

# Autómatas Celulares Lineales Reversibles

Proyecto de Tesis.

Seck Tuoh Mora Juan Carlos  
Matemáticas Aplicadas y Computación  
Universidad Nacional Autónoma de México  
Campus Acatlán

Asesor:  
Harold V. McIntosh

Agosto, 1997

# Índice

<b>1</b>	<b>Introducción</b>	<b>4</b>
<b>2</b>	<b>Conceptos Básicos de Autómatas Celulares.</b>	<b>5</b>
2.1	Origen de los Autómatas Celulares. . . . .	5
2.1.1	Trabajo de von Neumann. . . . .	5
2.1.2	Trabajo de John Conway. . . . .	5
2.1.3	Trabajo de Stephen Wolfram. . . . .	6
2.2	Funcionamiento de los Autómatas Celulares Lineales. . . . .	7
2.2.1	Conceptos de <i>estado y vecindad</i> . . . . .	7
2.2.2	Descripción del mecanismo de evolución. . . . .	8
2.2.3	Características de los Autómatas Celulares como sistemas dinámicos discretos y caóticos. . . . .	10
2.2.4	Clasificación de Wolfram. . . . .	11
2.2.5	Aplicaciones de los Autómatas Celulares. . . . .	12
2.3	Herramientas utilizadas en el estudio de los Autómatas Celulares Lineales. . . . .	15
2.3.1	Diagrama de de Brijjn. . . . .	15
2.3.2	Diagrama de Subconjuntos. . . . .	18
2.3.3	Diagrama de Parejas. . . . .	21
<b>3</b>	<b>Reversibilidad y su manifestación en los Autómatas Celulares Lineales.</b>	<b>23</b>
3.1	Antecedentes Históricos . . . . .	23
3.1.1	Trabajo realizado por Moore. . . . .	23
3.1.2	Trabajo realizado por Hedlund. . . . .	23
3.1.3	Trabajo realizado por Fredkin. . . . .	24
3.2	Reversibilidad . . . . .	24
3.2.1	Concepto de Reversibilidad . . . . .	24
3.2.2	Ejemplos de Sistemas Reversibles. . . . .	24
3.3	Características de los Autómatas Celulares Lineales Reversibles. . . . .	25
3.3.1	Concepto de Ancestro y Jardín del Edén. . . . .	25
3.3.2	Sobreyectividad e Inyectividad en un Autómata Celular Lineal Reversible . . . . .	26
3.4	Estudio de los Autómatas Celulares Lineales Reversibles mediante las herramientas utilizadas para su análisis . . . . .	39
3.4.1	Características en el diagrama de de Brijjn . . . . .	39
3.4.2	Características en el diagrama de Parejas . . . . .	41
3.4.3	Diagrama de Subconjuntos . . . . .	42
3.4.4	Ejemplos de Autómatas Celulares Lineales no Reversibles . . . . .	47
3.4.5	Ejemplos de Autómatas Celulares Lineales Reversibles . . . . .	50
3.5	Procedimiento para encontrar la regla "inversa" de un Autómata Celular Lineal Reversible . . . . .	53
3.5.1	Procedimiento de Fredkin . . . . .	53

<b>4</b>	<b>Métodos actuales para encontrar Autómatas Celulares Lineales Reversibles.</b>	<b>58</b>
4.1	Algoritmo de Hillman . . . . .	58
4.2	Algoritmo utilizando los Indices de Welch . . . . .	63
<b>5</b>	<b>Propuesta de un método para encontrar Autómatas Celulares Lineales Reversibles.</b>	<b>65</b>
5.1	Propiedades de la matriz de evolución de los Autómatas Celulares Lineales Reversibles . . . . .	65
5.1.1	Propiedades por cada estado . . . . .	67
5.1.2	Propiedades globales de la matriz . . . . .	68
5.2	Método para encontrar Autómatas Celulares Lineales Reversibles . .	73
5.3	Comentarios sobre el método propuesto . . . . .	77
<b>6</b>	<b>Presentación de los resultados obtenidos con el método propuesto</b>	<b>78</b>
6.1	Resultados para los Autómatas Celulares Lineales Reversibles (4,h) .	78
6.2	Resultados para los Autómatas Celulares Lineales Reversibles (5,h) .	81
6.3	Resultados para los Autómatas Celulares Lineales Reversibles (6,h) .	87
6.4	Comparación de resultados con los otros métodos de cálculo de reversibles . . . . .	93
<b>7</b>	<b>Conclusiones</b>	<b>94</b>
7.1	Perspectivas para mejorar el cómputo de los Autómatas Celulares Lineales Reversibles . . . . .	94
7.2	Futuro de las aplicaciones de los Autómatas Celulares Reversibles . .	95
<b>8</b>	<b>Apéndice A</b>	<b>96</b>

# 1 Introducción

Sin duda alguna la computación ha sido una de las áreas de investigación que más ha avanzado en los últimos 50 años; este avance no es casual, sino se debe a las necesidades y desarrollos que actualmente se tiene en el manejo de grandes volúmenes de información, el auge de las telecomunicaciones, la disminución del tiempo en cálculos matemáticos, la simulación de los fenómenos naturales o el estudio de los sistemas con comportamientos "caóticos" entre otras muchas.

Dentro de estos dos puntos finales, uno de los campos que más se ha investigado es el de autómatas celulares, principalmente porque su construcción e implementación en una computadora es muy sencilla, sin embargo la comprensión de sus propiedades, una de ellas la *reversibilidad* en tales sistemas, aún está lejos de lograrse por completo.

El siguiente trabajo se enfoca en el análisis de la reversibilidad en los *Autómatas Celulares Lineales* o en una dimensión; en la sección 2 se explica las propiedades básicas y herramientas que existen para el estudio de tales sistemas, la sección 3 expone los conceptos de reversibilidad, su manifestación en los autómatas celulares lineales y los trabajos que se han desarrollado al respecto, la sección 4 trata sobre dos algoritmos que buscan encontrar todos los posibles autómatas celulares lineales reversibles, la sección 5 propone un método propio para realizar tal tarea y la sección 6 presenta los resultados obtenidos por este método, haciendo una comparación con los algoritmos de la sección 4.

## 2 Conceptos Básicos de Autómatas Celulares.

Antes de hacer un análisis de un tema en especial sobre los Autómatas Celulares Lineales (ACL), es importante explicar que son, de donde surgen y las propiedades básicas que éstos tienen así como su nomenclatura y la aplicación de la Teoría de Gráficas para el estudio de los mismos.

### 2.1 Origen de los Autómatas Celulares.

#### 2.1.1 Trabajo de von Neumann.

El concepto de autómatas celular (AC) fué desarrollado en los finales de los años 40's por John von Neumann; en esta época la manufactura de automóviles y bienes eléctricos empezaba a automatizarse y von Neumann se interesaba en crear una máquina que pudiera manufacturarse a sí misma [18], es decir, autorreproducirse.

Como carecía de los medios físicos y dispositivos técnicos que le permitieran hacer su proyecto, trató de hacerlo utilizando una forma muy simple y abstracta siguiendo una sugerencia de su amigo Stanislaw M. Ulam; esencialmente, Ulam sugirió un espacio cuadrulado donde cada cuadro podía ser ocupado por una *célula* dada que podía tener un número finito de estados, el tiempo en este espacio avanzaba en lapsos discretos, cada célula hacía una transición a un nuevo estado dependiendo de su valor actual y el de las células en cada borde o *vecinos*.



Figura 1: Vecindad de von Neumann.

De este modo, von Neumann pudo desarrollar un AC que podía autorreproducirse, contaba con 29 estados diferentes para cada célula; cabe señalar que cuando en 1953 Watson y Crick descubrieron la estructura molecular del ADN observaron que tenía muchas de las características del modelo de von Neumann [23].

John von Neumann muere en 1957 y no publica su trabajo pero éste es editado por Burks [2] en 1966, en ese mismo tiempo Codd [3] trabaja en una variante del autómatas de von Neumann con solo 8 estados.

#### 2.1.2 Trabajo de John Conway.

En 1970 John Horton Conway desarrolló un juego matemático llamado *Life*, éste aparece publicado en la columna de Martin Gardner en Scientific American [6].

En realidad, Life no es un juego que alguien pueda jugar, sino que es un autómata celular que evoluciona en el tiempo y el espectador observa como ocurre esta evolución, recibe su nombre debido a que cada célula tiene dos estados posibles, vivo o muerto, la vecindad que utiliza Life es la siguiente:



Figura 2: Vecindad de Moore.

El nombre de esta vecindad se debe a su creador [2], una célula que está viva puede morir por tener muchas vecinas vivas alrededor (sobrepoblación) o muy pocas de éstas (aislamiento); pero si hay dos o tres células vivas en su vecindad la célula se mantiene viva, si una celda está vacía hay un “nacimiento” si en su vecindad existen exactamente tres células vivas.

Con estas simples reglas, el juego de Life puede desarrollar un comportamiento bastante complejo muy parecido al de un grupo de microbios en una gota de agua que se observan bajo un microscopio, es por eso que ganó gran popularidad y produjo que muchos investigadores se dedicaran al estudio de los AC.

### 2.1.3 Trabajo de Stephen Wolfram.

El interés por el estudio de temas tales como el caos o la complejidad se ha visto incrementado gracias al desarrollo y accesibilidad que han ganado los equipos de computación.

Aprovechando estas circunstancias, surge en los 80's el trabajo de Stephen Wolfram que investiga la complejidad a su nivel más fundamental, él basa su estudio de los sistemas complejos particionando éstos en sus elementos más simples y estudiando como esas partes interactúan para producir un comportamiento complejo, Wolfram señala que los sistemas complejos se rigen por componentes y leyes básicas muy simples, pero la complejidad aumenta debido a la gran cantidad de estos simples componentes interaccionando simultáneamente, es por eso que hace uso de los AC ya que con ellos se puede analizar en detalle su comportamiento que es completamente determinístico y aún así generar patrones complejos.

Un AC es un modelo muy simple en su construcción pero con las características necesarias para tener un comportamiento complejo, además de que resulta fácil su implementación en una computadora permitiendo modelar con ellos sistemas físicos, químicos o biológicos [24].

Un punto importante a resaltar es que mientras von Neumann y Conway se concentraron en desarrollar un AC con una cierta regla que cumpliera sus propósitos y estudiar su comportamiento, Wolfram se enfocó a analizar un número de reglas diferentes en un AC principalmente en el caso lineal.

## 2.2 Funcionamiento de los Autómatas Celulares Lineales.

### 2.2.1 Conceptos de *estado* y *vecindad*.

Un ACL se construye de la siguiente manera; se coloca una serie de celdas en línea, es decir un arreglo lineal de elementos los cuales se denominan *células*.

Cada célula puede tener un número finito de valores, ya sea un valor entero, una letra o un color, lo que se quiera que represente cada una, a estos valores se les denomina *estados*, todas las células tienen el mismo número posible de estados.

Las relaciones que existe entre las células constituyen los tipos de *vecindades* que forman, se puede tomar para una célula dada un radio de vecindad  $r$ , es decir  $r$  vecinos a cada lado de la misma, obteniendo un tamaño de vecindad igual a  $2r + 1$ , el número total de células que forman la vecindad.

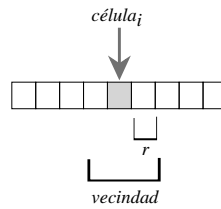


Figura 3: ACL donde cada célula tiene un vecino a cada lado ( $r = 1$ ), el tamaño total de la vecindad es  $2r + 1 = 3$ .

En los extremos del arreglo la célula inicial y final no tienen el mismo vecindario que las células interiores, por ésto para preservar la uniformidad de todos los vecindarios, la cadena puede ser imaginada como un anillo, así la primera célula del arreglo es la vecina derecha de la última y ésta es la vecina izquierda de la primera.

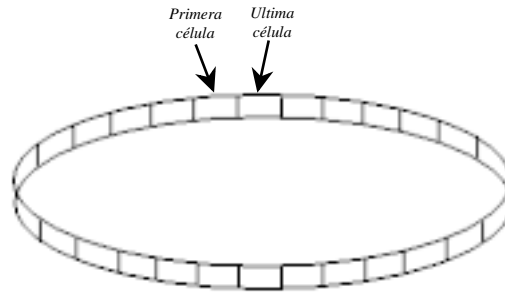


Figura 4: Cadena de células dispuesta en forma de anillo.

Wolfram propone una notación para los ACL, la cual es  $(k, r)$ , es decir existen  $k$  estados posibles para cada célula junto con  $r$  células de cada lado para formar su vecindario, por ejemplo, para un ACL(2,1) el vecindario contiene tres células y cada célula puede tener un estado de dos posibles, el número posible de vecindades diferentes es  $k^{2r+1}$ , en este caso  $2^3 = 8$ , si los estados posibles fueran 0, 1, las vecindades que tendríamos serían (000, 001, 010, 011, 100, 101, 110, 111).

### 2.2.2 Descripción del mecanismo de evolución.

Un ACL evoluciona de la siguiente forma, el nuevo valor de cualquier célula en el tiempo  $t_1$  está en función de los valores que tenga ella misma y sus vecinas en el tiempo  $t_0$ , todas las células en el arreglo actualizan su valor simultáneamente, el tiempo avanza en etapas discretas.

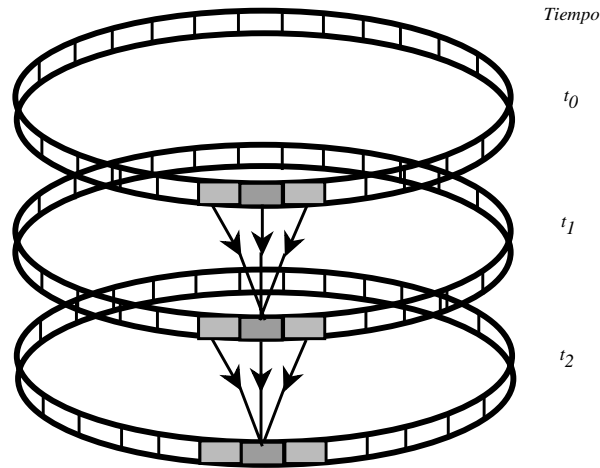


Figura 5: Forma en que evoluciona la  $i$ -ésima célula en un  $ACL(k,1)$ .

En realidad un vecindario no tiene que estar centrado en una célula [13], si el tamaño del mismo es par, la célula siguiente puede ocupar un lugar entre media célula.

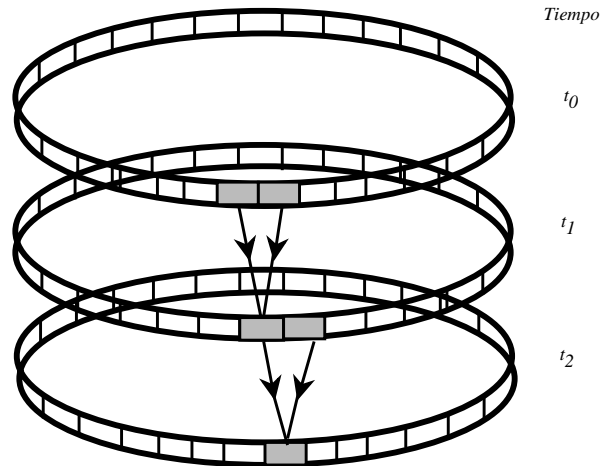


Figura 6: Forma en que evoluciona la  $i$ -ésima célula en un  $ACL(k,h)$ ,  $h = 0.5$ .



El patrón de valores que se observa a través de todo el arreglo se denomina el estado global del ACL en un tiempo dado [26], como todas las células en el arreglo actualizan su valor al mismo tiempo, ésto provoca que cambie el estado global del arreglo del tiempo  $t_i$  al  $t_{i+1}$ , a los diferentes estados globales que el arreglo puede tomar a través del tiempo se denomina *evolución* del ACL.

Cualquier patrón de estados en el arreglo puede ser una condición desde donde comience la evolución del ACL, a tal condición se le nombra *configuración inicial*, y al estado global en el tiempo  $i$  se le denomina *generación  $i$* , por último. la evolución de un ACL depende de su *regla de evolución* por la cual las células cambian de estado de una generación a la próxima, esta regla de evolución es una función de transición aplicada a cada célula y sus vecinos que se puede expresar en forma tabular.

$Tiempo_i$	Evolución	$Tiempo_{i+1}$
000	→	1
001	→	1
010	→	1
011	→	1
100	→	0
101	→	0
110	→	0
111	→	0

Tabla 1: Regla de evolución de un ACL(2,1).

Cada célula en una vecindad puede tomar  $k$  valores posibles por lo que el número de reglas de evolución totales de un ACL(k,r) se puede calcular con  $k^{k^{2r+1}}$ ; para un ACL(2,1) existen  $2^{2^3} = 256$  reglas posibles.

Para distinguir a los ACL, Wolfram propone utilizar sus reglas de evolución asociadas; como esta regla crece a medida que hay más estados y/o vecinos, se utiliza el equivalente decimal de la regla o cambios a otra base según convenga, a tal representación se le conoce como *Número Wolfram*, por ejemplo, para un ACL(2,1) podemos utilizar el equivalente decimal ya que la regla es un número binario, la siguiente tabla muestra la regla 15 para este caso:

Vecindades	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1
	0	1	0	1	0	1	0	1
Evolución	1	1	1	1	0	0	0	0
No. Binario	$2^0$	$2^1$	$2^2$	$2^3$	$2^4$	$2^5$	$2^6$	$2^7$
No. Decimal	1	2	4	8	0	0	0	0

Tabla 2: Regla de evolución 15 de un ACL(2,1).

Mientras tanto, para un ACL(4,h) se maneja una notación hexadecimal en bloques de dos, veamos la regla 0055AAFF:

0 0	0 0	1 1	1 1	2 2	2 2	3 3	3 3
0 1	2 3	0 1	2 3	0 1	2 3	0 1	2 3
0 0	0 0	1 1	1 1	2 2	2 2	3 3	3 3
4 <sup>1</sup> 4 <sup>0</sup>	4 <sup>1</sup> 4 <sup>0</sup>	4 <sup>1</sup> 4 <sup>0</sup>	4 <sup>1</sup> 4 <sup>0</sup>	4 <sup>1</sup> 4 <sup>0</sup>	4 <sup>1</sup> 4 <sup>0</sup>	4 <sup>1</sup> 4 <sup>0</sup>	4 <sup>1</sup> 4 <sup>0</sup>
0	0	5	5	10	10	15	15
0	0	5	5	A	A	F	F

Tabla 3: Regla de evolución 0055AAFF de un ACL(4,h).

### 2.2.3 Características de los Autómatas Celulares como sistemas dinámicos discretos y caóticos.

Se puede resumir que las características de un ACL es que cada célula tiene valores discretos, el espacio donde evoluciona es discreto, el sistema cambia su estado global en función de sus valores actuales y evoluciona a través del tiempo en pasos discretos; estas últimas características los definen como sistemas dinámicos de tipo discreto.

Un sistema o fenómeno se dice que es caótico si el comportamiento de los mismos es muy difícil o imposible de predecir, es decir, demuestran un comportamiento aperiódico el cual no desaparece [17] a esta aperiodicidad se le denomina caos, sin embargo, éste tiene peculiaridades muy particulares [11] como es que los miembros de sistemas caóticos son muy simples, el comportamiento del sistema es determinístico generado por reglas fijas que no contienen probabilidades o elementos de cambio pero generan un comportamiento impredecible, por lo que se puede concluir que determinismo y predicibilidad no son equivalentes [17].

Estas características se pueden claramente observar en un ACL, por ejemplo, la regla de evolución que rige su comportamiento es determinística por completo y es constante a través del tiempo, la evolución de una célula depende de ella misma y sus vecinas, por lo que el comportamiento global del sistema es resultado de las relaciones locales de las células del mismo, siendo con ésto capaz de desarrollar patrones muy complejos.

### 2.2.4 Clasificación de Wolfram.

Wolfram propone cuatro clases para clasificar a los ACL dependiendo de sus características más visibles, en la evolución de un ACL se observa que cada regla tiene su propia personalidad y que algunas reglas comparten tanto similitudes como diferencias, su clasificación es la siguiente.

- **CLASE 1.-** El ACL evoluciona después de un número finito de pasos de su estado inicial a un estado homogéneo único, es decir, todas las células tienen el mismo valor.

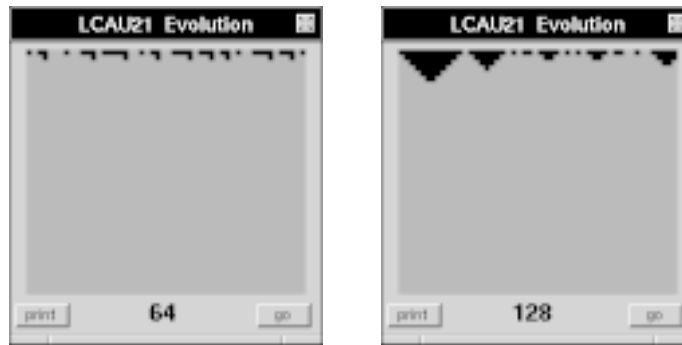


Figura 7: ACL's Clase 1.

- **CLASE 2.-** El ACL genera estructuras simples separadas desde un estado inicial dado.

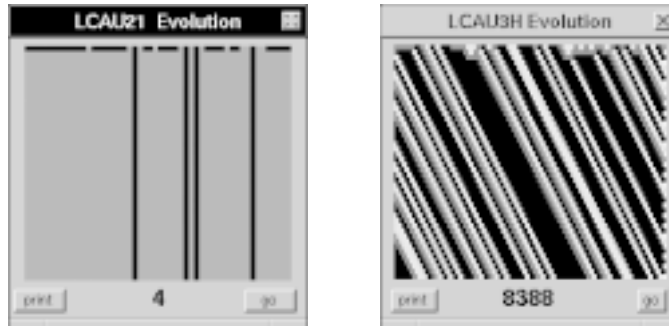


Figura 8: ACL's Clase 2.

- **CLASE 3.-** El ACL desde casi cualquier posible configuración inicial produce patrones aperiódicos o caóticos.

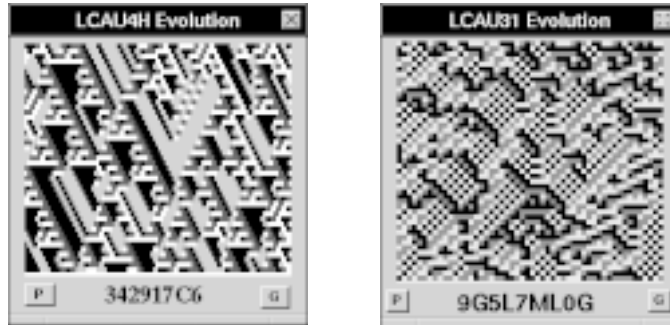


Figura 9: ACL's Clase 3.

- **CLASE 4.-** El ACL produce estructuras con comportamiento complejo (entre el orden y el caos).

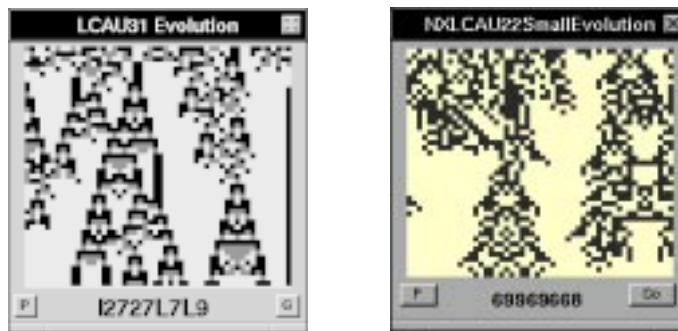


Figura 10: ACL's Clase 4.

### 2.2.5 Aplicaciones de los Autómatas Celulares.

Se puede identificar tres vertientes principales en donde se han utilizado AC para algún propósito:

1. *Simulación de sistemas naturales.*

Dentro de este contexto se busca simular sistemas en donde el comportamiento de los mismos se rija por la interacción local de sus componentes, de este modo se han podido modelar el crecimiento de cristales, incendios forestales, modelos de reacciones químicas como la reacción de Belousov-Zhabotinsky, mecánica de fluidos, patrones de pigmentación de piel, crecimiento de conchas marinas y corales, comportamiento de colonias de microorganismos entre otros; ejemplos de estas aplicaciones se pueden encontrar en [8, 24].

## 2. Estudios Teóricos.

En este campo se utilizan a los AC para estudiar áreas como complejidad, sistemas caóticos, termodinámica, entropía, computación en paralelo, computación universal, teoría de lenguajes computacionales o estudio de patrones fractales como se muestra en [12, 25].

## 3. Realización de tareas específicas.

Aquí se busca construir un AC que sea capaz de desarrollar un proceso en especial, esto puede ser desde creación de fondos para diseños artísticos, procesamiento de imágenes o encriptación de datos [8]; se han creado ACL que realizan tareas muy sencillas como podría ser un ACL(2,1) que dada una configuración inicial, produzca después de un número finito de pasos un estado global fijo dependiendo que estado sea mayoritario en la configuración inicial [4].

Se puede observar que estos campos de desarrollo no son excluyentes, ya que un mismo trabajo puede caer en las tres vertientes, un ejemplo puede ser el AC de von Neumann, ya que puede ser visto como una simulación de la autorreproducción de organismos microscópicos, se puede estudiar en éste el funcionamiento de un sistema complejo y es un AC que realiza una tarea en especial, la replica de él mismo.

Mostrando un ejemplo muy sencillo de una aplicación, propongo un ACL que ordene de izquierda a derecha en orden descendente una secuencia aleatoria de ceros y unos después de un número finito de pasos.

0110001100	$t_0$
↓	⋮
1111000000	$t_n$

Tabla 4: Ordenación de una secuencia de 0's y 1's.

Para resolver esta simple tarea se tomará un ACL(3,1) donde dos estados representan los valores de 0 y 1 a ordenar y el tercero será un margen para la ordenación, la regla de evolución de tal autómata es la siguiente:

0 0 0	0 0 0	0 0 0	1 1 1	1 1 1
0 0 0	1 1 1	2 2 2	0 0 0	1 1 1
0 1 2	0 1 2	0 1 2	0 1 2	0 1 2
0 1 0	0 0 0	2 2 2	0 1 0	1 1 1
$3^0$ $3^1$ $3^2$	$3^0$ $3^1$ $3^2$	$3^0$ $3^1$ $3^2$	$3^0$ $3^1$ $3^2$	$3^0$ $3^1$ $3^2$
3	0	Q	3	D

1 1 1	2 2 2	2 2 2	2 2 2
2 2 2	0 0 0	1 1 1	2 2 2
0 1 2	0 1 2	0 1 2	0 1 2
2 2 2	0 1 0	1 1 1	2 2 2
$3^0$ $3^1$ $3^2$	$3^0$ $3^1$ $3^2$	$3^0$ $3^1$ $3^2$	$3^0$ $3^1$ $3^2$
Q	3	D	Q

Tabla 5: Regla de evolución QD3QD3Q03 para un ACL(3,1).

Esta regla de evolución del ACL(3,1) se clasifica QD3QD3Q03 utilizando en ésta una notación de base 27 en bloques de 3 para este ACL en especial, la regla se lee empezando desde la vecindad 222.

Analizando un poco la regla de evolución, se observa que el estado 0 hace corrimientos a la derecha, el estado 1 a la izquierda y el estado 2 permanece estable al evolucionar, a continuación se presenta un ejemplo de su funcionamiento en donde la configuración inicial es una secuencia aleatoria de 0's y 1's, limitada a los lados con el estado 2.

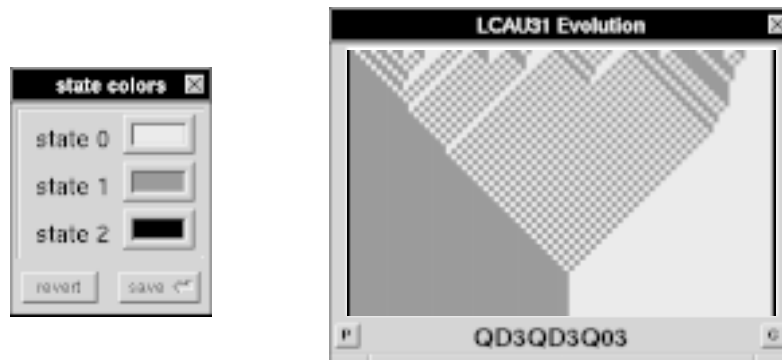


Figura 11: Ordenación de una secuencia aleatoria de 0's y 1's con un ACL(3,1) regla QD3QD3Q03.

En este ejemplo es posible ver de manera muy sencilla una computación en paralelo que hace el ACL propuesto al ordenar al mismo tiempo varias secuencias aleatorias de 0's y 1's.

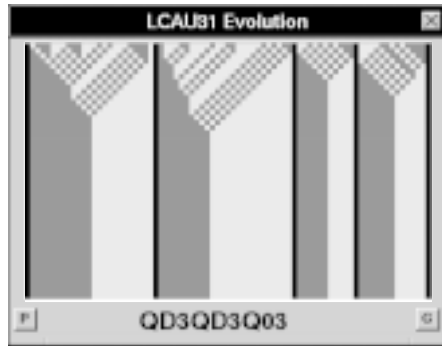


Figura 12: Ordenación “paralela” de secuencias aleatorias de 0's y 1's con un ACL.

## 2.3 Herramientas utilizadas en el estudio de los Autómatas Celulares Lineales.

Un análisis acerca de los ACL puede hacerse observando directamente como evoluciona el mismo, sin embargo existen 3 herramientas basadas en la Teoría de Gráficas que son de gran utilidad para el mismo objetivo ya que con éstas se consigue obtener características importantes del comportamiento local y global del ACL, además que presentar la dinámica de cualquier sistema por medio de gráficas tiene la ventaja de hacer más claro (visualmente hablando) su comportamiento, agregando que las representaciones matriciales de las mismas son también de gran utilidad.

### 2.3.1 Diagrama de de Bruijn.

El diagrama de *de Bruijn* nos representa como evolucionan los vecindarios en un ACL, sus orígenes vienen de la teoría del registro de corrimientos; los nodos del diagrama representan vecindades parciales del ACL, si éste tiene  $k$  estados y un tamaño de vecindad  $2r + 1$  el diagrama tendrá  $k^{2r}$  vértices y  $k^{2r+1}$  ligas.

Los nodos resultan de tomar cadenas diferentes de tamaño  $2r$  formadas con los  $k$  estados y las ligas unen los nodos para formar las vecindades completas donde los últimos  $2r - 1$  elementos del nodo donde parte la liga deben concordar con los primeros  $2r - 1$  elementos del nodo adonde llega, es decir, cuadrando como fichas de dominó y el color o etiqueta de la liga representa la evolución de la vecindad; tomemos un ACL(2,1) regla 90.

0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1
0	1	0	1	1	0	1	0

Tabla 6: Regla de evolución 90 de un ACL(2,1).

El diagrama de de Bruijn correspondiente tendrá  $2^2 = 4$  nodos con  $2^3 = 8$  ligas.

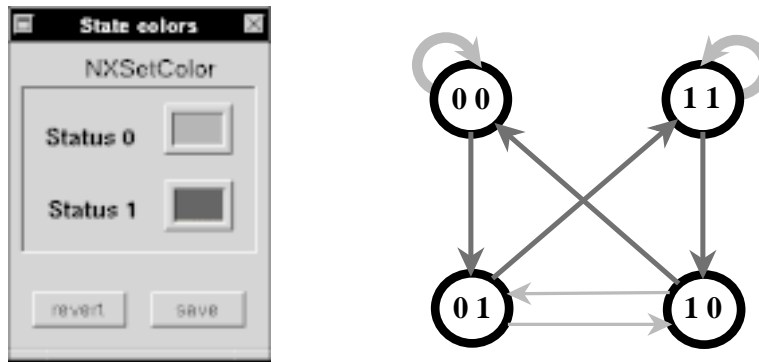


Figura 13: Diagrama de de Bruijn de un ACL(2,1) regla 90.

El ciclo en el nodo (00) significa que la vecindad 000 evoluciona en el estado 0, la liga que va del nodo (00) al (01) señala que la vecindad 001 evoluciona en 1 y así sucesivamente; para facilitar la representación gráfica se renombran los nodos con valores enteros del 0 en adelante, así el mismo diagrama toma la siguiente forma:

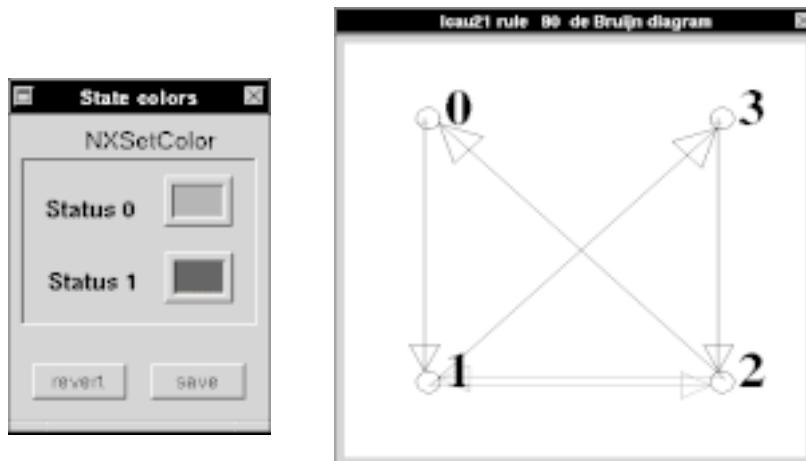


Figura 14: Diagrama de de Bruijn renombrado de un ACL(2,1) regla 90.



Esta gráfica también tiene una representación matricial a la cual se denomina *matriz de evolución*, donde los índices representan los nodos y sus elementos el valor de la liga que une del índice del renglón al de la columna.

$$\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\
 \hline
 0 & 0 & 1 & - & - \\
 1 & - & - & 0 & 1 \\
 2 & 1 & 0 & - & - \\
 3 & - & - & 1 & 0
 \end{array}$$

Tabla 7: Matriz de evolución de un ACL(2,1) regla 90.

Otro ejemplo del diagrama de de Bruijn ahora para un ACL(4,h).

0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	3	1	1	2	0	3	1	0	2	0	0	3	3	0	0

Tabla 8: Regla de evolución 0F08725C de un ACL(4,h).

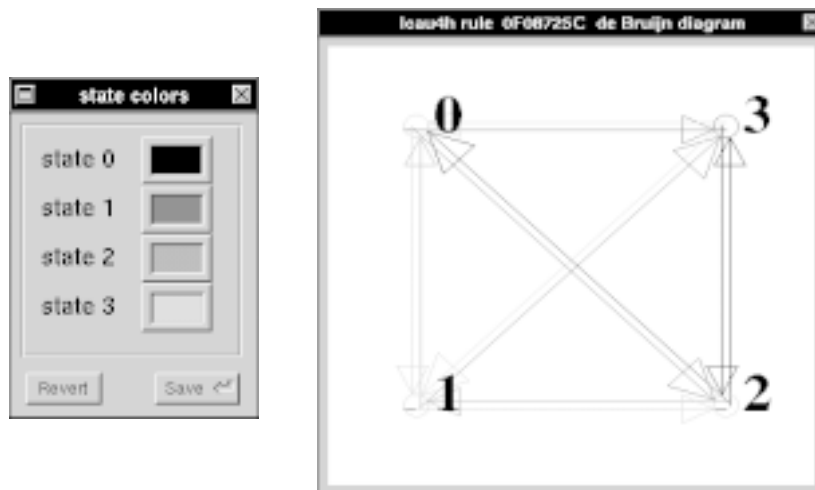


Figura 15: Diagrama de de Bruijn de un ACL(4,h) regla 0F08725C.

La matriz de evolución de dicho ACL(4,h) será:

$$\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\
 \hline
 0 & 0 & 3 & 1 & 1 \\
 1 & 2 & 0 & 3 & 1 \\
 2 & 0 & 2 & 0 & 0 \\
 3 & 3 & 3 & 0 & 0
 \end{array}$$

Tabla 9: Matriz de evolución del ACL(4,h) regla 0F08725C.

Una aplicación trivial del diagrama de de Bruijn es recobrar los valores de las células que al evolucionar producen una cadena dada de estados, por ejemplo para el ACL(4,h) la cadena 001 puede ser consecuencia de la evolución de las células 0002 ó 3203.

### 2.3.2 Diagrama de Subconjuntos.

El diagrama de de Bruijn nos da la oportunidad de trazar rutas en él y verificar que estados las originan, sin embargo una pregunta importante es saber si una ruta dada existe o es posible de construir en el diagrama [14]; para responder ésto se ha utilizado el diagrama de *Subconjuntos* desarrollado por Moore [15] el cual toma como base al diagrama de de Bruijn; los nodos son agrupados en subconjuntos que van desde el conjunto vacío, los subconjuntos de nodos unitarios, los subconjuntos de parejas distintas de nodos hasta el subconjunto que contenga todos los nodos; si el diagrama de de Bruijn tiene  $n$  nodos el diagrama de subconjuntos tendrá  $2^n$  nodos, veamos el caso para un ACL(2,1) regla 22.

0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	0

Tabla 10: Regla de evolución 22 del ACL(2,1).

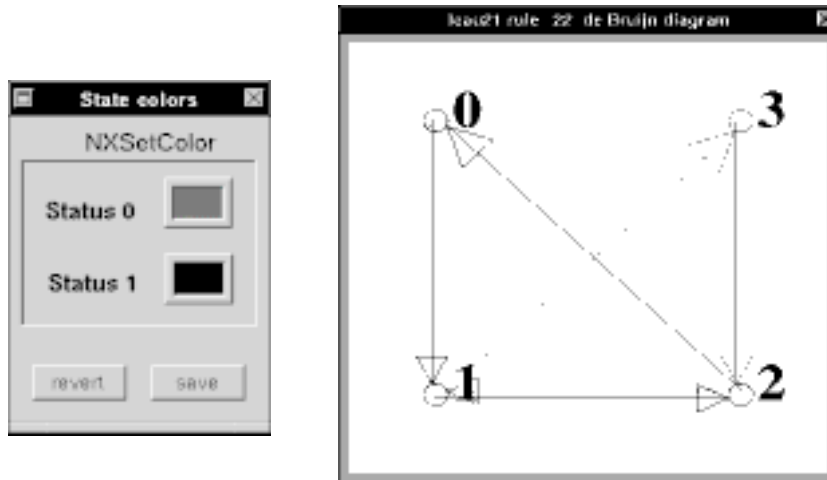


Figura 16: Diagrama de de Bruijn de un ACL(2,1) regla 22.

	0	1	2	3
0	0	1	-	-
1	-	-	1	0
2	1	0	-	-
3	-	-	0	0

Tabla 11: Matriz de evolución del ACL(2,1) regla 22.

El diagrama de subconjuntos tendrá  $2^4 = 16$  nodos que se enumerarán desde el 0 para el conjunto vacío hasta el 15 para el subconjunto total utilizando una notación binaria, es decir si tenemos 4 nodos se ordenan 0, 1, 2, 3; si el subconjunto lo forman los nodos (1, 2) en la secuencia de nodos tendríamos 0, 1, 1, 0 donde el 0 significa la ausencia del nodo y el 1 la presencia del mismo, con ésto el subconjunto (1, 2) le corresponde el valor 6 y así para todos los casos.

Subconjunto	Liga 0	Liga 1	Enumeración	Liga 0	Liga 1
$\emptyset$	$\emptyset$	$\emptyset$	0	0	0
0	0	1	1	1	2
1	3	2	2	8	4
2	1	0	4	2	1
3	2,3	$\emptyset$	8	12	0
0,1	0,3	1,2	3	9	6
0,2	0,1	0,1	5	3	3
0,3	0,2,3	1	9	13	2
1,2	1,3	0,2	6	10	5
1,3	2,3	2	10	12	4
2,3	1,2,3	0	12	14	1
0,1,2	0,1,3	0,1,2	7	11	7
0,1,3	0,2,3	1,2	11	13	6
0,2,3	0,1,2,3	0,1	13	15	3
1,2,3	1,2,3	0,2	14	14	5
0,1,2,3	0,1,2,3	0,1,2	15	15	7

Tabla 12: Subconjuntos del diagrama de de Bruijn del ACL(2,1) regla 22.

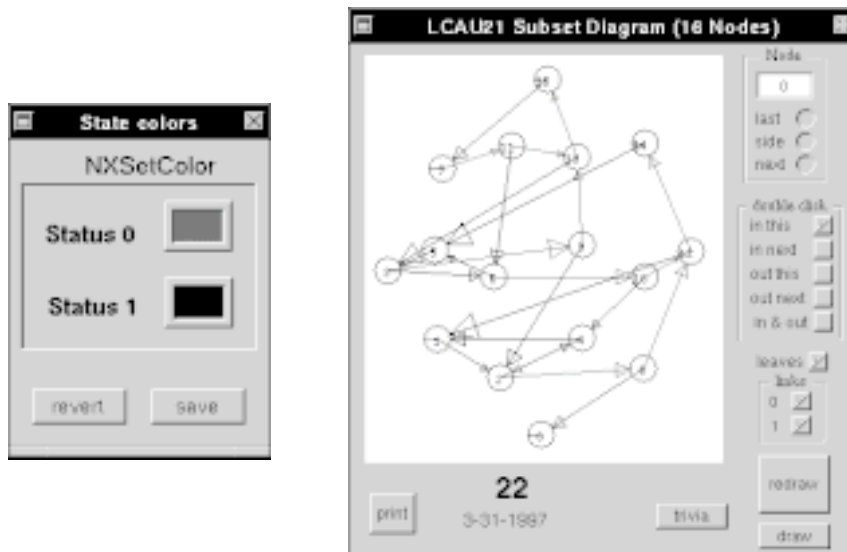


Figura 17: Diagrama de Subconjuntos de un ACL(2,1) regla 22.

Varias observaciones sobre la construcción del diagrama de subconjuntos son:

- A) El conjunto vacío liga a sí mismo con cualquier liga.
- B) Existe una liga del nodo  $a$  al nodo  $b$  si al menos un elemento del nodo  $a$  tiene enlace por medio de esa liga al nodo  $b$ , no importando que los demás elementos carezcan de tal conexión.

Otra importante razón para utilizar el diagrama de subconjuntos es que proporciona una funcionalidad que posiblemente no exista en el diagrama de de Bruijn ya que en este último dos ligas del mismo color pueden partir de un mismo nodo cosa que no ocurre en el diagrama de subconjuntos, donde cada nodo tiene una sola liga correspondiente a cada estado del ACL, el conjunto vacío asegura que en cada nodo todas las ligas estén definidas para todos los estados del autómata, la matriz de adyacencia del diagrama de subconjuntos para el ACL(2,1) regla 22 agrupando los subconjuntos de acuerdo a su tamaño es como sigue:

	15	14	13	11	7	12	10	9	6	5	3	8	4	2	1	0
15	1	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-
14	-	1	-	-	-	-	-	-	-	1	-	-	-	-	-	-
13	1	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-
11	-	-	1	-	-	-	-	-	1	-	-	-	-	-	-	-
7	-	-	-	1	1	-	-	-	-	-	-	-	-	-	-	-
12	-	1	-	-	-	-	-	-	-	-	-	-	-	-	1	-
10	-	-	-	-	-	1	-	-	-	-	-	-	1	-	-	-
9	-	-	1	-	-	-	-	-	-	-	-	-	-	1	-	-
6	-	-	-	-	-	-	1	-	-	1	-	-	-	-	-	-
5	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-
3	-	-	-	-	-	-	-	1	1	-	-	-	-	-	-	-
8	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	1
4	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1	-
2	-	-	-	-	-	-	-	-	-	-	-	1	1	-	-	-
1	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1	-
0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2

Tabla 13: Matriz del diagrama de subconjuntos del ACL(2,1) regla 22.

### 2.3.3 Diagrama de Parejas.

El producto cartesiano de dos gráficas tiene muchas aplicaciones, por ejemplo comparar rutas entre dos diagramas diferentes o dos rutas en el mismo diagrama, ésta es la función del diagrama de *Parejas* [14] el cual también toma como base al diagrama de de Bruijn; los nodos del diagrama de parejas son duplas de nodos del diagrama de de Bruijn, las ligas unen nodos donde ambos miembros de un nodo están conectados con la misma liga con los miembros del otro, de este modo las rutas del diagrama de parejas corresponden a parejas de rutas iguales en el diagrama de de Bruijn que por supuesto no tienen que compartir los mismos nodos, se puede decir que el diagrama original está implícito en el diagrama de parejas.

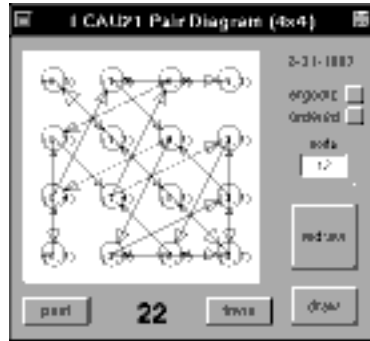


Figura 18: Diagrama de Parejas de un ACL(2,1) regla 22.

	00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33
00	1	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-
01	-	-	-	1	-	-	1	-	-	-	-	-	-	-	-	-
02	-	1	-	-	1	-	-	-	-	-	-	-	-	-	-	-
03	-	-	1	1	-	-	-	-	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-	-	1	-	-	1	-	-	-
11	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	1
12	-	-	-	-	-	-	-	-	1	-	-	-	-	1	-	-
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1
20	-	1	-	-	1	-	-	-	-	-	-	-	-	-	-	-
21	-	-	1	-	-	-	-	1	-	-	-	-	-	-	-	-
22	1	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-
23	-	-	-	-	-	-	1	1	-	-	-	-	-	-	-	-
30	-	-	-	-	-	-	-	-	1	-	-	-	1	-	-	-
31	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	1
32	-	-	-	-	-	-	-	-	-	1	-	-	-	1	-	-
33	-	-	-	-	-	-	-	-	-	-	1	1	-	-	1	1

Tabla 14: Matriz del diagrama de parejas del ACL(2,1) regla 22.

Comparando esta matriz con la matriz de evolución del mismo ACL se observa que la primera es una extensión de la segunda ya que donde aparece un cero en la matriz de evolución se presenta la matriz de conectividad de ceros en la de parejas, es decir éste es el resultado del producto tensorial (elemento x matriz) de la matriz de evolución con las matrices de conectividad por estados del diagrama de de Bruijn.

## 3 Reversibilidad y su manifestación en los Autómatas Celulares Lineales.

Una vez que sabemos qué es un Autómata Celular, cómo evoluciona y las herramientas que existen para su estudio, podemos adentrarnos al análisis de los Autómatas Celulares Lineales Reversibles (ACLR); en este capítulo se expondrán los trabajos que se han desarrollado en el tema de los ACLR, el concepto de reversibilidad, manifestaciones y aplicaciones de ésta en el mundo real, las características que se pueden observar en un ACLR y en los diferentes diagramas que describen su conducta así como los métodos que existen para encontrar de un ACLR dado, el ACL con el comportamiento inverso.

### 3.1 Antecedentes Históricos

Inicialmente los AC se utilizaron como modelos para estudiar el comportamiento de sistemas biológicos, reacciones químicas, procesamiento de imágenes; es decir los investigadores se interesaban más en explorar las consecuencias de un comportamiento más que analizar sus orígenes, como los AC primero se desarrollaron por biólogos y gente de computación, recibieron poca atención de los matemáticos por lo que la investigación de propiedades tales como la reversibilidad fue lenta.

#### 3.1.1 Trabajo realizado por Moore.

A mediados de los 50's, surgen preguntas de tipo filosófico sobre configuraciones que no pudieran ser generadas por un AC cualquiera, este tipo de cuestiones fueron las que originaron el trabajo realizado por Edward F. Moore en 1962 y su concepto de "*Jardín del Edén*" (que se explicará con detalle más adelante), este trabajo es reimpreso por Burks [2].

#### 3.1.2 Trabajo realizado por Hedlund.

Sin duda alguna, un trabajo culminante en este campo es llevado a cabo por Gustav A. Hedlund aunque curiosamente para fines completamente desligados del estudio de ACL.

Hedlund trabajaba para la Agencia Nacional de Seguridad de Estados Unidos en los 60's, su estudio se realiza en el área de la dinámica simbólica sobre mapeos continuos de secuencias de símbolos con propósitos de encriptación de datos, cabe señalar que los mapeos de este tipo son un ACL; su trabajo a lo largo de 10 años sobre este tema es publicado en 1969 [9] e irónicamente los resultados que con mucho detalle en éste se exponen siguen siendo desconocidos y poco entendidos para muchos estudiosos sobre ACL, a pesar de su íntima relación con éstos.

Un ejemplo de lo anteriormente dicho es el trabajo de Amoroso y Patt en 1972 [1] buscando ACLR(2,t) ( $t = 1.5$ ) por "fuerza bruta", es decir revisando todas las posibles reglas de evolución de dicho autómata una por una y encontrando cuales eran reversibles sin saber que un estudio más teórico y profundo había sido realizado por Hedlund en 1963; en 1978 Nasu [16] hace un cuidadoso análisis del trabajo de Hedlund y relaciona éste con los ACLR apoyado en la teoría de gráficas.

### 3.1.3 Trabajo realizado por Fredkin.

Otro desarrollo importante es el realizado por Edward Fredkin [5] sobre métodos de construcción de un ACR y aplicaciones de éstos a problemas físicos, Fredkin se interesa en las leyes físicas que se presentan en los AC y que modelos de éstos preservan la información del sistema, llegando a técnicas para manejar un comportamiento invertible con operaciones booleanas.

## 3.2 Reversibilidad

### 3.2.1 Concepto de Reversibilidad

Llamease reversible a un proceso en que el sistema puede volver a pasar por los estados o condiciones anteriores [7], en el ámbito de este trabajo un ACLR es aquel cuya evolución y el pasado entero de cualquier configuración puede ser descifrado.

La pregunta a contestar en este tópico es qué relación existe entre el mapeo local de un AC que asigna sucesores a los vecindarios y el mapeo global que produce la sucesión de configuraciones de una generación a la próxima, sabiendo que aplicando el mapeo local una configuración  $q$  evoluciona a una  $q'$  y en un ACLR el mapeo global es invertible, es decir si cada configuración que por definición tiene exactamente un sucesor debe tener entonces exactamente uno y solo un antecesor.

Para que esto sea posible, un ACLR debe cumplir con la propiedad de conservar la información del sistema, cada estado dado del mismo codifica toda la información necesaria para identificar la trayectoria particular de donde viene y ninguna parte de esta información se pierde en el curso de la evolución, lo que no sucede en un ACL irreversible que es disipativo, ya que pierde o disipa la información del sistema.

### 3.2.2 Ejemplos de Sistemas Reversibles.

Podemos encontrar muchos ejemplos de manifestaciones o aplicaciones de sistemas con un comportamiento reversible, por ejemplo el cambio de estado de una cantidad de agua de sólido a líquido, la dilatación o contracción del volumen de un gas sometido a cambios de temperatura; en el terreno de las aplicaciones tenemos la codificación de la información ya sea por motivos de seguridad (encriptación), almacenamiento en dispositivos como discos y cintas magnéticas donde se tiene que verificar la integridad de los datos ahí guardados así como en el transporte de la información en redes de comunicación donde el equipo receptor debe decodificar los datos que ha recibido.

Es por estas razones que los ACLR han adquirido un creciente interés y desarrollo ya sea por su capacidad para modelar fenómenos reversibles o para su utilización en el manejo de información con seguridad.



### 3.3 Características de los Autómatas Celulares Lineales Reversibles.

#### 3.3.1 Concepto de Ancestro y Jardín del Edén.

Antes de presentar las propiedades que debe cumplir un ACL para ser reversible, es importante conocer dos conceptos fundamentales en este campo; en un ACL por definición cada configuración evoluciona en un único sucesor, así la configuración  $x_t$  que da origen a la  $x_{t+1}$  se dice que es el *ancestro* de la misma.

Por otro lado, el trabajo realizado por Moore [2] se enfoca a estudiar que sucede si existen configuraciones que tuvieran más de un ancestro posible concluyendo que si en un ACL una configuración tiene varios ancestros, existen otras que no tienen ninguno, es decir configuraciones que no pueden ser generadas en el transcurso de la evolución del autómata y solo aparecen al principio de ésta, al conjunto de tales configuraciones se le denomina el *Jardín del Edén* de un ACL dado.

Analizemos el caso de un ACL(4,h).

0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
2	3	2	1	0	1	1	2	0	2	3	1	0	3	2	3

Tabla 15: Regla de evolución EC78946E de un ACL(4,h).

Haciendo una observación detallada de la regla de evolución, se nota que este autómata no podrá generar una cadena de dos o más 0's seguidos en su evolución, ya que las vecindades que generan este estado (10, 20, 30) no pueden traslaparse para formar una cadena continua de elementos que evolucionen todos en 0's.

Si tomamos diferentes configuraciones de dos, tres y cuatro células y codificamos cada una de éstas utilizando una base 4, vemos como evoluciona el ACL(4,h) desde una configuración inicial formada unicamente por el estado 0.

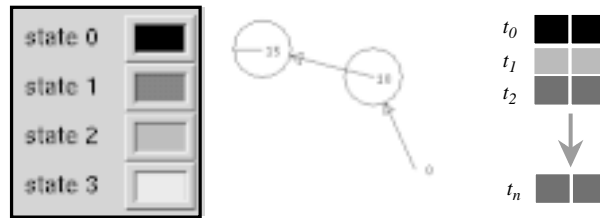


Figura 19: Evolución de una configuración de 2 células de un ACL(4,h) regla EC78946E.

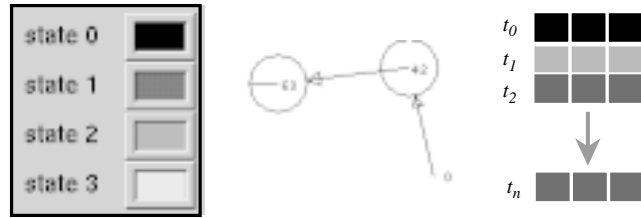


Figura 20: Evolución de una configuración de 3 células de un ACL(4,h) regla EC78946E.

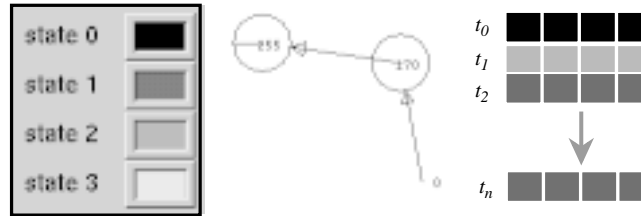


Figura 21: Evolución de una configuración de 4 células de un ACL(4,h) regla EC78946E.

Estas cadenas formadas por 0's no pueden aparecer como producto de la evolución de algún ancestro, así que éstas forman parte del Jardín del Edén de este ACL(4,h).

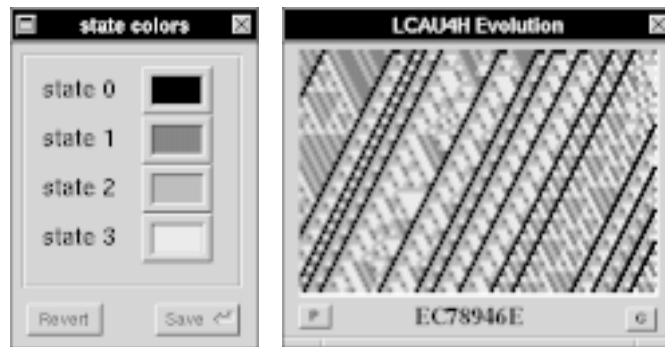


Figura 22: Evolución de un ACL(4,h) regla EC78946E, donde cadenas de dos o más 0's son parte del *Jardín del Edén* de tal ACL.

Con este ejemplo se muestra que una de las cualidades que debe cumplir un ACL para ser reversible es que cualquier configuración del mismo debe tener uno y solo un ancestro, además que en tal autómata no debe existir el Jardín del Edén.

### 3.3.2 Sobreyectividad e Inyectividad en un Autómata Celular Lineal Reversible

La parte central del estudio de un ACLR es su regla de evolución, ya que ésta nos dice como se comporta el sistema a través del tiempo, lo interesante es que esta

regla de evolución es de influencia local, o sea solo afecta las vecindades que forman parte de la configuración, sin embargo tal regla induce un mapeo global el cual es reversible.

Desde esta perspectiva podemos tomar a la evolución como una función que esencialmente transforma elementos de un conjunto a otro, donde tales elementos son las configuraciones globales que puede tener el ACL y la función el mapeo global producto de la regla de evolución.

Las funciones se pueden clasificar por el mapeo que producen obteniendo las siguientes definiciones [19]:

Una función  $f : X \rightarrow Y$  se dice

- **Inyectiva** o mapeo **uno a uno** si a cada miembro de  $Y$  le corresponde a lo más un miembro de  $X$ .
- **Sobreyectiva** o mapeo **sobre** si para cada  $y \in Y$  le corresponde al menos una  $x \in X$ .
- **Biyectiva** o mapeo que es **uno a uno** y **sobre** a la vez.

En [19] se describe claramente estos conceptos en los siguientes diagramas:

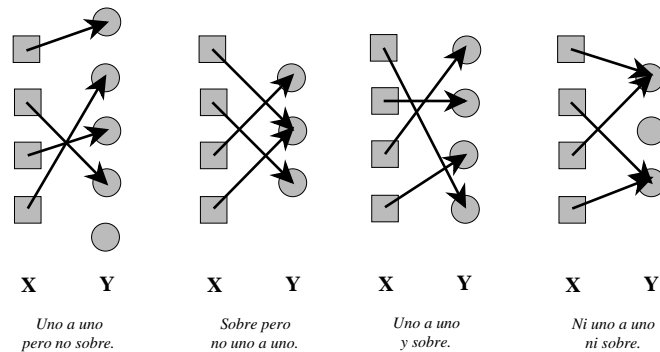


Figura 23: Diferentes tipos de mapeo de una función.

Si un mapeo  $f : X \rightarrow Y$  es biyectivo, entonces  $f^{-1} : Y \rightarrow X$  también define una función biyectiva; por lo tanto, caracterizar a los ACLR es encontrar que propiedades debe tener el mapeo inducido por la regla de evolución para que éste sea biyectivo, sabiendo además que existe una regla inversa a la original.

Usando un poco de nomenclatura de estructuras algebraicas podemos decir que el mapeo global define un *homomorfismo* de  $X$  a  $X$ , como el dominio como el contradominio de la función es el mismo, ésta es un *endomorfismo* y ya que tal función debe ser biyectiva (uno a uno y sobre) para que el ACL sea reversible, esta función debe definir un *automorfismo*.

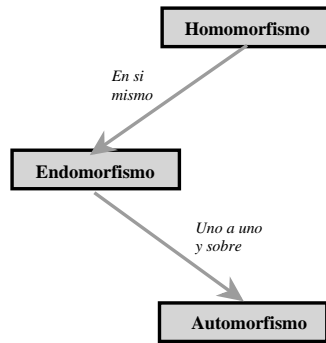


Figura 24: La regla de evolución de un ACLR define un automorfismo.

Tomando como base el trabajo de [9] y [16] se expondrán tres definiciones que exponen estos autores para caracterizar la regla de evolución de un ACLR, los cuales son *Multiplicidad Uniforme*, *Indices de Welch* y *Combinabilidad*.

- Multiplicidad Uniforme

La sección 5 en [9] demuestra que el mapeo global inducido por un ACL es sobreyectivo si y solo si cada mapeo menor contenido en éste es sobreyectivo y además, el número de ancestros de cada uno de éstos mapeos es igual al número de nodos en el diagrama de de Bruijn (multiplicidad), siendo este número igual para cada posible configuración (uniformidad), ilustremos este concepto con un ACLR(5,h).

	0	1	2	3	4
0	3	3	4	4	4
1	4	4	3	3	3
2	0	2	1	1	2
3	1	0	2	2	1
4	2	1	0	0	0

Tabla 16: Regla de evolución 0077A76AIJOOI de un ACLR(5,h).

Las configuraciones que pueden formar la cadena 0 son:

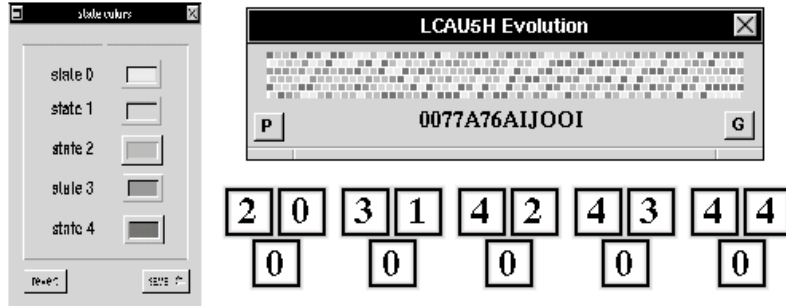


Figura 25: Configuraciones de tamaño 2 que evolucionan en 0.

Definamos ahora las configuraciones que evolucionan en la cadena 02.

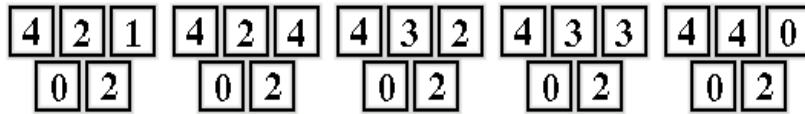


Figura 26: Configuraciones de tamaño 3 que evolucionan en 02.

Podemos hacer cada vez mas grandes estas cadenas y seguiremos encontrando que el número de ancestros de cada una de éstas es el mismo, definamos como  $\mathcal{A}=\{421,424,432,433,440\}$  al conjunto de configuraciones que son ancestros de 02 con  $a = \text{card}\mathcal{A}$ , definamos como  $\mathcal{K}=\{0,1,2,3,4\}$  al conjunto de estados del ACL con  $k = \text{card}\mathcal{K}$  y  $\text{Ancestros}(\mathcal{B})$  al conjunto de ancestros de una cadena  $\mathcal{B}$ , entonces tenemos que  $\text{Ancestros}(02\mathcal{K}) = \mathcal{AK}$ .

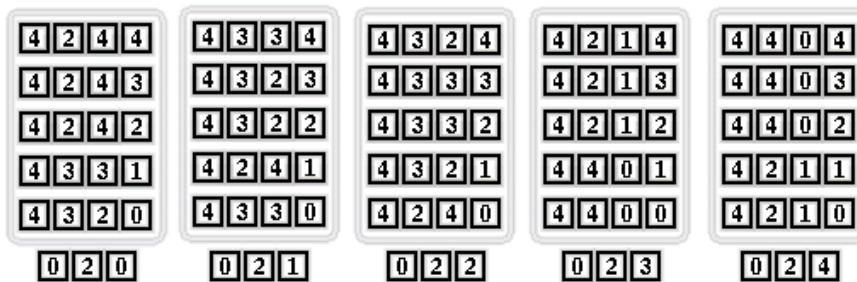


Figura 27: Conjunto de ancestros de las configuraciones 02K.

Tenemos que  $\text{card}\mathcal{AK} = ak$ ; supongamos que  $\text{card}\text{Ancestros}(02e) \geq a$  para todo  $e \in \mathcal{K}$ , si existe  $\text{card}\text{Ancestros}(02e) > a$  para cualquier  $e \in \mathcal{K}$ , tendríamos que  $\text{card}\text{Ancestros}(\mathcal{AK}) > ak$  lo que no es posible, por lo tanto se concluye que  $\text{card}\text{Ancestros}(02e) = a$  para cada  $e \in \mathcal{K}$ , este resultado se extiende para cualquier cadena y las extensiones que se hagan a la misma.

Encontremos ahora cual es el valor que toma  $a$  para cada caso; tomemos  $\text{Ancestros}(02) = \mathcal{A} = \{421,424,432,433,440\}$  con  $a = \text{card}\mathcal{A}$ ,  $\mathcal{K}=\{0,1,2,3,4\}$

al conjunto de configuraciones de tamaño  $2r$  con  $k = \text{card}\mathcal{K}$ , si tomamos la cadena  $02\mathcal{K}02$  tenemos que  $\text{Ancestros}(02\mathcal{K}02) = \mathcal{A}\mathcal{A}$  ya que al hacer cada posible concatenación de 2 cadenas del conjunto  $\mathcal{A}$  obtenemos toda posible instancia de la cadena  $02\mathcal{K}02$ , entonces  $\text{cardAncestros}(02\mathcal{K}02) = \text{card}\mathcal{A}\mathcal{A} = a^2$ , tomemos ahora un elemento del conjunto  $\mathcal{K}$ , (el 1 en este caso aunque puede ser cualquier elemento) y formemos la cadena  $02102$  con todos sus posibles ancestros.

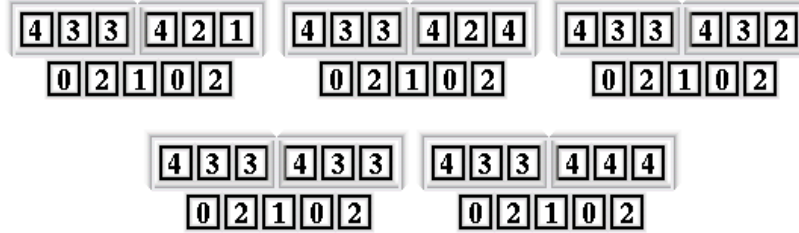


Figura 28: Elementos del conjunto  $\mathcal{A}$  de cadenas que evolucionan en  $02$  que forman los ancestros de la configuración  $02102$ .

Tenemos que  $\text{cardAncestros}(02102) = a$  lo que ocurre para cada posible elemento de  $\mathcal{K}$ , ya que  $\text{card}\mathcal{K} = k$ , tenemos que  $\text{cardAncestros}(02\mathcal{K}02) = ka = aa$ , de este modo  $a = k$ ; sabemos que  $k$  está dada por el número de configuraciones diferentes de tamaño  $2r$  que es la misma manera en que se definen los nodos en el diagrama de de Bruijn, por lo tanto se concluye que en un ACL con un mapeo global sobreyectivo cada cadena de estados debe tener tantos ancestros como el número de configuraciones diferentes que se puedan hacer de tamaño  $2r$  o como nodos existan en su diagrama de de Bruijn correspondiente.

- Índices de Welch

La definición de este concepto se debe a L. R. Welch colaborador de Hedlund, en la sección 14 en [9] se definen estos índices como sigue:

Sea  $A$  cualquier bloque de estados y  $\mathcal{B}_i$  el  $i$ -ésimo conjunto de extensiones por la derecha de  $A$  tal que  $AB$  evolucionen en la misma cadena, así definamos  $r_i = \text{card}\mathcal{B}_i$ , entonces se tomará  $R = \max\{r_i\}, 1 \leq i \leq n, n \geq 1$  es decir, la cardinalidad del conjunto  $\mathcal{B}_i$  con mayor número de elementos; se define de manera análoga  $L$  para  $BA$ , las extensiones compatibles por la izquierda de un bloque de estados.

Ejemplifiquemos este concepto con un ACL(5,h) reversible.

	0	1	2	3	4
0	3	3	4	4	4
1	4	4	3	3	3
2	2	2	1	1	2
3	0	0	2	2	1
4	1	1	0	0	0

Tabla 17: Regla de evolución 0067A26CIJOOI de un ACLR(5,h).

Tomemos el estado 0 y busquemos las extensiones a la derecha compatibles con éste tal que la evolución sea 3 ó 4:

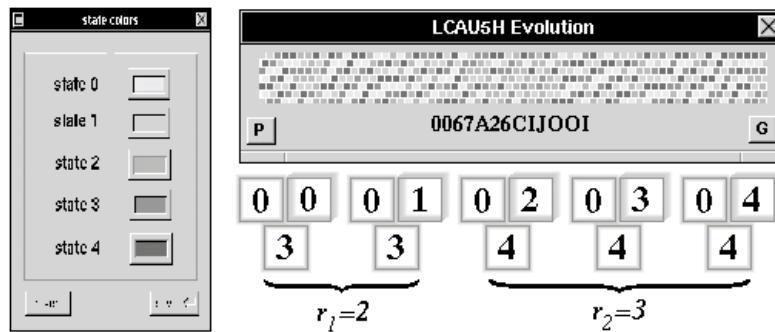


Figura 29: Regla de evolución 0067A26CIJOOI de un ACLR(5,h), y las extensiones derechas compatibles con 0 que evolucionan en 3 y 4.

Para este caso  $R = 3$  ya  $r_2$  es la cardinalidad del mayor conjunto compatible por la derecha  $\mathcal{B}$  de 0, ahora busquemos a  $\mathcal{B}$  tal que la evolución se 34 y 40.

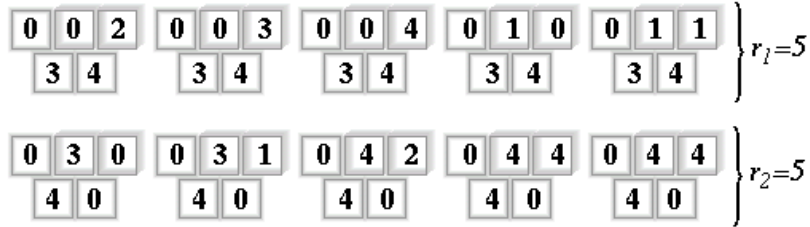


Figura 30: Extensiones derechas compatibles con 0 que evolucionan en 34 y 40.

Ahora  $R = r_1 = r_2 = 5$ , si ahora tomamos  $\mathcal{B}$  compatible por la derecha tal que la evolución se 342 tenemos:



Figura 31: Extensiones derechas compatibles con 0 que evolucionan en 342.

Podemos continuar de forma indefinida incrementado cada vez más las extensiones compatibles por la derecha del estado 0 y encontraremos que  $R$  se mantendrá con valor 5 para este ACL, lo mismo ocurre para todas las extensiones derechas de los demás estados.

Ahora busquemos  $\mathcal{B}$  compatible por la izquierda del mismo estado 0 tal que la evolución sea 3 y 4 y observemos lo que se obtiene:

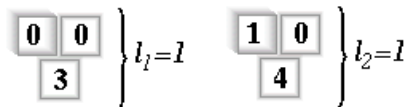


Figura 32: Extensiones izquierdas compatibles con 0 que evolucionan en 3 y 4.



En este caso  $L = l_1 = l_2 = 1$ , si extendemos la evolución a 23 ó 14 resulta:

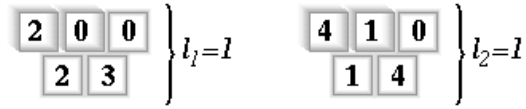


Figura 33: Extensiones izquierdas compatibles con 0 que evolucionan en 23 y 14.

Se observa que  $L$  sigue siendo 1. Definamos ahora  $A$  como un bloque de estados y  $\mathcal{A}$  como el conjunto de bloques de estados  $A$  en donde las extensiones  $L$  y  $R$  de cada  $A \in \mathcal{A}$  evolucionen en la misma cadena bajo la regla de evolución y denotemos a  $card\mathcal{A} = M$ , tomemos un ejemplo de todas las posibles cadenas evolucionen en 231.

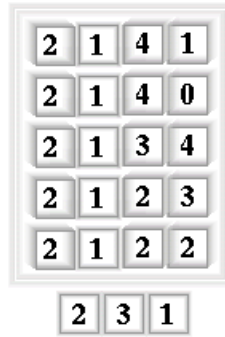


Figura 34: Ancestros de la cadena 231.

En este caso  $M = 1$  ya que hay un solo elemento en el cual sus extensiones por derecha y por izquierda interactúan con el para evolucionar en 231, una propiedad fundamental que Hedlund y Welch postulan es que si un  $ACL(k,r)$  define un mapeo global sobreyectivo, entonces la multiplicación de  $LMR = k^{2r}$  o igual al número de nodos en el diagrama de de Buijn, veamos como funciona esto para la cadena de estados 130 y sus posibles ancestros.

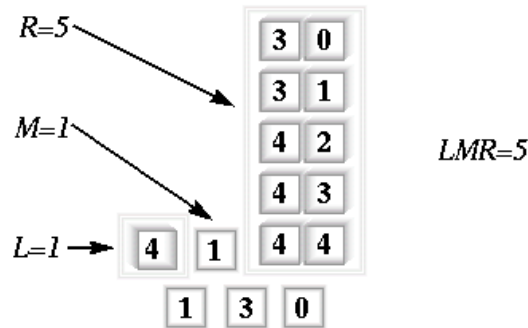


Figura 35: Ancestros de la cadena 130.

Se observa en este caso que  $M = 1$  para esta configuración, ya que el estado 1 y sus conjunto compatibles por la izquierda y derecha son los únicos que evolucionan en la cadena 130, con  $L = 1$  y  $R = 5$ , ilustremos ésto con otros ACLR.

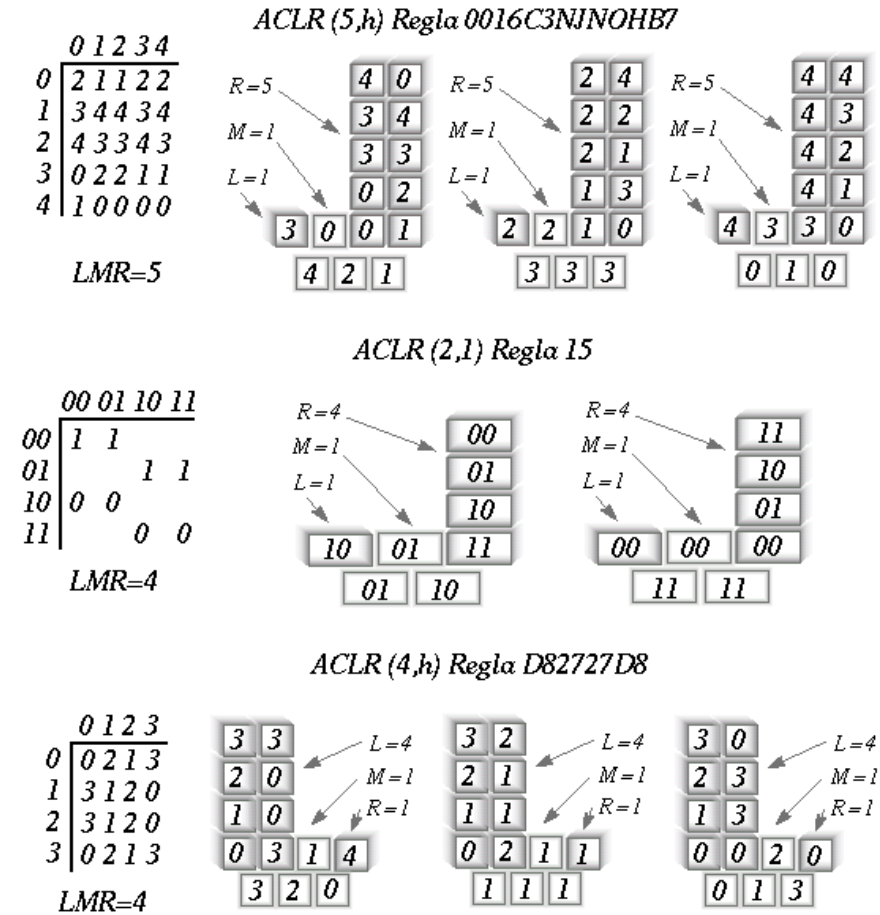
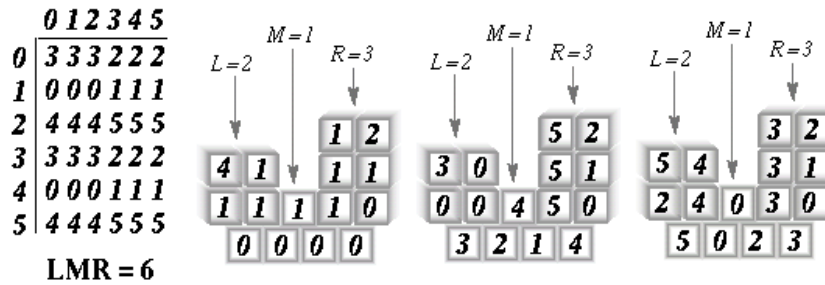


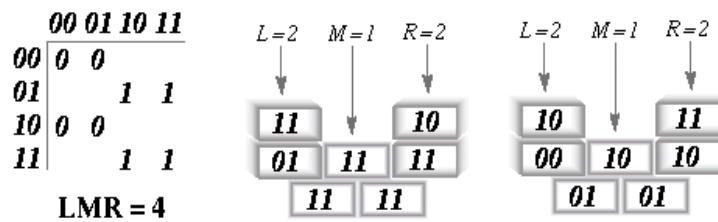
Figura 36: Ejemplos de ACLR y sus valores de  $LMR$ .

Por supuesto, no siempre  $L$  ó  $R$  deben ser igual a 1, ya que hay casos en el que el valor  $k^{2^r}$  puede ser formado por otras combinaciones que no contemplen 1; el único caso en el que  $L$  ó  $R$  deben ser 1 es cuando el valor de  $k^{2^r}$  es primo, a continuación se presentan algunos ejemplos de ACLR donde este valor no es primo.

**ACLR (6,h) Regla ZYS760EFLZYS760EFL**



**ACLR (2,1) Regla 204**



**ACLR (3,1) Regla 0QD0QD0QD**

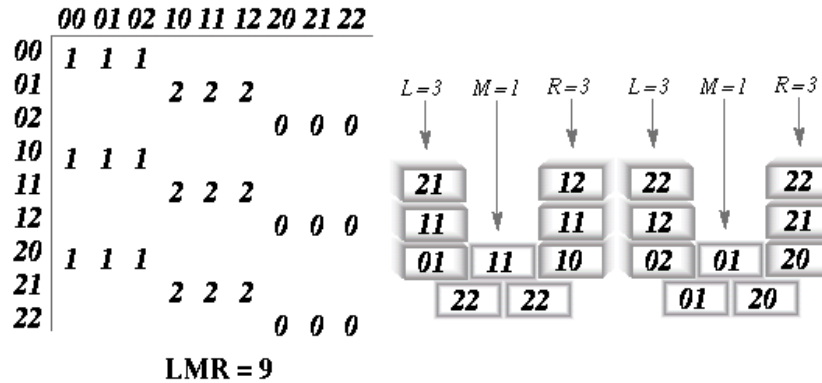


Figura 37: Ejemplos de ACLR con valores de  $LMR$  no primos.

- Combinabilidad

En la sección 16 de [9] y y en el trabajo de [16] se define este concepto el cual nos dice que un mapeo local es *debilmente combinable* si para toda cadena dada de estados de longitud  $l = 2r$  el conjunto  $\mathcal{B}$  de sus extensiones compatibles por la derecha debe tener una cardinalidad igual a  $R$  o a 0 y el conjunto  $\mathcal{C}$  de sus extensiones compatibles por la izquierda debe tener una cardinalidad igual a  $L$  o a 0, ilustremos un ejemplo de esta definición mostrando un ACL(5,h) no reversible.

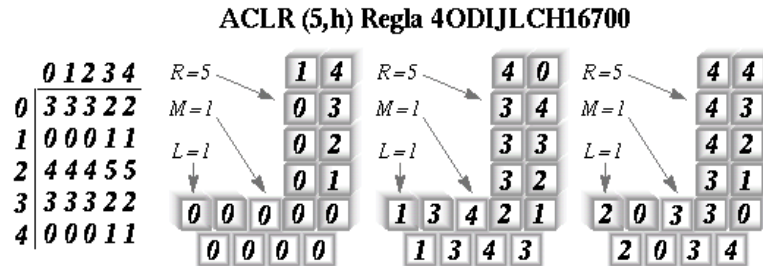


Figura 38: ACL no reversible (5,h) con cadenas cuyas valores de  $LMR = k^{2r}$ .

Aquí se observa que este ACL tiene configuraciones con valores de  $L = 1$ ,  $M = 1$  y  $R = 5$ , sin embargo tomemos la cadena formada alternando los estados 0, 2 y observemos sus posibles ancestros.

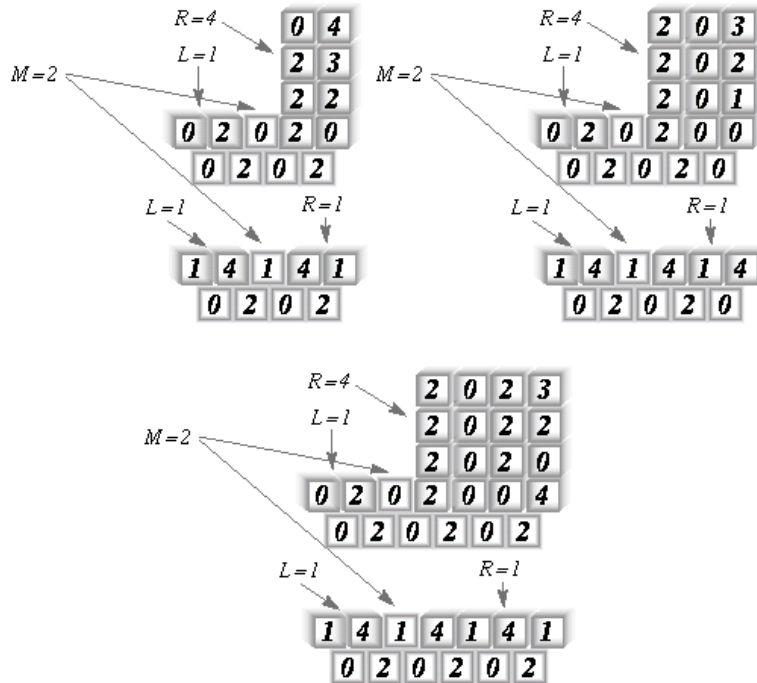


Figura 39: Ancestros de la cadena formada alternando los estados 0, 2.

En este caso una cadena de este tipo puede ser generada por las extensiones compatibles de dos estados distintos, un estado con  $R = 4$  y el otro con  $R = 1$ , es decir, el valor de  $M = 2$ , por lo que este ACL tiene múltiples ancestros para esta cadena en particular, lo que impide que sea reversible; lo mismo sucede con la cadena formada alternando los estados 1, 4.

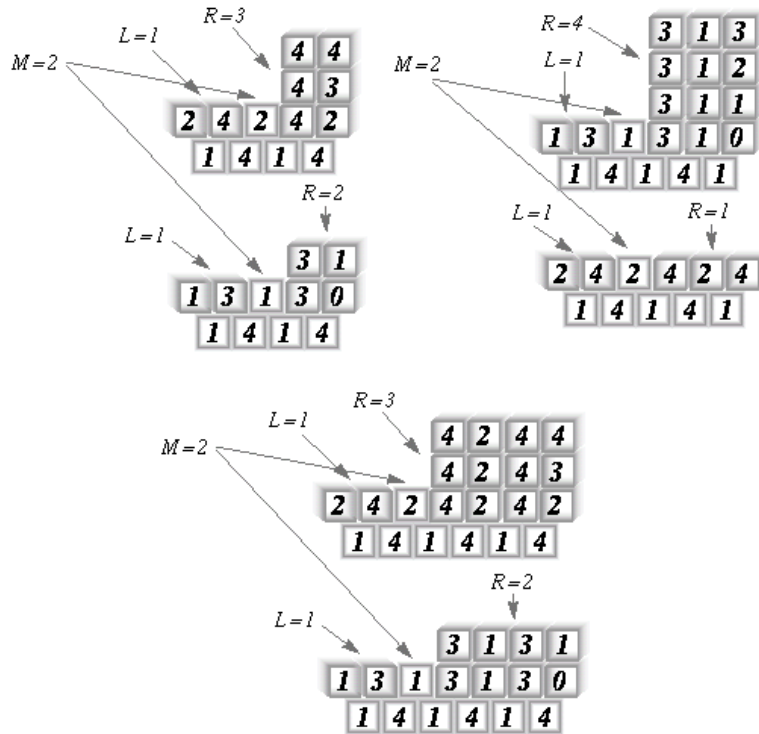


Figura 40: Ancestros de la cadena formada alternando los estados 1, 4.

En un ACLR el valor de  $M$  debe ser siempre igual a 1 y los ancestros de las cadenas de longitud  $l \geq 2r$  deben tener índices  $L$  y  $R$  con  $LR = k^{2r}$ , es decir cada posible ancestro de debe estar formado por las extensiones compatibles de una cadena de estados de longitud  $l = 2r$  en donde se cumpla que para el conjunto  $\mathcal{B}$  de extensiones compatibles por la derecha del mismo,  $\text{card}\mathcal{B} = \mathcal{R}$  y para el conjunto  $\mathcal{C}$  de extensiones compatibles por la izquierda del estado,  $\text{card}\mathcal{C} = \mathcal{L}$ , tomemos como ejemplo un ACLR(5,h).

**ACLR (5,h) Regla 001662DHOCIJO**

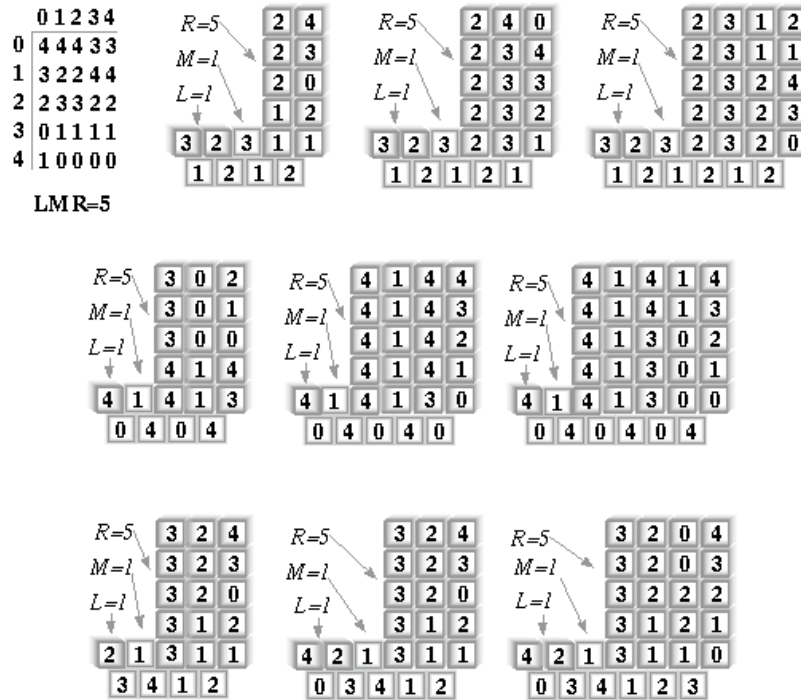


Figura 41: ACLR(5,h) donde el valor de  $M$  siempre es igual a 1.

La propiedad de que la regla de evolución de un ACL sea debilmente combinable obliga a que una cadena de estados sea producto de la evolución de una única cadena de estados de longitud  $l = 2r$  con sus extensiones compatibles derechas e izquierdas, evitando así que otra secuencia de estados evolucione en la misma cadena y ésta tenga múltiples ancestros.

### 3.4 Estudio de los Autómatas Celulares Lineales Reversibles mediante las herramientas utilizadas para su análisis

Una vez que hemos definido las características que debe cumplir la regla de evolución para inducir un mapeo global reversible, podemos ahora señalar que cualidades deben presentar los diagramas que describen el comportamiento de tal ACL para que éste sea invertible.

#### 3.4.1 Características en el diagrama de de Bruijn

El diagrama de de Bruijn es la herramienta básica que nos dice como evolucionan las diferentes vecindades que existen en un ACL; tomando en cuenta las características presentadas para definir un mapeo global biyectivo, podemos deducir que un ACL es reversible si en su diagrama de de Bruijn asociado cada tipo de arco aparece el mismo número de veces que el resto, además, si no existen dos o más ciclos iguales de una longitud dada formados por distintos nodos ya que esto significaría que una cadena dada tiene varias formas de producirse con diferentes estados, es decir, ancestros múltiples.

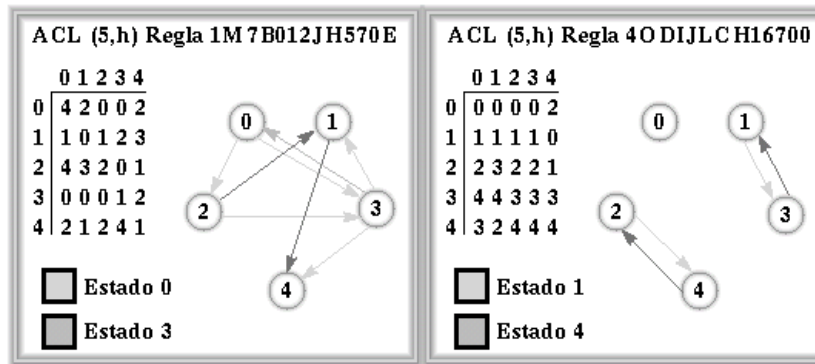


Figura 42: ACL(5,h) no reversibles y parte de sus diagramas de de Bruijn asociados.

En la gráfica superior se observa que en el caso del ACL de la izquierda la liga del estado 0 aparece más veces que la liga del estado 3 lo que se vio que era imposible para un ACLR por la definición de multiplicidad uniforme, en el caso del ACL de la derecha existen dos ciclos idénticos de los estados 1 y 4 formados por parejas distintas de nodos, en un caso los nodos 1, 3 y en el otro los nodos 2, 4, lo que define en la evolución del autómata que la cadena formada alternando los estados 1, 4 tiene múltiples ancestros.

Se definió que un ACL es reversible si para toda cadena de longitud  $l = 2r$  sus extensiones compatibles por la derecha o izquierda son 0 ó  $R$  y  $L$  respectivamente, así en su diagrama de de Bruijn debemos encontrar que empezando desde un único nodo  $x$  para formar toda ruta  $y$  de cierta longitud  $l \geq 1$  debemos tener  $R$  maneras distintas o ninguna, a su vez para formar la ruta  $y$  terminando únicamente en el nodo  $x$  debemos tener  $L$  maneras distintas o ninguna, con  $LR = k^{2r}$ .

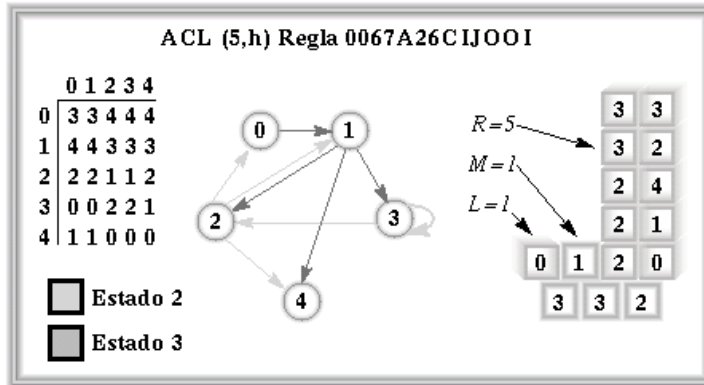


Figura 43: ACLR(5,h) regla 0067A26CIJOI y la ruta 332 en su diagrama de de Bruijn.

Como se observa en al gráfica en el diagrama de de Bruijn asociado al ACLR(5,h) regla 0067A26CIJOI existen 5 posibles formas de generar la ruta 32 empezando desde el nodo 1 ( $R = 5$ ) y solo existe una forma de generar la ruta 3 terminando en el nodo 1 ( $L = 1$ ), como no existe otra forma de generar la ruta 332 más que las descritas anteriormente, se tiene que  $M = 1$  ya que solo las extensiones compatibles por la izquierda y por la derecha del nodo 1 son las que producen dicha ruta.

En un ACLR toda ruta  $y$  debe poderse formar en su diagrama de de Bruijn asociado, es decir no debe existir el Jardín del Edén, pero cada ruta  $y$  de longitud  $l \geq 2r$  solo debe poderse formar llegando de  $L$  maneras distintas a un único nodo  $x$  y partiendo del mismo con  $R$  diferentes formas en el diagrama de de Bruijn.

Debido a que observar estas propiedades directamente en el diagrama de de Bruijn no es fácil ni cómodo, utilizamos para esto las otras dos herramientas basadas en este diagrama, el diagrama de parejas y el diagrama de subconjuntos.



### 3.4.2 Características en el diagrama de Parejas

Sabemos que los nodos en el diagrama de parejas son pares de nodos del diagrama de de Bruijn, y existe una liga del nodo  $x_i$  al nodo  $x_j$  si los elementos del nodo  $x_i$  se conectan con la misma liga con los elementos del nodo  $x_j$  en el diagrama de de Bruijn, tomando esta idea tenemos que la diagonal principal del diagrama de parejas no es más que el mismo diagrama de de Bruijn base.

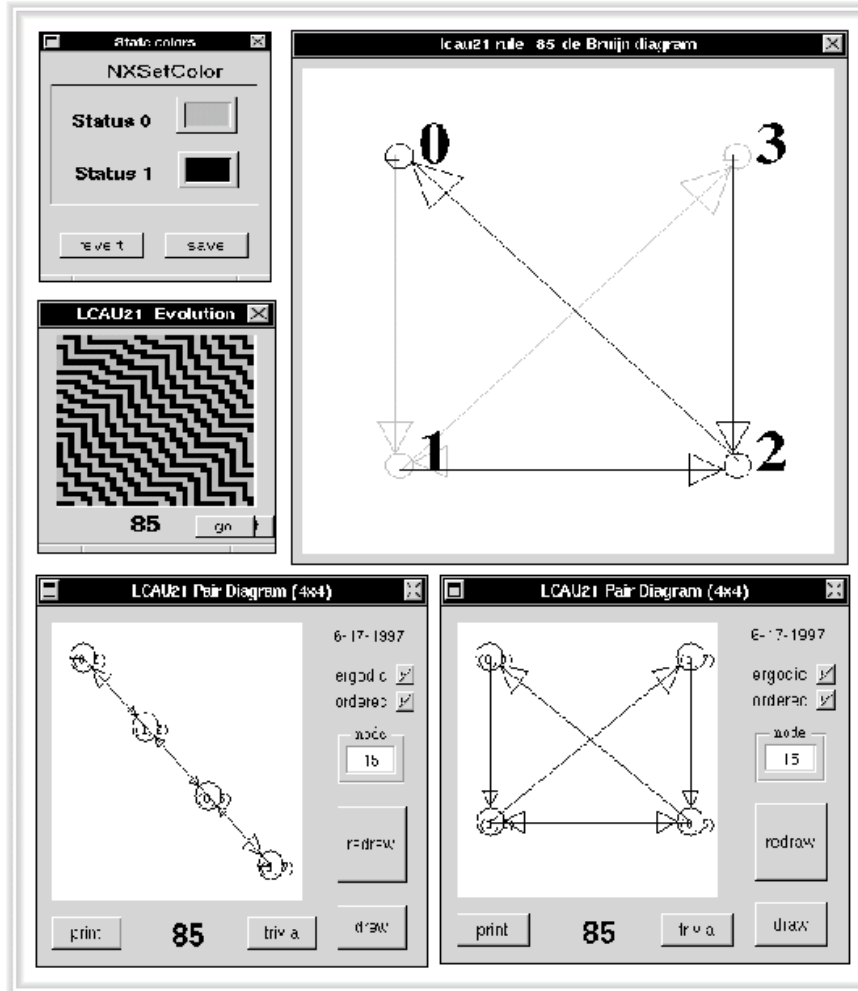


Figura 44: ACLR(2,1) regla 85 y su diagrama de parejas asociado.

En el esquema se observa un ACLR(2,1) regla 85 y su diagrama de de Bruijn así como la diagonal principal aislada de su diagrama de parejas, en la cual se volvieron a recomodar sus nodos para mostrar que tiene la misma estructura que el diagrama de de Bruijn.

Sin embargo, si existen ciclos fuera de la diagonal principal del diagrama de parejas indica que dentro del diagrama de de Bruijn una misma cadena puede generarse de varias formas distintas, o sea que tiene más de un ancestro posible lo que indica que el ACL no es reversible.

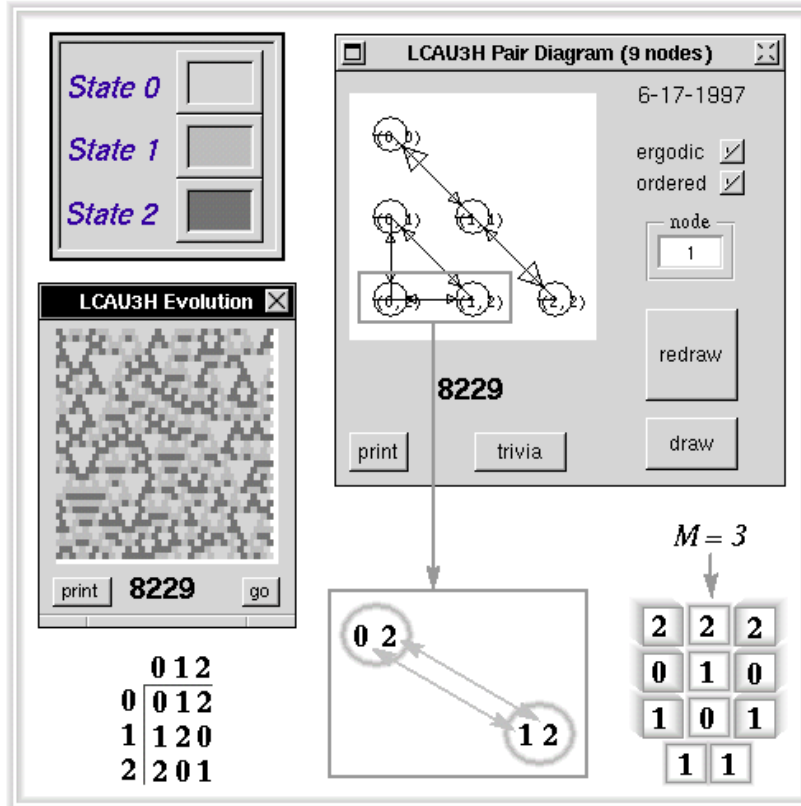


Figura 45: ACLR(3,h) regla 8229 y su diagrama de parejas asociado.

En este ejemplo la cadena de estados 11 puede ser producto de la evolución de las extensiones compatibles por ambos lados ya sea de 0, 1 o 2, es decir  $M = 3$ , ésto se observa claramente en el ciclo señalado en el diagrama de parejas que nos indica que dicha cadena tiene ancestros múltiples impidiendo que el ACL sea reversible.

Lo que nos muestra el diagrama de parejas es si existe una cadena de estados que tenga varios ancestros, es decir si la regla de evolución del ACL define un mapeo global inyectivo o no.

### 3.4.3 Diagrama de Subconjuntos

Como se vió anteriormente, se llama Jardín del Edén al conjunto de todas aquellas cadenas de estados que no puedan aparecer en la evolución del ACL más que en la configuración inicial; el diagrama de subconjuntos indica de manera muy clara

cuales son estas cadenas, ésto se observa si existe una ruta que vaya desde el subconjunto de máxima cardinalidad hasta el conjunto vacío y la secuencia de arcos de tal ruta es justamente la cadena que pertenece al Jardín del Edén de dicho ACL; esta ruta nos muestra que no importa desde que estado partamos en el diagrama de de Bruijn, llegará un momento en que no se pueda encontrar una liga para completar dicha ruta (justamente la liga que nos lleva al conjunto vacío), ejemplifiquemos esto de manera sencilla con un ACL(2,1) regla 231.

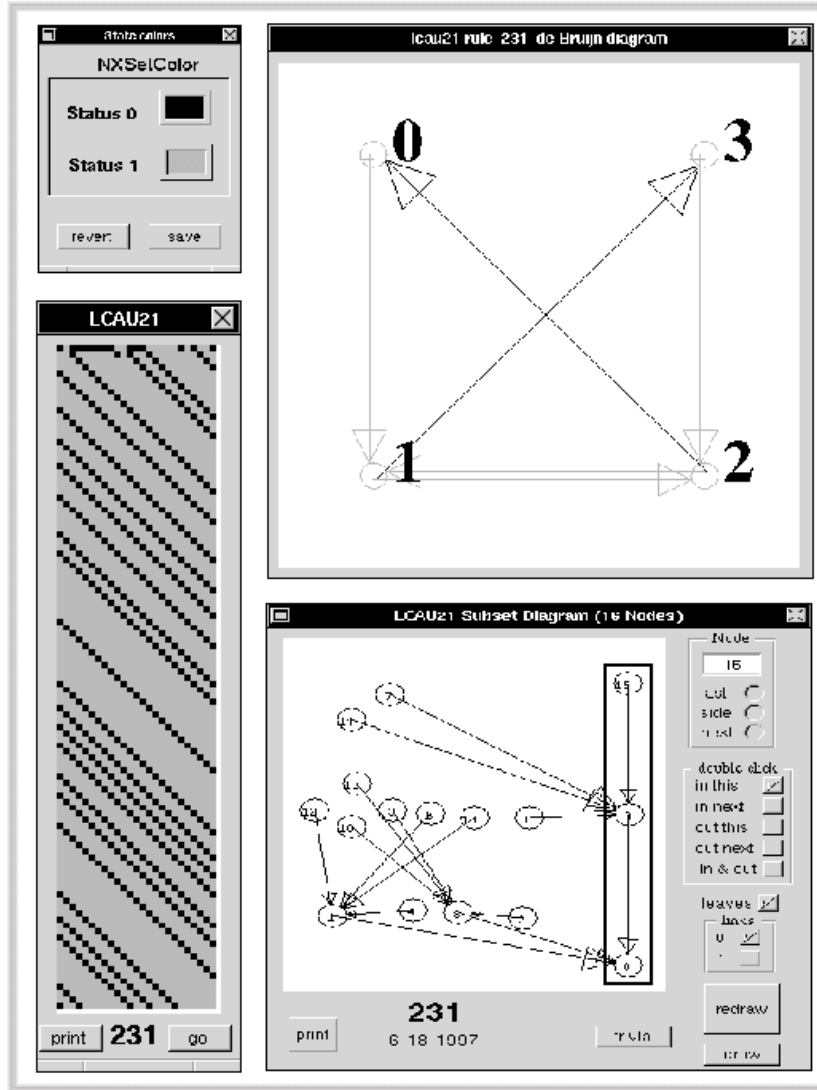


Figura 46: ACL(2,1) regla 231 y su diagrama de subconjuntos.

En este caso se ve claramente que en el diagrama de de Bruijn uno no puede formar la ruta 00 empezando desde cualquier nodo, esta es la misma ruta que

aparece en el diagrama de subconjuntos que va desde el nivel mas alto al conjunto vacio, y como se nota en la evolución del autómata, las cadenas formadas por dos o más 0's aparecen solo en la configuración inicial; el diagrama de subconjuntos nos indica que configuraciones pertenecen al Jardín del Edén, es decir, si el ACL es sobreyectivo o no, si tales rutas no existen, entonces la regla de evolución define un mapeo global sobreyectivo.

Pero aun podemos obtener más información acerca del ACL, en especial de los índices de Welch, esto es ya que un arco que parte de un nodo a otro en el diagrama de subconjuntos está conformado por el conjunto de arcos donde al menos un estado de subconjunto inicial tiene ligas con los estados de subconjunto final.

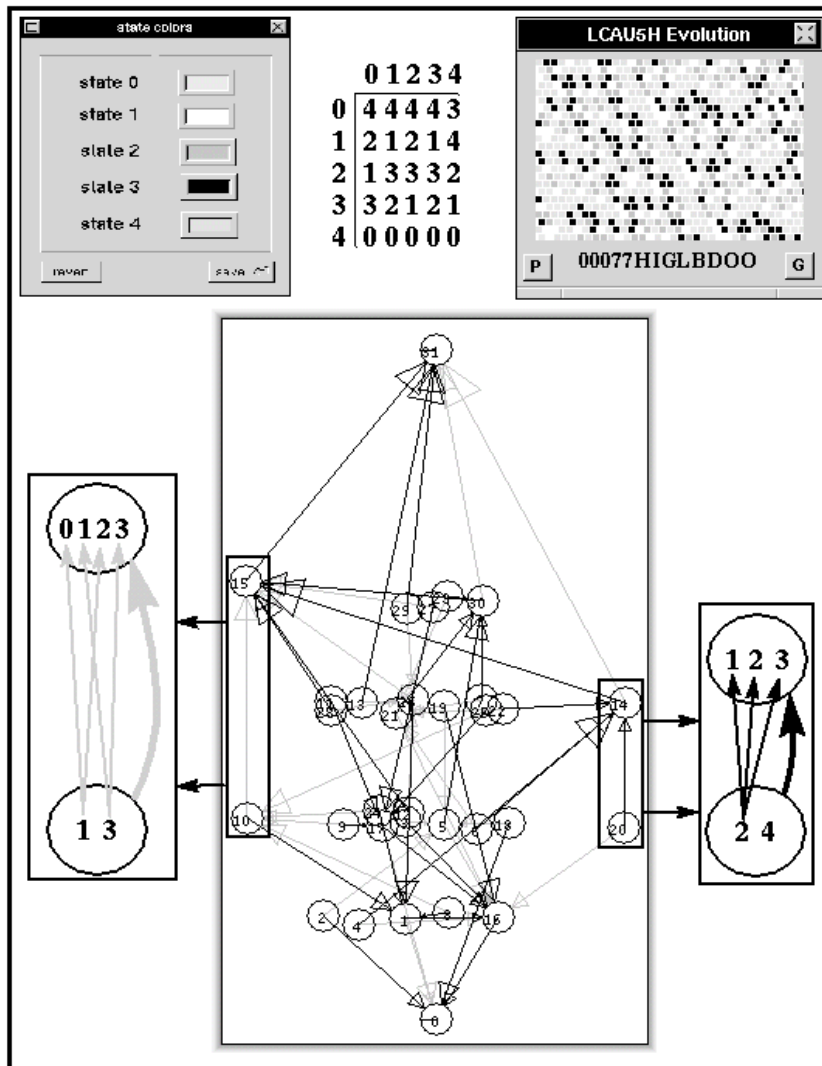


Figura 47: ACLR(5,h) regla 00077HIGLBDOO y su diagrama de subconjuntos.

En el diagrama de subconjuntos anterior se observa que el nodo 10 formado por los estados 1, 3, tiene una liga de valor 2 con el nodo 15 formado por los estados 0, 1, 2, 3, esta liga está constituida por las ligas que van del estado 1 del nodo 10 a los estados 0, 2 del nodo 15 y las ligas del estado 3 del nodo 10 a los estados 1, 3 del nodo 15; lo mismo ocurre en el caso de los nodos 20 y 14.

Si tomamos las rutas que parten desde los subconjuntos unitarios podemos conocer cual es la cardinalidad de los conjuntos compatibles por la derecha de cada estado observando a que subconjuntos llegan los arcos; si el ACL es reversible todas las rutas que empiezan desde los subconjuntos unitarios terminan en subconjuntos con cardinalidad  $R$  o en el conjunto vacío, cumpliendo así con la propiedad de combinabilidad que Nasu especifica.

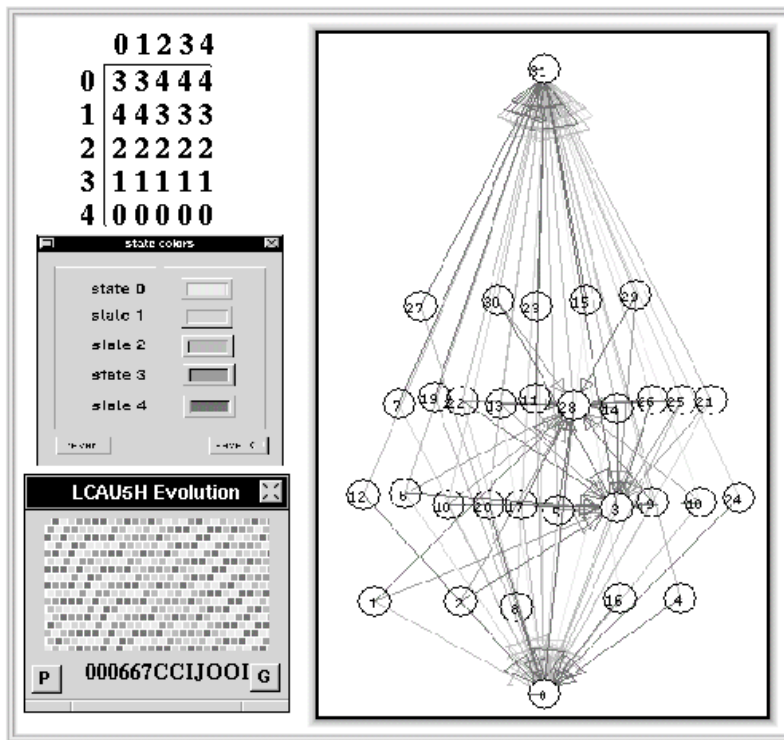


Figura 48: ACLR(5,h) regla 000667CCIJOOI y su diagrama de subconjuntos.

En la gráfica anterior de un ACLR(5,h), se observa que partiendo de las clases unitarias (1, 2, 4, 8, 16), todas las rutas "suben" hasta el nivel máximo o "bajan" al conjunto vacío, en este caso tendríamos que  $R = 5$ , podemos conocer el índice  $L$  tomando la reflexión de la regla de evolución original, esto es las vecindades de la regla de evolución con valores asimétricos se intercambian para reflejar la regla de evolución como un espejo, el efecto que tiene la reflexión de la regla original es convertir los conjuntos compatibles por la izquierda en conjunto compatibles por la derecha, si observamos ahora el diagrama de subconjunto de la reflexión de la regla estaremos conociendo su índice  $L$ .

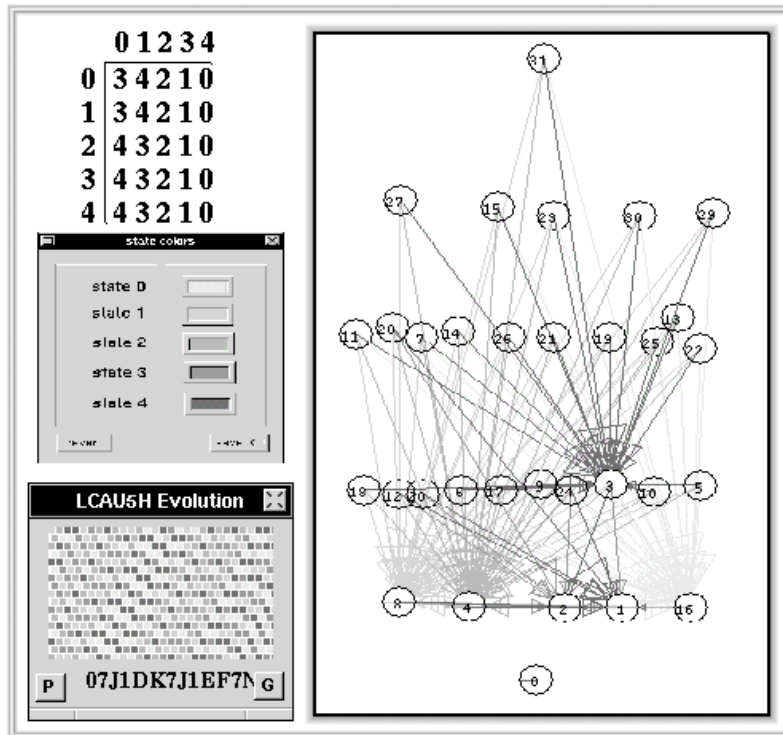


Figura 49: ACLR(5,h) reflexión de la regla 000667CCIJOOI y su diagrama de subconjuntos.

En este ejemplo se observa que el diagrama de subconjuntos carece de rutas del subconjunto total al conjunto vacío, todas las rutas tienden a subconjuntos con los valores de  $L = 1$  en la reflexión obteniendo que  $LR = k^{2r} = 5$  para el ACLR(5,h).

Para saber por medio del diagrama de subconjuntos si un ACL es reversible, por principio de cuentas no debe existir una ruta que vaya desde el subconjunto máximo al conjunto vacío, en el diagrama de la regla original toda ruta que parta desde las clases unitarias de llegar a un nivel de subconjuntos con cardinalidad  $R$  y nunca salir de este nivel sin ciclos intermedios entre el nivel unitario y el nivel  $R$ , lo mismo debe ocurrir para la reflexión de la regla de evolución ahora partiendo desde los subconjuntos unitarios y terminando en un nivel con cardinalidad  $L$  sin ciclos intermedios entre el nivel unitario y el nivel  $L$ , donde  $LR = k^{2r}$ .

### 3.4.4 Ejemplos de Autómatas Celulares Lineales no Reversibles

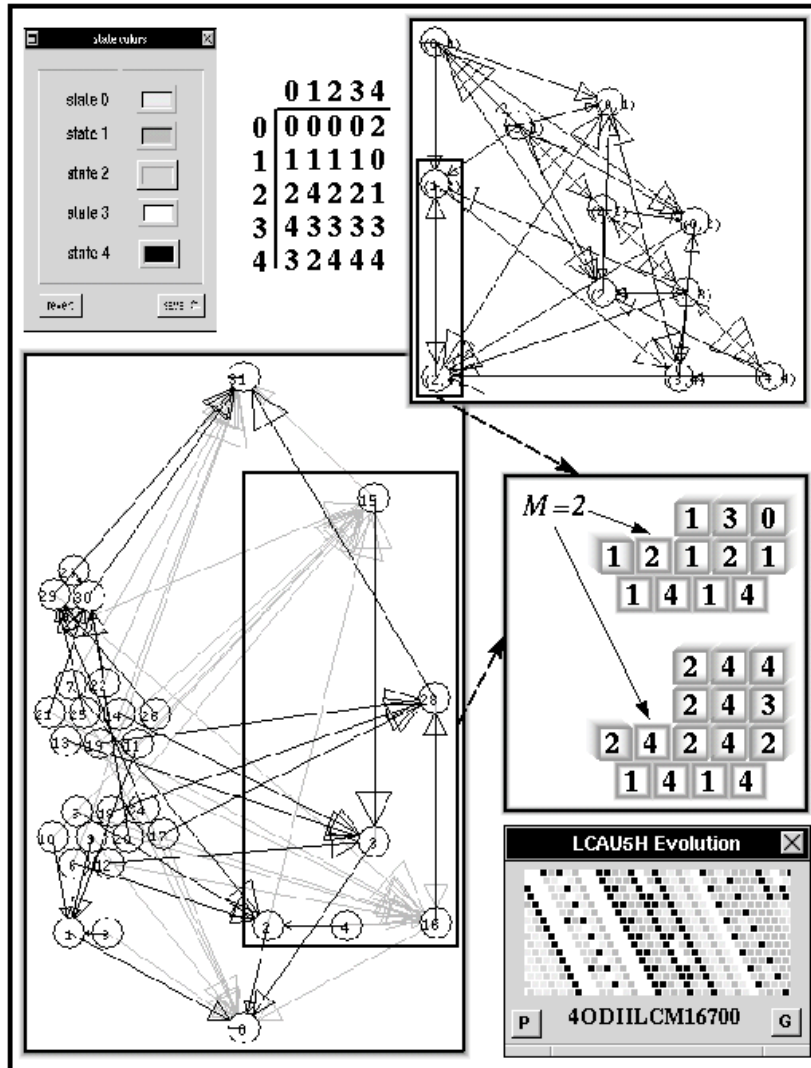


Figura 50: ACL(5,h) regla 4ODIILCH16700 donde existen ciclos en el diagrama de subconjuntos y en el de parejas.

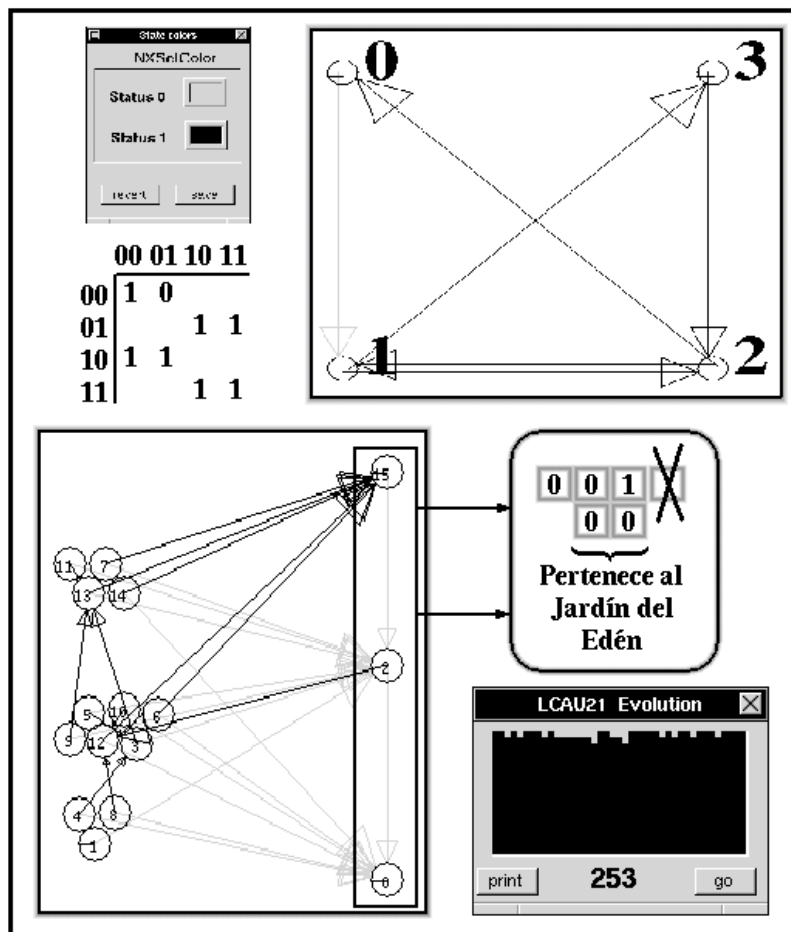


Figura 51: ACL(2,1) regla 253 en donde la cadena de estados 00 pertenece al Jardín del Edén.



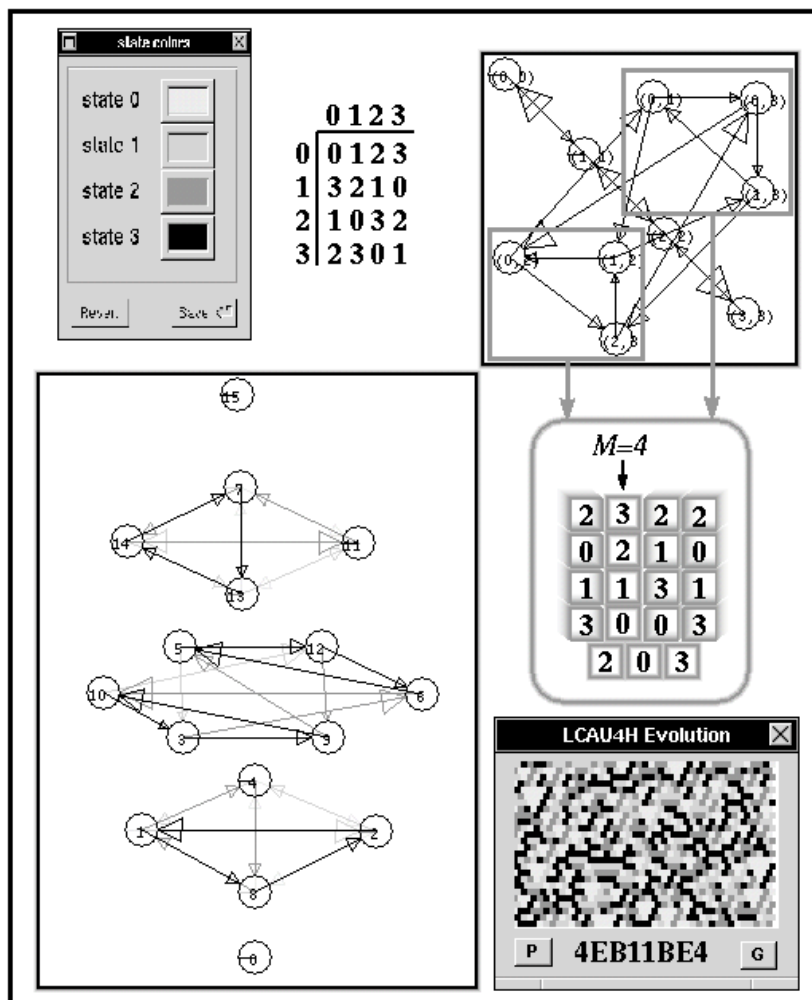


Figura 52: ACL(4,h) regla 4EB11BE4 el cual tiene un mapeo global sobreyectivo pero no reversible.

### 3.4.5 Ejemplos de Autómatas Celulares Lineales Reversibles

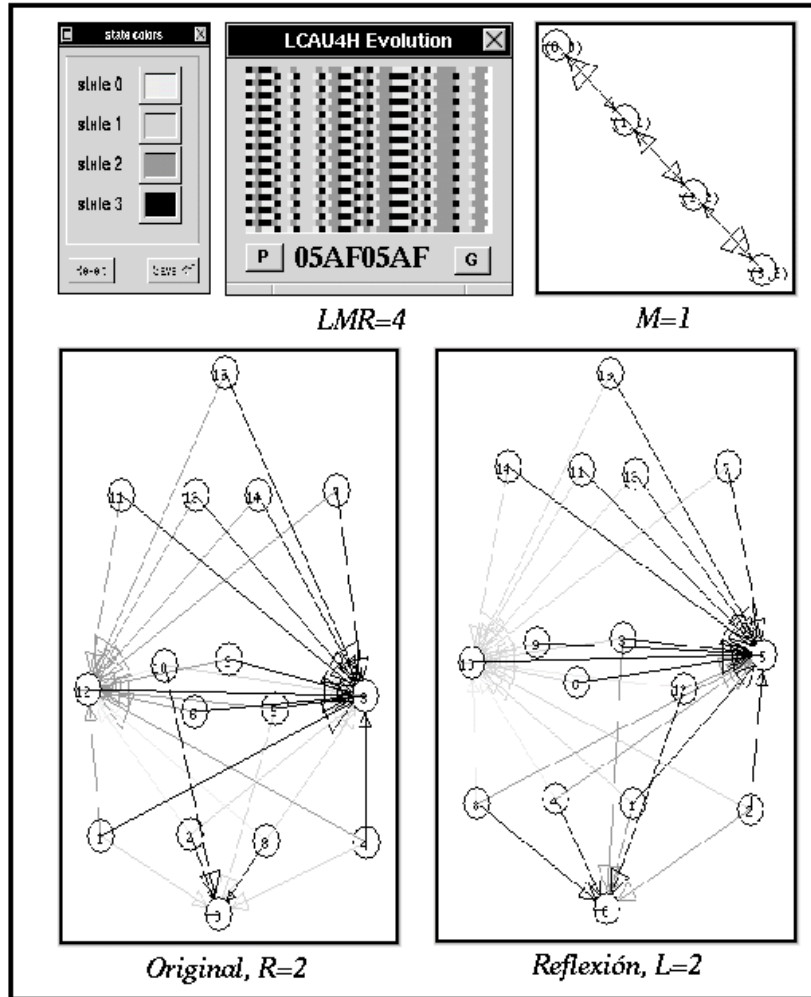


Figura 53: ACLR(4,h) regla 05AF05AF.

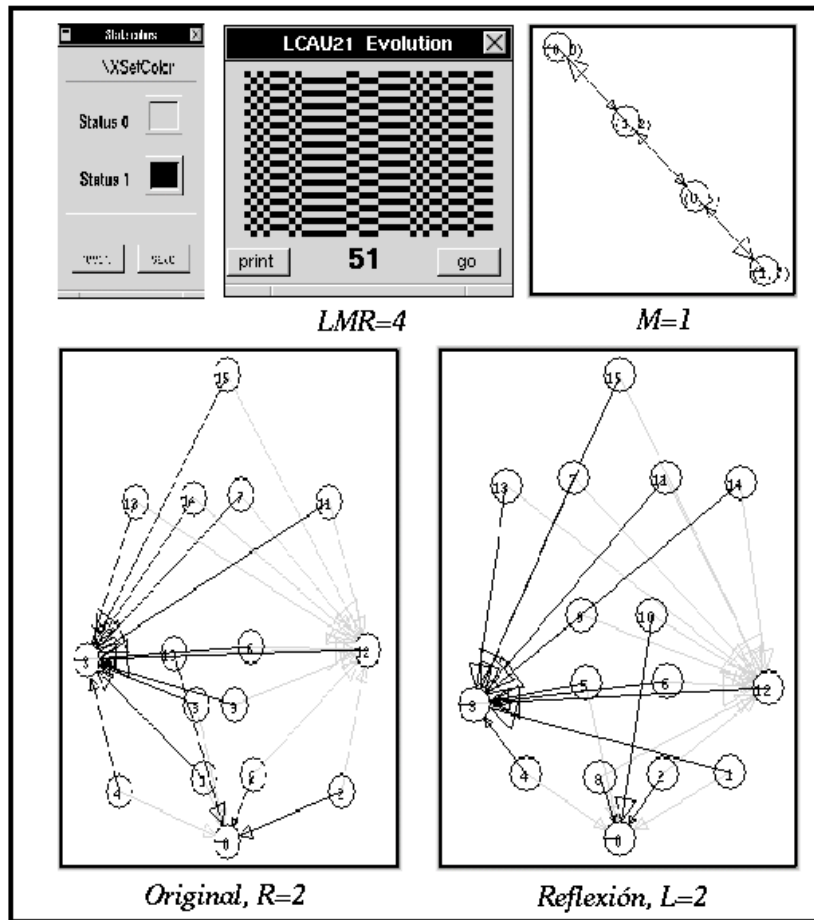


Figura 54: ACLR(2,1) regla 51.

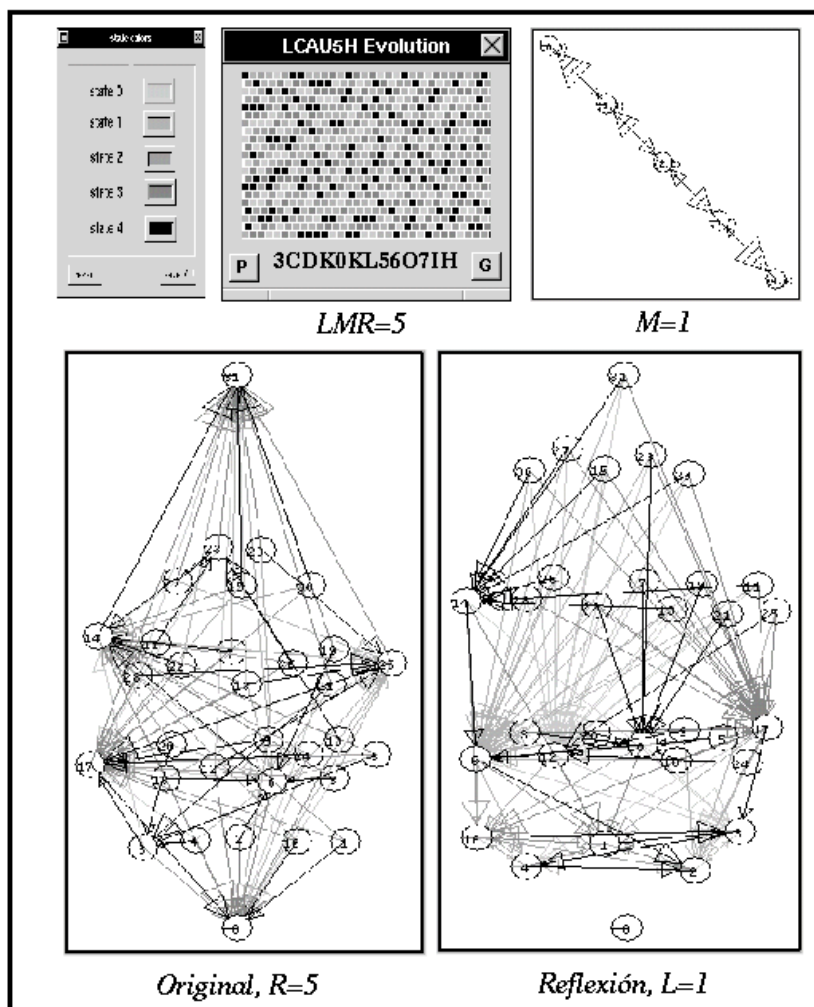


Figura 55: ACLR(5,h) regla 3CDK0KL56O7IH.

### 3.5 Procedimiento para encontrar la regla "inversa" de un Autómata Celular Lineal Reversible

Una vez que sabemos que un ACL es reversible, es importante tener un modo de determinar cuales son los posibles mapeos locales que aplicados al mismo ACLR puedan hacer regresar la evolución del mismo, es decir obtener las configuraciones pasadas del ACLR.

#### 3.5.1 Procedimiento de Fredkin

En [20] se describe de manera muy sencilla como funciona este proceso, supongamos que tenemos los cuadros aislados de una película que representan el movimiento de una bala, si tenemos dos cuadros consecutivos de la película, una con la posición presente de la bala y otra con la posición anterior de la misma, podemos reconstruir un tercer cuadro para la posición futura y este procedimiento puede ser iterado hacia adelante o hacia atrás; justamente esta es la idea básica del método que Fredkin desarrolla.

Su método utiliza dos generaciones de células, sea  $x_i^t$  el estado de la  $i$ ésima célula en la generación  $t$ , tenemos que la evolución de este estado está dada por

$$x_i^{t+1} = \phi(x_{i-1}^t, x_i^t, x_{i+1}^t)$$

para un ACL(2,1) donde  $\phi$  representa la regla de evolución del autómata; podemos inducir una nueva regla como

$$x_i^t + 1 = \phi(x_{i-1}^t, x_i^t, x_{i+1}^t) \otimes x_i^{t-1} = \Phi(x_i^{t-1}, x_{i-1}^t, x_i^t, x_{i+1}^t)$$

expandiendo a dos generaciones la regla de evolución, ésto a su vez se puede invertir como

$$x_i^t - 1 = \phi(x_{i-1}^t, x_i^t, x_{i+1}^t) \otimes x_i^{t+1} = \Phi(x_i^{t+1}, x_{i-1}^t, x_i^t, x_{i+1}^t)$$

para la función  $\otimes$  podemos utilizar una función la cual preserve la información del sistema como el or-exclusivo, el nor-exclusivo o cualquier función invertible de dos variables, veamos dos ejemplos de como funciona esta técnica, tomemos un ACLR(2,1) regla 240



Figura 56: ACLR(2,1) regla 240.

Tomemos ahora para  $\otimes$  la operación or-exclusivo

Operando 1	Operando 2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 18: Tabla de valores para el or-exclusivo.

Observemos ahora una configuración aleatoria de células y observemos como evoluciona.

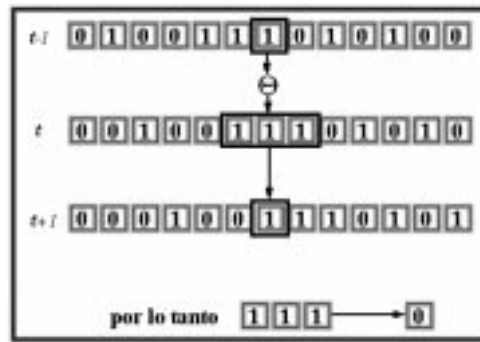


Figura 57: Utilizando el método de Fredkin en la evolución de una configuración aleatoria en un ACLR(2,1) regla 240.

Tomando por ejemplo la vecindad 111 en la generación  $t$  vemos que evoluciona en 1, además que la célula central de dicha vecindad en la generación  $t - 1$  es igual con 1, por lo tanto utilizando el método de Fredkin definimos una regla secundaria en la cual la vecindad 111 evolucione en 0 para que al hacer la operación or-exclusivo con la célula central del vecindario en la generación anterior produzca la misma evolución que la regla original, repitiendo el proceso para todas las vecindades obtenemos lo siguiente.

	0	1	2	3
0	0	1	-	-
1	-	-	0	1
2	1	0	-	-
3	-	-	1	0

Tabla 19: Regla secundaria que junto con la operación or-exclusivo realizan la misma evolución que la regla 240.

Si ahora formamos otra regla de evolución tomando en cuenta la regla secundaria y haciendo la operación or-exclusivo con la célula central del vecindario en la generación  $t + 1$  estaremos obteniendo justamente la regla inversa a la original, repitiendo este proceso para todas las vecindades se conforma la regla 170.

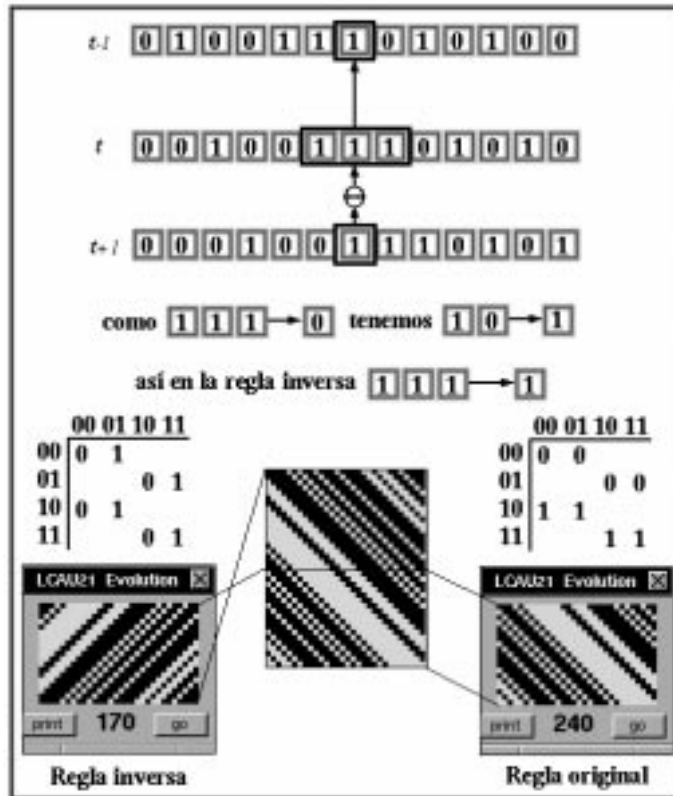


Figura 58: Regla 240 de un ACLR(2,1) y su regla inversa obtenida por el método de Fredkin.

Un ejemplo para un ACLR(3,h) regla 10179, tomemos ahora la suma modulo 3 para obtener la regla secundaria, y luego aplicamos la resta modulo 3 para obtener la regla inversa a la original, en este caso la 11355.

Op 1	Op 2	Suma mod $k$
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	0
2	0	2
2	1	0
2	2	1

Op 1	Op 2	Resta mod $k$
0	0	0
0	1	2
0	2	1
1	0	1
1	1	0
1	2	2
2	0	2
2	1	1
2	2	0

Tabla 20: Tabla de valores para las operaciones Suma modulo 3 y Resta modulo 3.

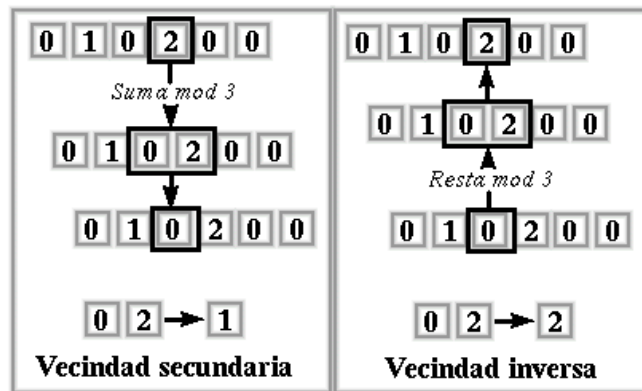


Figura 59: Usando el método de Fredkin con suma y resta módulo 3.



Vecindades	Regla Original	Regla Secundaria	Regla Inversa
00	0	0	0
01	0	1	2
02	0	2	1
10	2	2	0
11	2	0	2
12	2	1	1
20	1	1	0
21	1	2	2
22	1	0	1

Tabla 21: Regla 10179 de un ACLR(3,h) y su regla inversa 11355 obtenida por el procedimiento de Fredkin.

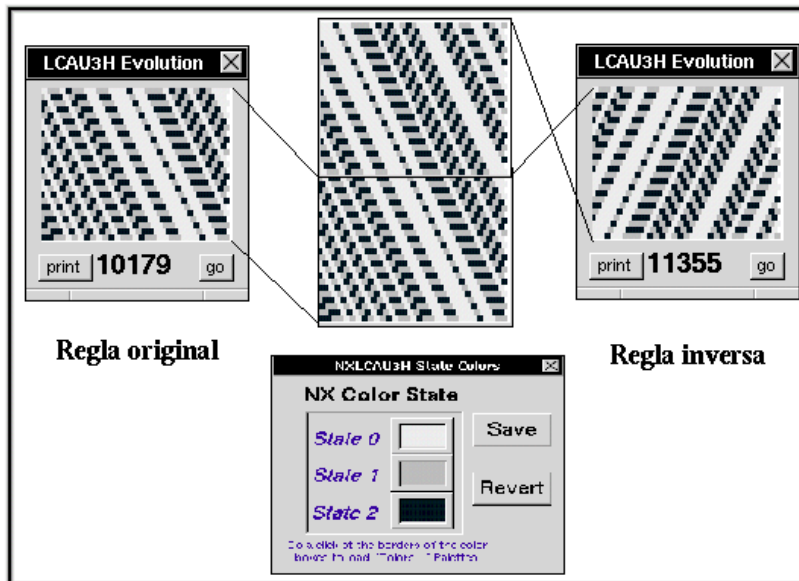


Figura 60: Evolución del ACLR(3,h) regla 10179 y de su regla inversa 11355.

## 4 Métodos actuales para encontrar Autómatas Celulares Lineales Reversibles.

A continuación se presentan los fundamentos y el funcionamiento de dos métodos (uno vigente y otro en desarrollo) que existen para encontrar los posibles ACLR.

### 4.1 Algoritmo de Hillman.

David Hillman [10] desarrolla un algoritmo para detectar todos los posibles ACLR con cualquier número de estados y cualquier tamaño de vecindad, el proceso es el siguiente:

Se van formando todos los posibles ancestros  $x_i$  de una cadena  $y$  de tamaño  $w$  y se almacenan en una tabla las parejas formadas por los  $2r$  elementos iniciales y finales de cada  $x_i$ , esto se hace para cada posible  $y$ , al final se tendrán un conjunto de tablas que guardan los  $2r$  elementos iniciales y finales de los posibles ancestros de cada cadena de ancho  $w$ , a tal conjunto se le denominará  $R_w$ ; se empieza con  $w = 1$ , para valores de  $w > 1$  simplemente se concatena  $R_{w-1}$  con  $R_1$  comparando cada tabla de  $R_{w-1}$  con cada tabla de  $R_1$ , si  $2r$  elementos finales de la tabla en  $R_{w-1}$  son iguales a los  $2r$  elementos iniciales de la tabla en  $R_1$  entonces se añade una nueva entrada en la tabla de  $R_w$  formada por los  $2r$  elementos iniciales en la tabla de  $R_{w-1}$  y los  $2r$  elementos finales en la tabla de  $R_1$ , las tablas de  $R_1$  se obtienen directamente de la regla de evolución, de esta manera el proceso puede ser implementado en una computación recursiva; si el ACL es reversible, debe existir para cada tabla en  $R_w$  una sola pareja en donde los  $2r$  elementos iniciales sean iguales con los  $2r$  elementos finales ya que esto indicaría que una y solo una cadena  $x_i^*$  de longitud  $w$  puede evolucionar en  $y$  para toda  $y$  de longitud  $w$ , es decir, cada posible cadena solo tendría un único ancestro y por lo tanto su evolución sería invertible.

Por supuesto, si  $w$  es grande el método se vuelve muy laborioso, sin embargo Hillman hace dos observaciones que minimizan este problema; en primera si las tablas de un conjunto  $R_i$  no tienen todas el mismo número de parejas entonces el ACL no es reversible pues este desequilibrio llevará a que la información del sistema se pierda gradualmente y con esto la reversibilidad del mismo; el segundo punto que Hillman señala es que como los valores que pueden tener cada tabla son finitos, entonces llegará un momento en que se repetirán estos valores; en síntesis, si cada tabla en  $R_j$  tiene el mismo número de parejas para  $j \geq 1$  y  $R_j = R_i$  para  $i < j$  entonces el ACL es reversible y el proceso termina.

Veamos un ejemplo de este algoritmo para ACL(3,h).

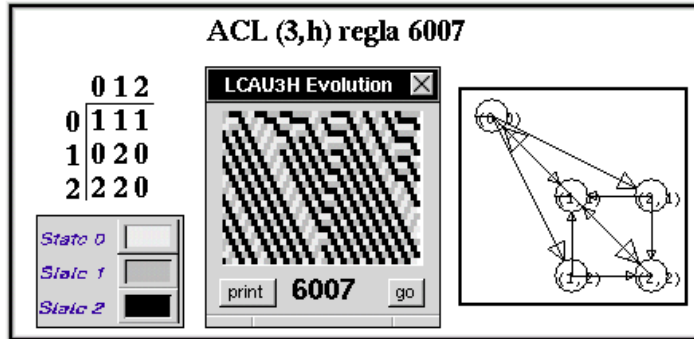


Figura 61: ACL(3,h) regla 6007.

Formemos las tablas de los  $2r$  elementos iniciales y finales para los ancestros de 0, 1 y 2.

Ancestros de 0	Ancestros de 1	Ancestros de 2
1 0	<b>0 0</b>	<b>1 1</b>
1 2	0 1	2 0
<b>2 2</b>	0 2	2 1

Tabla 22: Conjunto  $R_1$  para el ACL(3,h) regla 6007.

Para  $R_1$  se encuentra que cada posible cadena tiene un solo ancestro, ahora concatenemos  $R_1$  consigo mismo para obtener los ancestros de 00, 01 y 02.

Ancestros de 00	Ancestros de 01	Ancestros de 02
1 2	1 0	1 0
<b>2 2</b>	<b>1 1</b>	<b>1 1</b>
	1 2	2 0
		2 1

Tabla 23: Concatenación de  $R_1$  con  $R_1$ .

Como no se cumple la condición de que cada tabla tenga el mismo número de parejas, se detiene el algoritmo y ACL(3,h) no es reversible.

Veamos otro ejemplo para otro ACL(3,h).

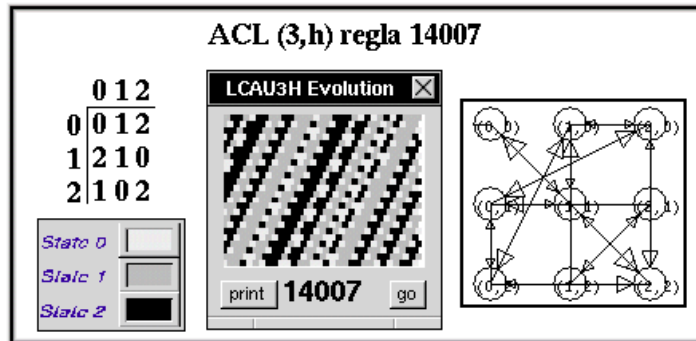


Figura 62: ACL(3,h) regla 14007.

Formemos las tablas de los  $2r$  elementos iniciales y finales para los ancestros de 0, 1 y 2.

Ancestros de 0	Ancestros de 1	Ancestros de 2
<b>0 0</b>	1 0	0 2
1 2	<b>1 1</b>	1 0
2 1	2 0	<b>2 2</b>

Tabla 24: Conjunto  $R_1$  para el ACL(3,h) regla 14007.

Para  $R_1$  también encontramos que cada posible cadena tiene un solo ancestro, ahora concatenemos  $R_1$  consigo mismo para obtener los ancestros de 00, 01 y 02.

Ancestros de 00	Ancestros de 01	Ancestros de 02
<b>0 0</b>	2 0	0 2
<b>1 1</b>	2 1	2 0
<b>2 2</b>	1 0	1 2

Tabla 25: Concatenación de  $R_1$  con  $R_1$ .

Aquí se observa que la cadena 00 tiene tres posibles ancestros mientras que las cadenas 01 y 02 no tienen ninguno, así que se detiene el algoritmo y ACL(3,h) no es reversible.

Veamos otro ejemplo para otro ACL(3,h).

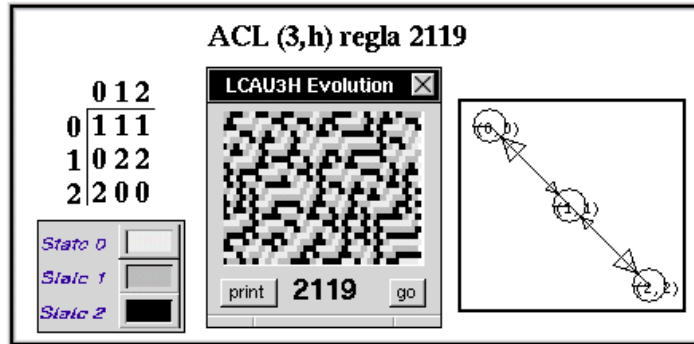


Figura 63: ACL(3,h) regla 2119.

Formemos las tablas de los  $2r$  elementos iniciales y finales para los ancestros de 0, 1 y 2.

Ancestros de 0	Ancestros de 1	Ancestros de 2
1 0	<b>0 0</b>	2 0
2 1	0 1	<b>1 1</b>
<b>2 2</b>	0 2	1 2

Tabla 26: Conjunto  $R_1$  para el ACL(3,h) regla 2119.

Para  $R_1$  también encontramos que cada posible cadena tiene un solo ancestro, ahora concatenemos  $R_1$  consigo mismo para obtener los ancestros de todas las posibles cadenas de longitud 2, en este caso solo obtenemos tres tablas distintas las cuales son:

2 0	1 0	<b>0 0</b>
2 1	<b>1 1</b>	0 1
<b>2 2</b>	1 2	0 2

Tabla 27: Conjunto  $R_2$ .

Si concatenamos ahora  $R_2$  con  $R_1$  obtenemos:

2 0	1 0	<b>0 0</b>
2 1	<b>1 1</b>	0 1
<b>2 2</b>	1 2	0 2

Tabla 28: Conjunto  $R_3$ .

Como este conjunto es igual a  $R_2$ , tenemos que el ACL(3,h) es reversible y se detiene el proceso.

En resumen, el algoritmo de Hillman va formando todos los posibles ancestros de las cadenas de longitud  $w$  empezando desde 1 y checa si hay un único ancestro para cada una de estas cadenas de longitud  $w + 2r$  que al tener los elementos  $2r$  iniciales y finales iguales, se pueda representar como una secuencia de estados también de longitud  $w$  que evoluciona en la cadena estudiada.

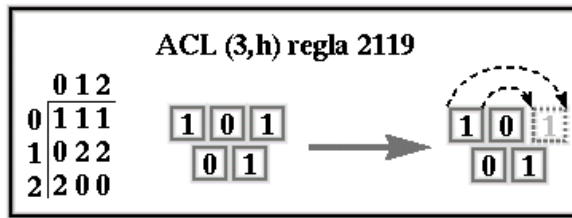


Figura 64: Ancestro único de la cadena 01 que puede representarse con la misma longitud.

## 4.2 Algoritmo utilizando los Indices de Welch

Una aproximación tentativa en tratar de utilizar los resultados de los trabajos en [9] y [16] fué desarrollada por el Dr. Harold V. McIntosh y Jose Manuel Gómez Soto; este algoritmo consiste en aprovechar los conceptos de multiplicidad uniforme e índices de Welch para la construcción de las posibles matrices de evolución de ACLR; por ejemplo, para un  $ACLR(5,h)$  se tiene la siguiente “plantilla” para generar las matrices de evolución:

	0	1	2	3	4
0	0				
1		1			
2			2		
3				3	
4	4	4	4	4	4

Tabla 29: Plantilla para generar matrices de evolución de  $ACLR(5,h)$ .

El resto de los lugares vacios se llenaban con todas permutaciones de estados posibles de tal manera que un elemento de cada columna fuera diferente al resto, esto es ya que al tener cinco nodos el diagrama de de Bruijn asociado, el valor de  $LMR = 5$ , por lo que se fija a  $M = 1$  y  $L = 1$ , lo que implica que no debe existir dos elementos iguales por columna.

La razón por la cual el último renglón de la matriz estaba lleno del estado 4 era para tener un estado en el cual fuera seguro que el conjunto de todas sus extensiones compatibles derechas tuviera una cardinalidad igual a  $R$ , cumpliendo para ese estado que  $LMR = 5$ , una vez llena la matriz, se generaba una evolución de este ACL y se observaba si cada posible vecindad de longitud 2 tenía un único estado para el cual evolucionar hacia atrás, de cumplir ésto, el ACL se tomaba como reversible, de no ser así se verificaban si las vecindades de longitud 3 si cumplían con tener un único estado en el pasado; este proceso continuaba hasta revisar vecindades de longitud 4.

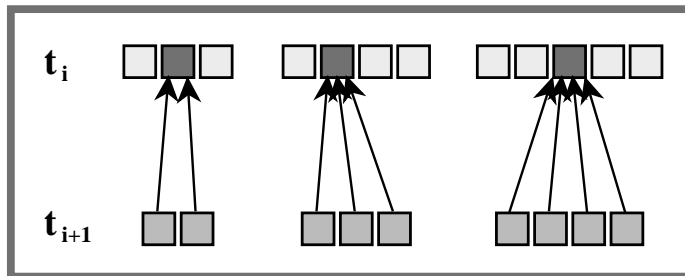


Figura 65: Verificación de que cada cadena tenga un único elemento en común en el pasado para cadenas de longitud 2, 3 y 4.

La ventaja de este planteamiento es que ya no hace la revisión de cada posible regla de evolución para un  $ACL(k,h)$  sino que solo analiza aquellas matrices de evolución en donde se tenga una diagonal principal donde cada estado sea diferente al resto y un renglon formado por un mismo valor para asegurar que la cardinalidad del conjunto de extensiones compatibles por la derecha del mismo fuera  $R$ , con lo que el tiempo de computación que toma hacer el cálculo se reduce de manera significativa.



## 5 Propuesta de un método para encontrar Autómatas Celulares Lineales Reversibles.

En esta sección se propone un método para encontrar todos los posibles ACLR de  $k$  estados y un radio de vecindad  $r$ ; cabe señalar que al principio dicho método fué desarrollado basado en observaciones experimentales de ACLR y después se depuró aplicando los resultados obtenidos por [9] y [16].

### 5.1 Propiedades de la matriz de evolución de los Autómatas Celulares Lineales Reversibles

Basado en los conceptos que anteriormente se han presentado, se mostrará que propiedades debe cumplir la matriz de evolución de un ACL para ser reversible, después se utilizarán estas propiedades en el diseño de un mecanismo que genere todos los posibles ACLR para un número de estados  $k$  y un radio de vecindad  $r$ .

Este es el momento adecuado para hacer una observación importante; para un ACL( $k,r$ ) el número de posibles reglas reversibles va creciendo exponencialmente conforme aumenta  $k$  y/o  $r$ ; de esta manera tenemos que existen desde unas cuantas reglas reversibles para el caso de un ACL(2, $h$ ) o ACL(2,1) hasta varias miles para un ACL(5, $h$ ) y millones de reglas reversibles para un ACL(6, $h$ ); es por esta razón que se utiliza el concepto de “cluster” (grupo, racimo) para agrupar las diferentes reglas reversibles que puedan existir en un ACL( $k,h$ ).

Tomemos como ejemplo un ACLR(4, $h$ ):

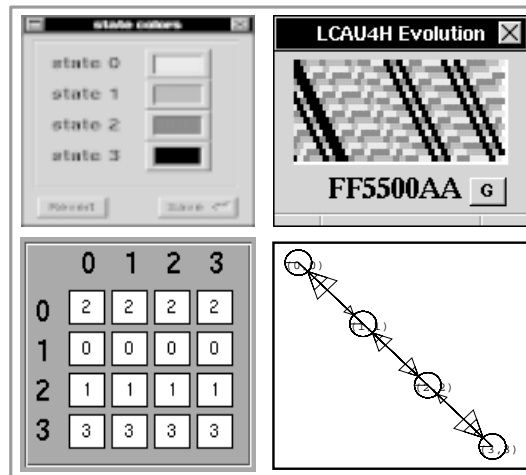


Figura 66: ACLR(4, $h$ ) regla FF5500AA.

Si hacemos una misma permutación de renglones, de columnas o de estados a la matriz de evolución obtendremos otro ACLR que pertenece al mismo cluster, repitiendo este proceso para todas las posibles permutaciones de 4 elementos podemos obtener todas las variantes de la matriz de evolución, por último tomemos

aquella variante cuyo número wolfram sea el menor que se haya obtenido (lexicográficamente hablando) así obtendremos el representante de este cluster (cluster mínimo).

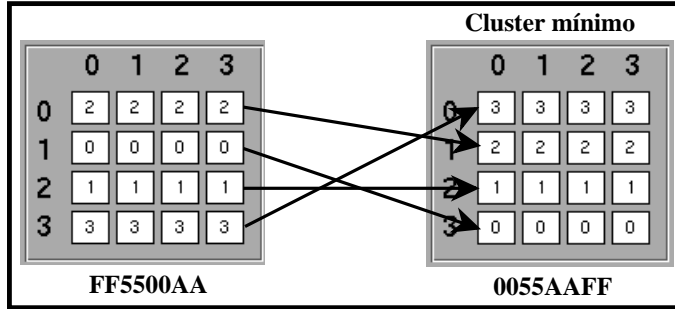


Figura 67: Cluster mínimo del ACLR(4,h) regla FF5500AA.

Aplicando el procedimiento para cada regla reversible, éstas se pueden agrupar en distintos clusters, teniendo que cada miembro mínimo de un cluster puede representar a cientos o miles de reglas reversibles dependiendo del tamaño de vecindad y número de estados del ACL.

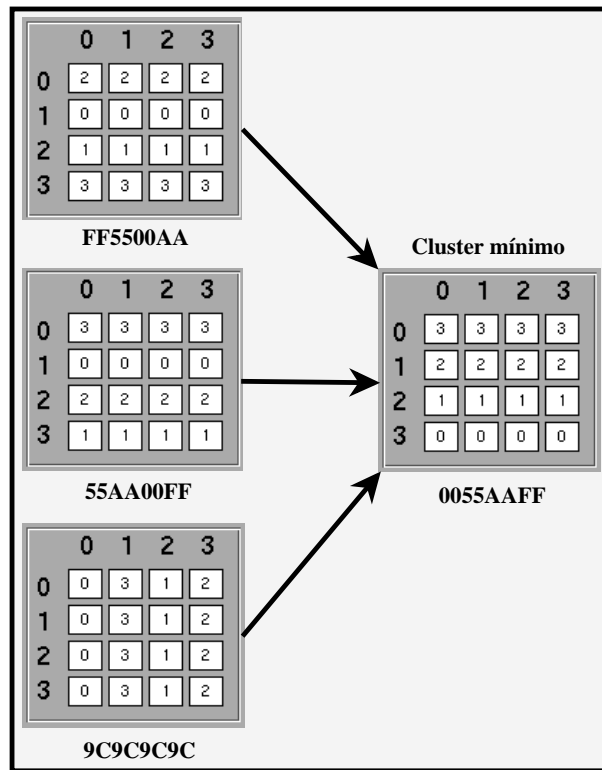


Figura 68: Cluster mínimo de varios ACLR(4,h).

Ahora veamos que propiedades debe cumplir la matriz de evolución de un ACL(k,h) para ser reversible.

### 5.1.1 Propiedades por cada estado

Primero se definirá que propiedades debe tener cada estado dentro de la matriz de evolución para construir las posibles reglas reversibles.

- Cada estado debe aparecer una sola vez en la diagonal principal

Esto nos garantiza que cada estado tengan un ancestro, pero si en la diagonal principal existen más de un elemento del mismo estado, se tendrán múltiples formas de ese mismo estado, lo que no puede ocurrir en un ACLR.

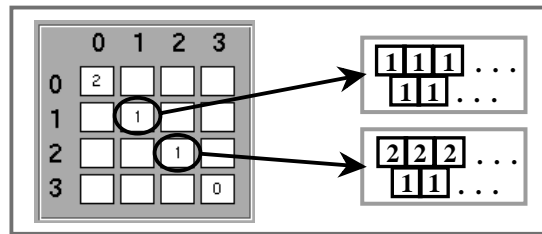


Figura 69: ACL(4,h) con múltiples ancestros para una cadena de 1's.

Como se ve en la figura, una secuencia de 1's puede ser construida por una cadena de 1's o de 2's, es decir, tiene varios ancestros posibles lo que no permite que sea reversible.

- Cada estado debe aparecer el mismo número de veces que el resto en la matriz de evolución

En un ACLR la multiplicidad uniforme de cada cadena debe ser respetada, si en principio un estado puede generarse de un mayor número de maneras que otro estamos contradiciendo este requerimiento.

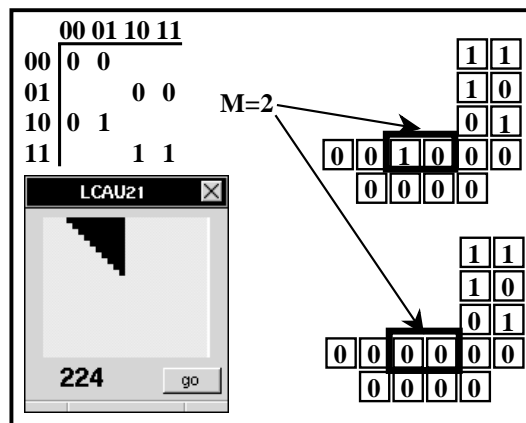


Figura 70: ACL(2,1) con múltiples ancestros para una cadena de 0's.

En la figura anterior se observa que una cadena formada de 0's tiene un valor de  $M = 2$  debido a que el estado 0 aparece más veces que el estado 1; lo que produce que en la evolución del ACL el estado 1 vaya desapareciendo conforme transcurre el tiempo.

De esta manera se observa que la multiplicidad uniforme debe cumplirse en la formación de cada estado dentro de la regla de evolución y como se definió en la sección 3, el valor de los posibles ancestros de cada estado debe ser igual a  $k^{2r}$  ó al número de nodos del diagrama de de Bruijn.

### 5.1.2 Propiedades globales de la matriz

Una vez definidas que propiedades debe cumplir cada estado en un ACLR, podemos utilizar las mismas para construir solo aquellas matrices de evolución que cumplan dichas propiedades y no todas las posibles reglas; sin embargo, el problema de la reversibilidad en un ACLR también depende de la interacción que los estados tengan unos con otros; esta interacción debe ser tal que permita conservar la información del sistema para poder reconstruir la evolución del mismo ya sea hacia adelante o hacia atrás en el tiempo, esto quiere decir que en un ACLR cada configuración global debe ser posible de generar a través de la evolución del autómata, pero por uno y solo un ancestro posible, evitando que exista el Jardín del Edén o ancestros múltiples.

Para lograr este objetivo se utilizarán los otros dos conceptos importantes expuestos en [9] y [16], que son los índices de Welch y la combinabilidad, en realidad estos conceptos están muy relacionados entre sí ya que la combinabilidad es solo checar que toda ruta posible con una longitud mayor o igual que  $2r$  tenga los mismos valores de los índices  $L$  y  $R$  cumpliendo que  $L * R = k^{2r}$ .

La pregunta aquí es cómo podemos observar ésto en la matriz de evolución de un ACL; en realidad la matriz de evolución puede ser descompuesta en matrices de conectividad, una para cada estado, cada matriz de conectividad tendrá un 1 si existe una liga con el valor de su estado que va de un nodo a otro en el diagrama de de Bruijn.

M. E.	M. C. estado 0	M. C. estado 1	M. C. estado 2
$  \begin{array}{c}  \\  \begin{array}{ccc}  0 & 1 & 2 \\  \hline  0 & 1 & 1 \\  1 & 2 & 2 \\  2 & 0 & 0  \end{array}  \end{array}  $	$  \begin{array}{c}  \\  \begin{array}{ccc}  0 & 1 & 2 \\  \hline  0 & 0 & 0 \\  1 & 0 & 0 \\  2 & 1 & 1  \end{array}  \end{array}  $	$  \begin{array}{c}  \\  \begin{array}{ccc}  0 & 1 & 2 \\  \hline  1 & 1 & 1 \\  0 & 0 & 0 \\  0 & 0 & 0  \end{array}  \end{array}  $	$  \begin{array}{c}  \\  \begin{array}{ccc}  0 & 1 & 2 \\  \hline  0 & 0 & 0 \\  1 & 1 & 1 \\  0 & 0 & 0  \end{array}  \end{array}  $

Tabla 30: Matriz de evolución de ACLR(3,h) regla 715 y sus matrices de conectividad.

Las matrices de conectividad nos dan también las rutas de tamaño 1 que existen en el diagrama de de Bruijn y podemos saber si tales rutas cumplen con la propiedad de combinabilidad examinando la matriz de conectividad para cada estado, por

ejemplo, para el ACLR(3,h) anterior, se observa que para la matriz de conectividad del estado 0 empezando desde el renglón 2 puedo llegar a los estados 0,1 y 2, es decir el valor de la cardinalidad del conjunto de extensiones compatibles por la derecha del estado 2 que genere un 0 al evolucionar es 3, como no puede existir otra cardinalidad mayor por el principio de multiplicidad uniforme se tiene que  $R = 3$ ; ahora si examinamos en la misma matriz las columnas 0, 1 y 2, vemos que cada una de éstas son parte final de la liga que empieza desde el renglon 2, es decir, cada uno de estos estados tiene una cardinalidad del conjunto de extensiones compatibles por la izquierda igual a 1 para generar un 0, debido a que cada columna está formada en este caso por elementos todos distintos tenemos que este valor es también máximo y  $L = 1$ ; ahora bien, el estado 2 es el único que cumple con tener valores de  $R = 3$  y  $L = 1$  para generar un 0, por lo tanto  $M = 1$ , al final tenemos que  $LMR = k^{2r} = 3$  cumpliendo con la propiedad de los índices de Welch, si hacemos extensivo este estudio para las demás matrices de conectividad obtenemos los mismos resultados, satisfaciendo de esta manera la condición de combinabilidad.

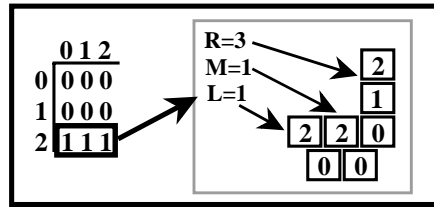


Figura 71: Matriz de conectividad del estado 0 del ACLR(3,h) regla 715 y como se manifiesta los índices de Welch en ésta.

Este comportamiento se observa claramente en el diagrama de subconjuntos asociado al ACLR(3,h); partiendo desde el estado 2 se puede llegar con un 0 a todos los nodos posibles del diagrama de de Bruijn, así en el diagrama de subconjuntos existe una liga que va desde el subconjunto unitario 4 al subconjunto completo 7 ( $R = 3$ ) mientras que con la regla de evolución reflejada la dirección de las ligas en su diagrama de subconjuntos simboliza la evolución desde donde termina la vecindad hasta donde inicia en la regla original, con lo que tenemos que para generar un 0 terminando en el estado 2 solo existe una extensión compatible izquierda que es otro estado 2, o lo que es lo mismo un ciclo de longitud 1 en el diagrama de subconjuntos en el nodo 4 ( $L = 1$ ), como desde el nodo 4 (estado 2) en ambos diagramas de subconjuntos es el único donde podemos extendernos por la derecha o izquierda para generar 0, tenemos que  $M = 1$  y se cumple que  $LMR = 3$ .

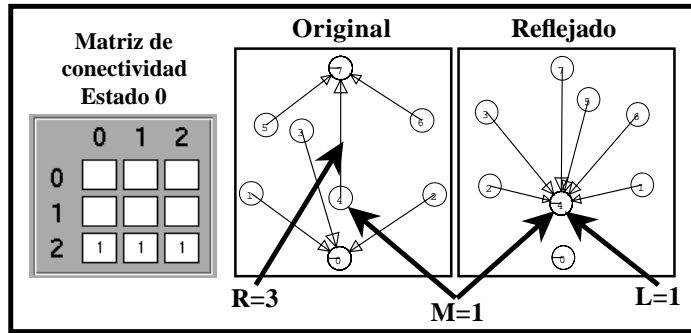


Figura 72: Diagramas de subconjuntos y valores de  $L$ ,  $M$  y  $R$  para la matriz de conectividad del estado 0 del ACLR(3,h) regla 715.

Este comportamiento se puede observar para los demás estados en los diagramas de subconjuntos tanto de la regla original como de la regla reflejada.

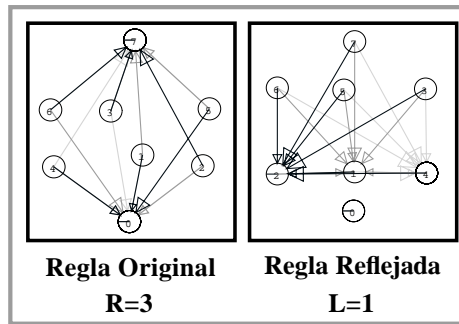


Figura 73: Diagramas de subconjuntos (regla original y reflejada) del ACLR(3,h) regla 715.

Sin embargo, al checar solo las matrices de conectividad de un ACL solo estamos estudiando las rutas de longitud 1 ó lo que es lo mismo los ancestros de las cadenas de un elemento; pero hemos visto que la propiedad de combinabilidad no necesariamente tiene que presentarse en las cadenas de longitud 1 sino que puede aparecer hasta cadenas que tengan un mayor número de estados, entonces cómo podemos utilizar las matrices de conectividad para estudiar cadenas de longitud mayor que 1 ó equivalentemente, rutas de longitud mayor que 1 en el diagrama de de Bruijn.

Es bien conocido en la Teoría de Gráficas que si una matriz de conectividad se eleva a cierta potencia estará representando las rutas que existen en la gráfica asociado con longitud igual a esa potencia, así ésto se puede extender más ya que la multiplicación de dos matrices de conectividad de una gráfica representa las rutas que existen en el mismo formadas por los arcos cuyos valores corresponden a los de las matrices de conectividad.

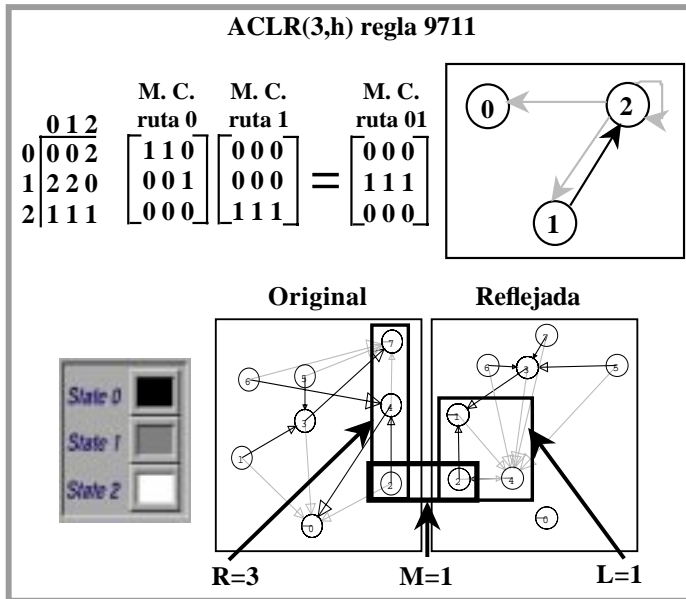


Figura 74: Indices de Welch para la cadena 01 del ACLR(3,h) regla 9711.

En la figura observamos las matrices de conectividad de los estados 0 y 1, en el caso de la matriz del estado 0 por si sola no tiene el valor necesario de los índices de Welch para considerarla como reversible, pero al multiplicarla por la matriz del estado 1 para obtener la matriz de conectividad de la cadena 01, se observa en el resultado que el valor de los índices satisface la propiedad de combinabilidad; si extendemos este análisis para todas las matrices de conectividad de las cadenas de longitud 2 y este comportamiento se presenta en todos los casos, se satisface por completo la propiedad de combinabilidad y el ACL es reversible, en caso contrario, se tendrán que checar las matrices de conectividad de las cadenas de longitud 3 y así sucesivamente.

En síntesis, para leer el valor de los índices de Welch desde una matriz de conectividad y saber si está cumpliendo con el principio de combinabilidad el número de elementos diferentes a 0 por cada renglón debe ser igual a 0 ó  $R$ , el número de elementos diferentes de 0 por cada columna debe ser igual a 0 ó  $L$  y la suma de elementos en la diagonal principal debe ser 1, es decir, un solo ciclo es permitido para producir la secuencia de estados representada por la matriz de conectividad; si cada matriz de conectividad correspondiente a cada cadena posible de una longitud dada (que puede ser mayor o igual a  $2r$ ) cumple estas características y con  $L * R = k^{2r}$  se concluye que para cada una de estas cadenas el valor de  $M = 1$  y por lo tanto el ACL es reversible.

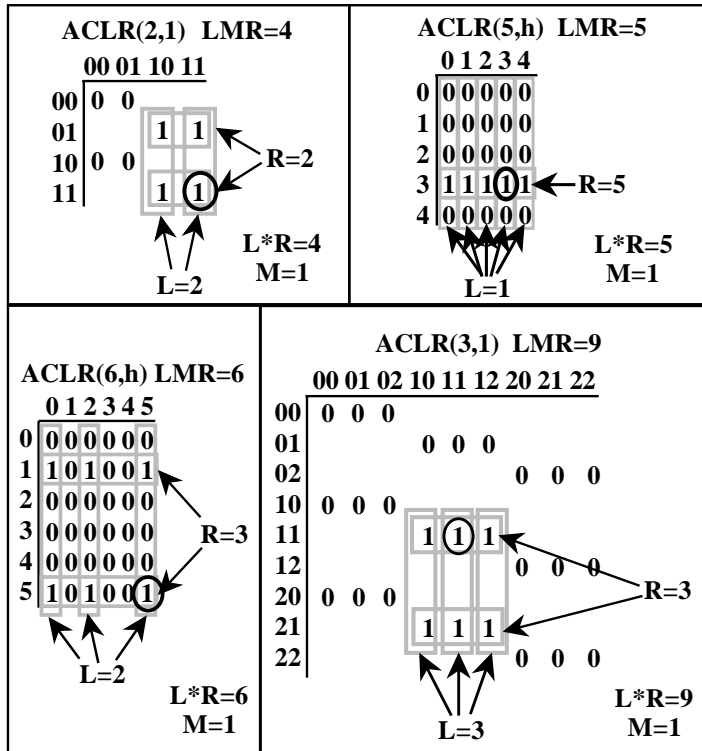


Figura 75: Diferentes formas de matrices de conectividad para ACLR, con un solo 1 en la diagonal principal y valores de  $L * R = k^{2r}$  induciendo que  $M = 1$ .

Para un ACL no reversible el proceso anterior generará matrices de conectividad en donde ya sea que la suma de los elementos de la matriz sea diferente a  $k^{2r}$  y por lo tanto no se cumpla la multiplicidad uniforme ó la suma de elementos en la diagonal principal sea mayor que 1 con lo que existirían multiples ancestros para una cadena dada.

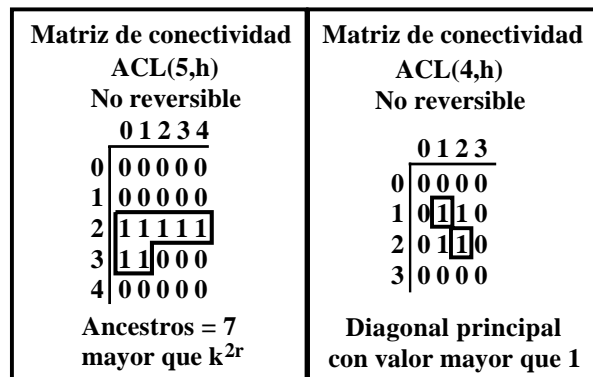


Figura 76: Matrices de conectividad para ACL no reversibles.



## 5.2 Método para encontrar Autómatas Celulares Lineales Reversibles

A continuación se presentará un proceso el cual utiliza las propiedades anteriormente expuestas para calcular todos las posibles reglas reversibles para un ACL(k,r), como forma de ejemplificar lo anterior se tomará el caso de un ACL(4,h) pero este proceso se puede adaptar para cualquier caso; la estrategia a seguir es ir generando todas las matrices de evolución posibles que cumplan las propiedades por estados individuales necesarias para que el ACL sea reversible, para ésto se tomará una plantilla para la matriz de evolución donde los elementos de la diagonal principal sean diferentes entre sí y permanezcan sin cambio durante el proceso.

		0	1	2	3
0	0				
1			1		
2				2	
3					3

Tabla 31: Plantilla para generar todas las matrices de evolución posibles candidatas a ser ACLR(4,h).

Sobre esta plantilla se empezará a colocar los elementos de cada estado desde 0 a  $k - 1$  (en el ejemplo anterior de 0 a 3), cumpliendo con que cada estado debe aparecer el mismo número de veces que los demás.

Un proceso adicional en el método es verificar conforme se actualiza cada estado que no se formen dos o más ciclos iguales de longitud 2 con nodos distintos (cosa que no puede ocurrir en un ACLR pues indica que una secuencia de dos elementos puede ser generada por varias cadenas de dos elementos); la forma de verificar ésto es formar una matriz auxiliar en donde cada renglón representará un estado del ACL y los elementos del mismo una codificación de las vecindades que lo generan, esto puede ser elevar al cubo el valor de cada célula que pertenece al ancestro del estado y sumar los valores.

Si existen dos valores iguales en un renglón significa que existe un ciclo de longitud 2 formado por un mismo estado, de este modo una secuencia de cualquier longitud formada por este estado tendría dos ancestros posibles uno compuesto por los elementos del ciclo y otro por las coordenadas de la diagonal principal donde se localiza el estado.

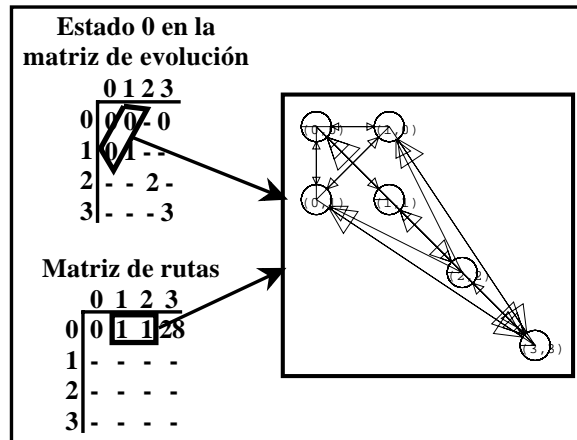


Figura 77: Matriz de rutas del estado 0 en un ACL(4,h) donde se observan múltiples ancestros.

Si un renglón tiene dos valores que aparecen en otro renglón significa que existen dos ciclos iguales de longitud 2 formado por distintos nodos, así una secuencia formada por los elementos del ciclo tiene como posibles ancestros cadenas compuestas ya sea por los nodos de uno de los ciclos ó por los nodos del otro, evitando que sea reversible.

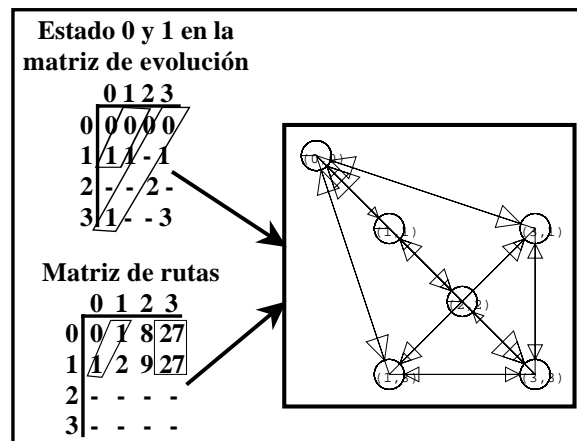


Figura 78: Matriz de rutas del estado 0 y 1 en un ACL(4,h) donde se observan múltiples ancestros.

Aunque este paso no es necesario para detectar ACLR es de gran utilidad ya que nos ayudará a descartar muchas matrices de evolución de manera rápida y por lo tanto agilizar el proceso de cálculo.

Una vez que la matriz se ha llenado la matriz de evolución completamente con los elementos de cada estado, se procede a verificar si las matrices de conectividad para cadenas de un elemento cumplen con el principio de combinabilidad de Nasu,

de no ser así se verifican las matrices de conectividad de cadenas de dos elementos y se continúa de esta forma ya sea hasta que todas las matrices de conectividad generadas cumplan con ser debilmente combinables (y por lo tanto el ACL es reversible) obteniendo el cluster mínimo del mismo para guardarlo, o que se cumpla en cualquier matriz de conectividad alguna de las condiciones presentadas para descartar la regla de evolución como reversible.

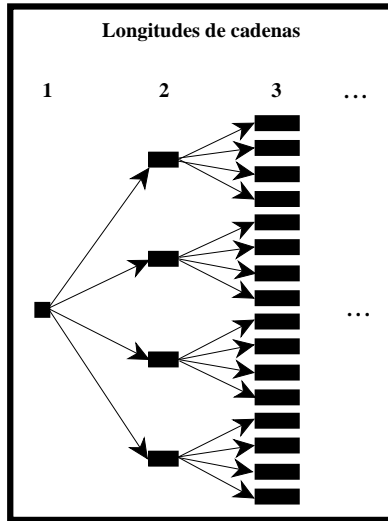


Figura 79: Crecimiento del número de matrices de conectividad a revisar para cadenas de longitud 1 en adelante para un  $ACL(4,h)$ .

El proceso completo se realiza para toda matriz de evolución generada; se empezará a llenar la plantilla de la matriz de evolución con el estado  $n$  para  $0 \leq n \leq k - 1$ ; en síntesis el algoritmo sería de la siguiente forma:

1. Actualizar en la matriz de evolución del  $ACL(k,r)$  los elementos del estado  $n$
2. Checar ciclos de longitud 2, si estos existen, regresar a 1
3. Si  $n < k - 1$  se pasa a 6
4. Checar las matrices de conectividad empezando para las cadenas de longitud 1 en adelante; si se presenta alguna de las condiciones de no reversibilidad, pasar a 7
5. Obtener el cluster mínimo del ACLR, compararlo con los ya obtenidos y si es nuevo guardarlo en un archivo
6. Hacer el proceso ahora para el estado  $n + 1$
7. Si existen más formas de colocar los elementos del estado  $n$  en la matriz de evolución regresar a 1; si no salir del proceso

El siguiente diagrama refleja el flujo del programa; el listado del mismo para el caso (4,h) se presenta en el Apéndice A.

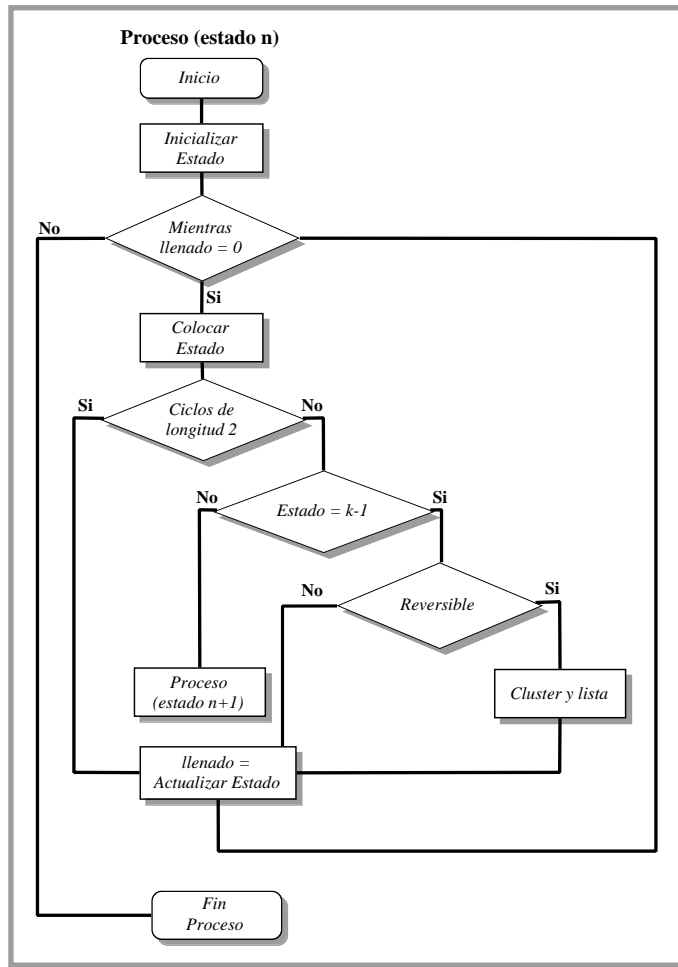


Figura 80: Flujo del proceso para obtener  $ACLR(k,r)$ .

### 5.3 Comentarios sobre el método propuesto

El algoritmo aquí presentado se ha implementado para encontrar todas las reglas reversibles para los ACL(4,h), (5,h) y (6,h) a continuación se presentan algunos apuntes sobre el comportamiento de mismo.

En este aspecto cabe destacar que el proceso genera todas las posibles matrices donde cada estado aparezca el mismo número de veces que el resto, es decir mientras más estados se tengan, el número de matrices generadas aumenta dramáticamente, debemos añadir a esto el tiempo requerido para saber si una posible matriz de evolución define un ACLR ya que el número de matrices de conectividad a estudiar aumenta exponencialmente conforme aumente la longitud de las cadenas a analizar; una vez que una matriz de evolución define un ACLR se debe calcular el cluster mínimo, cálculo el cual también aumenta su tiempo de manera exponencial conforme existan mayor número de estados.

Todo esto nos lleva a que para el caso de un ACL(4,h) el tiempo de cálculo de todas las posibles reglas reversibles nos lleva 20 segundos, para el caso (5,h) sabemos que uno de los índices de Welch debe ser 1 y se puede aprovechar esto fijando por ejemplo a  $L = 1$ , generando solo aquellas matrices de evolución donde cada columna este formada por elementos todos distintos; por lo que el tiempo de cálculo requerido para los posibles reversibles (5,h) es de dos minutos y medio aproximadamente; en el caso (6,h) tenemos que no se puede fijar el valor de un índice a 1, por lo que se tiene que generar todas las posibles combinaciones de estados para producir las posibles matrices de evolución reversibles, siguiendo un esquema parecido al caso (5,h) se calcularon solo aquellos reversibles (6,h) donde  $L = 1$  y el proceso tardó un poco más de 20 horas.

Otro aspecto a mencionar es que el proceso propuesto no necesita almacenar en memoria las matrices de conectividad que vayan generando, estas simplemente se obtienen de la matriz de evolución producida y para obtener las matrices de evolución de cadenas de longitud mayor que 1 se usa un proceso recursivo, dejando el manejo de la memoria en manos del compilador.

## 6 Presentación de los resultados obtenidos con el método propuesto

En esta sección se muestran los resultados obtenidos por el método propuesto para encontrar los ACLR en los casos (4,h),(5,h) y (6,h), en los primeros dos casos se mostrará la lista completa de ACLR encontrados y algunos ejemplos de evolución de los mismos y en el último caso ya que el listado resulta muy extenso, solo se presentarán algunos ejemplos de evolución; en cada autómata se presenta el espécimen que el programa de computación genera así como el cluster mínimo del mismo.

Para los casos con menor número de estados (2,h) y (3,h) solo existe uno y dos ACLR respectivamente, por lo que no se presentan resultados de los mismos.

### 6.1 Resultados para los Autómatas Celulares Lineales Reversibles (4,h)

AUTOMATAS CELULARES LINEALES REVERSIBLES 4H con L=1, R=4	
Cluster Mínimo No. 1: 0055AAFF	Especimen: FFAA5500
Cluster Mínimo No. 2: 0055BAEF	Especimen: FBAE5500
Cluster Mínimo No. 3: 0055AFFA	Especimen: FAAF5500
Cluster Mínimo No. 4: 0056E9BF	Especimen: FE6B9500
Cluster Mínimo No. 5: 006ABFD5	Especimen: FEA95700
Cluster Mínimo No. 6: 0066BFD9	Especimen: FAAD5700
Cluster Mínimo No. 7: 0154AFFA	Especimen: FAAF1540
Cluster Mínimo No. 8: 0550AFFA	Especimen: FAAF0550

AUTOMATAS CELULARES LINEALES REVERSIBLES 4H con L=2, R=2	
Cluster Mínimo No. 1: 05AF05AF	Especimen: F5A0F5A0
Cluster Mínimo No. 2: 05AF05EB	Especimen: F5A0B5E0
Cluster Mínimo No. 3: 05AF0FA5	Especimen: F5A0A5F0

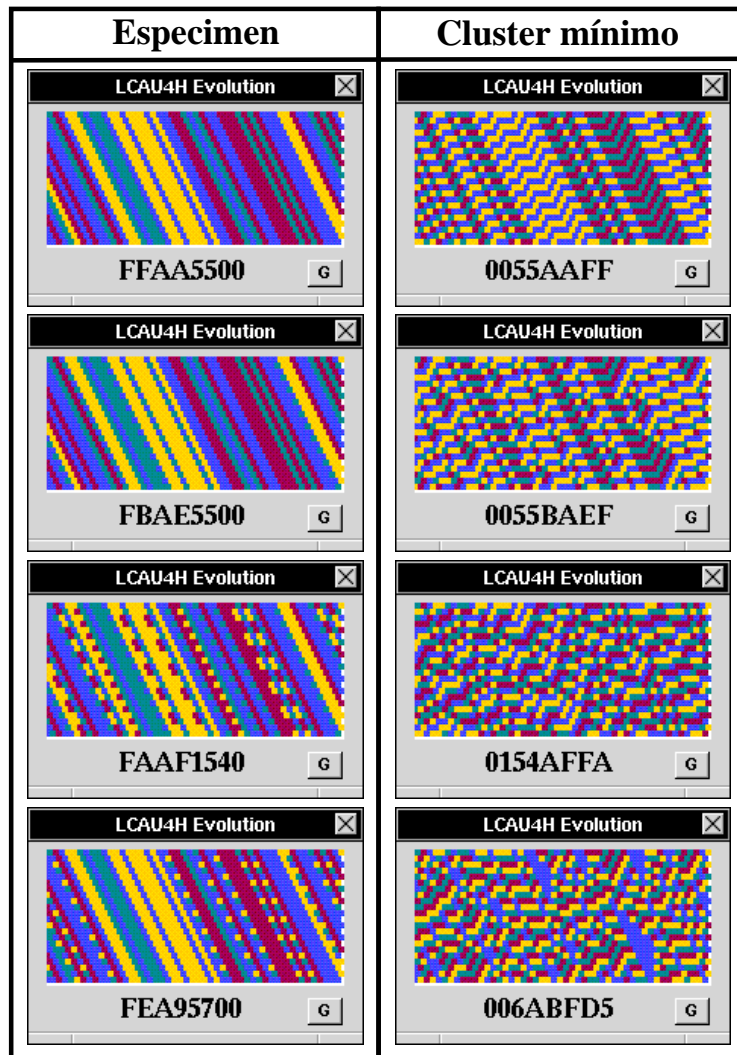


Figura 81: ACLR(4,h) con  $L = 1$  y  $R = 4$ .

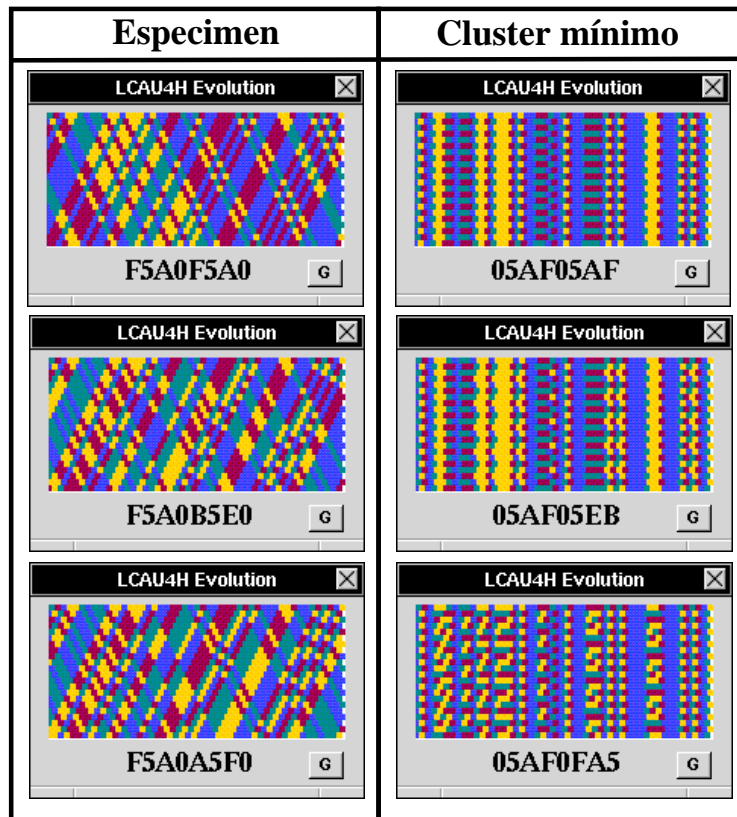


Figura 82: ACLR(4,h) con  $L = 2$  y  $R = 2$ .



## 6.2 Resultados para los Automatas Celulares Lineales Reversibles (5,h)

AUTOMATAS CELULARES LINEALES REVERSIBLES 5H con L=1, R=5	
Cluster Mínimo No. 1: 000667CCIIJOO	Especimen: 4OOIHC66500
Cluster Mínimo No. 2: 000667CCINJNO	Especimen: 4NOINHCC66500
Cluster Mínimo No. 3: 000667CCIOONI	Especimen: 4NJIOHCC66500
Cluster Mínimo No. 4: 000667CCIJOOI	Especimen: 4NIIOMCC66500
Cluster Mínimo No. 5: 000667CDJIEJO	Especimen: 4OJDJICC66500
Cluster Mínimo No. 6: 000667CDOIDJO	Especimen: 4OIDJNCC66500
Cluster Mínimo No. 7: 000667HCJOODI	Especimen: 4ONIHMC66500
Cluster Mínimo No. 8: 000667HCJJOEI	Especimen: 4NNIMMCH66500
Cluster Mínimo No. 9: 000667HCJJIEO	Especimen: 4OJIHHC66500
Cluster Mínimo No. 10: 000667HCJNJDO	Especimen: 4NJIMHMC66500
Cluster Mínimo No. 11: 000667DIJCEJO	Especimen: 4OIIHMC66500
Cluster Mínimo No. 12: 000667DDJOOHH	Especimen: 4NIIMMCM66500
Cluster Mínimo No. 13: 000667DIJEOJC	Especimen: 4OIDHNCM66500
Cluster Mínimo No. 14: 000667DJJOJHC	Especimen: 4ODDJNCH66500
Cluster Mínimo No. 15: 000667HEJJOEDI	Especimen: 4ODDINCM66500
Cluster Mínimo No. 16: 000667DIIOOMC	Especimen: 4OIIHCCO66500
Cluster Mínimo No. 17: 000667DDIIOOMH	Especimen: 4NIIMCCO66500
Cluster Mínimo No. 18: 000667JOJCEDI	Especimen: 4ODIJCCJ66500
Cluster Mínimo No. 19: 000667JEMIIMM	Especimen: 4NDIOCCJ66500
Cluster Mínimo No. 20: 00066CCBJNJIO	Especimen: 4NJIOGCCB6500
Cluster Mínimo No. 21: 00066CCBJOOII	Especimen: 4ONILCCB6500
Cluster Mínimo No. 22: 00066CCBJJOOI	Especimen: 4NIIOLCCB6500
Cluster Mínimo No. 23: 00066CCGJOOID	Especimen: 4ENILMCB6500
Cluster Mínimo No. 24: 00066CHBMIIOO	Especimen: 4ONHLC6500
Cluster Mínimo No. 25: 00066CJJMC8NN	Especimen: 4NJIOBCDB6500
Cluster Mínimo No. 26: 00066CIIMC8OO	Especimen: 4ONIIBC6500
Cluster Mínimo No. 27: 00066CJJHC9NN	Especimen: 4NIIOBC6500
Cluster Mínimo No. 28: 00066DCBNIHOO	Especimen: 4OMIILCDB6500
Cluster Mínimo No. 29: 00066DCBONHIO	Especimen: 4NHIOLCDB6500
Cluster Mínimo No. 30: 00066DCBNJMOI	Especimen: 4NHIOGC6500
Cluster Mínimo No. 31: 00066CIIHM9MO	Especimen: 4EOIIBMDB6500
Cluster Mínimo No. 32: 00066CIIMM8MO	Especimen: 4ENIIBMEB6500
Cluster Mínimo No. 33: 00066DCLNIHOE	Especimen: 4EMIILMDB6500
Cluster Mínimo No. 34: 00066CDINC8OO	Especimen: 4ONIHBCJB6500
Cluster Mínimo No. 35: 00066DEOOC7II	Especimen: 4OMIHL6500

AUTOMATAS CELULARES LINEALES REVERSIBLES 5H con L=1, R=5 (continuación)	
Cluster Mínimo No. 36: 00066CCDNI8OO	Especimen: 4ONDILCCG6500
Cluster Mínimo No. 37: 00066DDGNCHOO	Especimen: 4ONDHLCHG6500
Cluster Mínimo No. 38: 00066DCDNI7OO	Especimen: 4ONDIBCEG6500
Cluster Mínimo No. 39: 00066CIGMCIOO	Especimen: 4OMDILCDG6500
Cluster Mínimo No. 40: 00066DELOCMII	Especimen: 4ONDHBCJG6500
Cluster Mínimo No. 41: 00066CDGNCIOO	Especimen: 4OMDHLCIG6500
Cluster Mínimo No. 42: 00066CHDMI8OO	Especimen: 4OMDHGCJG6500
Cluster Mínimo No. 43: 00067CC6JNJIO	Especimen: 4NJIOG7CC6500
Cluster Mínimo No. 44: 00067CC6JOOII	Especimen: 4ONIL7CC6500
Cluster Mínimo No. 45: 00067CC6IOONI	Especimen: 4NNINL7CC6500
Cluster Mínimo No. 46: 00067CC6IJOOI	Especimen: 4NIIOL7CC6500
Cluster Mínimo No. 47: 00067DC6ENJIO	Especimen: 4NJIOB7DC6500
Cluster Mínimo No. 48: 00067DC6EOOII	Especimen: 4ONIIB7EC6500
Cluster Mínimo No. 49: 00067DC6EJOJI	Especimen: 4NNINB7EC6500
Cluster Mínimo No. 50: 00067DC6DOONI	Especimen: 4OIIJB7EC6500
Cluster Mínimo No. 51: 00067DC6DJOOI	Especimen: 4NIIOB7EC6500
Cluster Mínimo No. 52: 00067CC8NI8OO	Especimen: 4ONDIL7CH6500
Cluster Mínimo No. 53: 00067JC6DIDOO	Especimen: 4ONDIB7EH6500
Cluster Mínimo No. 54: 00067HI6JOOCH	Especimen: 4OMDIL7DH6500
Cluster Mínimo No. 55: 00067DC9EO9II	Especimen: 4ON8ID7EC6500
Cluster Mínimo No. 56: 00067DC8NI7OO	Especimen: 4OM8IN7DC6500
Cluster Mínimo No. 57: 00067JC7DI8OO	Especimen: 4ON8IC7EH6500
Cluster Mínimo No. 58: 00067HIGJOO C7	Especimen: 4OM8IM7DH6500
Cluster Mínimo No. 59: 0007CC66JOOII	Especimen: 4ONIIMCB66A00
Cluster Mínimo No. 60: 00077DOOBB9II	Especimen: 4NNINMCB66A00
Cluster Mínimo No. 61: 00076DOONIH7B	Especimen: 4NIIOMCB66A00
Cluster Mínimo No. 62: 0007CC68NI8OO	Especimen: 4ONDINCB66A00
Cluster Mínimo No. 63: 00077DOODB98I	Especimen: 4OIDJNCB66A00
Cluster Mínimo No. 64: 00077DO9NIH7L	Especimen: 4NNIMMCG66A00
Cluster Mínimo No. 65: 00077DOOBG9HI	Especimen: 4OIIHMCL66A00
Cluster Mínimo No. 66: 00076DOENIH7L	Especimen: 4NIIMMCL66A00
Cluster Mínimo No. 67: 0007CCG8LI8OO	Especimen: 4ONDHNCG66A00
Cluster Mínimo No. 68: 00077DOODG97I	Especimen: 4OIDHNCL66A00
Cluster Mínimo No. 69: 00077IOOND C86	Especimen: 4ODDJNCG66A00
Cluster Mínimo No. 70: 0007CCG9JO98I	Especimen: 4ODDINCL66A00

AUTOMATAS CELULARES LINEALES REVERSIBLES 5H con L=1, R=5 (continuación)	
Cluster Mínimo No. 71: 0007CCIIJOO66	Especimen: 4ONIICCB66K00
Cluster Mínimo No. 72: 00077DOODIJ76	Especimen: 4NNINCCB66K00
Cluster Mínimo No. 73: 00077CIIJOO76	Especimen: 4OIIJCCB66K00
Cluster Mínimo No. 74: 00076CIIJOO7B	Especimen: 4NIIOCCB66K00
Cluster Mínimo No. 75: 00077CIIGOOM6	Especimen: 4OIDJDCB66K00
Cluster Mínimo No. 76: 0007CC8IJOOG6	Especimen: 4OMDINCB66F00
Cluster Mínimo No. 77: 00077HOONID76	Especimen: 4OHDJNCB66F00
Cluster Mínimo No. 78: 00077DO9DIJ7L	Especimen: 4NNIMCCG66K00
Cluster Mínimo No. 79: 00077CIIJEO96	Especimen: 4OIIHCCL66K00
Cluster Mínimo No. 80: 00076CIIJEO9B	Especimen: 4NIMCCL66K00
Cluster Mínimo No. 81: 0007CCJOJ698I	Especimen: 4ODIJCCG66K00
Cluster Mínimo No. 82: 00077JIGGBCOO	Especimen: 4NDIOCCG66K00
Cluster Mínimo No. 83: 00077JIHGB7OO	Especimen: 4NDINCCCL66K00
Cluster Mínimo No. 84: 00076CIDJOO7G	Especimen: 4OIDHDCL66K00
Cluster Mínimo No. 85: 00077HIGLBDOO	Especimen: 4ODDJDCG66K00
Cluster Mínimo No. 86: 00077CI8JOO7G	Especimen: 4ODDIDCL66K00
Cluster Mínimo No. 87: 0007CC88JOOGG	Especimen: 4OMDHNCG66F00
Cluster Mínimo No. 88: 0007CCJ9LILL	Especimen: 4OHDHNCL66F00
Cluster Mínimo No. 89: 00077DOOBDJI6	Especimen: 4OCDJNCG66F00
Cluster Mínimo No. 90: 00077DOOBIJH6	Especimen: 4OCDINCL66F00
Cluster Mínimo No. 91: 00076DJOOIH7B	Especimen: 4NIIOLCBB6A00
Cluster Mínimo No. 92: 00076CJOOII7B	Especimen: 4NIIOBCBB6K00
Cluster Mínimo No. 93: 00076CIJJOJ7B	Especimen: 4NHIOBCBB6F00
Cluster Mínimo No. 94: 0007CHLLNID99	Especimen: 4ED8IMMLG6A00
Cluster Mínimo No. 95: 0007CH99NIDLL	Especimen: 4ED8ICMLG6K00
Cluster Mínimo No. 96: 00076DJNOIM7B	Especimen: 4NIIOL7BC6A00
Cluster Mínimo No. 97: 00076CJNOIN7B	Especimen: 4NIIOB7BC6K00
Cluster Mínimo No. 98: 0007DJICEL86O	Especimen: 4NHIO67EC6F00
Cluster Mínimo No. 99: 0007DJICB9NO6	Especimen: 4NHIO67DC6K00
Cluster Mínimo No.100: 001662CDOIDJO	Especimen: 4OIDJNCC16600
Cluster Mínimo No.101: 001662CHIOOND	Especimen: 4EIIJMMC16600
Cluster Mínimo No.102: 001662HCHOONI	Especimen: 4OIIHMC16600
Cluster Mínimo No.103: 001662HEHOENI	Especimen: 4OIDHNCM16600
Cluster Mínimo No.104: 001662DIIMEMO	Especimen: 4NOIMCCI16600
Cluster Mínimo No.105: 001662DIIOOMC	Especimen: 4OIIHCCO16600

AUTOMATAS CELULARES LINEALES REVERSIBLES 5H con L=1, R=5 (continuación)	
Cluster Mínimo No.106: 001662DIIEOOC	Especimen: 4NIIMCCO16600
Cluster Mínimo No.107: 001662DDIOOMH	Especimen: 4NCIOMCI16600
Cluster Mínimo No.108: 001662DHOCIJO	Especimen: 4OIDHDCO16600
Cluster Mínimo No.109: 001672C7IJOOI	Especimen: 4NIIOM7C26600
Cluster Mínimo No.110: 001673JNOIMC7	Especimen: 49ODHD CI46600
Cluster Mínimo No.111: 0016C3NJNOHB7	Especimen: 46ODHICH4L600
Cluster Mínimo No.112: 0016C3ONNIMB7	Especimen: 46ODIDCD4L600
Cluster Mínimo No.113: 0016C3JOOIHB7	Especimen: 46OIHCCI4L600
Cluster Mínimo No.114: 0016C3JNOIMB7	Especimen: 46ODHD CI4L600
Cluster Mínimo No.115: 00177DJJBB4NN	Especimen: 4NIIOM7B26B00
Cluster Mínimo No.116: 00166CCAIJOOI	Especimen: 4NIIOLCC16700
Cluster Mínimo No.117: 00167CC5JJOJI	Especimen: 4OIIJL7C26700
Cluster Mínimo No.118: 00167CC5IJOOI	Especimen: 4NIIOL7C26700
Cluster Mínimo No.119: 0017C267IJOOI	Especimen: 4NIIOLCB16C00
Cluster Mínimo No.120: 001773JJBBENN	Especimen: 4NIIOL7B26C00
Cluster Mínimo No.121: 006652DIIOOMC	Especimen: 4OIIHCCO06650
Cluster Mínimo No.122: 006652DIIEOOC	Especimen: 4NIIMCCO06650
Cluster Mínimo No.123: 006652DIJEJJC	Especimen: 4NEIOCCI06650
Cluster Mínimo No.124: 006652DJIOJMC	Especimen: 4OIDHDCO06650
Cluster Mínimo No.125: 006652DDIOOMH	Especimen: 4OCDJNCI06650
Cluster Mínimo No.126: 0067AC6AIJOOI	Especimen: 4EC88HMO0G850
Cluster Mínimo No.127: 0066ADJJC74NN	Especimen: 4NIIOLCB06C50
Cluster Mínimo No.128: 00667CC0IJOOI	Especimen: 4NIIOL7C067A0
Cluster Mínimo No.129: 0067A26CIJOOI	Especimen: 4NIIOL7B06CA0
Cluster Mínimo No.130: 0076A8JJC74NN	Especimen: 4NIIOKCB56C50
Cluster Mínimo No.131: 007672C1IJOOI	Especimen: 4NIIOK7CA66A0
Cluster Mínimo No.132: 0077A76AIJOOI	Especimen: 4NIIOK7BA6BA0

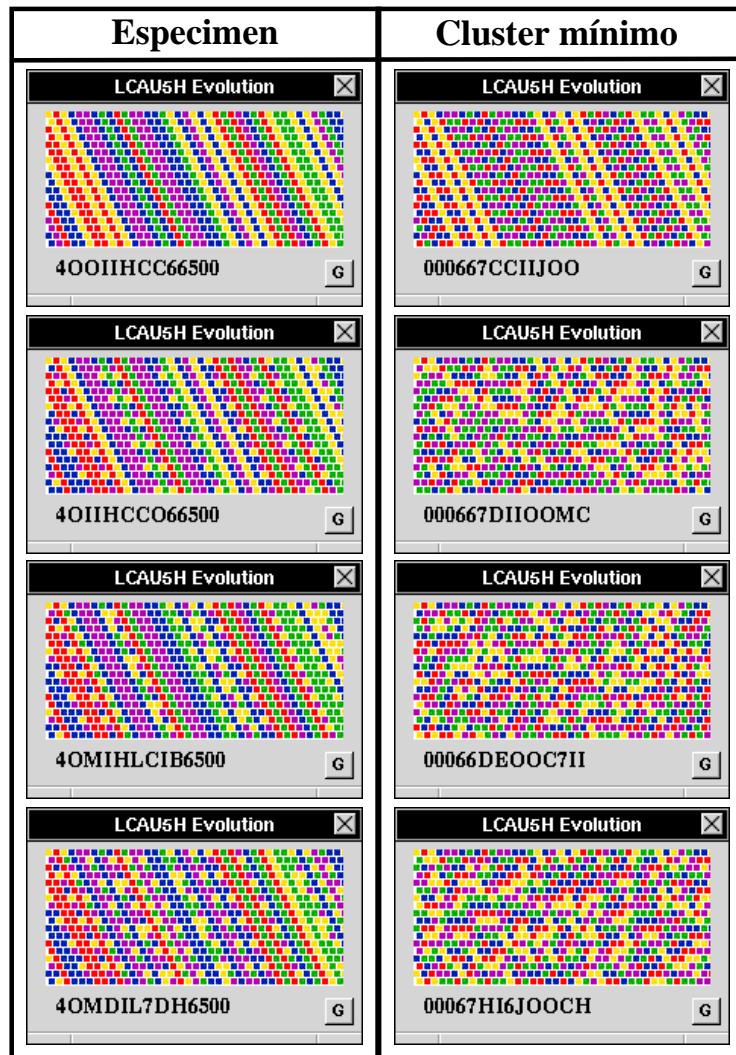


Figura 83: ACLR(5,h) con  $L = 1$  y  $R = 5$ .

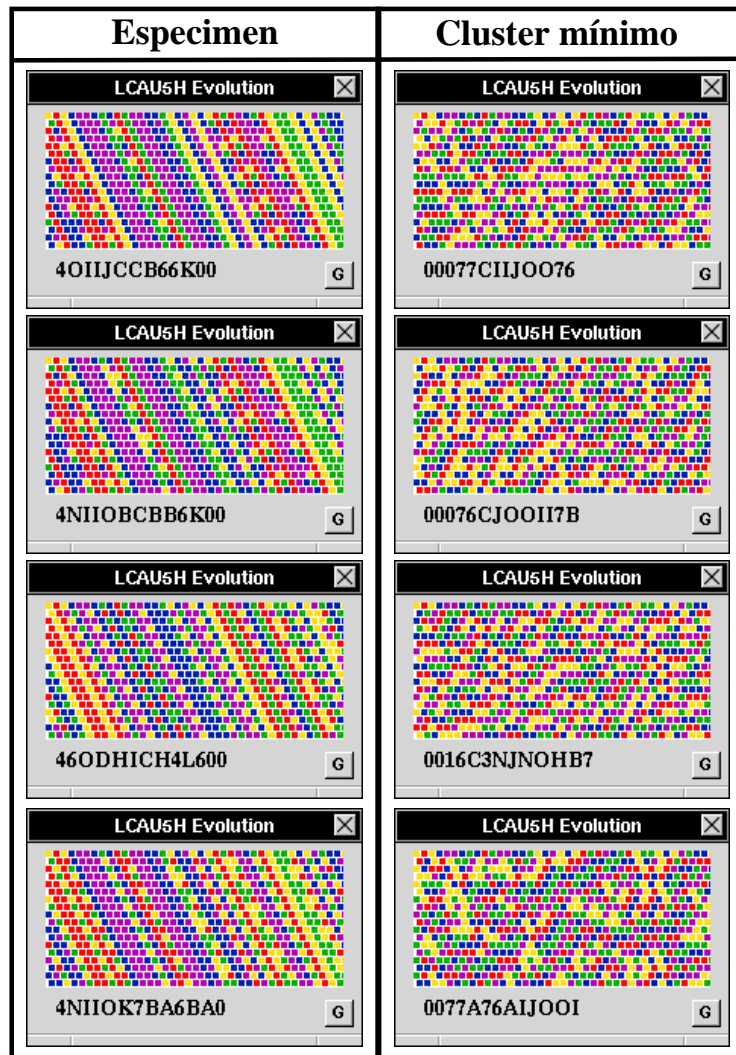


Figura 84: ACLR(5,h) con  $L = 1$  y  $R = 5$ .

### 6.3 Resultados para los Automatas Celulares Lineales Reversibles (6,h)

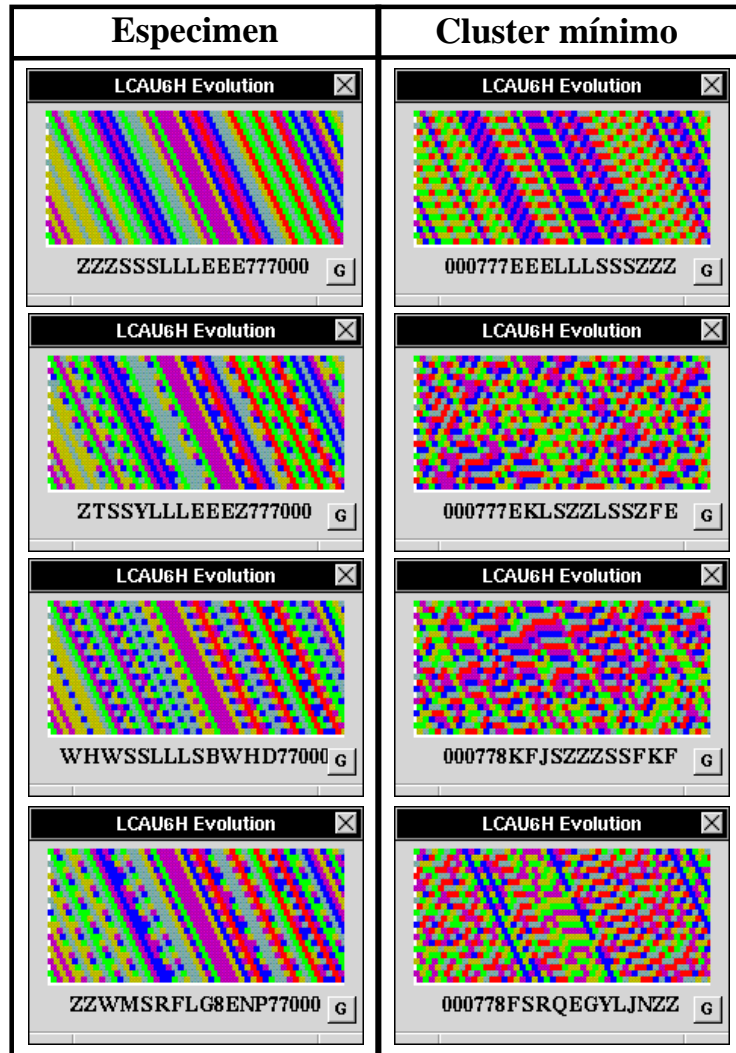


Figura 85: ACLR(6,h) con  $L = 1$  y  $R = 6$ .

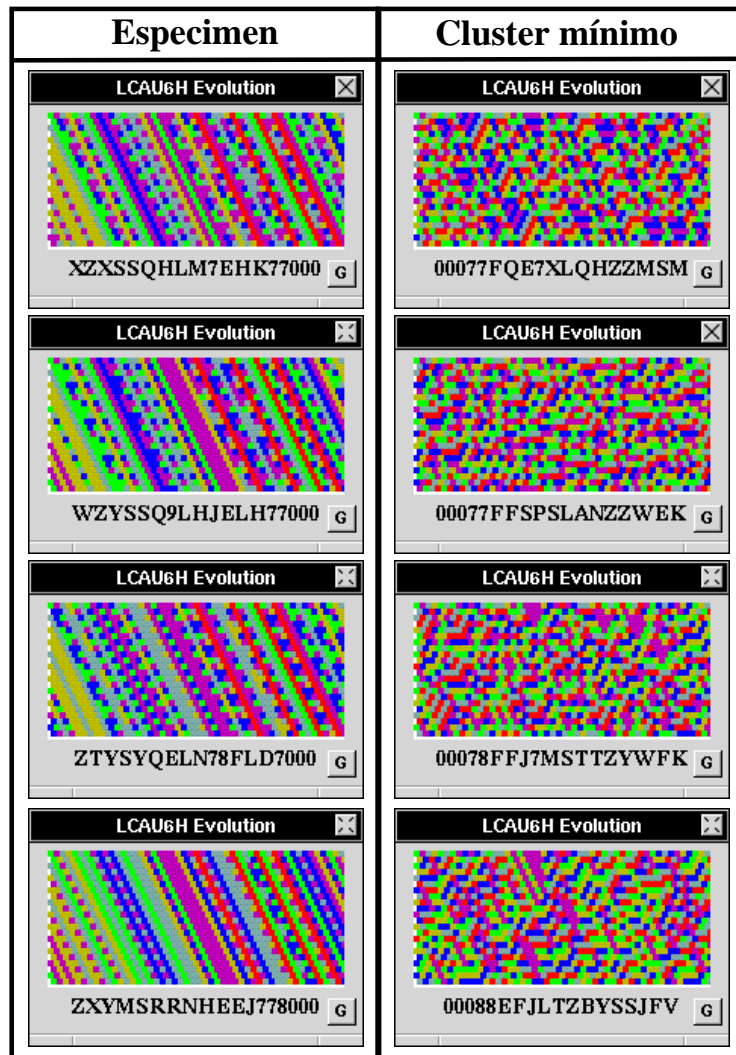


Figura 86: ACLR(6,h) con  $L = 1$  y  $R = 6$ .



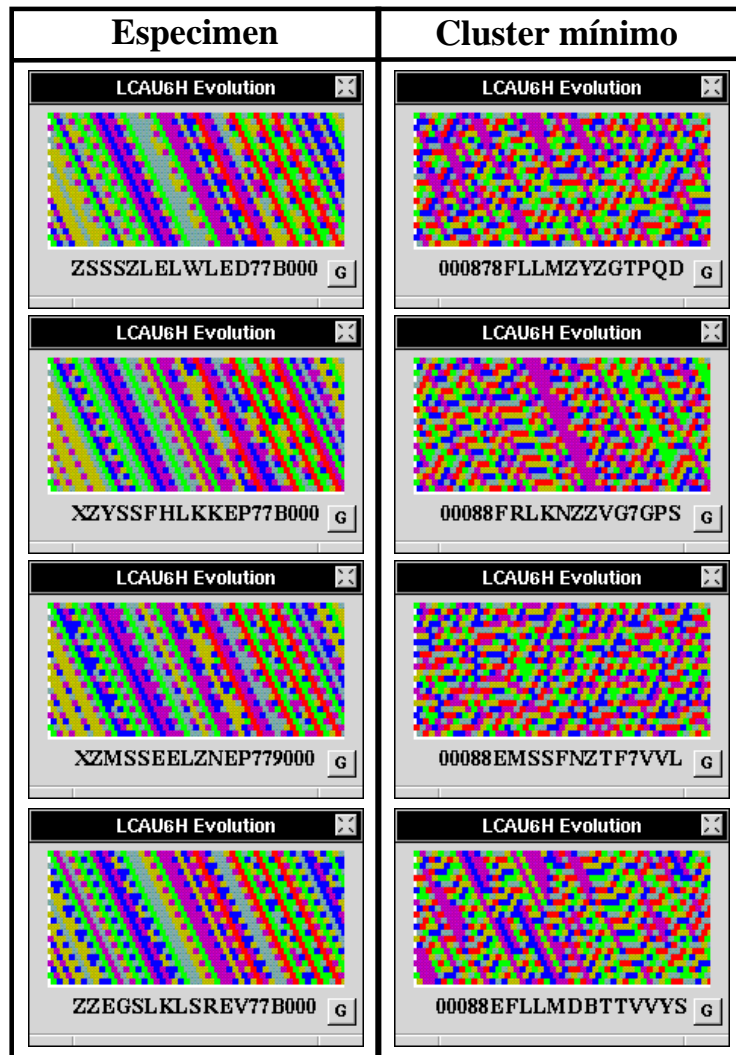


Figura 87: ACLR(6,h) con  $L = 1$  y  $R = 6$ .

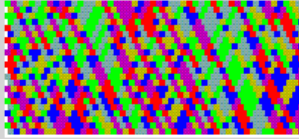
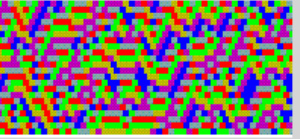
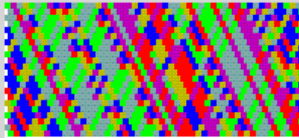
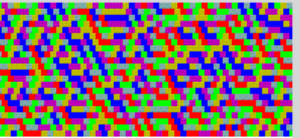
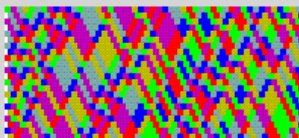
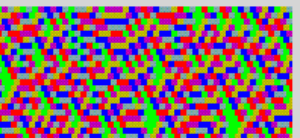
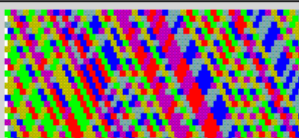
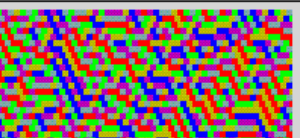
Especimen	Cluster mínimo
<p>LCAU6H Evolution <input type="checkbox"/></p>  <p>VTBS8ELI0SKKVTBF60 <input type="checkbox"/></p>	<p>LCAU6H Evolution <input type="checkbox"/></p>  <p>017FMSKWZKWZ31SCP7 <input type="checkbox"/></p>
<p>LCAU6H Evolution <input type="checkbox"/></p>  <p>VX7GESNI0SQEZ7JI0 <input type="checkbox"/></p>	<p>LCAU6H Evolution <input type="checkbox"/></p>  <p>017EFMMTZEFNMTY161 <input type="checkbox"/></p>
<p>LCAU6H Evolution <input type="checkbox"/></p>  <p>ZWESP7LI0MQEXV7TI0 <input type="checkbox"/></p>	<p>LCAU6H Evolution <input type="checkbox"/></p>  <p>017EFLSTZ0HZGP7Q3L <input type="checkbox"/></p>
<p>LCAU6H Evolution <input type="checkbox"/></p>  <p>WENAS7NI0WWKSP7NI0 <input type="checkbox"/></p>	<p>LCAU6H Evolution <input type="checkbox"/></p>  <p>017EFMNYTEFMMZT071 <input type="checkbox"/></p>

Figura 88: ACLR(6,h) con  $L = 2$  y  $R = 3$ .

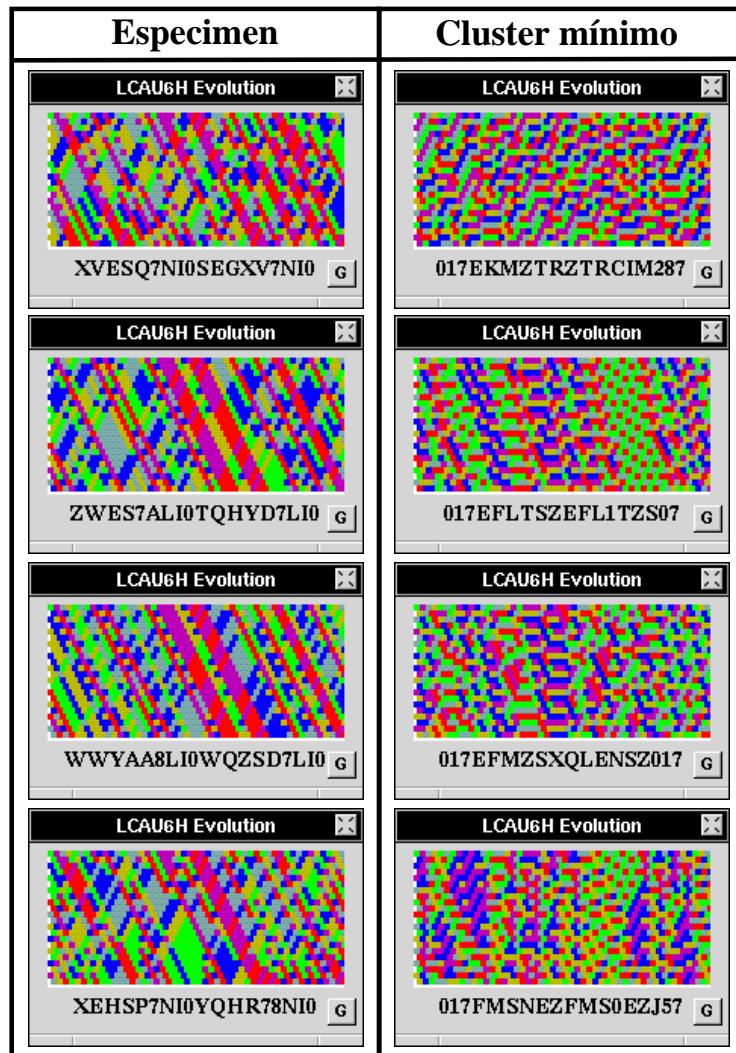


Figura 89: ACLR(6,h) con  $L = 2$  y  $R = 3$ .

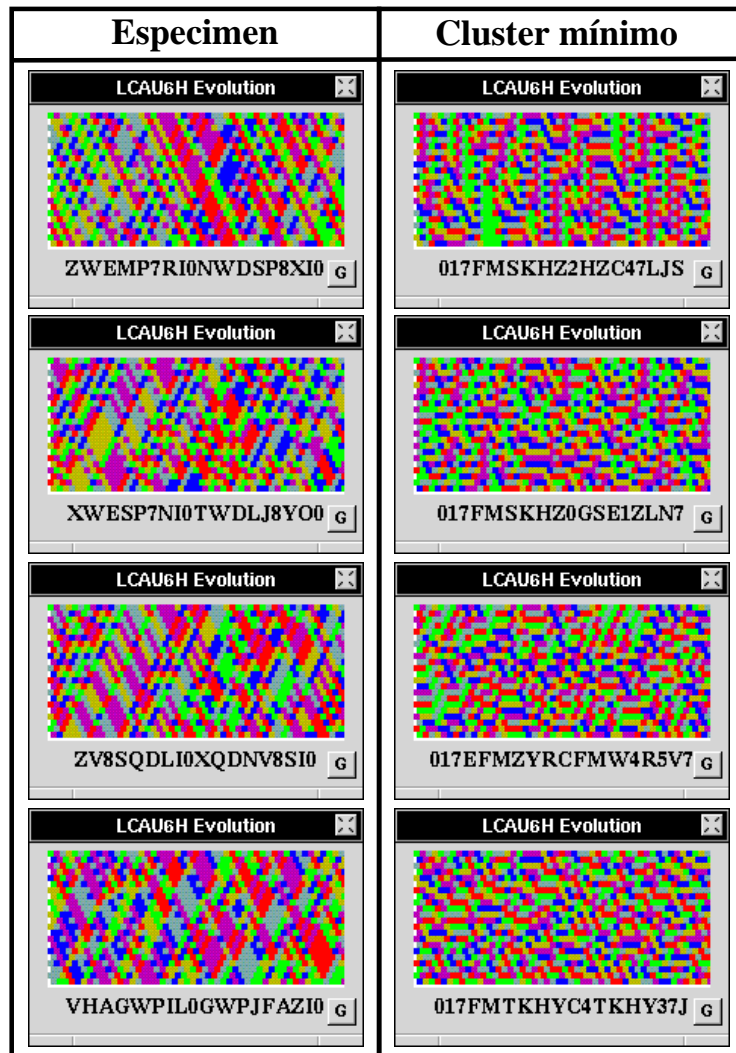


Figura 90: ACLR(6,h) con  $L = 2$  y  $R = 3$ .

## 6.4 Comparación de resultados con los otros métodos de cálculo de reversibles

El proceso aquí propuesto ha obtenido los mismos resultados comparado con el algoritmo de Hillman para los casos (3,h) y (4,h), en el caso del (5,h) Hillman obtuvo con su algoritmo de Hillman 385 clusters mínimos, sin embargo al parecer la forma de definir sus clusters fué diferente a la presentada en este escrito por lo que con una posterior revisión de los mismos que Hillman está haciendo actualmente se espera que concuerde con nuestros resultados.

Cabe señalar las grandes semejanzas que existen entre el algoritmo que Hillman desarrolló y el proceso que aquí se expone, ya que las tablas que Hillman produce en su proceso para los posibles ancestros de una cadena en realidad pueden ser vistas como las coordenadas donde aparecen 1's en la matriz de conectividad de la misma y la concatenación que hace él de tablas para obtener los ancestros de cadenas de mayor longitud es similar a multiplicar matrices de conectividad para obtener otras matrices de cadenas de mayor longitud.

Sin embargo la implementación del algoritmo de Hillman debe tomar en cuenta almacenar todas las tablas que se vayan formando conforme crezca la longitud de las cadenas a revisar y realizar comparaciones entre el grupo de tablas de ancestros para las cadenas de longitud  $n$  con las tablas para cadenas de longitud menor.

En el proceso presentado en este trabajo la recursión nos permite no tener que guardar las matrices de conectividad conforme se vayan generando (esta tarea se deja al compilador) y no se tienen que realizar comparaciones de una matriz de conectividad con las demás de cadenas de longitud menor, sino que usamos el criterio de los índices de Welch para saber si las matrices de conectividad manifiestan un mapeo global reversible, de hecho la implementación de Hillman podría mejorarse agregando estas mismas condiciones y evitando hacer comparaciones entre tablas, por ejemplo, Hillman dice que cada tabla debe tener el mismo número de elementos que el resto, ésto no es más que aplicar el concepto de multiplicidad uniforme, la comparación que hace entre tablas podría evitarse si mejor en cada tabla checara que una columna tuviera  $L$  elementos distintos y en la otra  $R$  elementos distintos, la condición que establece que solo una pareja debe ser formada por elementos iguales es revisar que solo exista un único ciclo permitido para generar la cadena; así se tendría que para cada tabla  $LR = k^{2r}$ , por lo que  $M = 1$ .

De esta manera mientras que el algoritmo de Hillman tarda en calcular las reglas reversibles de un ACL(5,h) más de 13 horas, la implementación que se expone aquí con una programación muy simple tarda menos de 5 minutos, y se espera todavía mejorar este tiempo.

En el caso del algoritmo que usa índices de Welch, el tiempo de cálculo no es muy diferente, sin embargo este proceso solo contemplaba ACLR donde un renglón estuviera formada por elementos todos iguales entre sí, cosa que no siempre sucede para los reversibles.

## 7 Conclusiones

### 7.1 Perspectivas para mejorar el cómputo de los Autómatas Celulares Lineales Reversibles

El trabajo desarrollado por Hedlund y Nasu nos han dado una caracterización completa acerca del comportamiento y las cualidades que debe mostrar la reglas de evolución para inducir un mapeo global reversible no importando cuantas células tenga el anillo donde se implementa el ACL, sin embargo aún quedan varias cuestiones por investigar, por ejemplo el algoritmo propuesto empieza a checar las matrices de conectividad para cadenas de longitud 1 en adelante, pero si el ACL es reversible, en que momento se manifestará el principio de combinabilidad que Nasu señala, es decir, para qué longitud de cadena las matrices de conectividad serán debilmente combinables; si tuvieramos la respuesta a esta cuestión podríamos checar solo las matrices de conectividad para las cadenas de dicha longitud y ahorrarnos las demás observaciones, con lo que el tiempo de cálculo de reversibles sería menor.

Otro aspecto a tratar es el tiempo de cálculo del cluster mínimo, esta cuestión aunque es puramente operativa en la implementación del proceso es importante ya que mientras más estados y/o más grande sea la vecindad en el ACL, más tiempo se consumirá en calcular el cluster mínimo; ya que este cálculo se basa en hacer permutaciones por renglones, columnas y estados, tenemos por ejemplo que para un (4,h) tenemos  $4!$  de posibilidades para estas permutaciones, para un (5,h) tenemos  $5!$  y así sucesivamente; por lo que se debe encontrar métodos más eficientes para agilizar el cálculo de estos clusters mínimos.

También tenemos que el proceso genera todas las posibles matrices de ciertas características y de éstas selecciona aquellas que cumplan con ser debilmente combinables, el problema con esta aproximación es que conforme aumenta el número de estados y el tamaño de vecindad, las posibles matrices a generar aumentan exponencialmente; pero por qué no inducir la generación directa de ACLR; parece ser que los ACLR(k,r) estan formados por subautomatas reversibles, es decir, que un ACLR(6,h) puede ser una extensión de un ACLR(5,h) o de un ACLR(4,h) junto con un ACLR(2,h) y así sucesivamente, si esta idea es correcta, el ACLR(2,h) podría ser la base para generar todos los demás ACLR para un mayor número de estados y de este modo no se tendrían que generar todas las posibles matrices de evolución para filtrar de éstas a los ACLR, sino que se generarían directamente matrices de evolución que fueran extensiones de un ACLR con menor número de estados, con lo que el número de matrices a generar se podría reducir dramáticamente.

Otra alternativa interesante es aplicar la teoría de matrices para la detección de ACLR, ya sea haciendo uso de eigenvalores y eigenvectores, ver como se comportan éstos en las matrices de conectividad de un ACLR así como la utilización de productos cartesianos; por otra parte el estudio aquí presentado detecta los valores de  $L$  y  $R$ , verifica si  $L * R = k^{2r}$  y en caso afirmativo se induce que  $M = 1$ , pero por qué no plantear un proceso el cual leyera directamente el valor de  $M$ , esto tendría la ventaja que si se cumple que  $M = 1$  para cada posible cadena de cierta longitud, se obtendría simultaneamente la regla inversa de tal ACLR; otra herramienta que sería muy útil desarrollar es un diagrama de Welch en el cual se pudiera leer directamente si existe un único estado en el cual sus valores de  $L$  y  $R$  sean los únicos

ancestros para una secuencia de estados dada.

Como se observa aún queda mucho por hacer para realizar el cálculo de todas las posibles reglas reversibles para un cierto  $ACL(k,r)$  sobre todo para estudiar casos donde  $k$  y  $r$  sean mucho mayores y ofrezcan por lo tanto una gama de comportamientos más complejos e interesantes.

## 7.2 Futuro de las aplicaciones de los Autómatas Celulares Reversibles

Esta última parte puede resultar algo tentativa, pues en realidad actualmente no existen muchas aplicaciones para los AC, los problemas que se han atacado por este medio han sido casos muy específicos y particulares en donde se ha observado que el comportamiento local de las partes es fundamental en el funcionamiento del sistema. Sin embargo, se pueden observar dos campos principales donde se pueden utilizar ACLR para realizar una tarea específica, la codificación de información y la encriptación de datos, en ambas aplicaciones se debe aplicar un proceso el cual transforme la información original ya sea para su almacenamiento seguro en dispositivos magnéticos u ópticos o para ocultarla de manera rápida y eficiente. Por ejemplo, en cuestión de codificación de datos para su almacenamiento en discos duros se busca que estos se graben de tal manera que cambios en los mismos provocados por causas no previstas (como cambios de corriente) no causen daños que no se puedan reparar, es por ésto que entre muchas cosas se busca almacenar la información de manera que no existan largas cadenas de bits con un mismo valor, pero a la vez esta información se debe poder decodificar rápidamente para que los datos recuperen su forma original; es por esto que se podría pensar en la implementación de ACLR que cumplieran ciertas características (como evitar formar largas cadenas formadas por un mismo elemento) para realizar dicha tarea. En la cuestión de encriptación de datos, la implementación de ACLR resulta inmediata, y se puede obtener un grado de complejidad bastante interesante, por ejemplo, se podría utilizar simplemente un  $ACLR(2,1)$  para encriptar la información de un archivo binario lo cual resultaría muy trivial y poco seguro ya que existen pocos reversibles en el caso  $(2,1)$ , pero si agrupamos los bits en grupos de dos tendríamos que el archivo tendría cuatro tipos de elementos distintos y se podría implementar un  $ACLR(4,h)$  por ejemplo en donde existen once clusters mínimos cada uno con decenas de reglas reversibles distintas, siguiendo esta idea podríamos agrupar ahora la información en grupos de tres bits obteniendo ocho posibles estados, si se decidiera aplicar un  $ACLR(8,h)$  tendríamos millones de posibles reglas que inducieran un mapeo global reversible, aumentando el tamaño de vecindad crece también dramáticamente las posibilidades de obtener más ACLR para aplicarlos como encriptadores, además que su decodificación resulta sencilla teniendo la regla inversa, la que a su vez puede tener un tamaño de vecindad mayor a la regla original, lo que agrega un grado mayor de seguridad.

## 8 Apéndice A

A continuación se presenta el programa de computación que calcula los ACLR para el caso (4,h), cabe señalar que el mismo proceso puede ser adaptado para cualquier tipo de ACLR(k,r) pues la forma de detección es la misma para todos los casos.

```
#include "stdio.h"
#include "stdlib.h"
#include "ctype.h"
#include "string.h"
#include "math.h"

/***** DEFINICION DE VALORES *****/

/*Numero de estados del ACL */
#define numestados 4

/*Longitud de la regla de evolucion */
#define longregla 16

/*Longitud del numero wolfram */
#define longcadena 8

/* Valor de producto L*R */
#define indice 4

/*Numero de iteraciones a realizar para buscar el cluster minimo */
#define numit 6

/*Numero total de permutaciones */
#define numper 24

/*Indice izquierdo a buscar */
#define izquierdo 2

/*Indice derecho a buscar */
#define derecho 2

/***** VARIABLES GLOBALES QUE EL PROGRAMA UTILIZA *****/

/*Permutaciones posibles para buscar el cluster minimo*/
int permutaciones[numper][numestados]={
{0,1,2,3},{0,1,3,2},{0,2,1,3},{0,2,3,1},{0,3,1,2},{0,3,2,1},
{1,0,2,3},{1,0,3,2},{1,2,0,3},{1,2,3,0},{1,3,0,2},{1,3,2,0},
```



```

{2,0,1,3},{2,0,3,1},{2,1,0,3},{2,1,3,0},{2,3,0,1},{2,3,1,0},
{3,0,1,2},{3,0,2,1},{3,1,0,2},{3,1,2,0},{3,2,0,1},{3,2,1,0}};

/*Matriz de evolucion del ACL */
int matriz[numestados][numestados];

/*Matriz de las coordenadas de elementos del ACL */
int coordenadas[numestados][numestados-1];

/*Matriz de rutas en el ACL */
int rutas[numestados][numestados];

/*Condicion de llenado para cada estado */
int cond;

/* Matrices auxiliares para obtener la conectividad de cada estado y el cluster
minimo de un ACLR */
int matrizb[numestados][numestados];
int matrizc[numestados][numestados];
int matrizd[numestados][numestados];

/*Arreglo que recibe el numero wolfram del ACL */
char numwolfram[longcadena];

/*Arreglo que guarda la regla de evolucion del ACL */
char reglaevolucion[longregla];

/*Valor del tamao de vecindad de la regla inversa si el ACL es reversible */
int maxvecindad=2;

/*Guarda la permutacion a realizar en la matriz de evolucion para buscar el cluster
minimo*/
int permutacion[numestados];

/*Guarda los cambios de estados a realizar en la matriz de evolucion para bus-
car el cluster minimo */
int estados[numestados];

/*Cadenas que almacenaran numeros wolfram para encontrar el cluster minimo
*/
char nw[longcadena];
char clustemp[longcadena];
char final[longcadena];

/*Guarda cuantos estados hay de un mismo valor por renglon o columna */
int estren,estcol;

```

```

/*Guarda que estados y que renglones se van a permutar */
int renglon[numestados];

/*Cuenta cuantas permutaciones tienen que hacerse para encontrar el cluster mínimo */
int numedos;

/*Numero de clusters mínimos encontrados y archivo para guardarlos */
int totalclusters;
FILE *hoja;
int clusters[20000][longcadena];

/*Total de ACLR generados por el proceso */
int total;

/*Variables para visualizar el progreso del cálculo */
int porcentaje=0;
int valor;

/*Valores de los índices de Welch */
int R,L;

/*****DECLARACION DE LAS FUNCIONES DEL PROGRAMA*****/

void imprimiravance();
void inicio();
void borrar(int edo);
int lugar(int posicion);
void inicializar(int edo);
void llenado(int edo);
int incrementar(int edo,int posicion);
int actualizar(int edo);
int checarciclos(int edo);
void conectividad(int edo, int aux[numestados][numestados]);
void multiplo(int matrizb[numestados][numestados],
int matrizc[numestados][numestados]);
int obtenerindices(int matriz[numestados][numestados],int edo,int largo);
int checarindices(int matriz[numestados][numestados], int largo);
int reversible();
void wolfram(int matriz[numestados][numestados]);
void contarestados();
void encontrar(int matriz[numestados][numestados]);
void asignar(int ind, int array[numestados]);
void permutar(int array[numestados],int matriz[numestados][numestados]);

```

```

void cambioest();
void menor();
void cluster();
int lista();
void calculo(int edo);

/***** FUNCIONES DEL PROGRAMA *****/

/*Presentar resultados en pantalla */
void imprimiravance()
{
    int i,j;
    printf("\n");
    for(i=0;i<numestados;i++)
    {
        for(j=0;j<numestados;j++)
            printf("%3d",matriz[i][j]);
        printf("\n");
    }
    printf("\nACLR(4,h): %s",nw);
    printf("\nCluster del ACLR(4,h): %s",final);
    printf("\nTotal de clusters minimos encontrados: %2d \n",totalclusters);
}

/*Inicializar matriz de evolucion y sus coordenadas*/
void inicio()
{
    int i,j;
    for(i=0;i<numestados;i++)
        for(j=0;j<numestados;j++)
            matriz[i][j]=-1;
    for(i=0;i<numestados;i++)
        matriz[i][i]=i;
    for(i=0;i<numestados-1;i++)
        coordenadas[0][i]=i+1;
}

/* Borrar elementos de la matriz de evolucion */
void borrar(int edo)
{
    int i,j;
    int v1;
    for(i=0;i<numestados;i++)
        for(j=0;j<numestados;j++)
            if(i!=j)

```

```

        if(matriz[i][j]==edo)
            matriz[i][j]=-1;
    }

/*Checa si el lugar a ocupar en la matriz de evolucion esta disponible */
int lugar(int posicion)
{
    int t1,t2;
    t1=posicion/numestados;
    t2=posicion-(t1*numestados);
    return(matriz[t1][t2]);
}

/*Inicializa las coordenadas del estado siguiente si un estado paso las pruebas anteriores */
void inicializar(int edo)
{
    int i,j;
    int t1,t2;
    for(i=0;i<numestados-1;i++)
        for(j=0;j<(numestados*numestados);j++)
            if(lugar(j)==-1)
                {
                    t1=j/numestados;
                    t2=j-(t1*numestados);
                    matriz[t1][t2]=edo;
                    coordenadas[edo][i]=j;
                    break;
                }
}

/*Pone en la matriz de evolucion los elementos del estado segun la matriz de coordenadas */
void llenado(int edo)
{
    int i,t1,t2;
    for(i=0;i<numestados-1;i++)
        {
            t1=coordenadas[edo][i]/numestados;
            t2=coordenadas[edo][i]-(t1*numestados);
            matriz[t1][t2]=edo;
        }
}

/*Funcion que escoge que coordenada actualizar*/
int incrementar(int edo,int posicion)
{

```

```

int flag;
int valor,valor2;
int i;
int t1,t2;
for(i=posicion;i=numestados-2;i++)
{
    flag=0;
    if(i<posicion)
        coordenadas[edo][i]=valor2;
    while(flag==0)
    {
        valor=(++coordenadas[edo][i]);
        if(valor<=(numestados*numestados))
        {
            borrar(edo);
            return -1;
        }
        t1=valor/numestados;
        t2=valor-(t1*numestados);
        if(matriz[t1][t2]==-1)
        {
            matriz[t1][t2]=edo;
            flag=1;
        }
    }
    valor2=valor;
}
borrar(edo);
return 0;
}

/* Actualiza las coordenadas del estado */
int actualizar(int edo)
{
    int i;
    for(i=numestados-2;i<=0;i--)
        if(incrementar(edo,i)==0)
            break;
    if(i==-1)
        return -1;
    return 0;
}

/* Checar si hay ciclos de longitud 2 en la matriz de evolucion */
int checarciclos(int edo)
{

```

```

int i,j,k;
int ciclo;
int v1,v2,p;
/*Obtiene la localizacion de el edo en la matriz y maximiza esta posicion */
p=0;
for(i=0;i<numestados;i++)
  for(j=0;j<numestados;j++)
    if(matriz[i][j]==edo)
      {
        rutas[edo][p]=((i*i)+(j*j));
        p++;
      }
/*Compara si en el mismo estado hay ciclos */
for(i=0;i<numestados;i++)
{
  v1=rutas[edo][i];
  if(i<0)
    for(j=0;j<i;j++)
      {
        v2=rutas[edo][j];
        if(v2==v1)
          return -1;
      }
}
/*controla estados anteriores */
for(i=0;i<edo;i++)
{
  ciclo=0;
  for(j=0;j<numestados;j++)
  {
    for(k=0;k< numestados;k++)
    {
      v1=rutas[i][k];
      v2=rutas[edo][j];
      if(v1==v2)
      {
        ciclo++;
        if(ciclo>1)
          return -1;
        break;
      }
    }
  }
}
return 0;
}

```

```

/*Obtiene la matriz de conectividad del ACL de un edo dado y la guarda en aux*/
void conectividad(int edo, int aux[numestados][numestados])
{
    int i,j;
    for(i=0;i<numestados;i++)
        for(j=0;j<numestados;j++)
            aux[i][j]=0;
    for(i=0;i<numestados;i++)
        for(j=0;j<numestados;j++)
            if(matriz[i][j]==edo)
                aux[i][j]=1;
}

/*Multiplica dos matrices dadas y guarda el resultado en matrizd*/
void multiplo(int matrizb[numestados][numestados], int matrizc[numestados][numestados])
{
    int i,j,k;
    int temp;
    for(i=0;i<numestados;i++)
        for(j=0;j<numestados;j++)
            {
                temp=0;
                for(k=0;k<numestados;k++)
                    temp+=matrizb[i][k]*matrizc[k][j];
                matrizd[i][j]=temp;
            }
}

/*Obtiene el valor de los indices de Welch para ver si el ACL es reversible*/
int obtenerindices(int matriz[numestados][numestados],int edo)
{
    int i,j,k;
    int valor;
    int diagonal;
    int matrizaux[numestados][numestados];
    /*Copia la matriz de conectividad que recibe */
    for(i=0;i<numestados;i++)
        for(j=0;j<numestados;j++)
            matrizaux[i][j]=matriz[i][j];
    /*Multiplica la matriz de conectividad que recibe por la matriz de conectividad
de el estado 0 */
    conectividad(edo,matrizc);
    multiplo(matrizaux,matrizc);
    /*Si la suma de elementos es diferente que el valor del "indice", no se cumple con

```

```

la multiplicidad uniforme */
valor=0;
for(j=0;j;numestados;j++)
  for(k=0;k;numestados;k++)
    valor+=matrizd[j][k];
if(valor!=indice)
{
  /* No se cumple la Multiplicidad Uniforme */
  return -1;
}
/*Para que el ACL sea reversible, debe haber un solo elemento en la diagonal
principal y  $L^*R=k\hat{2}r$  */
diagonal=0;
for(j=0;j;numestados;j++)
  if(matrizd[j][j]!=0)
    diagonal++;
if(diagonal!=1)
{
  /*Multiples Ancestros */
  return -1;
}
/*Obtiene cuantos renglones cumplen con el valor del indice derecho*/
R=0;
for(j=0;j;numestados;j++)
{
  valor=0;
  for(k=0;k;numestados;k++)
    if(matrizd[j][k]!=0)
      valor++;
  if(valor==derecho)
    R++;
}
/*Obtiene cuantas columnas cumplen con el valor del indice izquierdo*/
L=0;
for(j=0;j;numestados;j++)
{
  valor=0;
  for(k=0;k;numestados;k++)
    if(matrizd[k][j]!=0)
      valor++;
  if(valor==izquierdo)
    L++;
}
/*Checa si los indices cumplen con  $L^*R=k\hat{2}r$ */
valor=0;
if(((R*L)==indice)&&(diagonal==1))

```



```

    valor=1;
    if(valor==0)
    {
        valor=(obtenerindices(matrizd,edo));
        if(valor==-1)
            return -1;
    }
    return 0;
}

/*Recibe una matriz de conectividad, la multiplica las matrices de conectividad
de cada estado y verifica si  $L \cdot R = k \hat{2}r$ , en este caso, guarda en maxvecindad el
numero de arcos de la ruta para saber el valor de  $2r+1$  en la regla inversa */
int checarindices(int matriz[numestados][numestados])
{
    int i,j,k;
    int valor;
    int diagonal;
    int matrizaux[numestados][numestados];
    /*Copia la matriz de conectividad que recibe */
    for(i=0;i<numestados;i++)
        for(j=0;j<numestados;j++)
            matrizaux[i][j]=matriz[i][j];
    /*Multiplica la matriz de conectividad que recibe por cada matriz de conectividad
de los demas estados */
    for(i=0;i<numestados;i++)
    {
        conectividad(i,matriz);
        multiplo(matrizaux,matriz);
        /*Si la suma de elementos es diferente que el valor del "indice", no se cumple
con la multiplicidad uniforme */
        valor=0;
        for(j=0;j<numestados;j++)
            for(k=0;k<numestados;k++)
                valor+=matrizd[j][k];
        if(valor!=indice)
        {
            /*No se cumple la Multiplicidad Uniforme*/
            return -1;
        }
    }
    /*Para que el ACL sea reversible, debe haber un solo elemento en la diagonal
principal y  $L \cdot R = k \hat{2}r$  */
    diagonal=0;
    for(j=0;j<numestados;j++)
        if(matrizd[j][j]!=0)
            diagonal++;
}

```

```

if(diagonal;1)
{
  /*Multiples Ancestros*/
  return -1;
}
/*Obtiene cuantos renglones cumplen con el valor del indice derecho*/
R=0;
for(j=0;j<numestados;j++)
{
  valor=0;
  for(k=0;k<numestados;k++)
    if(matrizd[j][k]!=0)
      valor++;
  if(valor==derecho)
    R++;
}
/*Obtiene cuantas columnas cumplen con el valor del indice izquierdo*/
L=0;
for(j=0;j<numestados;j++)
{
  valor=0;
  for(k=0;k<numestados;k++)
    if(matrizd[k][j]!=0)
      valor++;
  if(valor==izquierdo)
    L++;
}
/*Checa si los indices cumplen con  $L \cdot R = k^2 r$ */
valor=0;
if(((R*L)==indice)&&(diagonal==1))
  valor=1;
/*Si no se cumple la condicion anterior se vuelve a hacer el proceso */
if(valor==0)
{
  valor=(checarindices(matrizd,vecindad+1));
  if(valor==-1)
    return -1;
}
}
return 0;
}

/*Checa si el automata es reversible*/
int reversible()
{
  int i;

```

```

for(i=0;i<numestados;i++)
{
    conectividad(i,matrizb);
    if((checharindices(matrizb,2))==-1)
        return -1;
}
return 0;
}

```

```

/*Calcula el numero wolfram de una matriz de evolucion */
void wolfram(int matriz[numestados][numestados])
{
    int i,j;
    int temp=0;
    for(i=0;i<numestados;i++)
        for(j=0;j<numestados;j++)
            reglaevolucion[(i*numestados)+j]=matriz[i][j];
    for(i=longcadena-1;i<=0;i--)
    {
        temp=(numestados*reglaevolucion[(i*2)+1])+reglaevolucion[(i*2)];
        if(temp<10)
            nw[longcadena-1-i]=toascii(48+temp);
        else
            nw[longcadena-1-i]=toascii(55+temp);
    }
}

```

```

/*Busca si es necesario trasponer la matriz para encontrar el cluster minimo */
void contarestados()
{
    int i,j;
    int temp;
    estren=estcol=0;
    for(i=0;i<numestados;i++)
    {
        temp=0;
        for(j=0;j<numestados;j++)
            if(matriz[i][i]==matriz[i][j])
                temp++;
        if(temp<estren)
            estren=temp;
    }
    for(i=0;i<numestados;i++)
    {

```

```

temp=0;
for(j=0;j<numestados;j++)
    if(matriz[i][j]==matriz[j][i])
        temp++;
if(temp!=estcol)
    estcol=temp;
}
}

```

/\*Encuentra que renglones tienen mas elementos de un mismo estado y guarda en arreglo renglon esta informacion \*/

```
void encontrar(int matriz[numestados][numestados])
```

```

{
    int i,j;
    int temp,temp2,aux;
    temp2=aux=0;
    numedos=1;
    renglon[aux]=0;
    /*Encuentra que renglon tiene mas elementos de un mismo estado */
    for(i=0;i<numestados;i++)
    {
        temp=0;
        for(j=0;j<numestados;j++)
            if(matriz[i][j]==matriz[j][i])
                temp++;
        if(temp>temp2)
        {
            temp2=temp;
            renglon[aux]=i;
        }
    }
    /*Encuentra si otros renglones tienen el mismo numero de elementos de un mismo estado */
    for(i=0;i<numestados;i++)
    {
        temp=0;
        if(i!=renglon[aux])
        {
            for(j=0;j<numestados;j++)
                if(matriz[i][j]==matriz[j][i])
                    temp++;
            if(temp==temp2)
            {
                renglon[numedos]=i;
                numedos++;
            }
        }
    }
}

```

```

    }
  }
}

```

```

/*Asigna a un arreglo la permutacion de la matriz permutaciones */
void asignar(int ind, int array[numestados])
{
  int i;
  for(i=0;i<numestados;i++)
    array[i]=permutaciones[ind][i];
}

```

```

/*Permuta los renglones y las columnas de la matriz de evolucion y guarda esta en
matrizb */
void permutar(int array[numestados],int matriz[numestados][numestados])
{
  int i,j;
  for(i=0;i<numestados;i++)
    for(j=0;j<numestados;j++)
      matriz[i][j]=matriz[array[(numestados-1)-i]][array[(numestados-1)-j]];
}

```

```

/*Cambia los estados de la matriz de evolucion */
void cambioest()
{
  int i,j,k;
  int contador;
  int temp;
  contador=0;
  for(i=numestados-1;i>=0;i--)
  {
    for(j=numestados-1;j>=0;j--)
    {
      temp=matriz[i][j];
      for(k=0;k<contador;k++)
        if(estados[k]==temp)
          break;
      if(k==contador)
      {
        estados[contador]=temp;
        contador++;
      }
    }
  }
}

```

```

        if(contador==numestados)
            break;
    }
    for(i=0;i<numestados;i++)
        for(j=0;j<numestados;j++)
            for(k=0;k<numestados;k++)
                if(matrizc[j][k]==estados[i])
                    matrizd[j][k]=i;
}

/*Compara dos numeros wolfram y conserva el menor lexicograficamente */
void menor()
{
    int temp;
    int i,j;
    temp=strcmp(clustemp,nw);
    if(temp<0)
        strcpy(clustemp,nw);
}

/*Obtiene el cluster minimo de una matriz de evolucion */
void cluster()
{
    int i,j;
    contarestados();
    if(estren!=estcol)
    {
        wolfram(matriz);
        strcpy(clustemp,nw);
        encontrar(matriz);
        for(i=0;i<numedos;i++)
            for(j=0;j<numit;j++)
            {
                asignar((renglon[i]*numit)+j,permutacion);
                permutar(permutacion,matriz);
                cambioest();
                wolfram(matrizd);
                menor();
            }
        strcpy(final,clustemp);
    }
    if(estcol==estren)
    {
        for(i=0;i<numestados;i++)
            for(j=0;j<numestados;j++)

```

```

        matrizb[i][j]=matriz[j][i];
wolfram(matrizb);
strcpy(clustemp,nw);
encontrar(matrizb);
for(i=0;i

```

```

        fprintf(hoja,"%c",clusters[totalclusters-1][j]);
        fprintf(hoja," Especimen: %s",nw);
        fclose(hoja);
        valor=1;
    }
    return valor;
}

/*Calcula las matrices de evolucion estocasticas por estado */
void calculo(int edo)
{
    int i;
    int cond;
    cond=0;
    inicializar(edo);
    while(cond==0)
    {
        llenado(edo);
        if(checarciclos(edo)==0)
        {
            conectividad(edo,matrizb);
            if((obtenerindices(matrizb,edo,2))==0)
            {
                if(edo==numestados-1)
                {
                    if(reversible()==0)
                    {
                        total++;
                        cluster();
                        if(lista()==1)
                            imprimiravance();
                    }
                }
            }
            else
                calculo(edo+1);
        }
    }
    borrar(edo);
    cond=actualizar(edo);
}

/*Cuerpo principal del programa */
void main()
{
    int i,j;

```



```
totalclusters=total=0;
inicio();
calculo(0);
printf("\nTotal de matrices generadas: %d\n",total);
}
```

## Bibliografía

- [1] S. Amoroso and Y. N. Patt, *Decision Procedures for Surjectivity and Injectivity of Parallel Maps for Tesselation Structures*, Journal of Computer and System Sciences 6, pp 448-464, 1972.
- [2] Arthur W. Burks (editor), *Essays on Cellular Automata*, University of Illinois Press, 1970.
- [3] E. F. Codd, *Cellular Automata*, Academic Press, 1968.
- [4] James P. Crutchfield and M. Mitchell, *The evolution of emergent computation*, Proceedings of the National Academy of Sciences, USA 92 (23), pp 10472-10476, 1995.
- [5] Edward Fredkin, *Digital Mechanics - An Informational Process Based on Reversible Universal Cellular Automata*, Physica D 45, pp 254-270, 1990.
- [6] Martin Gardner, *Mathematical Games - The fantastic combinations of John Conway's new solitaire game Life*, Scientific American, October 1970, pp 120-123.
- [7] Gran Diccionario Enciclopédico Ilustrado, Volumen X, Selecciones Reader's Digest, 1979.
- [8] Howard Gutowitz, *Cellular Automata, Theory and Experiment*, MIT Press, 1991.
- [9] Gustav A. Hedlund, *Endomorphisms and automorphisms of the shift dynamical system*, Mathematical Systems Theory 3, pp 320-375, 1969.
- [10] David Hillman, *The structure of reversible one-dimensional cellular automata*, Physica D 52, pp 277-292, 1991.
- [11] Aron V. Holden, *Chaos*, Princenton University Press, 1986.
- [12] L. Margara, *Cellular Automata and Chaos*, Ph.D. Thesis, 1995.
- [13] Harold V. McIntosh, *Linear Cellular Automata*, Universidad Autónoma de Puebla, 1990.
- [14] Harold V. McIntosh, *Linear Cellular Automata via de Bruijn Diagrams*, por publicarse.
- [15] Harold V. McIntosh, *Reversible Cellular Automata*, por publicarse.
- [16] Masakazu Nasu, *Local Maps Inducing Surjective Global Maps of One Dimensional Tesselation Automata*, Mathematical Systems Theory 11, pp 327-351, 1978.
- [17] Heinz-Otto-Peitgen, Hartman Jurgens, Dietmar Soupe, *Chaos and Fractals, New Frontiers of Science*, Springer-Verlog, 1992.

- [18] William Poundstone, *The Recursive Universe*, William Morrow and Company Inc., 1985.
- [19] Harold S. Stone, *Discrete Mathematics*, CCS Editorial, Stanford University, 1973.
- [20] Tommaso Toffoli, Norman Margolus, *Cellular Automata Machines*, The MIT Press, Cambridge Massachusetts, 1987.
- [21] Tommaso Toffoli, *Computation and Construction Universality of Reversible Cellular Automata*, Journal of Computer and System Sciences 15, pp 213-231, 1977.
- [22] Burton H. Voorhees, *Computational Analysis of One Dimensional Cellular Automata*, World Scientific, Sigapore, 1995.
- [23] M. Mitchell Waldrop, *Complexity*, Touchstone, 1992.
- [24] Stephen Wolfram, *Theory and Applications of Cellular Automata*, World Scientific, Singapore, 1986.
- [25] Stephen Wolfram, *Universality and Complexity in Cellular Automata*, Physica D10, pp 1-35, 1984.
- [26] Andrew Wuensche, Mike Lesser, *The Global Dynamics of Cellular Automata*, Santa Fe Institute, 1992.

## Lista de Figuras

1	Vecindad del AC de von Neumann . . . . .	5
2	Vecindad de Moore utilizada en Life . . . . .	6
3	Vecindad en un ACL . . . . .	7
4	Cadena de células dispuesta en anillo . . . . .	7
5	Evolución de la <i>i-ésima</i> célula en un ACL . . . . .	8
6	Evolución de la <i>i-ésima</i> célula en un ACL vecindad $h$ . . . . .	8
7	ACL's Clase 1 . . . . .	11
8	ACL's Clase 2 . . . . .	11
9	ACL's Clase 3 . . . . .	12
10	ACL's Clase 4 . . . . .	12
11	Ordenación de una secuencia aleatoria de 0's y 1's con un ACL . . . . .	14
12	Ordenación " <i>paralela</i> " de secuencias aleatorias de 0's y 1's . . . . .	15
13	Diagrama de de Bruijn de un ACL(2,1) regla 90. . . . .	16
14	Diagrama de de Bruijn renombrado de un ACL(2,1) regla 90. . . . .	16
15	Diagrama de de Bruijn de un ACL(4,h) regla 0F08725C. . . . .	17
16	Diagrama de de Bruijn de un ACL(2,1) regla 22. . . . .	19
17	Diagrama de Subconjuntos de un ACL(2,1) regla 22. . . . .	20
18	Diagrama de Parejas de un ACL(2,1) regla 22. . . . .	22
19	Evolución de una configuración de 2 células de un ACL(4,h) regla EC78946E. . . . .	25
20	Evolución de una configuración de 3 células de un ACL(4,h) regla EC78946E. . . . .	26
21	Evolución de una configuración de 4 células de un ACL(4,h) regla EC78946E. . . . .	26
22	Evolución de un ACL(4,h) regla EC78946E, donde cadenas de dos o más 0's son parte del <i>Jardín del Edén</i> de tal ACL. . . . .	26
23	Diferentes tipos de mapeo de una función. . . . .	27
24	La regla de evolución de un ACLR define un automorfismo. . . . .	28
25	Configuraciones de tamaño 2 que evolucionan en 0. . . . .	29
26	Configuraciones de tamaño 3 que evolucionan en 02. . . . .	29
27	Conjunto de ancestros de las configuraciones $02\mathcal{K}$ . . . . .	29
28	Elementos del conjunto $\mathcal{A}$ de cadenas que evolucionan en 02 que forman los ancestros de la configuración 02102. . . . .	30
29	Regla de evolución 0067A26CIJOOI de un ACLR(5,h), y las extensiones derechas compatibles con 0 que evolucionan en 3 y 4. . . . .	31
30	Extensiones derechas compatibles con 0 que evolucionan en 34 y 40. . . . .	32
31	Extensiones derechas compatibles con 0 que evolucionan en 342. . . . .	32
32	Extensiones izquierdas compatibles con 0 que evolucionan en 3 y 4. . . . .	32
33	Extensiones izquierdas compatibles con 0 que evolucionan en 23 y 14. . . . .	33
34	Ancestros de la cadena 231. . . . .	33
35	Ancestros de la cadena 130. . . . .	33
36	Ejemplos de ACLR y sus valores de $LMR$ . . . . .	34
37	Ejemplos de ACLR con valores de $LMR$ no primos. . . . .	35
38	ACL no reversible (5,h) con cadenas cuyas valores de $LMR = k^{2r}$ . . . . .	36
39	Ancestros de la cadena formada alternando los estados 0, 2. . . . .	36

40	Ancestros de la cadena formada alternando los estados 1, 4. . . . .	37
41	ACLR(5,h) donde el valor de $M$ siempre es igual a 1. . . . .	38
42	ACL(5,h) no reversibles y parte de sus diagramas de de Bruijn asociados. . . . .	39
43	ACLR(5,h) regla 0067A26CIJOOI y la ruta 332 en su diagrama de de Bruijn. . . . .	40
44	ACLR(2,1) regla 85 y su diagrama de parejas asociado. . . . .	41
45	ACLR(3,h) regla 8229 y su diagrama de parejas asociado. . . . .	42
46	ACL(2,1) regla 231 y su diagrama de subconjuntos. . . . .	43
47	ACLR(5,h) regla 00077HIGLBDOO y su diagrama de subconjuntos. . . . .	44
48	ACLR(5,h) regla 000667CCIJOOI y su diagrama de subconjuntos. . . . .	45
49	ACLR(5,h) reflexión de la regla 000667CCIJOOI y su diagrama de subconjuntos. . . . .	46
50	ACL(5,h) regla 4ODIJLCH16700 donde existen ciclos en el diagrama de subconjuntos y en el de parejas. . . . .	47
51	ACL(2,1) regla 253 en donde la cadena de estados 00 pertenece al Jardín del Edén. . . . .	48
52	ACL(4,h) regla 4EB11BE4 el cual tiene un mapeo global sobreectivo pero no reversible. . . . .	49
53	ACLR(4,h) regla 05AF05AF. . . . .	50
54	ACLR(2,1) regla 51. . . . .	51
55	ACLR(5,h) regla 3CDK0KL56O7IH. . . . .	52
56	ACLR(2,1) regla 240. . . . .	53
57	Utilizando el método de Fredkin en la evolución de una configuración aleatoria en un ACLR(2,1) regla 240. . . . .	54
58	Regla 240 de un ACLR(2,1) y su regla inversa obtenida por el método de Fredkin. . . . .	55
59	Usando el método de Fredkin con suma y resta módulo 3. . . . .	56
60	Evolución del ACLR(3,h) regla 10179 y de su regla inversa 11355. . . . .	57
61	ACL(3,h) regla 6007. . . . .	59
62	ACL(3,h) regla 14007. . . . .	60
63	ACL(3,h) regla 2119. . . . .	61
64	Ancestro único de la cadena 01 que puede representarse con la misma longitud. . . . .	62
65	Verificación de que cada cadena tenga un único elemento en común en el pasado para cadenas de longitud 2, 3 y 4. . . . .	63
66	ACLR(4,h) regla FF5500AA. . . . .	65
67	Cluster mínimo del ACLR(4,h) regla FF5500AA. . . . .	66
68	Cluster mínimo de varios ACLR(4,h). . . . .	66
69	ACL(4,h) con múltiples ancestros para una cadena de 1's. . . . .	67
70	ACL(2,1) con múltiples ancestros para una cadena de 0's. . . . .	67
71	Matriz de conectividad del estado 0 del ACLR(3,h) regla 715 y como se manifiesta los índices de Welch en ésta. . . . .	69
72	Diagramas de subconjuntos y valores de $L$ , $M$ y $R$ para la matriz de conectividad del estado 0 del ACLR(3,h) regla 715. . . . .	70

73	Diagramas de subconjuntos (regla original y reflejada) del ACLR(3,h) regla 715. . . . .	70
74	Indices de Welch para la cadena 01 del ACLR(3,h) regla 9711. . . . .	71
75	Diferentes formas de matrices de conectividad para ACLR, con un solo 1 en la diagonal principal y valores de $L * R = k^{2r}$ induciendo que $M = 1$ . . . . .	72
76	Matrices de conectividad para ACL no reversibles. . . . .	72
77	Matriz de rutas del estado 0 en un ACL(4,h) donde se observan múltiples ancestros. . . . .	74
78	Matriz de rutas del estado 0 y 1 en un ACL(4,h) donde se observan múltiples ancestros. . . . .	74
79	Crecimiento del número de matrices de conectividad a revisar para cadenas de longitud 1 en adelante para un ACL(4,h). . . . .	75
80	Flujo del proceso para obtener ACLR(k,r). . . . .	76
81	ACLR(4,h) con $L = 1$ y $R = 4$ . . . . .	79
82	ACLR(4,h) con $L = 2$ y $R = 2$ . . . . .	80
83	ACLR(5,h) con $L = 1$ y $R = 5$ . . . . .	85
84	ACLR(5,h) con $L = 1$ y $R = 5$ . . . . .	86
85	ACLR(6,h) con $L = 1$ y $R = 6$ . . . . .	87
86	ACLR(6,h) con $L = 1$ y $R = 6$ . . . . .	88
87	ACLR(6,h) con $L = 1$ y $R = 6$ . . . . .	89
88	ACLR(6,h) con $L = 2$ y $R = 3$ . . . . .	90
89	ACLR(6,h) con $L = 2$ y $R = 3$ . . . . .	91
90	ACLR(6,h) con $L = 2$ y $R = 3$ . . . . .	92