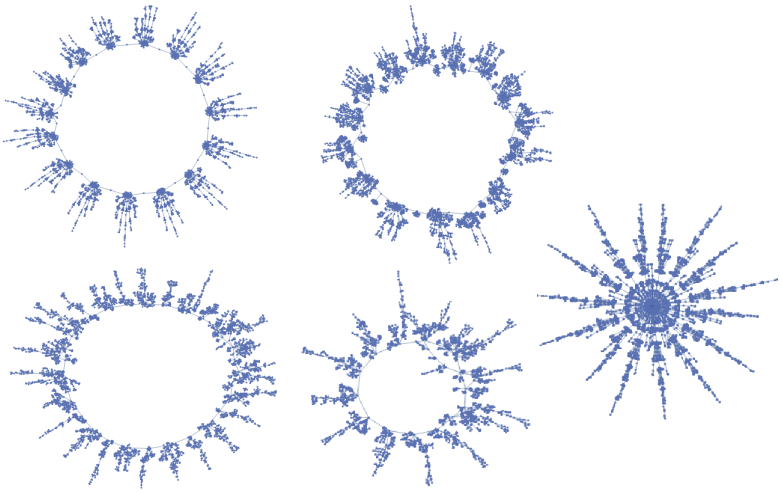
i.

(u) Regla 27



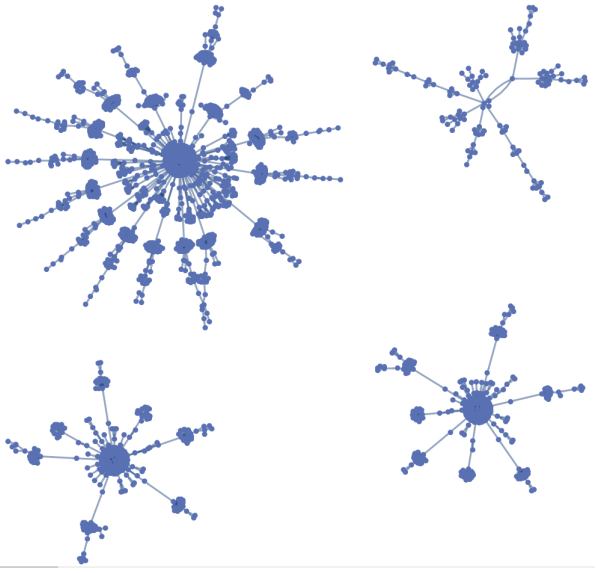
i.

(v) Regla 31



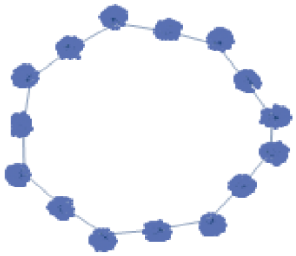
i.

(w) Regla 33



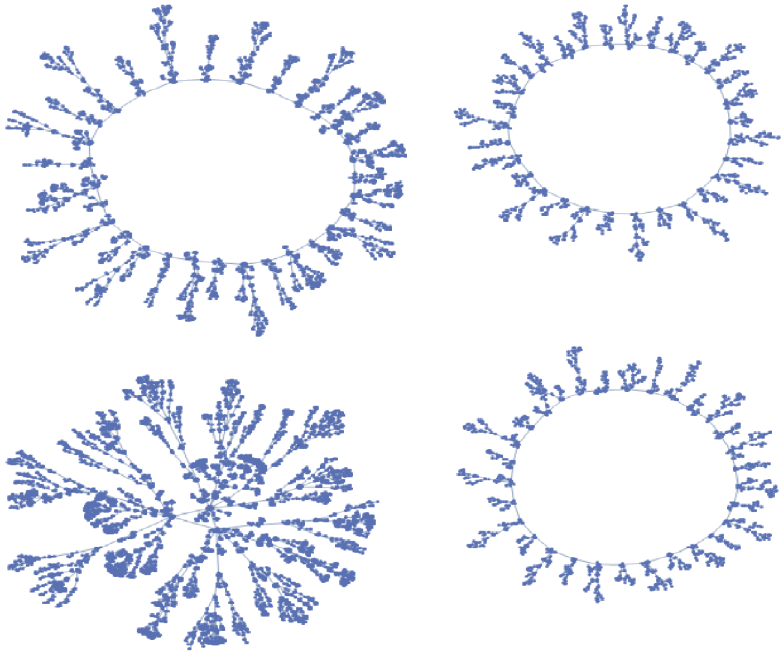
i.

(x) Regla 34



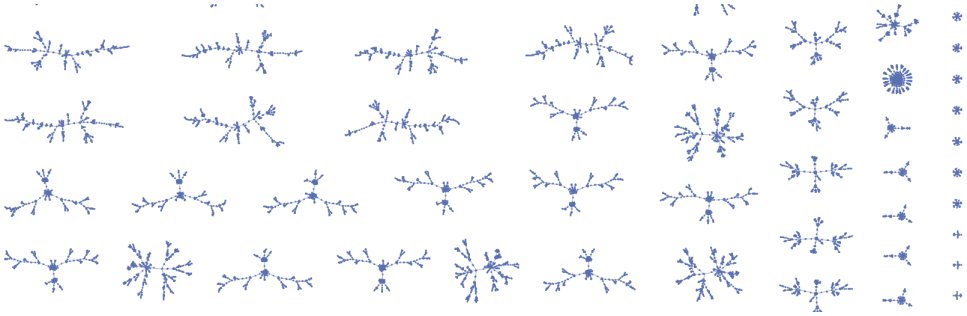
i.

(y) Regla 35



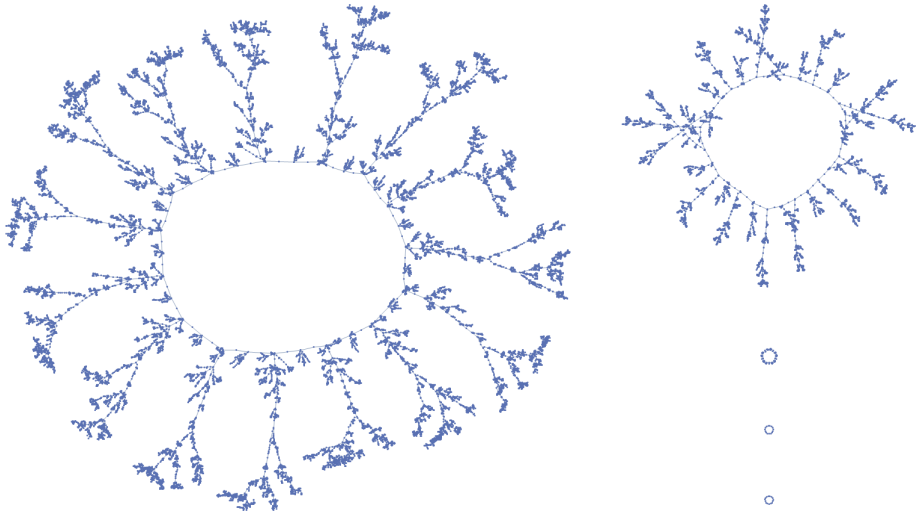
i.

(z) Regla 37



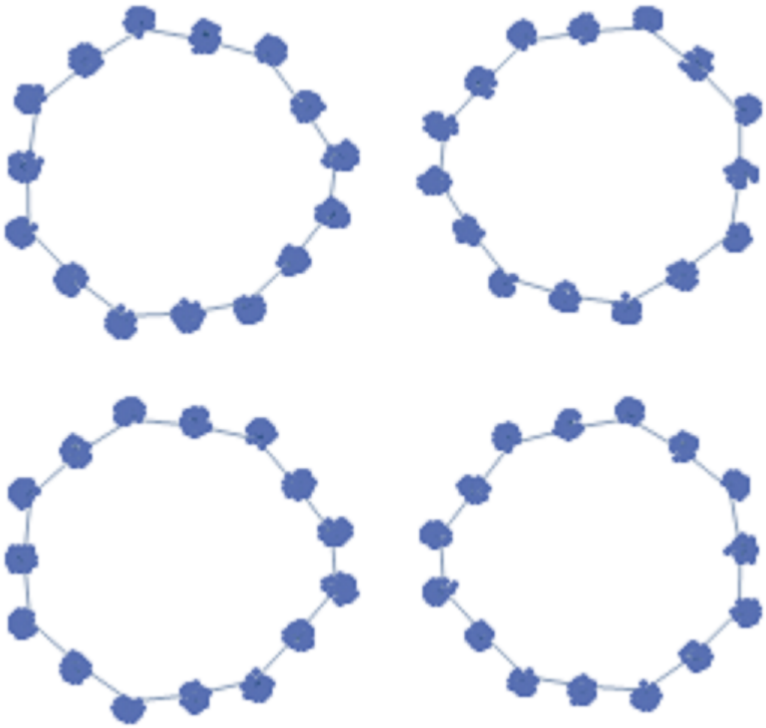
i.

() Regla 41



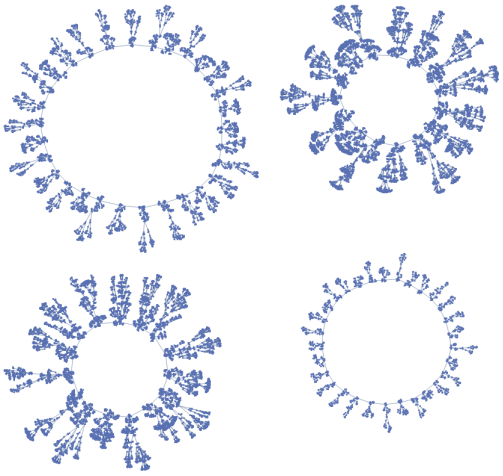
i.

() Regla 42



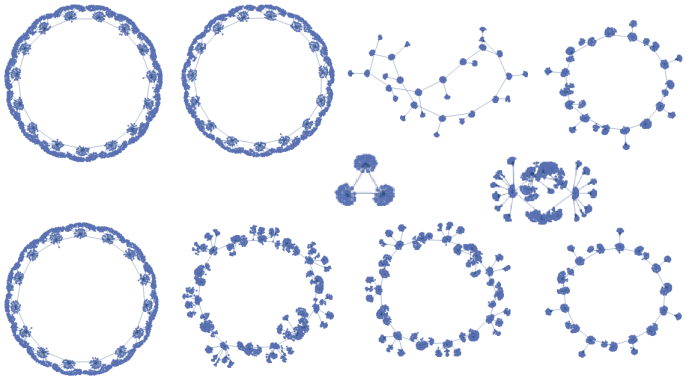
i.

() Regla 43



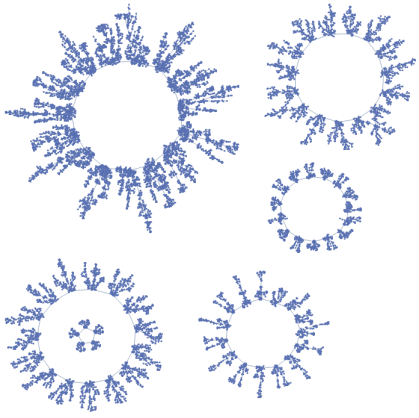
i.

() Regla 46



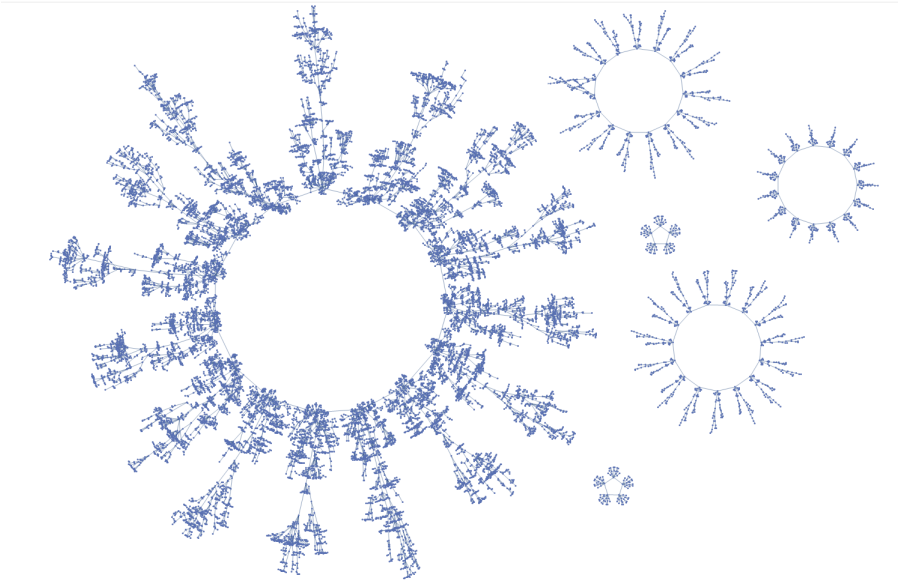
i.

() Regla 56

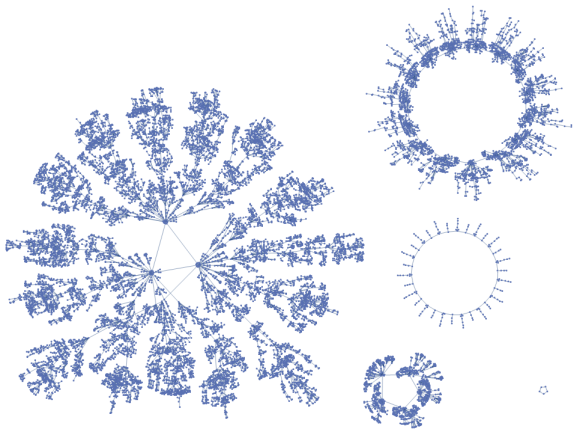


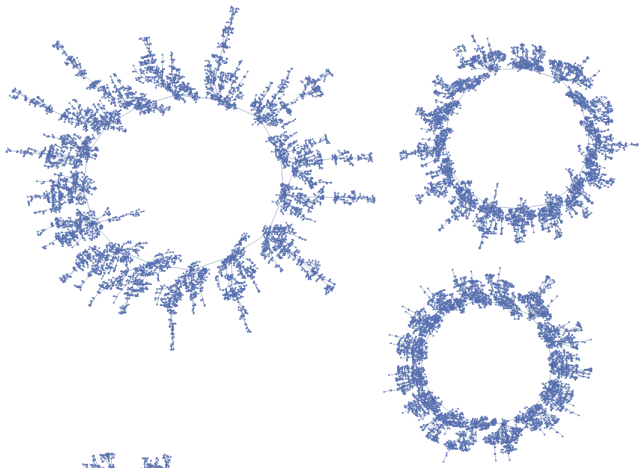
i.

() Regla 57

i. 

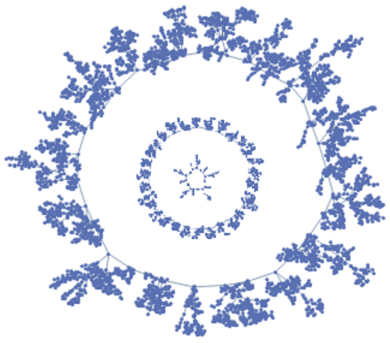
() Regla 58

i. 



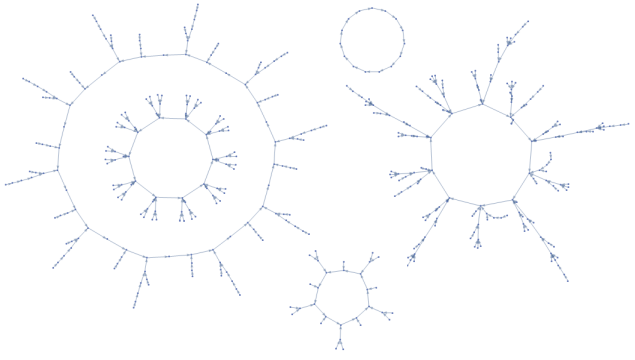
ii.

() Regla 74



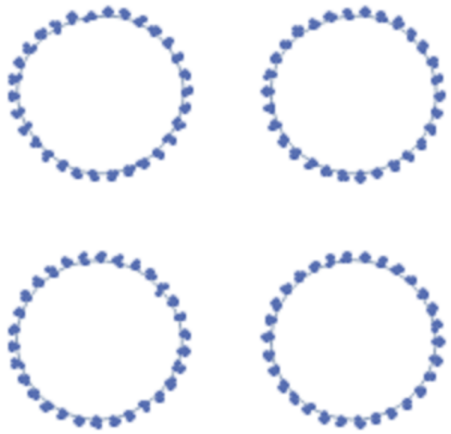
i.

() Regla 88



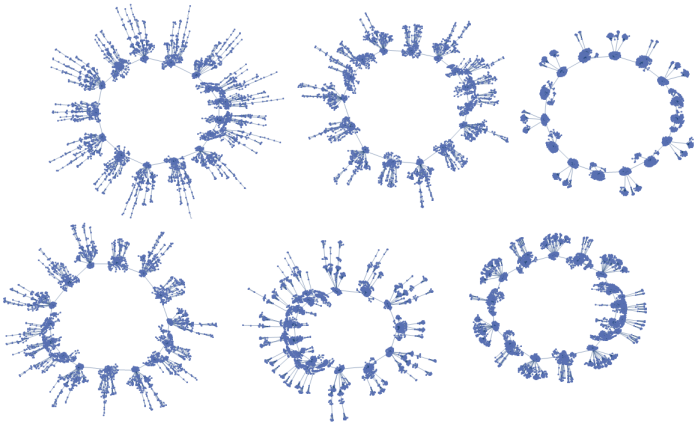
i.

() Regla 105



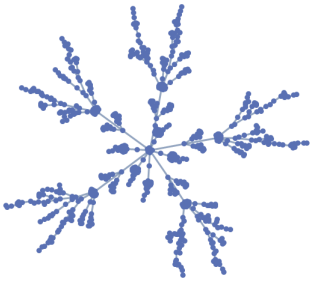
i.

() Regla 130



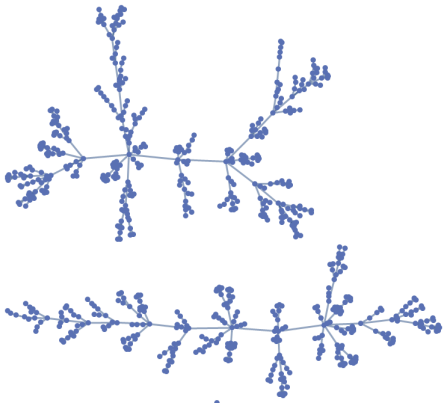
i.

() Regla 133



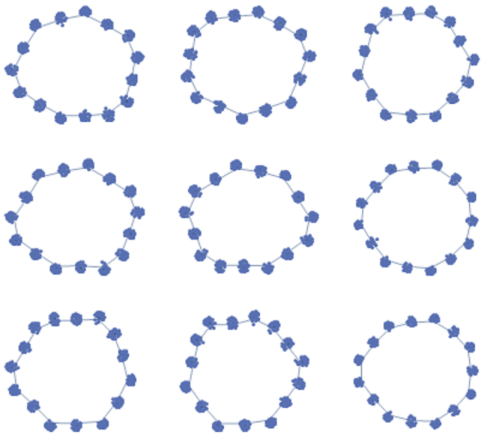
i.

() Regla 134



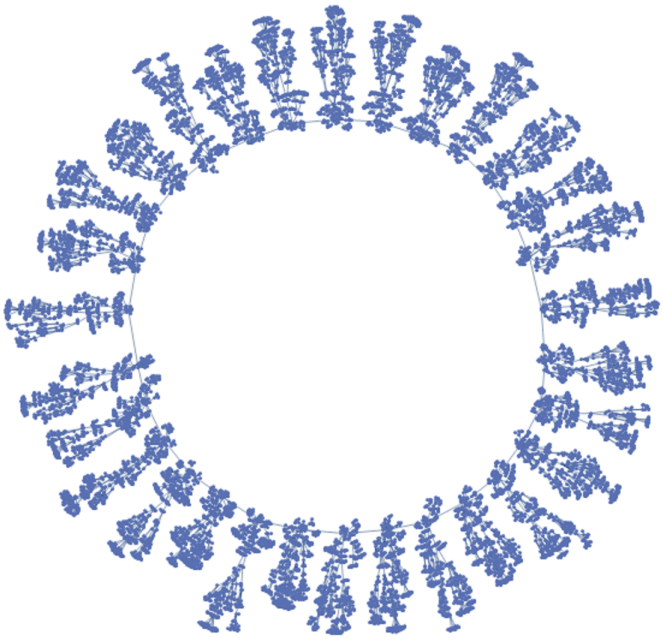
i.

() Regla 138



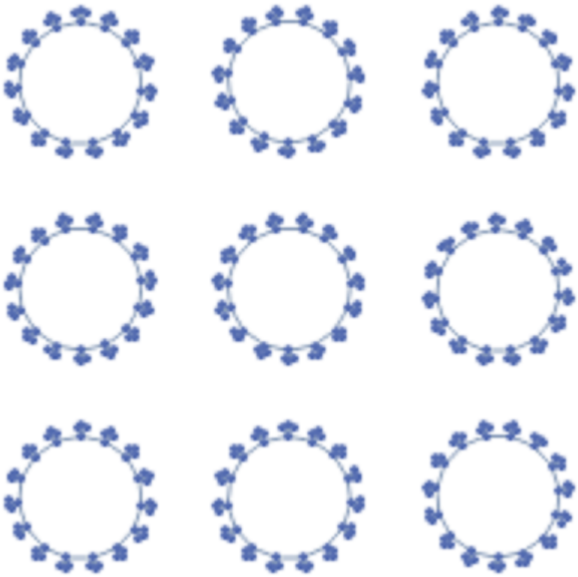
i.

() Regla 142



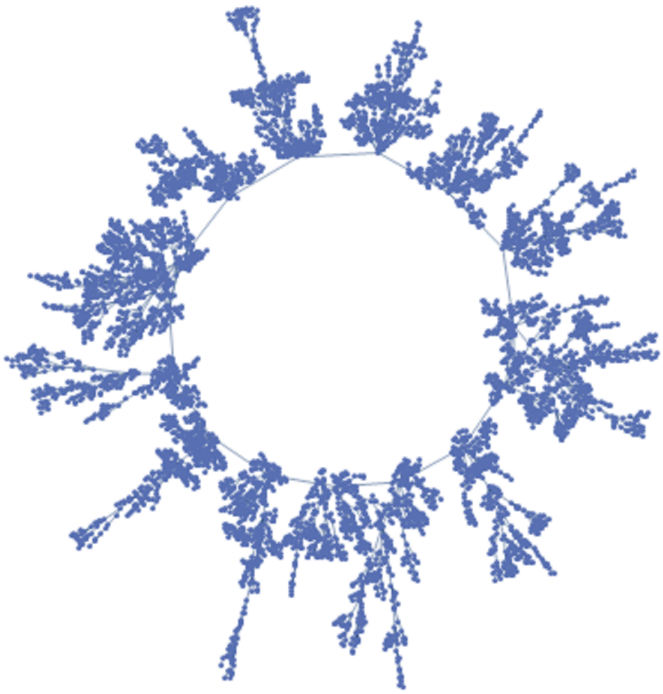
i.

() Regla 150



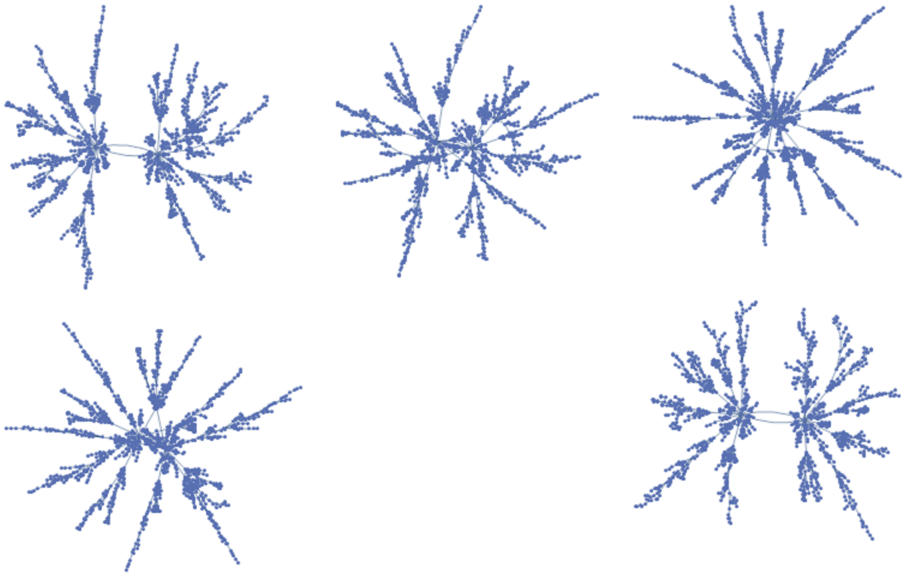
i.

() Regla 152



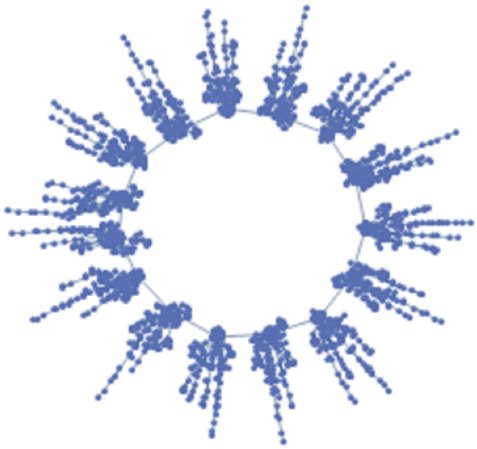
i.

() Regla 156



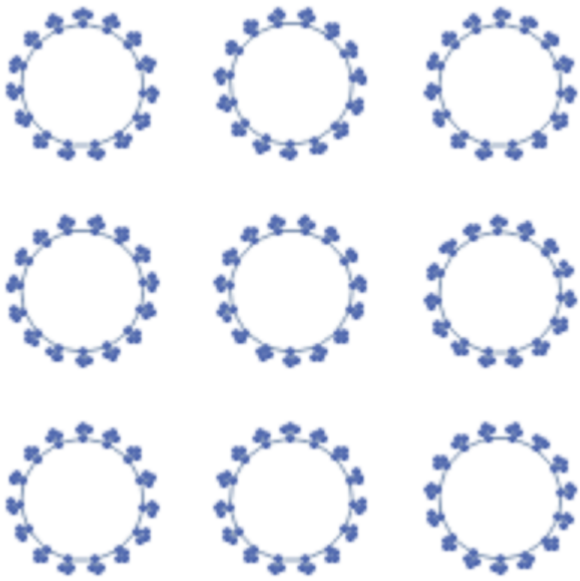
i.

() Regla 162



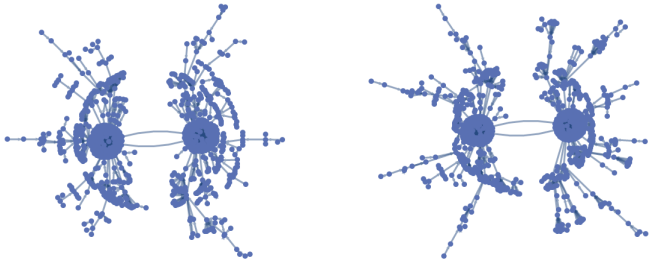
i.

() Regla 170



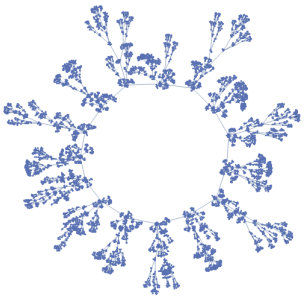
i.

() Regla 178



i.

() Regla 184



i.

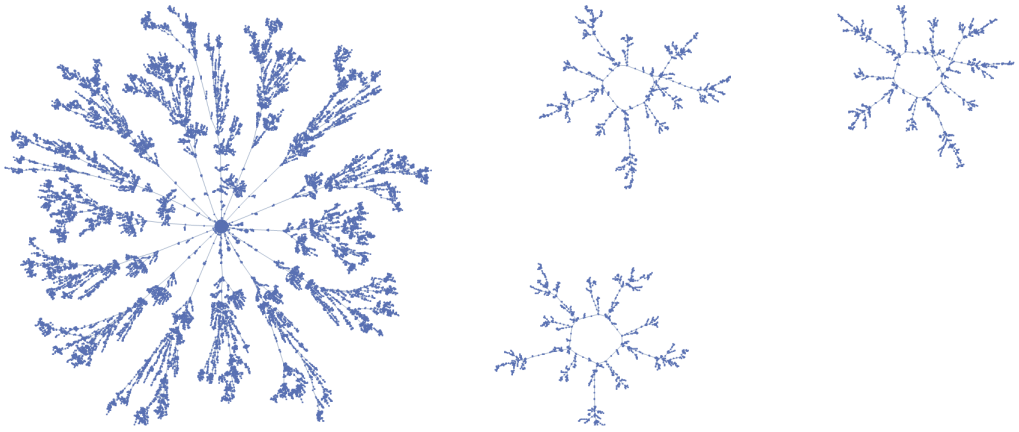
2.4.3 Clase III - Complejos

1. Descripción

   (a) Este tipo de autómatas presentan diversos atractores, los cuale spresentan comportamientos estáticos o cíclicos, llegando a cada uno de ellos con estados diferentes lo que al final produce cierto tipo de patrones combinados.

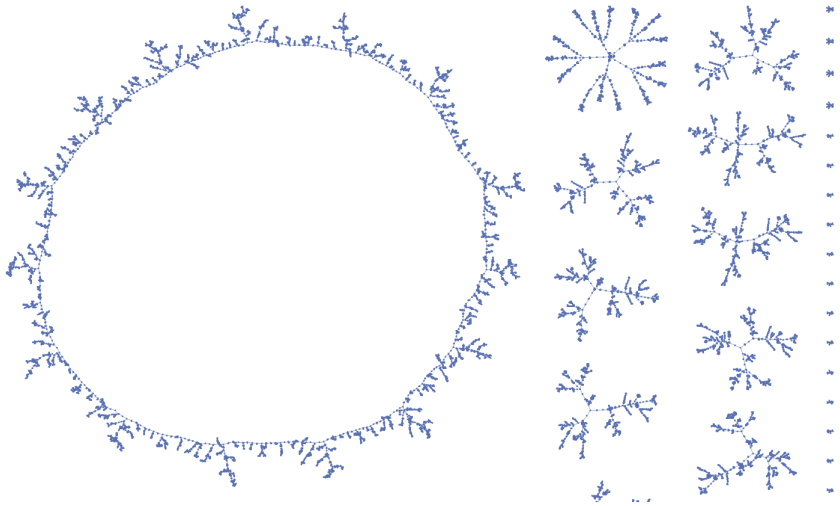
2. Reglas

   (a) Regla 22



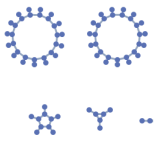
i.

(b) Regla 54



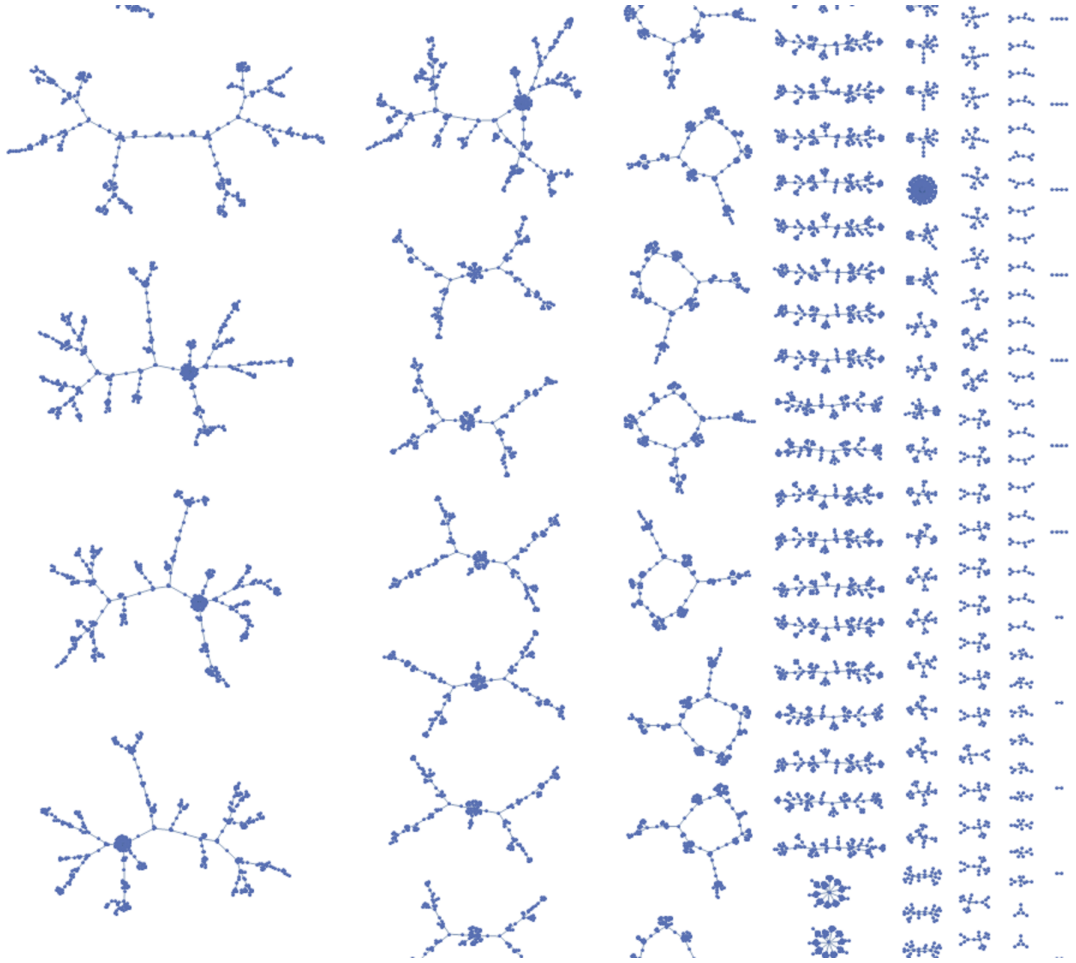
i.

(c) Regla 60



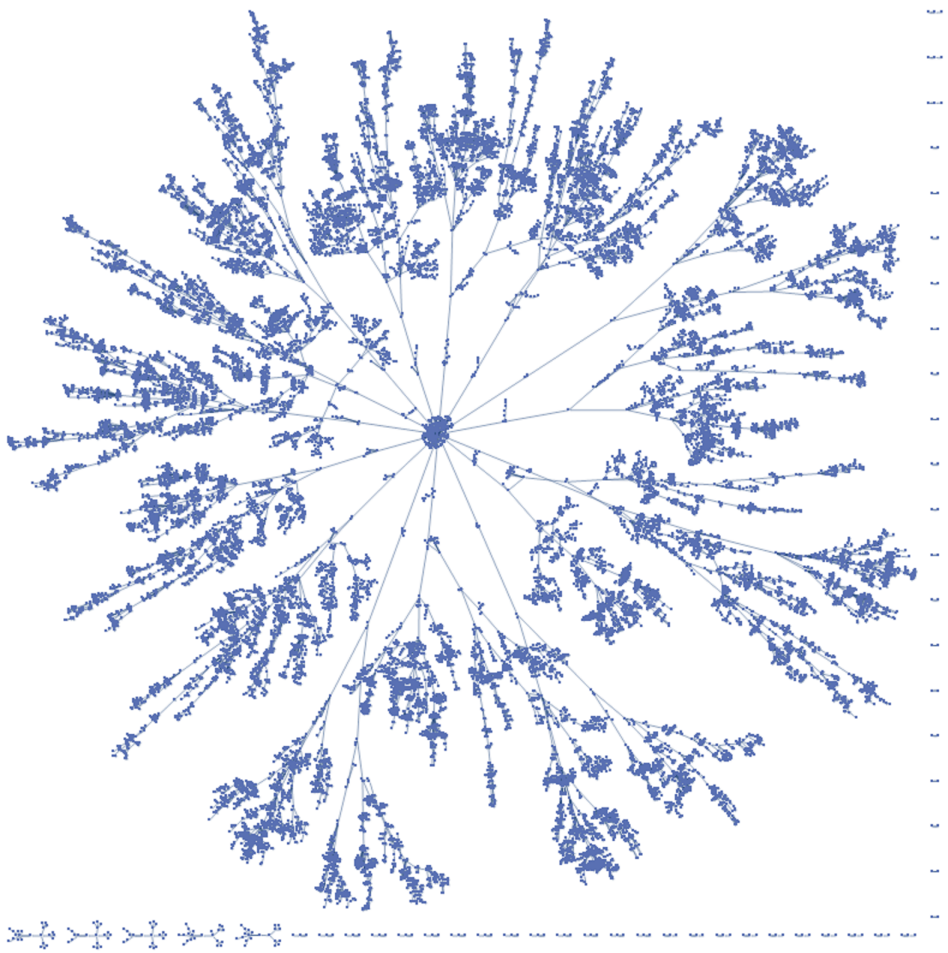
i.

(d) Regla 73



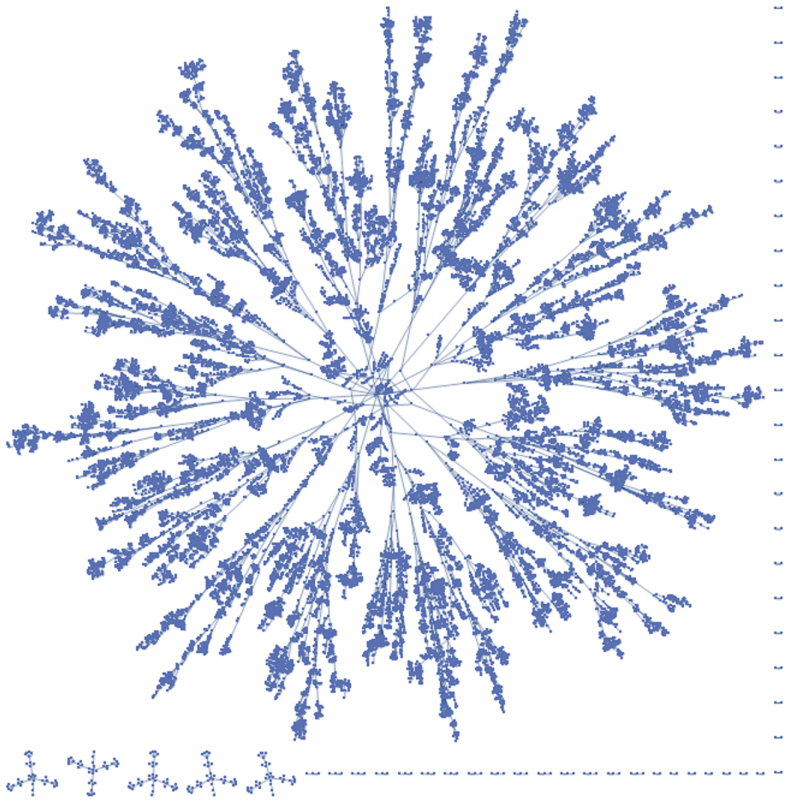
i.

(e) Regla 146



i.

(f) Regla 161

i. 

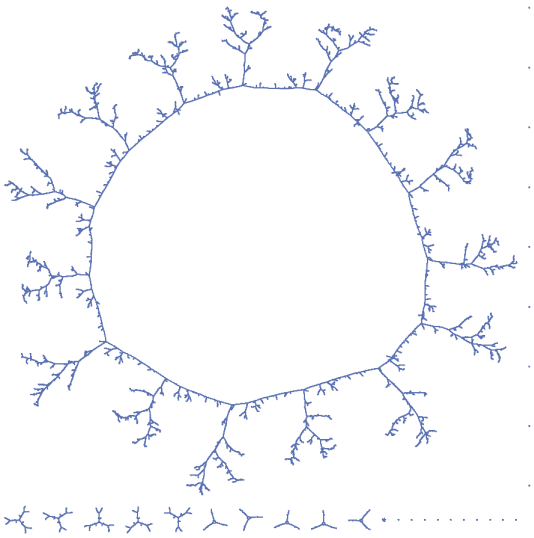
2.4.4 Clase IV - Caóticos

1. Descripción

   (a) Este tipo de autómatas celulares presenta combinaciones de los atractores de autómatas celulares clase I y clase II pero a su vez tiene ciclos muy muy largos lo que genera comportamientos erráticos pero con secciones en aparente orden, suelen tener raman tupdias cortas y girar en torno a un ciclo.

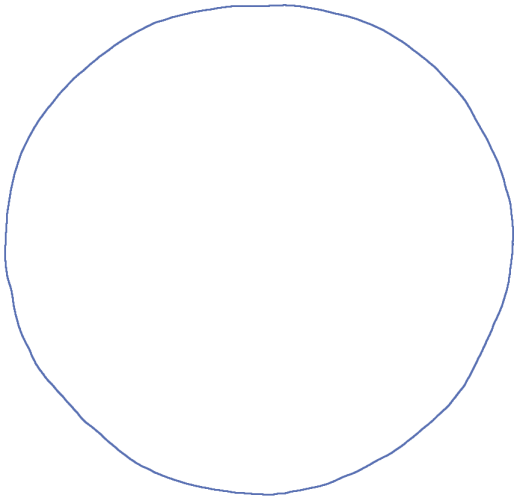
2. Reglas

   (a) Regla 30



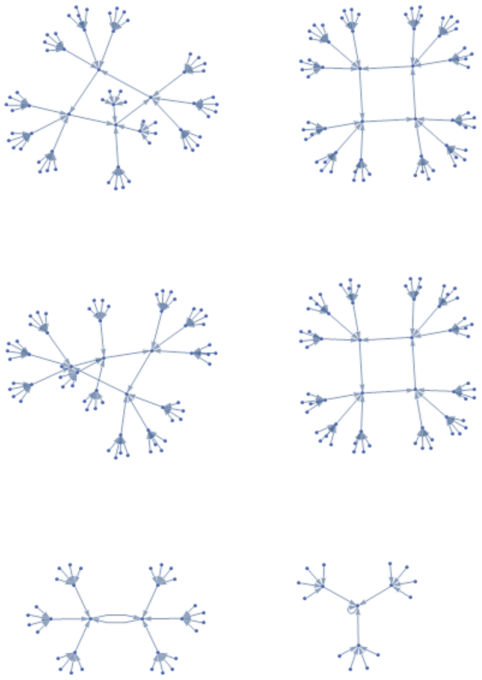
i.

(b) Regla 45



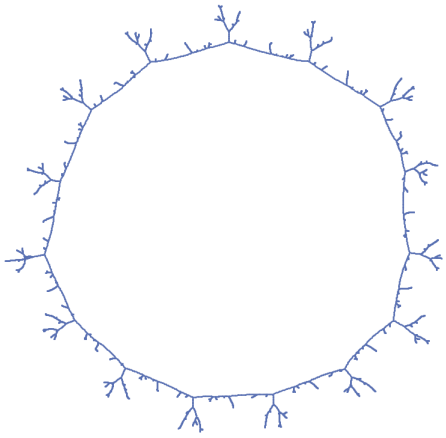
i.

(c) Regla 90

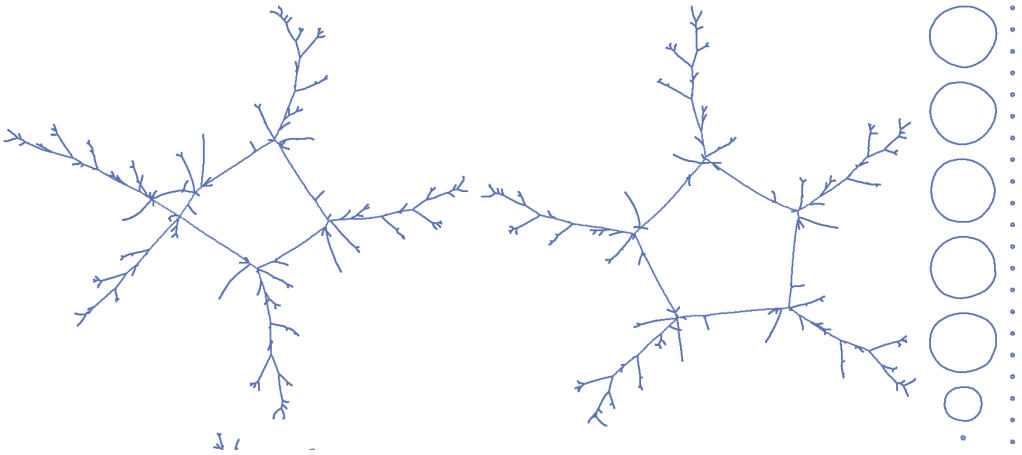


i.

(d) Regla 106

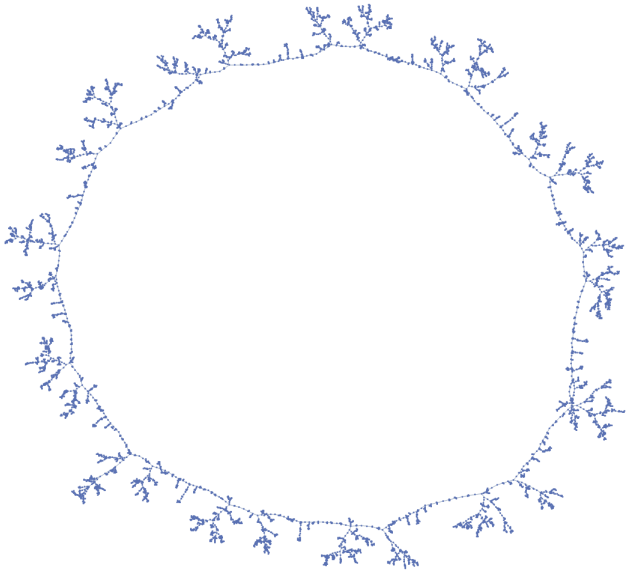


i.



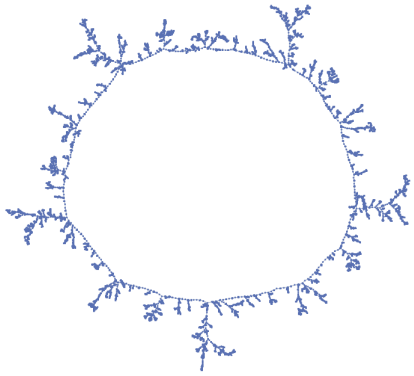
ii.

(e) Regla 110



i.

(f) Regla 137



i.