UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ACATLAN

MATEMATICAS APLICADAS Y COMPUTACION

AUTOMATAS CELULARES LINEALES REVERSIBLES

TESIS QUE PRESENTA:
JUAN CARLOS SECK TUOH MORA

PARA OBTENER EL GRADO DE
LICENCIADO EN MATEMATICAS APLICADAS Y COMPUTACION

DIRECTOR DE TESIS:
SERGIO VICTOR CHAPA VERGARA

MEXICO, D. F. ABRIL, 1998

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ACATLAN

MATEMATICAS APLICADAS Y COMPUTACION

AUTOMATAS CELULARES LINEALES REVERSIBLES

Tesis que presenta:
Juan Carlos Seck Tuoh Mora

para obtener el grado de

LICENCIADO EN MATEMATICAS APLICADAS
Y COMPUTACION

Director de Tesis:
Sergio Victor Chapa Vergara

Indice

Introducción

Sin duda alguna la computación ha sido una de las áreas de investigación que más ha avanzado en los últimos 50 años. Este vertiginoso crecimiento se debe principalmente a dos aspectos. El primero, al desarrollo del transistor en 1947 y por ende el desarrollo de toda la microelectrónica. El segundo, fue el desarrollo de una serie de ideas y fundamentos teóricos cuyos resultados presagiaron las modernas computadoras.

En publicaciones independientes Emil L. Post por un lado y Alan M. Turing por el otro, concivieron lo que ahora llamamos como autómata y descrito en forma similar como máquinas. De los dos aspectos anteriores se ha derivado lo que actualmente conocemos como computadoras y computación las cuales de una manera interrelacionadas han catalizado el avance de ambas.

Una de las ideas de Turing es acerca de la existencia de un autómata universal, en el sentido que éste pudiera calcular cualquier secuencia que cualquier otro automata especial pudiera, proveyendo solamente de un conjunto apropiado de órdenes de entrada. esto parece paradójico, puesto que en principio uno puede imaginar autómatas de varios tamaños y complejidades.

En esencia lo que demuestra Turing es que cualquier autómata puede ser descrito por un conjunto finito de instrucciones y cuando éste es alimentado a su autómata universal en su turno imita al caso especial.

John von Neumann, intrigado enormemente con las ideas anteriores, en 1947 empieza a trabajar en dos amplios frentes: primero, quería averiguar que tan complejo podría ser construir un aparato auto reproductivo y segundo, quería investigar el problema de como organizar dispositivos que debieran ser hechos de partes aisladas.

Estos planteamientos no surgieron azarosamente, el interés de von Neumann en la teoría de autómatas se combino por su temprana preparación en lógica y los recientes intereses en computadoras y neurofisiología.

En 1948 von Neumann ofreció algunas pláticas en donde planteaba sus conjeturas acerca de un modelo cinético en donde los principios en el cual se conduce están estrechamente relacionados a los principios de Turing. Además, discutía las nociones fundamentales de computación y supuestos acerca de la complejidad. Es interesante que la conjetura de von Neumann presentó mucha similitud con lo planteado por Watson y Crick [25] en la estructura molecular del ADN y ARN. de manera muy clara lo que se estableció en sus escritos y conversaciones acerca del autómata es el mecanismo de duplicación de material genético para la reproducción.

Aunque los autómatas celulares de manera preliminar pero simple pueden describirse en términos de dos conceptos (configuración y reglas de transición); el estudio de propiedades, taxonomía y análisis requiere de una gran cantidad de herramienta matemática y experimentación.

Los resultados han alcanzado problemas fundamentales y temas como los fractales y sistemas dinámicos; en particular, una clase de autómatas celulares son los reversibles en donde el concepto es totalmente similar al de termodinámica, en donde el sistema puede regresar por los estados anteriores.

El objeto principal de este trabajo es el de estudiar y mostrar algunos resultados acerca de los autómatas celulares lineales reversibles.

El trabajo está organizado de la siguiente forma:

- El capítulo 1 explica el funcionamiento, las propiedades básicas y la terminología de los autómatas celulares así como las herramientas que existen para el estudio de tales sistemas.
- El capítulo 2 expone los conceptos de reversibilidad y su manifestación en los autómatas celulares lineales, la caracterización de los mismos se apoya en dos trabajos fundamentales sobre el tema; uno acerca del mapeo en un sistema de corrimiento, desarrollado por los matemáticos Gustav Adolf Hedlund y otro acerca de mapeos locales y mapeos globales en un autómata celular, elaborado por Masakasu Nasu.
- El capítulo 3 trata sobre dos algoritmos (uno concluido y el otro en desarrollo) cuyo objetivo es detectar todos los posibles autómatas celulares lineales reversibles.
- El capítulo 4 expone el funcionamiento de un método propio para realizar la tarea de detectar en un autómata celular lineal las características presentadas en el capítulo 2.
- El capítulo 5 presenta los resultados obtenidos por el método propuesto haciendo una comparación con los algoritmos presentados en el capítulo 3.
- Se dan las conclusiones finales del trabajo y se presentan observaciones sobre el futuro tanto en el cálculo de automatas reversibles como sus posibles aplicaciones.
- El apéndice A contiene el listado para un caso particular del cálculo de autómatas celulares lineales reversibles.

Las gráficas de evolución, así como los diagramas representativos de autómatas celulares lineales que se presentan en este trabajo se generaron utilizando el sistema NXLCAU para NeXtSTEP desarrollado por el Dr. Harold V. McIntosh en el Centro de Aplicación de Microcomputadoras de la Universidad Autónoma de Puebla; este sistema está disponible en su página en Internet: *www.cs.cinvestav.mx/mcintosh/cellular.html.*

Capítulo 1

Conceptos básicos de Autómatas Celulares.

Antes de hacer un análisis sobre la reversibilidad en los autómatas celulares lineales, es de relevancia primero explicar el concepto de autómata celular, de donde surge esta teoría, sus definiciones, nomenclatura y las propiedades básicas que estos tienen.

Se hace una pequeña revisión sobre la utilización de los autómatas celulares en diferentes campos de aplicación y por último se describe la aplicación que se hace de la Teoría de Gráficas para el estudio de los autómatas celulares lineales.

1.1 Origen de los Autómatas Celulares.

1.1.1 Trabajo de von Neumann.

El concepto de autómata celular fué desarrollado en los finales de los años 40'tas por John von Neumann; en esta época la manufactura de automóviles y bienes eléctricos empezaba a automatizarse, en el ámbito de la computación el trabajo de Turing estaba obteniendo cada vez más interés y von Neumann se interesó en encausar su trabajo hacia esta dirección aunque se puede congeturar en que el verdadero interés de von Neumann era crear una máquina que pudiera manufacturarse a sí misma [20], 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.

Ulam esencialmente sugirió un espacio cuadriculado donde cada cuadro podía ser ocupado por una *célula* dada, la cual podía tener un *número finito* de estados que cambiaban en el tiempo que avanza en lapsos discretos. Cada célula hacía una transición a un nuevo estado dependiendo de su valor actual y el de sus vecinos; las células que se encuentran en sus bordes (fig. 1.1).

Figura 1.1: Vecindad de von Neumann.

De este modo, von Neumann pudo desarrollar un autómata que podía autorreproducirse y contaba con 29 estados diferentes para cada célula, cada estado cumplía una función sencilla para la *duplicación* del patrón inicial. Las anteriores características del modelo de von Neumann presentaron semejanzas a la estructura del ADN descubierta por Watson y Crick en 1953 [25]. John von Neumann murió en 1957 sin publicar su trabajo pero éste es editado por Arthur W. Burks [2] en 1966 y en ese mismo tiempo Edward Codd [3] trabaja en una variante de 8 estados más versátil en su comportamiento del autómata de von Neumann.

1.1.2 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 de forma interactiva, sino que es un autómata celular que a través del tiempo evoluciona y el espectador observa. El nombre de Life se debe a que cada célula tiene dos estados posibles, vivo o muerto y la vecindad se debe a Moore [2], con la configuración que aparece en la figura 1.2:

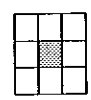


Figura 1.2: Vecindad de Moore.

Las reglas que rigen el comportamiento de Life son las siguientes:

- Una célula que está viva muere por tener 4 o más vecinas vivas a su alrededor (sobrepoblación).
- Una célula que está viva muere por tener 1 o menos vecinas vivas a su alrededor (aislamiento).
- Si existen 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 autómatas celulares.

1.1.3 Trabajo de Stephen Wolfram.

La disponibilidad que se tiene actualmente de un mejor equipo de computación ha permitido modelar problemas complejos que llegan alcanzar un comportamiento caótico, propiciando nuevas vertientes de investigación.

En los 80'tas surge el trabajo de Stephen Wolfram en donde se investiga la complejidad en el nivel más fundamental. Basándose en el particionamiento de un sistema en sus elementos más simples, el comportamiento de la interacción de las partes es estudiado en términos de un comportamiento global complejo.

Stephen Wolfram señala que los sistemas complejos se rigen con componentes muy simples y leyes básicas que los gobiernan. Sin embargo, a pesar del establecimiento simple del modelo la interacción simultánea de los componentes sencillos produce una gran complejidad del sistema. es por esto que Wolfram hace uso de los autómatas celulares, ya que con ellos se puede analizar su comportamiento en detalle; aunque completamente determinísticos pueden generar patrones complejos con comportamiento estadístico y hasta caótico.

La construcción de los autómatas celulares es sencilla y con su fácil implementación en una computadora ha permitido modelar sistemas físicos, químicos o biológicos [26].

Comparando el trabajo de von Neumann y Conway con el de Wolfram, es importante resaltar que mientras los primeros se concentraron en desarrollar un sólo autómata con una cierta regla que cumplieran sus propósitos y estudiaron su comportamiento; Wolfram, se enfocó a analizar un cierto número de reglas diferentes en un autómata celular principalmente en el caso lineal.

Se puede sugerir que Wolfram optó estudiar el caso unidimensional ya que existen en éste menos posibilidades de comportamiento que en casos de mayor dimensión además de su sencillez de implementación y representación en una computadora con lo que su estudio puede ser más fácil y detallado; siguiendo esta misma filosofía este trabajo abordará sólo el caso unidimensional ya que aunque es el más sencillo de estudiar aún existen temas dentro del mismo que permanecen sin una comprensión total (como es el caso de la reversibilidad).

1.2 Funcionamiento de los Autómatas Celulares Lineales.

1.2.1 Conceptos de estado y vecindad.

Un autómata celular lineal se construye de la siguiente manera:

a) En una línea se colocan una serie de celdas llamadas células, cuya representación puede ser llevada a un arreglo lineal de los elementos.

b) Cada célula puede tener un número finito de estados distinguibles mediante valores enteros, letras o atributos, lo que se quiera que represente cada una, a estos valores se les denomina *estados.*

c) Todas las células tienen el mismo número posible de estados.

En los automatas celulares lineales las células se interrelacionan formando un tipo de vecindad, donde cada célula tiene un radio de vecindad r, o sea r vecinos a cada lado (fig. 1.3) para obtener una vecindad de interacción de tamaño $2r + 1$ como número total de células.

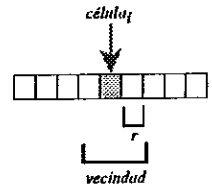


Figura 1.3: Autómata celular lineal donde cada célula tiene un vecino a cada lado ($r = 1$), el tamaño total de la vecindad es $2r + 1 = 3$.

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 (fig. 1.4).

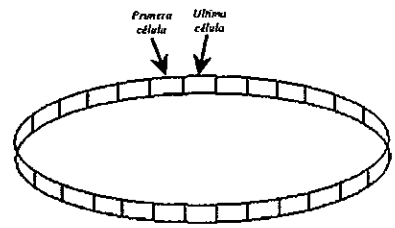


Figura 1.4: Cadena de células dispuesta en forma de anillo.

Wolfram propone la notación (k, r) para los autómatas celulares lineales, en donde k denota los estados posibles que toma cada célula y r el radio de células vecinas de cada lado para formar su vecindario.

Por ejemplo, para un autómata celular lineal $(2, 1)$ el vecindario contiene tres células y cada célula puede tener un estado de dos posibles $(0, 1)$. El número posible de vecindades diferentes es k^{2r+1}, en este caso $2^3 = 8$, dando como resultado las siguientes vecindades: (000,001,010,011,100, 101,110,111).

1.2.2 Descripción del mecanismo de evolución.

Un autómata celular lineal 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 simultaneamente y el tiempo avanza en etapas discretas (fig. 1.5).

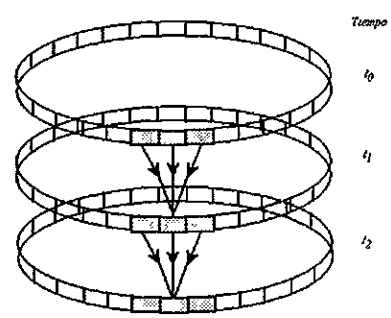


Figura 1.5: Forma en que evoluciona la i-*ésima* célula en un autómata (k,1).

En realidad, los vecindarios no necesariamente tienen que estar centrados y tener un número impar de células, Harold V. McIntosh [15] establece vecindarios de tamaño par; en donde la célula en la etapa siguiente significa que puede ocupar un lugar entre media célula (fig. 1.6).

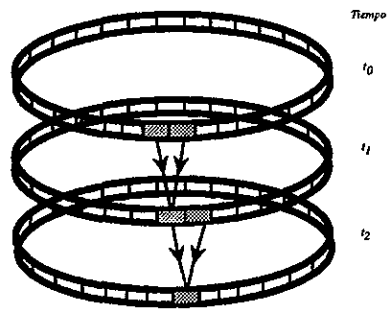


Figura 1.6: Forma en que evoluciona la i-*ésima* célula en un autómata (k,h), $h = 0.5$.

El patrón de valores que se observa a través de todo el arreglo se denomina el estado global del autómata en un tiempo dado [28], la *evolución* de un autómata son los diferentes estados globales que el arreglo puede tomar a través del tiempo, estos se forman en el momento en que todas las células en el arreglo actualizan su valor al mismo tiempo, lo que provoca que cambie el estado global del arreglo del tiempo t_i al t_{i+1}.

Cualquier patrón de estados en el arreglo puede ser una condición desde donde comienze la evolución del autómata, a tal condición se le nombra *configuración inicial*, y al estado global en el tiempo i se le denomina *generación i-esima.*

Por último. la evolución de un autómata celular lineal 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 puede expresarse en forma tabular (tabla 1.1).

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

Tabla 1.1: Regla de evolución de un autómata (2,1).

Debido a que cada célula en una vecindad puede tomar k valores se tiene que el número de reglas de evolución de un autómata (k,r) es $k^{k^{2r+1}}$.

Wolfram hace una clasificación de los autómatas celulares mediante sus reglas de evolución. Como la regla a medida que hay más estados y/o vecinos se torna más complicada, Wolfram utiliza su equivalente decimal o cambios a otra base según convenga, quedando en la literatura la designación como *Número de Wolfram* [15], por ejemplo, para un autómata (2,1) se utiliza su equivalente decimal pues la regla es un número binario, la tabla 1.2 muestra la regla 15:

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 1.2: Regla 15 de un autómata (2,1).

Mientras tanto, para un autómata $(4, 1/2)$ se maneja una notación hexadecimal en bloques de dos; cabe señalar que cuando el radio de vecindad está dado en medios, en la notación se acostumbra sustituirlos por letras, para el caso de $1/2$ su utiliza h, para $3/2$ se utiliza t, para $5/2$ se utiliza f y así sucesivamente, aplicando lo anterior tenemos que se trata de un autómata (4,h); veamos en la tabla 1.3 la regla de evolución 0055AAFF:

	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
Vecindades	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
Evolución	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
Base 4	4^1	4^0	4^1	4^0	4^1	4^0	4^1	4^0	4^1	4^0	4^1	4^0	4^1	4^0	4^1	4^0
Base 10	0		0		5		5		10		10		15		15	
Base 16	0		0		5		5		A		A		F		F	

Tabla 1.3: Regla de evolución 0055AAFF de un autómata (4,h).

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

Las características de un autómata celular lineal son 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; en síntesis, son 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 [19], a ésta aperiodicidad se le denomina caos, sin embargo, éste tiene pecularíedades muy particulares [13] 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 [19], entendiendo por determinismo un comportamiento no variante y por predicibilidad saber con certeza el estado futuro de un sistema.

Estas características se pueden claramente observar en un autómata celular lineal, 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.

Ya que los autómatas celulares cumplen con las características de los sistemas discretos y caóticos se pueden aplicar para el estudio y simulación de sistemas biológicos y físicos (estudio de cultivos microbianos, crecimiento dendrítico, estudio del magnetismo, dinámica de fluidos entre otras muchas) que comparten estas mismas características además de mostrar una conducta que va del caos al orden y visceversa; Wolfram muy posiblemente observando estos distintos comportamientos postula una clasificación de los autómatas celulares de acuerdo a su evolución.

1.2.4 Clasificación de Wolfram.

Para los autómatas celulares, Wolfram propone cuatro clases, las cuales dependen de sus características visibles. En la evolución de un autómata se observa que cada regla tiene sus particularidades, compartiendo algunas veces similitudes y diferencias. La clasificación de Wolfram es la siguiente:

- **CLASE 1.**- El autómata 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 (fig. 1.7).

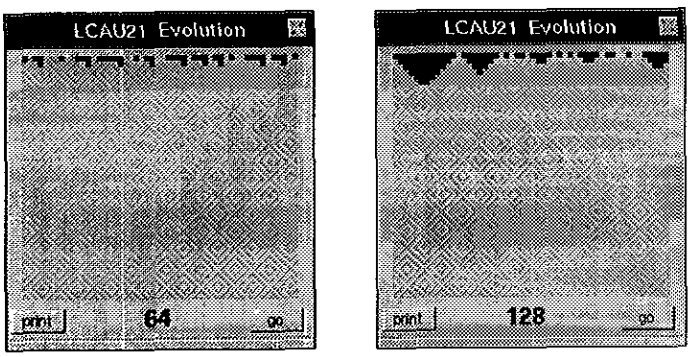


Figura 1.7: Autómatas celulares Clase 1.

- **CLASE 2.**- Desde casi cualquier configuración inicial, el autómata generará después de un corto tiempo, estructuras simples separadas las cuales tienen un comportamiento periódico (fig. 1.8).

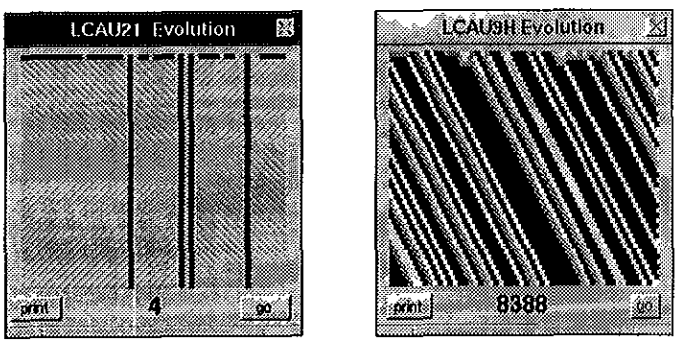


Figura 1.8: Autómatas celulares Clase 2.

- **CLASE 3.-** El autómata desde casi cualquier posible configuración inicial produce patrones aperiódicos o caóticos (fig. 1.9).

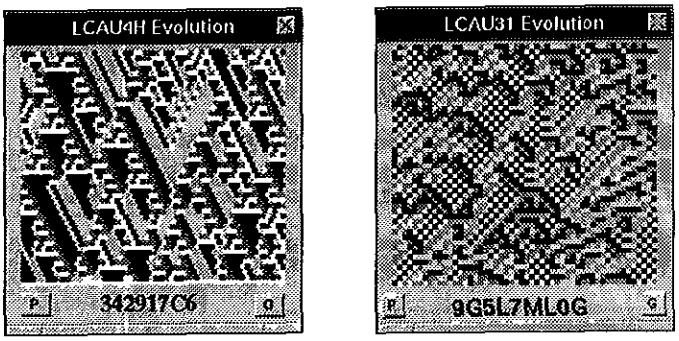


Figura 1.9: Autómatas celulares Clase 3.

- **CLASE 4.-** El autómata produce estructuras con comportamiento complejo, entre el orden y el caos (fig. 1.10).

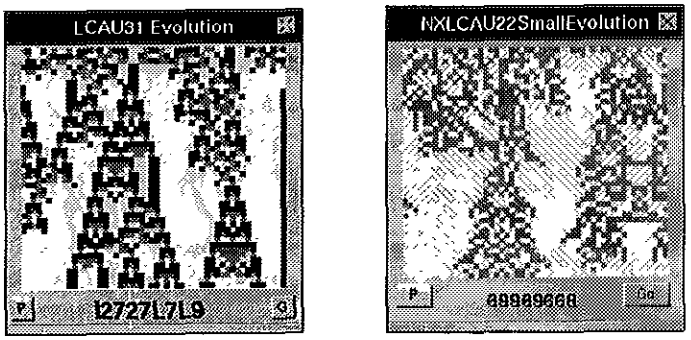


Figura 1.10: Autómatas celulares Clase 4.

1.2.5 Aplicaciones de los Autómatas Celulares.

Se puede identificar tres vertientes principales en donde se han utilizado autómatas celulares 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 propuesta por dos químicos rusos, Boris P. Belousov y Anatol M. Zhabotinsky que lleva sus nombres (reacción Belousov-Zhabotinsky o BZ), mecánica de fluidos, patrones de pigmentación de piel, estudio de morfogénesis (trabajo de Alan Turing), crecimiento de conchas marinas y corales, comportamiento de colonias de microorganismos entre otros; ejemplos de estas aplicaciones se pueden encontrar en [10, 26].

2. *Estudios Teóricos.*

   En este campo se estudia a los autómatas celulares como un sistema formal y todo lo que ésto implica (desarrollo de axiomas, proposiciones, teoremas, lemas, etc.) además de utilizarlos para estudiar áreas como complejidad, sistemas caóticos, termodinámica, entropía, computación en paralelo, computación universal, teoría de lenguajes computacionales o estudio de patrones fractáles como se muestra en [14, 27].

3. *Realización de tareas específicas.*

   Aquí se busca construir un autómata que sea capaz de desarrollar un proceso en especial, esto puede ser desde creación de fondos para diseños artísticos, procesamiento de imágenes y encriptación de datos [10]; se han creado autómatas que realizan tareas muy sencillas como podría ser un autómata (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]; diseño de circuitos integrados (VLSI), detección de globulos rojos y blancos en la sangre, códigos de detección de errores entre otras.

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 autómata 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 autómata que realiza una tarea en especial, la replica de él mismo.

Mostrando un ejemplo muy sencillo de una aplicación, se propone un autómata que ordene de izquierda a derecha en orden descendente una secuencia aleatoria de ceros y unos después de un número finito de pasos (tabla 1.4).

0110001100	t_0
$\downarrow$	$\vdots$
1111000000	t_n

Tabla 1.4: Ordenación de una secuencia de 0's y 1's.

Para resolver esta simple tarea se tomará un autómata (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 se muestra en la tabla 1.5.

Vecindades	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	
Evolución	0	1	0	0	0	0	2	2	2	0	1	0	1	1	1	
Base 3	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	
Base 27	3			0			Q			3			D			

Vecindades	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
Evolución	2	2	2	0	1	0	1	1	1	2	2	2
Base 3	3^0	3^1	3^2	3^0	3^1	3^2	3^0	3^1	3^2	3^0	3^1	3^2
Base 27	Q			3			D			Q		

Tabla 1.5: Regla de evolución QD3QD3Q03 para un autómata (3,1).

Esta regla de evolución del autómata (3,1) se clasifica QD3QD3Q03 utilizando en ésta una notación de base 27 en bloques de 3 para este autómata 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 (fig. 1.11) en donde la configuración inicial es una secuencia aleatoria de 0's y 1's, limitada a los lados con el estado 2.

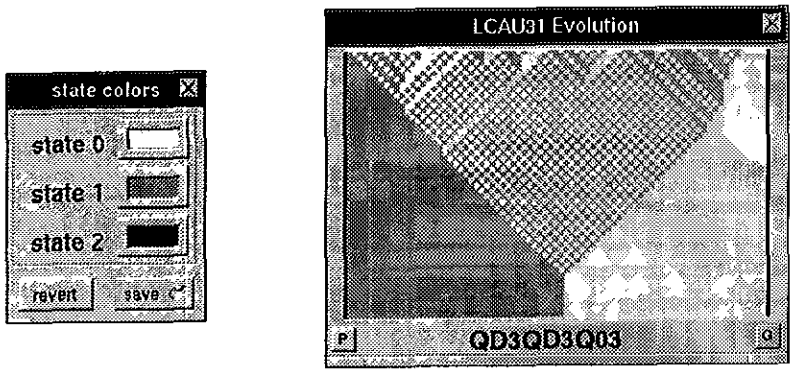


Figura 1.11: Ordenación de una secuencia aleatoria de 0's y 1's con un autómata (3,1) regla QD3QD3Q03.

En este ejemplo (fig. 1.12) es posible ver de manera muy sencilla una computación en paralelo que hace el autómata propuesto al ordenar al mismo tiempo varias secuencias aleatorias de 0's y 1's.

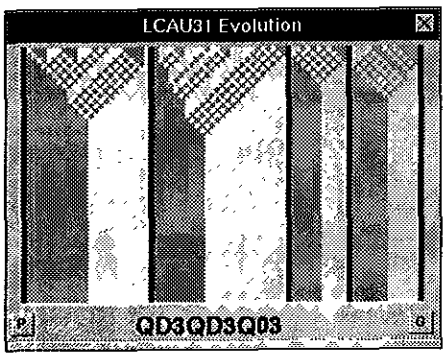


Figura 1.12: Ordenación "*paralela*" de secuencias aleatorias de 0's y 1's con un autómata celular lineal.

1.3 Herramientas utilizadas en el estudio de los Autómatas Celulares Lineales.

Los autómatas celulares lineales se pueden estudiar mediante su evolución, sin embargo para llevar a cabo análisis más profundos se hace uso de la teoría de gráficas con tres herramientas principales: diagramas de de Bruijn, diagramas de subconjuntos y diagramas de parejas.

Con estos se obtienen importantes características tanto del comportamiento local como global del autómata. Las gráficas tienen la ventaja de hacer más claro el comportamiento de la dinámica de cualquier sistema, además, de la posibilidad de que con las representaciones matriciales podemos analizar los autómatas encontrando nuevas propiedades y caracterizaciones.

1.3.1 Diagrama de de Bruijn.

El diagrama de de Bruijn surge debido a una serie de trabajos dedicados al estudio de registros de corrimientos de una secuencia de símbolos para la codificación de información, el nombre del

diagrama se debe al trabajo desarrollado por N. G. de Bruijn a mediados de los 40's como señala Solomon Golomb [9] quien trabajó también en la misma idea.

En esta área, el diagrama de *de Bruijn* nos representa como evolucionan los vecindarios en un autómata celular lineal; cada nodo del diagrama representa un conjunto de $2r$ células por lo que habrá tantos nodos como posibles conjuntos distintos de células de tamaño $2r$.

Estos nodos representan vecindades parciales del autómata; 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, traslapando como fichas de dominó, la importancia de las ligas originadas por el traslape de dos nodos es que el color o etiqueta de la liga representa la evolución de la vecindad. Tomando los conceptos anteriores mostremos la forma genérica que tendría el diagrama de de Bruijn (fig. 1.13) para diversos tipos de autómatas (k,r).

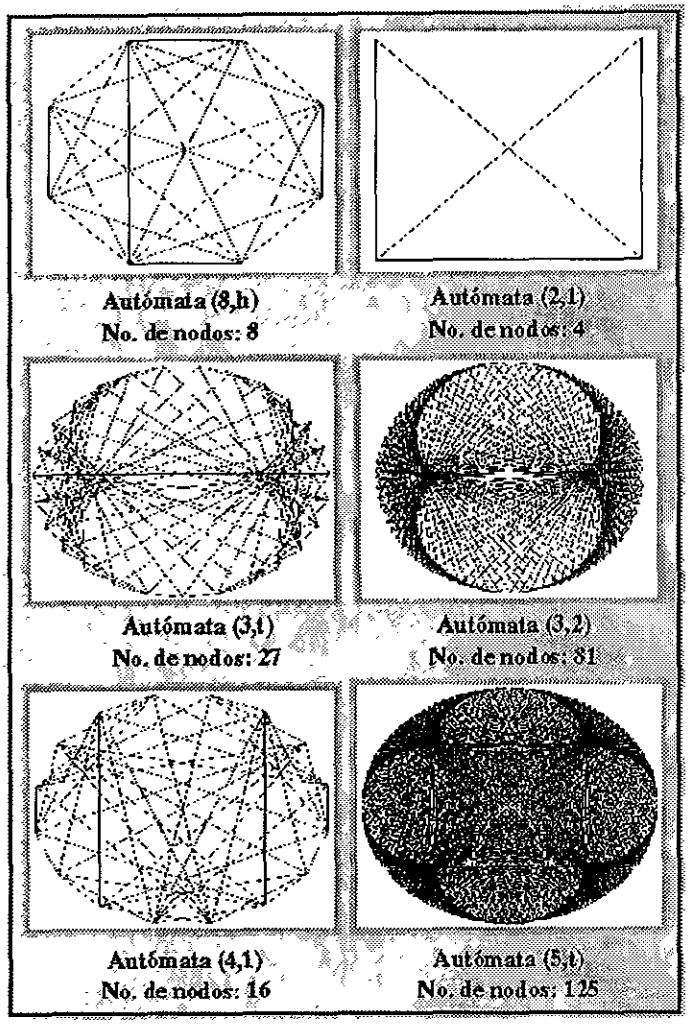


Figura 1.13: Forma genérica del diagrama de de Bruijn para diversos tipos de autómatas celulares.

Como podemos observar, conforme crece el número de estados y/o el tamaño de vecindad aumenta rapidamente el número de líneas del diagrama de de Bruijn asociado, lo que se traduce en problemas de representación gráfica.

Considere un ejemplo particular, un autómata (2,1) regla 90 (tabla 1.6).

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	0	1	0	1	1	0	1	0

Tabla 1.6: Regla de evolución 90 de un autómata (2,1).

El diagrama de de Bruijn tendrá $2^2 = 4$ nodos con $2^3 = 8$ ligas (fig. 1.14).

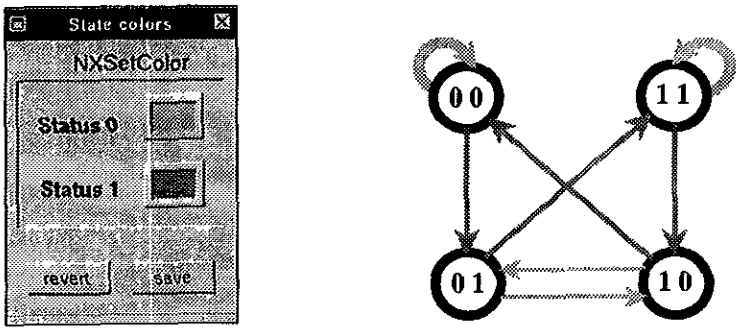


Figura 1.14: Diagrama de de Bruijn de un autómata (2,1) regla 90.

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 se renombran los nodos con valores enteros del 0 en adelante y el diagrama toma la siguiente forma (fig. 1.15):

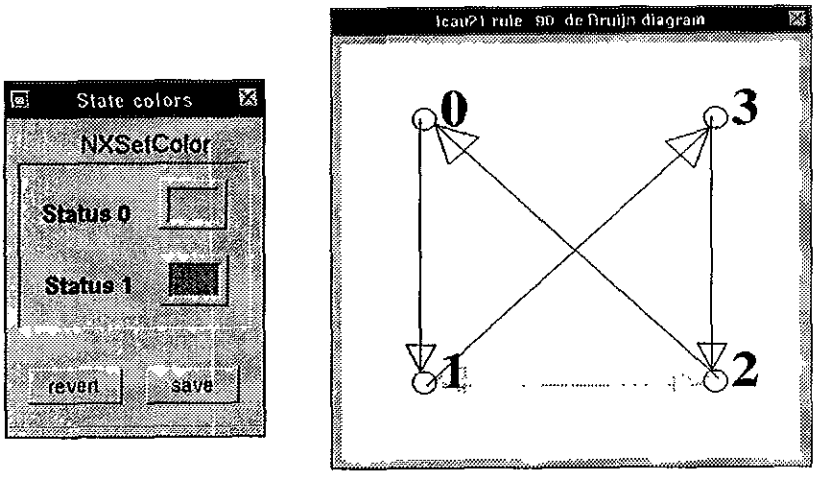


Figura 1.15: Diagrama de de Bruijn renombrado de un autómata (2,1) regla 90.

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 (tabla 1.7).

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

Tabla 1.7: Matriz de evolución de un autómata (2,1) regla 90.

Otro ejemplo del diagrama de de Bruijn ahora para un autómata (4,h) (tabla 1.8 y fig. 1.16).

Vecindades	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
Evolución	0	3	1	1	2	0	3	1	0	2	0	0	3	3	0	0

Tabla 1.8: Regla de evolución 0F08725C de un autómata (4,h).

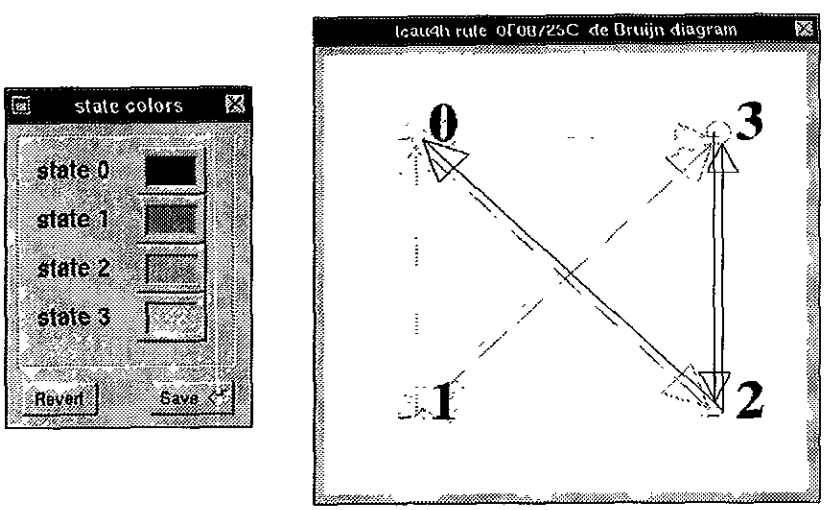


Figura 1.16: Diagrama de de Bruijn de un autómata (4,h) regla 0F08725C.

La matriz de evolución de dicho autómata (4,h) será (tabla 1.9):

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

Tabla 1.9: Matriz de evolución del autómata (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 autómata (4,h) la cadena 001 puede ser consecuencia de la evolución de las células 0002 ó 3203.

1.3.2 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 en el diagrama [16]; para responder esto se ha utilizado el diagrama de *Subconjuntos* desarrollado por Moore [17]. Este diagrama es muy aplicado en la teoría general de autómatas para saber si una cierta cadena es parte del lenguaje de un autómata determinístico, un antecedente del uso de tal diagrama en el estudio de autómatas celulares se puede encontrar en el trabajo de Amoroso y Patt [1].

Para construir el diagrama de subconjuntos se 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 autómata (2,1) regla 22 (tabla 1.10, fig. 1.17 y tabla 1.11).

Vecinades	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	0	1	1	0	1	0	0	0

Tabla 1.10: Regla de evolución 22 del autómata (2,1).

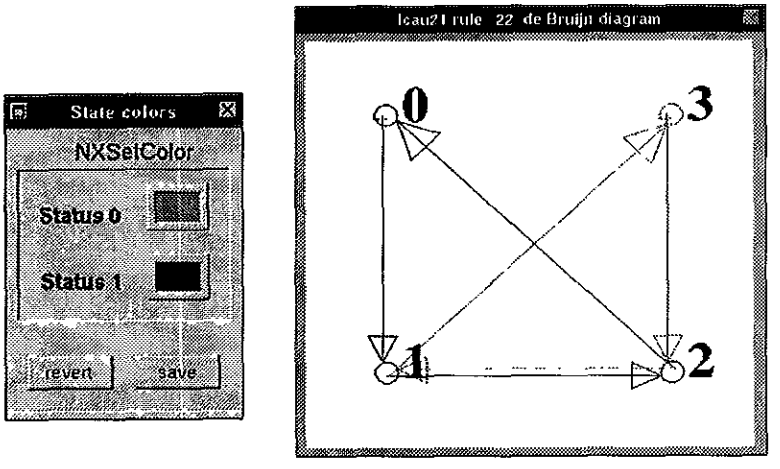


Figura 1.17: Diagrama de de Bruijn de un autómata (2,1) regla 22.

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

Tabla 1.11: Matriz de evolución del autómata (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 (tabla 1.12 y fig. 1.18).

Subconjunto	Liga 0	Liga 1	Enumeración	Liga 0	Liga 1
∅	∅	∅	0	0	0
0	0	1	1	1	2
1	3	2	2	8	4
2	1	0	4	2	1
3	2,3	∅	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 1.12: Subconjuntos del diagrama de de Bruijn del autómata (2,1) regla 22.

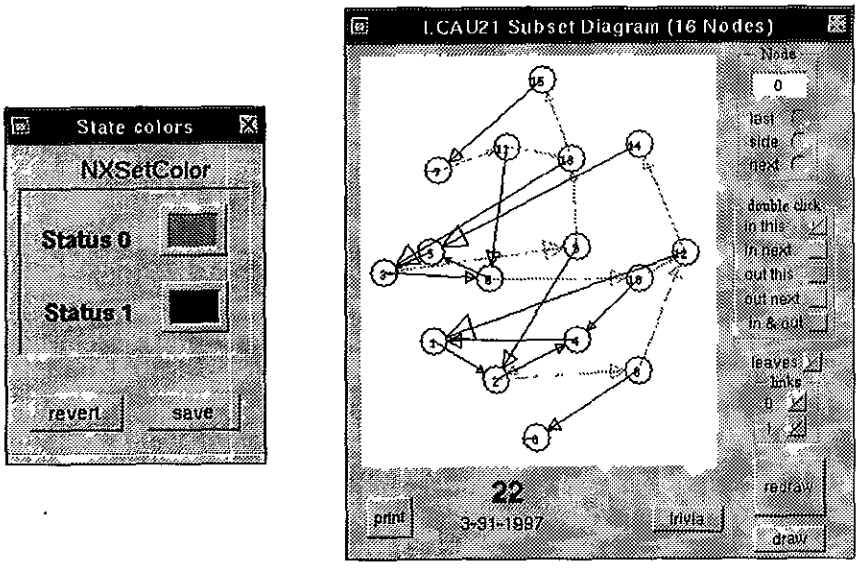


Figura 1.18: Diagrama de Subconjuntos de un autómata (2,1) regla 22.

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 o más 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 autómata, 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 autómata (2,1) regla 22 agrupando los subconjuntos de acuerdo a su tamaño es como sigue (tabla 1.13).

	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 1.13: Matriz del diagrama de subconjuntos del autómata (2,1) regla 22.

1.3.3 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. Esta es la función del diagrama de *Parejas* [16] bien conocido en el estudio de la teoría de autómatas en general; en este caso el diagrama 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 en el diagrama de parejas unen nodos donde ambos miembros de un nodo están conectados con la misma liga en el diagrama de de Bruijn con los miembros del otro (fig. 1.19 y tabla 1.14). 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; el diagrama de de Bruijn está implícito en la diagonal principal del diagrama de parejas.

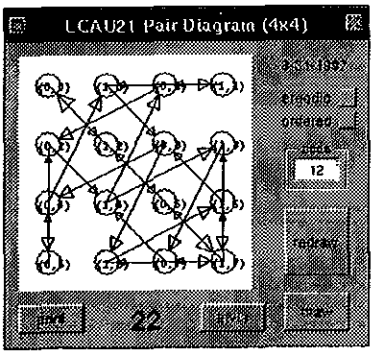


Figura 1.19: Diagrama de Parejas de un autómata (2,1) regla 22.

	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 1.14: Matriz del diagrama de parejas del autómata (2,1) regla 22.

Comparando esta matriz con la matriz de evolución del mismo autómata 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 ésto 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.

1.4 Comentarios Concluyentes.

Los trabajos de von Neumann, Conway y Wolfram han inspirado a muchos otros en sus respectivas épocas para trabajar en el análisis y desarrollo de la teoría y aplicación de los autómatas celulares. Estos son una herramienta alternativa interesante, facilmente reproducibles en una computadora, para el estudio de sistemas dinámicos en donde el comportamiento local de sus partes sea fundamental.

La construcción y el funcionamiento de un autómata celular lineal son muy sencillos, teniendo una gama de comportamientos que van desde ser muy simples hasta ser verdaderamente complejos incluyendo conductas caóticas.

La teoría de gráficas sin duda favorece el entendimiento del campo donde se aplique gracias a la percepción visual que ofrece. En los autómatas celulares no es la excepción, ya que nos ofrece un modo directo y elegante de mostrar las propiedades y características que cumplen los autómatas celulares lineales.

En principio, el diagrama de de Bruijn, el de subconjuntos y el de parejas
responden tres cuestiones básicas acerca del comportamiento de un autómata celular lineal:

- Saber si el autómata es capaz de formar una secuencia dada de estados.
- Como es posible formar dichas secuencias.
- De cuantas máneras diferentes se pueden construir las mismas.

Capítulo 2

Reversibilidad y su manifestación en los Autómatas Celulares Lineales.

Una vez que hemos presentado la definición y propiedades de los autómatas celulares lineales, así como las principales herramientas para el análisis de los mismos, en seguida pasaremos a la parte central de este trabajo que es el estudio de los Autómatas Celulares Lineales Reversibles.

El objetivo de este capítulo, en principio, es dar un marco teórico de los autómatas celulares lineales reversibles los cuales son una clase especial importante debido a que conservan la información del sistema a través de su evolución.

También se presenta el concepto de reversibilidad para autómatas celulares y características que incluyen conceptos tan importantes como el de ancestro y jardín del edén. Con los diferentes diagramas: de Bruijn, parejas y subconjuntos, se analizan estas conductas y comportamientos.

Finalmente se presentan los métodos para encontrar de un autómata celular lineal reversible dado, el autómata con el comportamiento inverso.

2.1 Antecedentes Históricos

Inicialmente los autómatas celulares se utilizaron como modelos para estudiar el comportamiento de sistemas biológicos, reacciones químicas, procesamiento de imá-genes; es decir los investigadores se interesaban más en explorar las consecuencias de un comportamiento más que analizar sus orígenes, como los autómatas celulares 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.

2.1.1 Trabajo realizado por Moore.

A mediados de los 50'tas, surgen preguntas de tipo filosófico sobre configuraciones que no pudieran ser generadas por un autómata 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].

2.1.2 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 autómatas celulares.

Hedlund trabajaba para la Agencia Nacional de Seguridad de Estados Unidos en los 60'tas, su estudio se realiza en el área de la dinámica simbólica sobre mapeos continuos de secuencias de símbolos presumiblemente con propósitos de encriptación de información, aunque Hedlund nunca reconoció dicha cuestión. Este es un caso muy parecido al de Turing en la segunda guerra mundial; su trabajo a lo largo de 10 años sobre este tema es publicado en 1969 [11] e irónicamente los resultados que con mucho detalle en éste se exponen tienen una íntima relación con la teoría de autómatas celulares pero desafortunadamente este trabajo sigue siendo desconocido y poco entendido para muchos estudiosos sobre el tema.

Un ejemplo de lo anteriormente dicho es el trabajo de Amoroso y Patt [1] en 1972. Ellos buscaron autómatas reversibles (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 Masakasu Nasu [18] hace un cuidadoso análisis del trabajo de Hedlund y relaciona éste con los autómatas reversibles apo-yado en la teoría de gráficas.

2.1.3 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 autómatas celulares y que modelos de éstos preservan la información del sistema, llegando a técnicas para manejar un comportamiento invertible con operaciones booleanas.

2.2 Reversibilidad

2.2.1 Concepto de Reversibilidad

En el diccionario [7] se define el concepto de reversible como un proceso en que el sistema puede volver a pasar por los estados o condiciones anteriores. En el ámbito de este trabajo un autómata celular reversible es aquel cuya evolución y el pasado entero de cualquier configuración puede ser descifrado.

Del concepto esencial de reversibilidad de poder encontrar cualquier configuración en su evolución y el pasado, surge la pregunta a contestar *¿cuál es la relación funcional que existe entre el mapeo local de un autómata 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 reversible el mapeo global es invertible, es decir, si cada configuración que por definición tiene exactamente un sucesor debe tener entonces exactamente uno y sólo un antecesor.

Para que esto sea posible, un autómata reversible 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 autómata irreversible que es disipativo, ya que pierde o disipa la información del sistema.

2.2.2 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 solido 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 autómatas reversibles 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.

2.3 Características de los Autómatas Celulares Lineales Reversibles.

2.3.1 Concepto de Ancestro y Jardín del Edén.

Antes de presentar las propiedades que tienen los autómatas celulares lineales reversibles, es importante conocer dos conceptos fundamentales. En un autómata celular 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 concluyendo que si en un autómata 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 sólo aparecen al principio de ésta, al conjunto de tales configuraciones se le denomina el *Jardín del Edén* de un autómata dado.

Analizemos el caso de un autómata (4,h) (tabla 2.1).

Vecindades	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
Evolución	2	3	2	1	0	1	1	2	0	2	3	1	0	3	2	3

Tabla 2.1: Regla de evolución EC78946E de un autómata (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 autómata (4,h) desde una configuración inicial formada unicamente por el estado 0 (fig. 2.1).

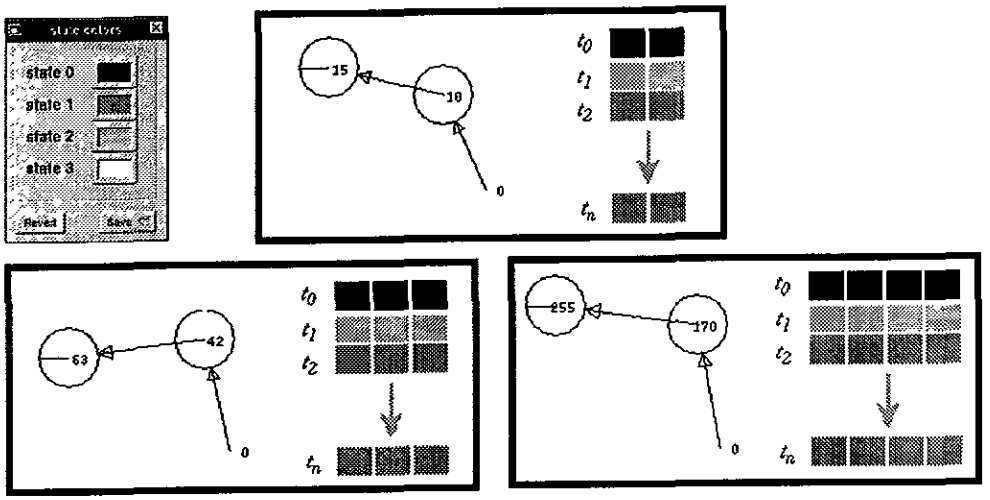


Figura 2.1: Evolución de una configuracón de 2,3 y 4 células de un autómata (4,h) regla EC78946E.

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 autómata (4,h) (fig. 2.2).

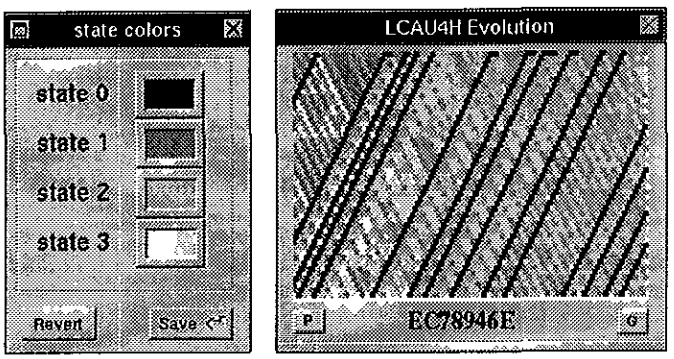


Figura 2.2: Evolución de un autómata (4,h) regla EC78946E, donde cadenas de dos o más $0's$ son parte del *Jardín del Edén* de tal autómata.

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

2.3.2 Sobreyectividad e Inyectividad en un Autómata Celular Lineal Reversible

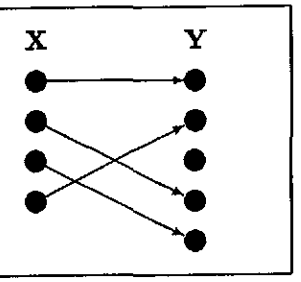
La parte central del estudio de un autómata reversible es su regla de evolución, ya que ésta nos dice como se comporta el sistema a través del tiempo, lo interesante es que esta regla de evolución es de influencia local, o sea sólo afecta las vecindades que forman parte de la configuración, sin embargo tal regla induce un mapeo global el cual es reversible.

De lo anterior se desprende que la evolución es una función que escencialmente transforma elementos de un conjunto a otro, donde tales elementos son las configuraciones globales que puede tener el autómata y la función el mapeo global producto de la regla de evolución.

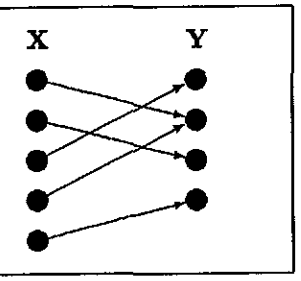
El comportamiento que tienen los autómatas celulares lineales reversibles depende de las propiedades de la función de mapeo global la cual puede determinar la reversibilidad. El mapeo que se lleva de un conjunto X a otro Y, es una función f que se clasifica en: inyectiva, sobreyectiva y biyectiva [21].

Una función $f : X \to Y$ se dice que es:

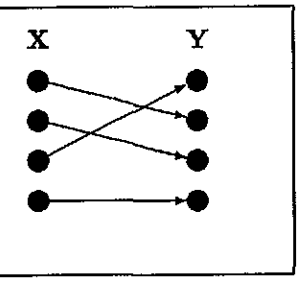
INYECTIVA:
Mapeo uno a uno, a cada miembro de Y le corresponde a lo más un miembro de X.



SOBREYECTIVA:
Mapeo sobre, a cada miembro de Y le le corresponde al menos un miembro de X.



BIYECTIVA:
Mapeo uno a uno y sobre, a cada miembro de Y le corresponde uno y sólo un miembro de X.



Si un mapeo $f : X \to Y$ es biyectivo, entonces $f^{-1} : Y \to X$ también define una función biyectiva; por lo tanto, caracterizar a los autómatas reversibles 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 *homeomorfismo* de X a X, es decir, conservan la topología del sistema; 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 autómata sea reversible, esta función debe definir un *automorfismo* (fig. 2.3).

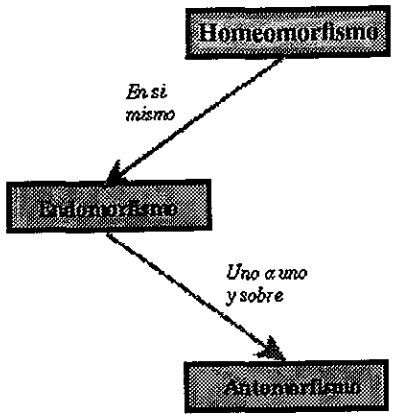


Figura 2.3: La regla de evolución de un autómata reversible define un automorfismo.

2.4 Tres conceptos fundamentales: Multiplicidad Uniforme, Indices de Welch y Combinabilidad

Tomando como base el trabajo de Hedlund [11] y Nasu [18], tres definiciones serán expuestas para caracterizar la regla de evolución de un autómata reversible, las cuales son *Multiplicidad Uniforme, Indices de Welch* y *Combinabilidad.*

2.4.1 Multiplicidad Uniforme

En el trabajo de Hedlund [11] sección 5 se demuestra que el mapeo global inducido a un autómata celular lineal es sobreyectivo sí y sólo sí cada mapeo local contenido en éste es también sobreyectivo.

La característica que cumple el mapeo local especificado en la regla de evolución es que el número de ancestros de cada cadena de estados es igual al número de posibles secuencias con un tamaño igual a dos veces el radio de vecindad o 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 reversible (5,h) (tabla 2.2).

	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 2.2: Regla de evolución 0077A76AIJOOI de un reversible (5,h).

Las configuraciones que pueden formar la cadena 0 son (fig. 2.4):

Figura 2.4: Configuraciones de tamaño 2 que evolucionan en 0.

Definamos las configuraciones que evolucionan en la cadena 02 (fig. 2.5).

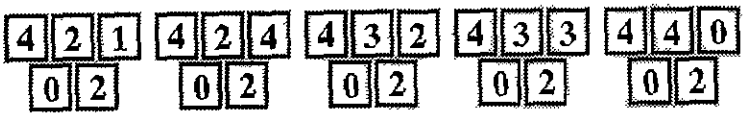


Figura 2.5: Configuraciones de tamaño 3 que evolucionan en 02.

Se utilizará esta cadena en particular para mostrar como funciona la Multiplicidad Uniforme, para esto se tomarán las siguientes definiciones:

- $\mathcal{A} = \{421, 424, 432, 433, 440\}$ al conjunto de configuraciones ancestros de 02
- $|\mathcal{A}|$ como la cardinalidad de $\mathcal{A}$
- $\mathcal{K} = \{0, 1, 2, 3, 4\}$ al conjunto de configuraciones diferentes de tamaño $2r$
- $|\mathcal{K}|$ como la cardinalidad de $\mathcal{K}$
- k como cualquier elemento de $\mathcal{K}$
- $Ancestros(\mathcal{B})$ al conjunto de ancestros de una cadena $\mathcal{B}$ dada

Tomando como base la cadena 02, la expresión $02k$ representa las secuencias formadas por la cadena 02 y cualquier elemento de $\mathcal{K}$ a la derecha de la misma (fig. 2.6).

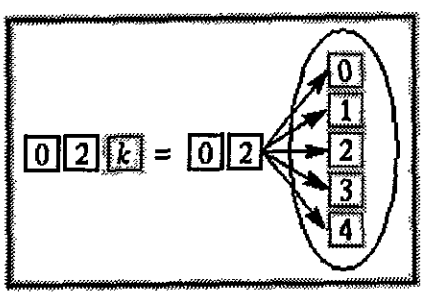


Figura 2.6: Secuencias de la forma $02k$.

Llamemos $\mathcal{AK}$ a todas las posibles cadenas formadas por la concatenación de las secuencias de $\mathcal{A}$ con las de $\mathcal{K}$, como el mapeo inducido por la regla de evolución es sobreyectivo, $\mathcal{AK} = Ancestros(02k)$, es decir, la evolución de todas las cadenas de $\mathcal{AK}$ producen todas las secuencias de la forma $02k$ sin que falte ninguna k (fig. 2.7).

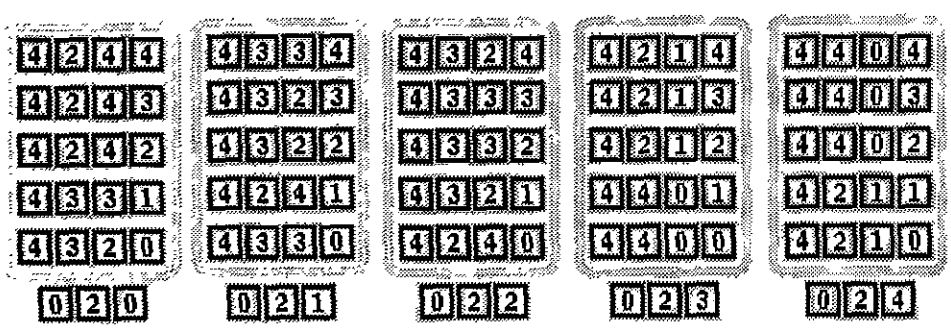


Figura 2.7: Conjunto de ancestros de las configuraciones $02k$.

En base a las definiciones tenemos que la cardinalidad de $\mathcal{AK} = |\mathcal{A}||\mathcal{K}|$.

Supongamos que la cardinalidad $Ancestros(02k) \geq |\mathcal{A}|$ para toda $k \in \mathcal{K}$; si se presenta que la cardinalidad de $Ancestros(02k) > |\mathcal{A}|$ para alguna k, tendríamos que la cardinalidad de $Ancestros(02k) > |\mathcal{A}||\mathcal{K}|$ o sea que la cardinalidad de $Ancestros(02k) >$ cardinalidad $\mathcal{AK}$ lo que no es posible; a continuación (fig. 2.8) se muestra un ejemplo de esta situación, si se presentara que la secuencia 022 (por tomar cualquiera) tuviera un número de ancestros mayor que las demás.

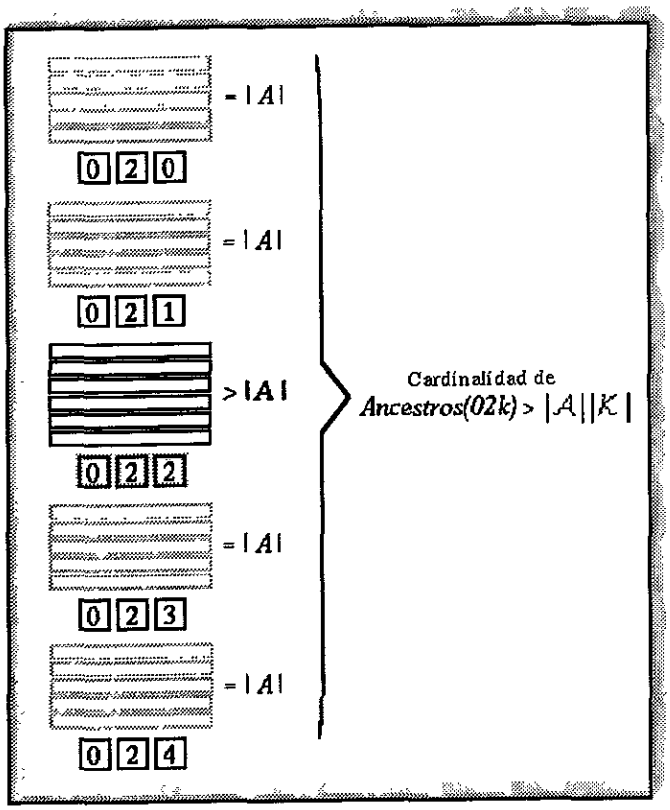


Figura 2.8: Si el número de ancestros de alguna secuencia $02k$ es mayor que $|A|$ se cae en una contradicción.

De forma análoga si para alguna k la cardinalidad de $Ancestros(02k) < |\mathcal{A}||\mathcal{K}|$, se tendría que la cardinalidad de $Ancestros(02k) <$ cardinalidad $\mathcal{AK}$ lo cual tampoco es posible; por lo que se concluye que la cardinalidad $Ancestros(02k) = |\mathcal{A}|$. En síntesis, hemos obtenido que el número de ancestros de la cadena 02 es el mismo para las extensiones adicionales que se hagan a la misma; este resultado se extiende para cualquier cadena.

Encontremos ahora cual es el valor que toma $|A|$ para cada caso; si formamos la cadena $02k02$ tenemos que $Ancestros(02k02) = \mathcal{AA}$ ya que al hacer cada posible concatenación de 2 cadenas de $\mathcal{A}$ obtenemos que la evolución de las mismas producen toda posible instancia de la cadena $02k02$, sin que falte ninguna k ya que el mapeo es sobreyectivo (fig. 2.9).

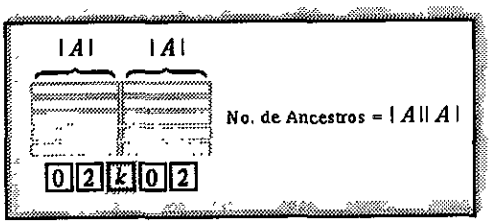


Figura 2.9: Ancestros de la configuración $02k02$.

La cardinalidad de $Ancestros(02k02) =$ cardinalidad de $\mathcal{AA} = |A||A|$.

Tomemos ahora cada elemento del conjunto $\mathcal{K}$ y formemos la cadena $02k02$ y sus posibles ancestros (fig. 2.10).

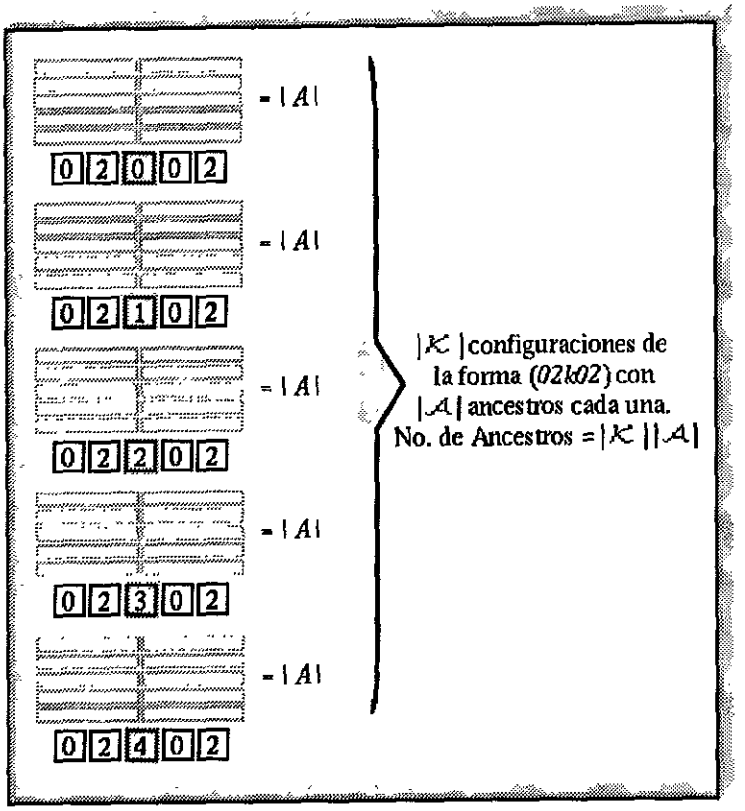


Figura 2.10: Cardinalidad de los ancestros de las secuencias de tipo $02k02$.

Como se mostró anteriormente, la cardinalidad de $Ancestros(02k02) = |A|$ para cualquier $k \in \mathcal{K}$; ya que la cardinalidad de $\mathcal{K} = |K|$, el número de cadenas distintas de la forma $02k02$ es $|K|$, cada una de éstas con $|A|$ ancestros; por lo que se obtiene que la cardinalidad de $Ancestros(02k02) = |K||A| = |A||A|$, de este modo $|A| = |K|$.

Sabemos que $\mathcal{K}$ 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 autómata 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.

2.4.2 Indices de Welch

La idea de los índices de Welch se debe a L. R. Welch, colaborador de Hedlund en [11]. Welch expone tres índices; L, R, y M, los cuales sirven para caracterizar la forma de los ancestros de una cadena de estados.

Primero definamos el índice R:

- Sea A cualquier bloque de estados.
- Sea $\mathcal{B}_i$ un conjunto de bloques de estados, todos de igual longitud (que puede ser mayor o igual que uno).
- Para que un bloque pertenezca a $\mathcal{B}_i$, debe cumplir que, al concatenarse a la derecha de A, produzca la misma cadena que el resto de los otros bloques al aplicarle la regla de evolución; es decir, todos los elementos del conjunto $A\mathcal{B}_i$ deben evolucionar en la misma cadena.
- Definamos r_i igual a la cardinalidad de $\mathcal{B}_i$, entonces se tomará el índice por la derecha $R = \max\{r_i\}, 1 \leq i \leq n, n \geq 1$ es decir, R toma el valor de la cardinalidad del conjunto $\mathcal{B}_i$ con mayor número de elementos

La definición de L es análoga:

- Sea A cualquier bloque de estados.
- Sea $\mathcal{B}_i$ un conjunto de bloques de estados, todos de igual longitud (que puede ser mayor o igual que uno).
- Para que un bloque pertenezca a $\mathcal{B}_i$, debe cumplir que, al concatenarse a la izquierda de A, produzca la misma cadena que el resto de los otros bloques al aplicarle la regla de evolución; es decir, todos los elementos del conjunto $\mathcal{B}_iA$ deben evolucionar en la misma cadena.
- Definamos l_i igual a la cardinalidad de $\mathcal{B}_i$, entonces se tomará el índice por la izquierda $L = \max\{r_i\}, 1 \leq i \leq n, n \geq 1$ es decir, L toma el valor de la cardinalidad del conjunto $\mathcal{B}_i$ con mayor número de elementos

A cada conjunto $\mathcal{B}_i$ se le dirá *compatible* con A ya sea por la derecha o por la izquierda según corresponda.

Definamos ahora el índice M:

- Sea A un bloque de estados.
- Sea $\mathcal{A}_i$ un conjunto cuyos elementos son bloques de estados A.
- Para que un bloque de estados A pertenezca a $\mathcal{A}_i$, debe cumplir que junto con sus conjuntos compatibles por la derecha y por la izquierda, evolucione en la misma cadena que el resto de los demás bloques en $\mathcal{A}_i$ bajo la regla de evolución; es decir, todos los elementos de $\mathcal{A}_i$ deben evolucionar en la misma cadena.
- Denotemos m_i igual a la cardinalidad de $\mathcal{A}_i$; entonces se tomará al índice $M = \min\{r_i\}, 1 \leq i \leq n, n \geq 1$ es decir, M toma el valor de la cardinalidad del conjunto $\mathcal{A}_i$ con menor número de elementos.

Ejemplifiquemos estos conceptos con un autómata (5,h) reversible (tabla 2.3).

	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 2.3: Regla de evolución 0067A26CIJOOI de un reversible (5,h).

Primero se obtendrá el índice R de este autómata; tomemos el estado 0 y busquemos sus conjuntos compatibles a la derecha, tal que la evolución sea 3 ó 4 (fig. 2.11).

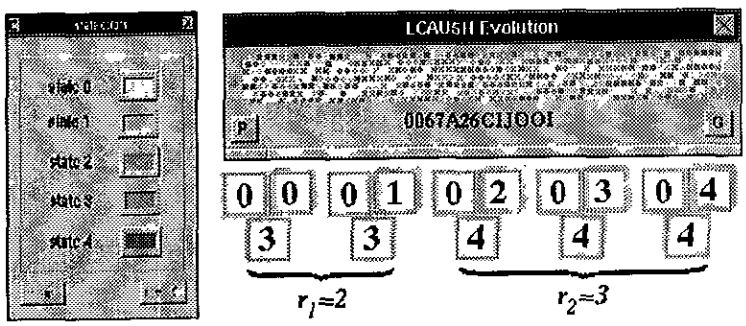


Figura 2.11: Regla de evolución 0067A26CIJOOI de un reversible (5,h), y las extensiones derechas de 0 que evolucionan en 3 y 4.

Para este caso $R = 3$ ya que r_2 es la cardinalidad del mayor conjunto compatible por la derecha de 0, ahora busquemos los conjuntos $\mathcal{B}_i$ de modo que la evolución sea 34 y 40 (fig. 2.12).

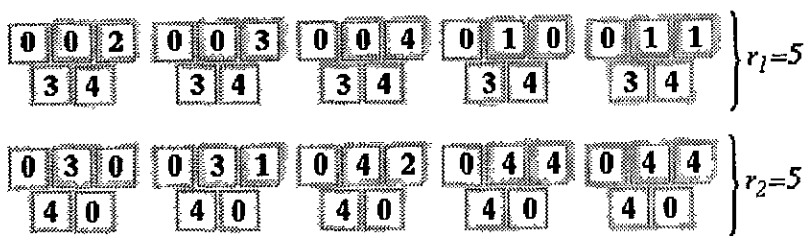


Figura 2.12: Conjuntos compatibles derechos con 0 que evolucionan en 34 y 40.

Ahora $R = r_1 = r_2 = 5$, si buscamos $\mathcal{B}$ compatible por la derecha tal que la evolución se 342 tenemos (fig. 2.13):

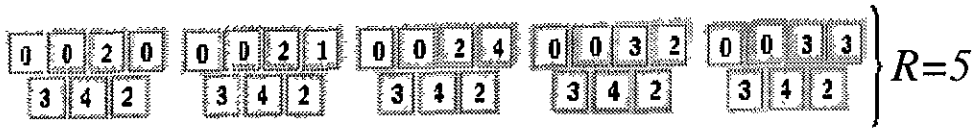


Figura 2.13: Conjunto compatible derecho con 0 que evoluciona en 342.

Podemos continuar de forma indefinida incrementado la longitud de los bloques del conjunto $\mathcal{B}$ compatible por la derecha del estado 0 y encontraremos que R se mantendrá con valor 5 para este autómata, lo mismo ocurre para todas los conjuntos compatibles derechos de los demás estados.

Ahora encontremos el valor del índice L; busquemos los conjuntos $\mathcal{B}_i$ compatibles por la izquierda del estado 0 tal que la evolución sea 3 y 4, observemos lo que se obtiene (fig. 2.14):

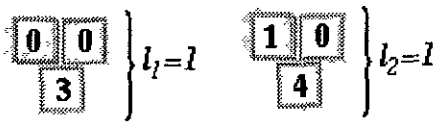


Figura 2.14: Conjuntos compatibles izquierdos con 0 que evolucionan en 3 y 4.

En este caso $L = l_1 = l_2 = 1$, extendamos la evolución a 23 ó 14 (fig. 2.15).

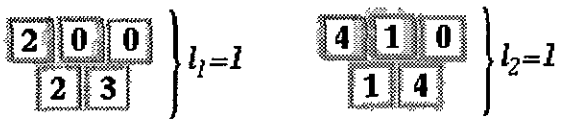


Figura 2.15: Conjuntos compatibles izquierdos con 0 que evolucionan en 23 y 14.

Se observa que L sigue siendo 1, si seguimos buscando para el estado 0 los conjuntos compatibles por la izquierda de una mayor longitud, encontraremos que L seguirá siendo 1; esto ocurre para todos los conjuntos compatibles izquierdos de los demás estados.

A continuación obtengamos el valor del índice M; tomemos como ejemplo todas las posibles cadenas que evolucionen en 231 (fig. 2.16).

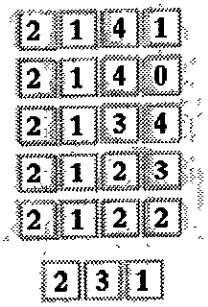


Figura 2.16: Ancestros de la cadena 231.

En este caso $M = 1$ ya que hay un sólo elemento (el estado 1) el cual cumple que su evolución junto con sus conjuntos compatibles por derecha y por izquierda genere la cadena 231.

Una propiedad fundamental que Hedlund y Welch postulan es que si un autómata (k,r) define un mapeo global sobreyectivo, entonces se cumple que la multiplicación de $LMR = k^{2r}$ o igual al número de nodos en el diagrama de de Bruijn, veamos como funciona esto para la cadena 130 y sus posibles ancestros (fig. 2.17).

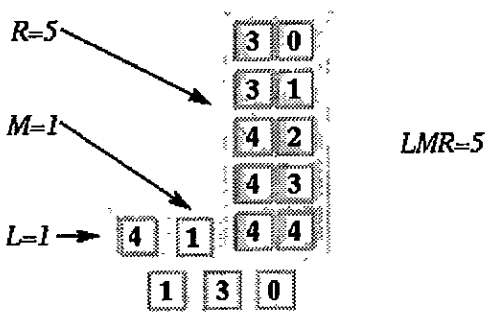


Figura 2.17: Ancestros de la cadena 130.

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 esto con otros autómatas reversibles (fig. 2.18).

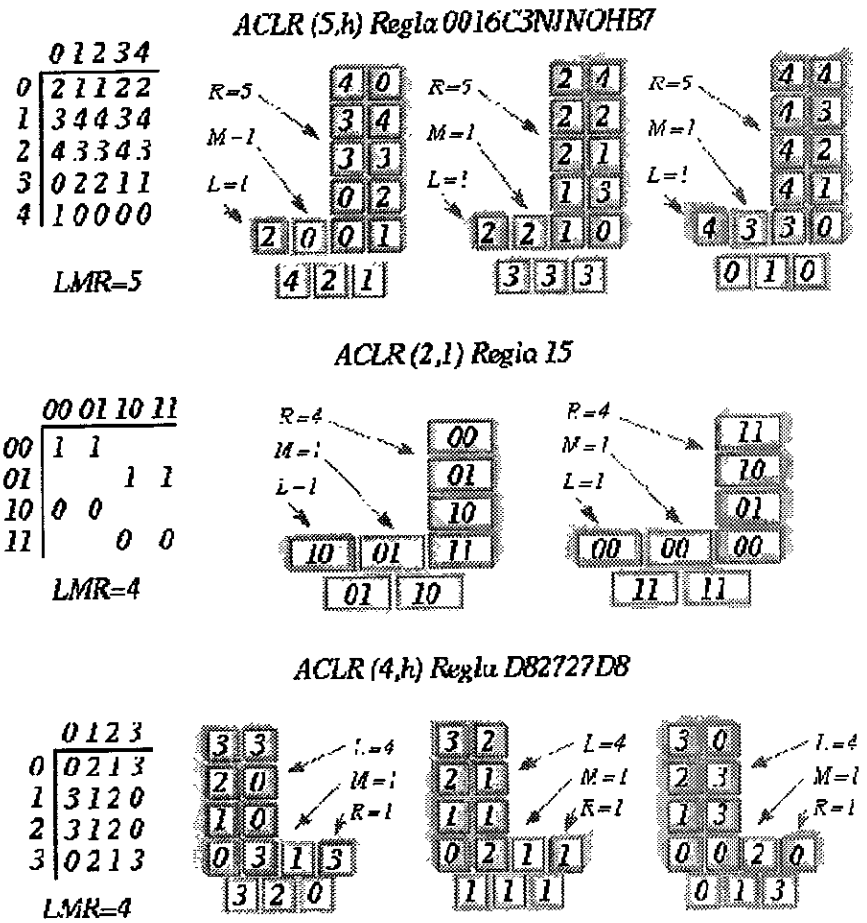


Figura 2.18: Ejemplos de autómatas reversibles y sus valores de LMR.

Por supuesto, no siempre L ó R deben ser igual a 1, ya que hay casos en el que el valor k^{2r} 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^{2r} es primo. A continuación se presentan algunos ejemplos de autómatas reversibles donde este valor no es primo (fig. 2.19).

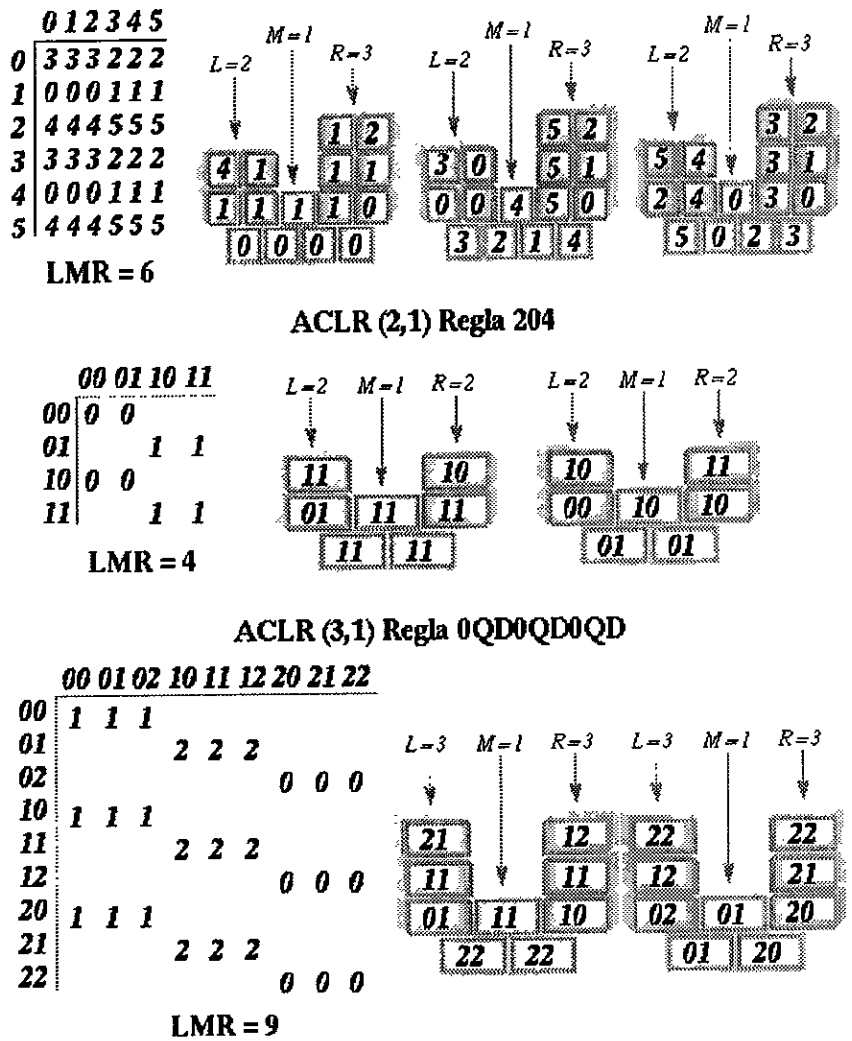


Figura 2.19: Ejemplos de autómatas reversibles con valores de LMR no primos.

2.4.3 Combinabilidad

En la sección 16 del trabajo de Hedlund [11] y en el trabajo de Nasu [18], el concepto de combinabilidad se define como sigue:

- Un mapeo local es *debilmente combinable* si para toda cadena dada de estados de longitud $l = 2r$ el conjunto $\mathcal{B}$ compatible por la derecha tiene una cardinalidad igual a R o a cero y el conjunto $\mathcal{C}$ compatible por la izquierda tiene una cardinalidad igual a L o a cero.

Ilustremos esta definición con un autómata (5,h) no reversible (fig. 2.20).

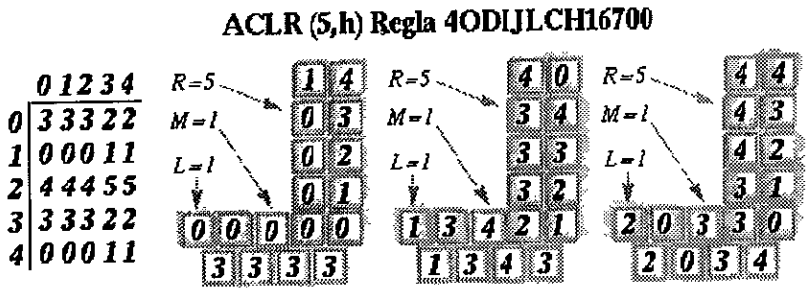


Figura 2.20: Autómata no reversible (5,h) con algunos valores de $LMR = k^{2r}$.

Aquí se observa que este autómata 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 (fig. 2.21).

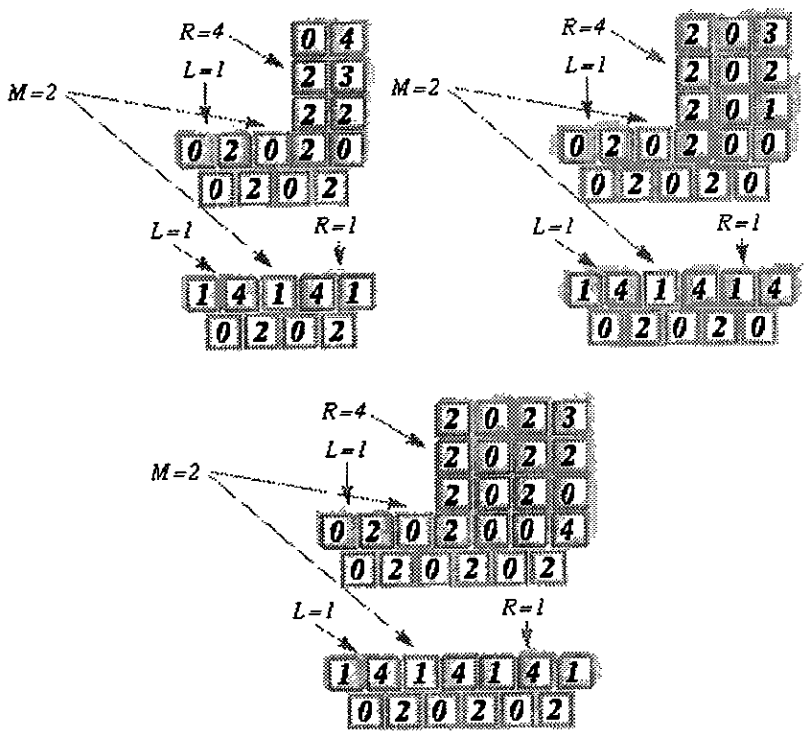


Figura 2.21: Ancestros de la cadena formada alternando los estados $0, 2$.

En este caso una cadena de este tipo puede ser generada por los conjuntos compatibles de dos estados distintos (0 y 1), es decir, se tiene que $M = 2$; además, un estado tiene a $R = 4$ y el otro tiene a $R = 1$; por lo que este autómata tiene múltiples ancestros para esta cadena en particular, lo que impide que sea reversible. Lo mismo sucede con los ancestros de la cadena formada alternando los estados $1, 4$ (fig. 2.22).

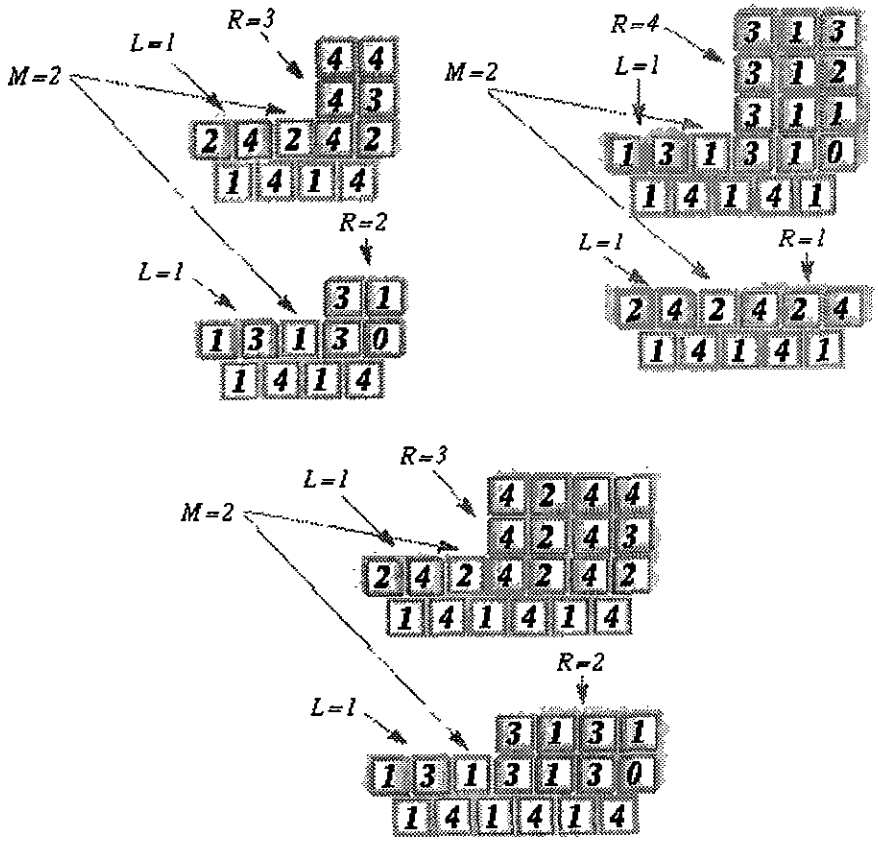


Figura 2.22: Ancestros de la cadena formada alternando los estados $1, 4$.

De este modo tenemos que las condiciones que debe cumplir un autómata celular lineal para ser reversible son las siguientes:

- Los ancestros de las cadenas con longitud mayor o igual a $2r$ deben tener todos índices L y R tal que $LR = k^{2r}$ (multiplicidad uniforme).
- Cada ancestro debe estar formado por los conjuntos compatibles de una cadena de estados de longitud igual a $2r$, en donde se cumpla que la cardinalidad del conjunto $\mathcal{B}$ compatible por la derecha sea igual R y la cardinalidad del conjunto $\mathcal{C}$ compatible por la izquierda sea igual a L.
- El valor del índice M para los ancestros de toda cadena de longitud mayor o igual a $2r$ debe ser siempre igual a uno (un único elemento en el cual regresar atrás en la evolución).

Tomemos como ejemplo un reversible (5,h) (fig. 2.23).

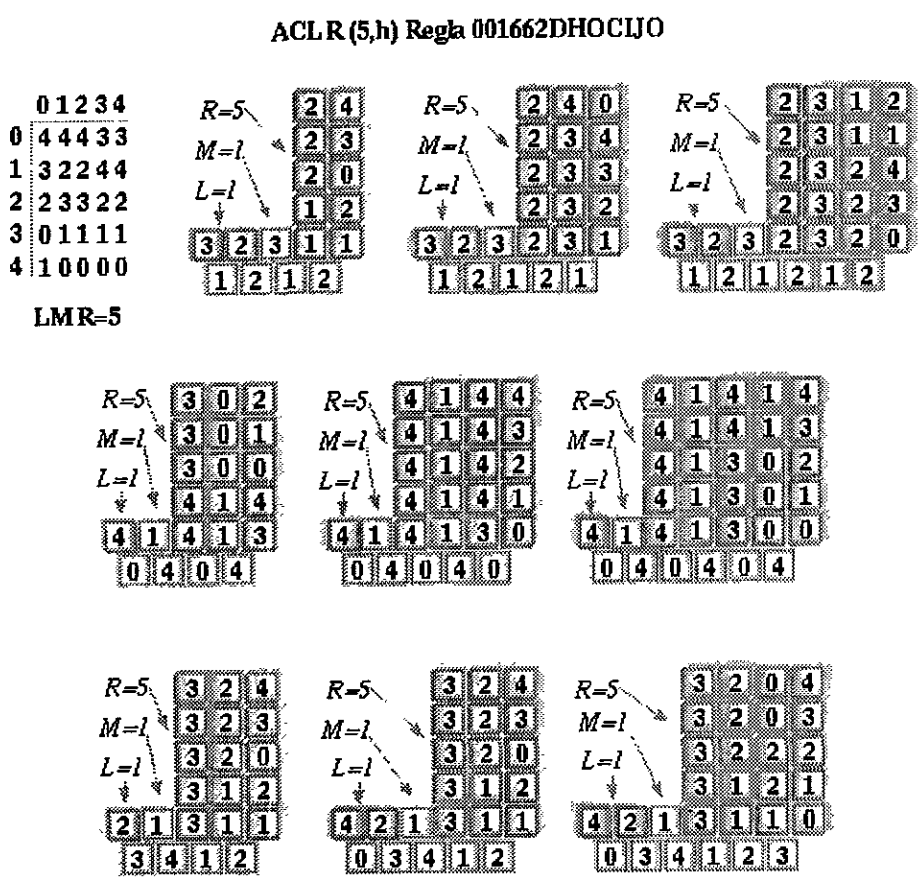


Figura 2.23: Reversible (5,h) donde el valor de M siempre es igual a 1.

La propiedad de que la regla de evolución de un autómata celular lineal 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 conjuntos compatibles derechos e izquierdos, evitando así que otra secuencia de estados evolucione en la misma cadena y ésta tenga múltiples ancestros.

2.5 Análisis de los Autómatas Celulares Lineales Reversibles mediante las herramientas gráficas

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 autómata para que éste sea invertible.

2.5.1 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 autómata celular; tomando en cuenta las características presentadas para definir un mapeo global biyectivo, podemos deducir que un autómata 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 ésto significaría que una cadena dada tiene varias formas de producirse con diferentes estados, es decir, ancestros múltiples (fig. 2.24).

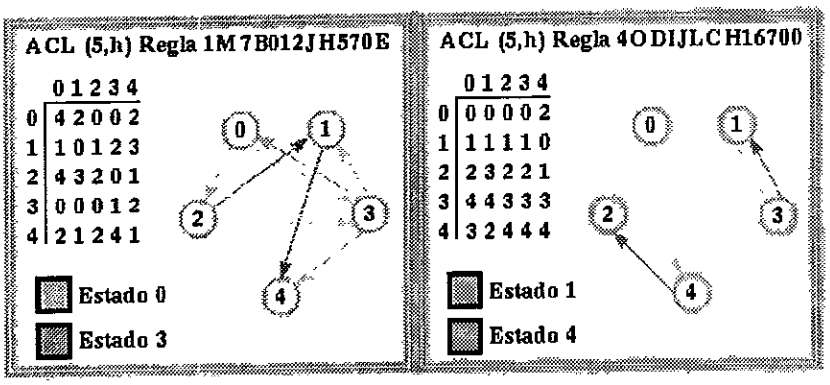


Figura 2.24: Autómata (5,h) no reversibles y parte de sus diagramas de de Bruijn asociados.

En la gráfica superior se observa que en el caso del autómata de la izquierda la liga del estado 0 aparece más veces que la liga del estado 3 lo que se vió que era imposible para un autómata reversible por la definición de multiplicidad uniforme, en el caso del autómata 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 multiples ancestros.

Se definió que un autómata 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 unicamente en el nodo x debemos tener L maneras distintas o ninguna, con $LR = k^{2r}$ (fig. 2.25).

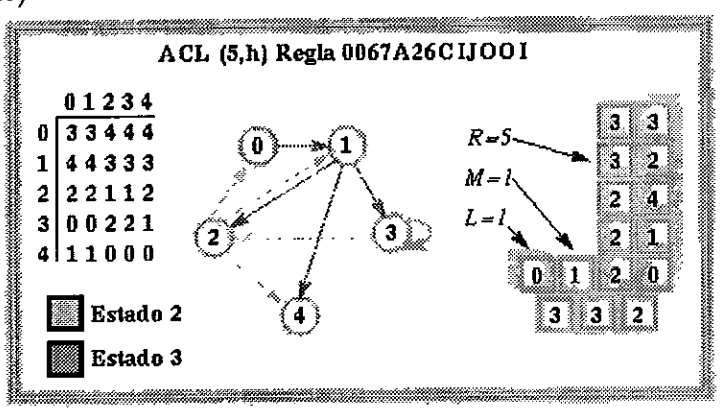


Figura 2.25: Reversible (5,h) regla 0067A26CIJOOI y la ruta 332 en su diagrama de de Bruijn.

Como se observa en la fig. 2.25 en el diagrama de de Bruijn asociado al reversible (5,h) regla 0067A26CIJOOI existen 5 posibles formas de generar la ruta 32 empezando desde el nodo 1 ($R = 5$) y sólo 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 sólo las extensiones compatibles por la izquierda y por la derecha del nodo 1 son las que producen dicha ruta.

En un autómata reversible toda ruta debe poderse formar en su diagrama de de Bruijn asociado, es decir no debe existir el Jardín del Edén, pero cada ruta de longitud $\geq 2r$ sólo debe

poderse formar llegando de L maneras distintas a un único nodo 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.

2.5.2 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 (fig. 2.26).

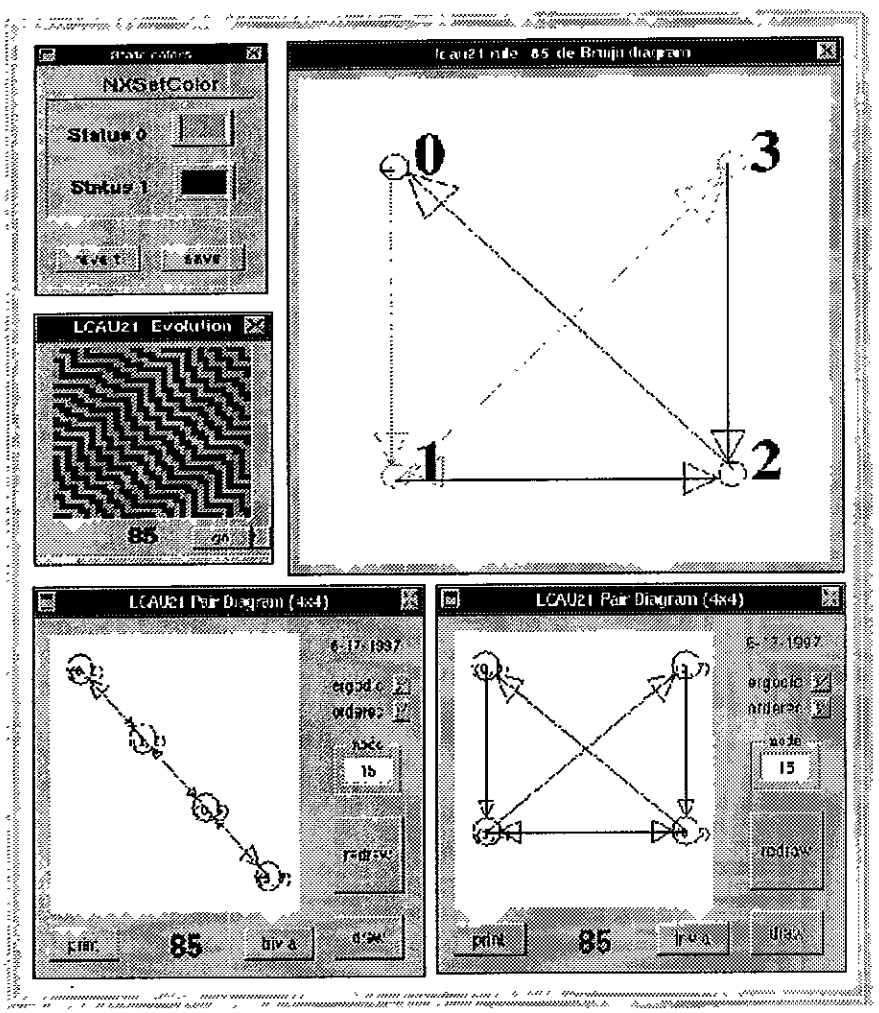


Figura 2.26: Reversible (2,1) regla 85 y su diagrama de parejas asociado.

En el esquema se observa un reversible (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 reacomodar 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 autómata no es reversible (fig. 2.27).

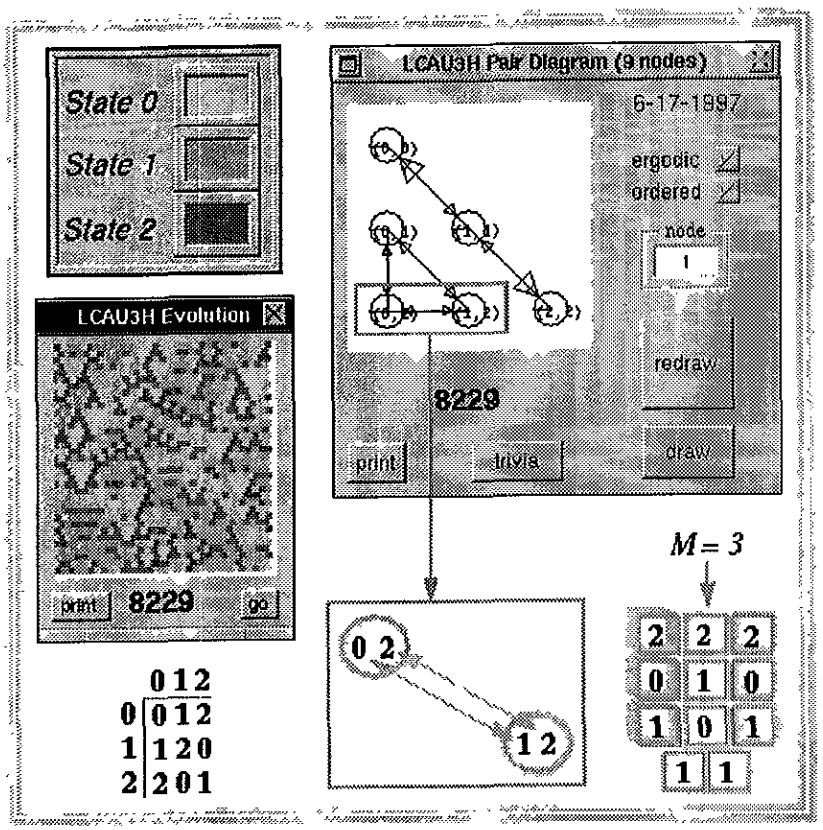


Figura 2.27: Reversible (3,h) regla 8229 y su diagrama de parejas asociado.

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 autómata 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 autómata define un mapeo global inyectivo o no.

2.5.3 Características en el 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 autómata más que en la configuración inicial; el diagrama de subconjuntos indica de manera muy clara cuales son estas cadenas. Esto se observa si existe una ruta que vaya desde el subconjunto de máxima cardinalidad hasta el conjunto vacio y la secuencia de arcos de tal ruta es justamente la cadena que pertenece al Jardín del Edén de dicho autómata. 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 vacio), ejemplifiquemos esto con un autómata (2,1) regla 231 (fig. 2.28).

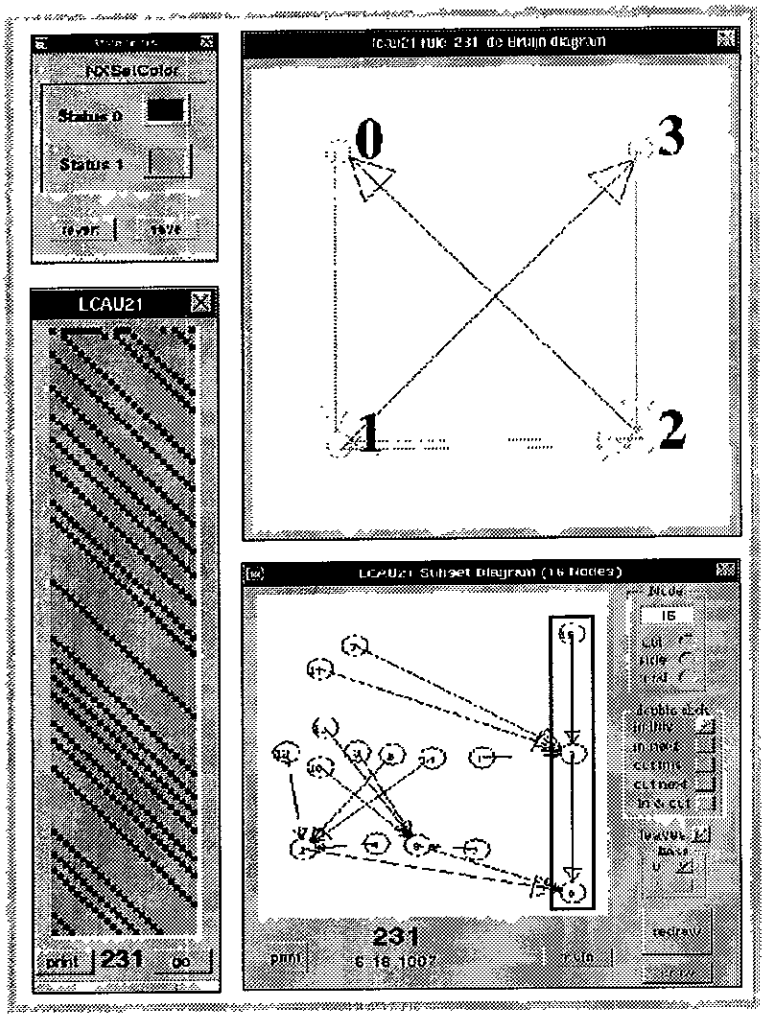


Figura 2.28: Autómata (2,1) regla 231 y su diagrama de subconjuntos.

En el autómata $(2,1)$ (fig. 2.28) se observa 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 más alto al conjunto vacío, y como se nota en la evolución del autómata, las cadenas formadas por dos o más $0's$ aparecen sólo en la configuración inicial; el diagrama de subconjuntos nos indica que configuraciones pertenecen al Jardín del Edén, es decir, si el autómata 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 autómata, en especial de los índices de Welch, 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 (fig. 2.29).

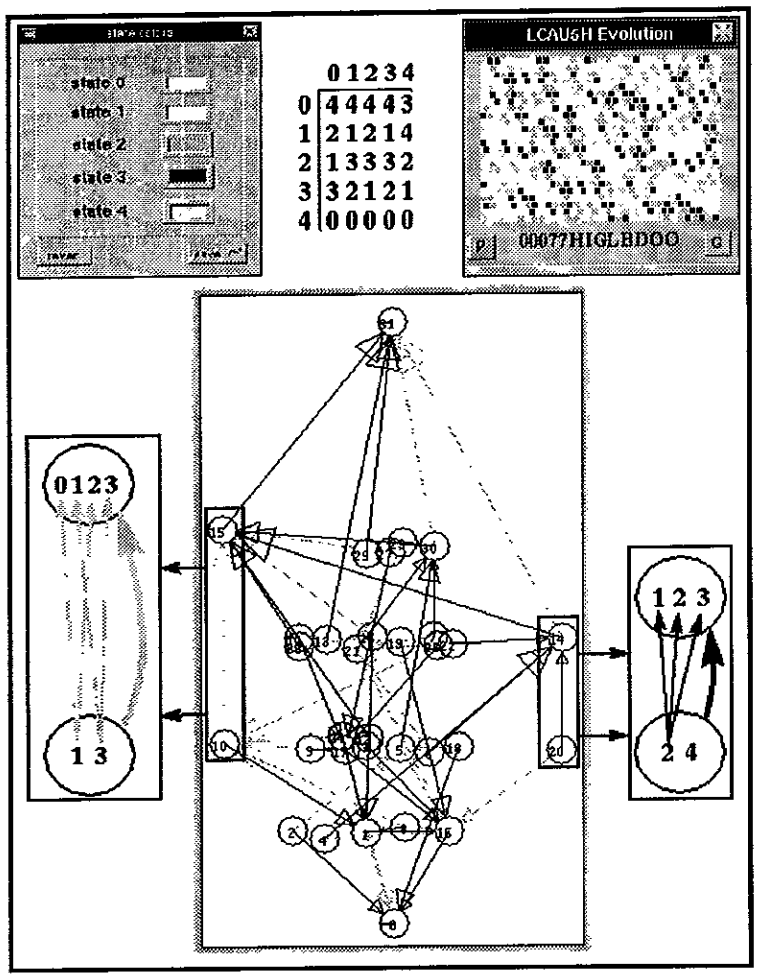


Figura 2.29: Reversible (5,h) regla 00077HIGLBDOO y su diagrama de subconjuntos.

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 autómata 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 (fig. 2.30).

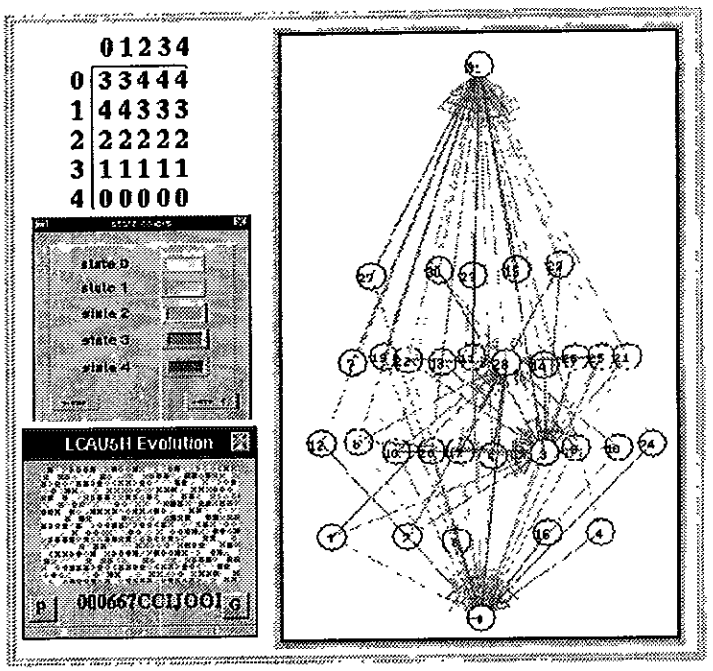


Figura 2.30: Reversible (5,h) regla 000667CCIJOOI y su diagrama de subconjuntos.

En la fig. 2.30 de un reversible (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 (fig. 2.31).

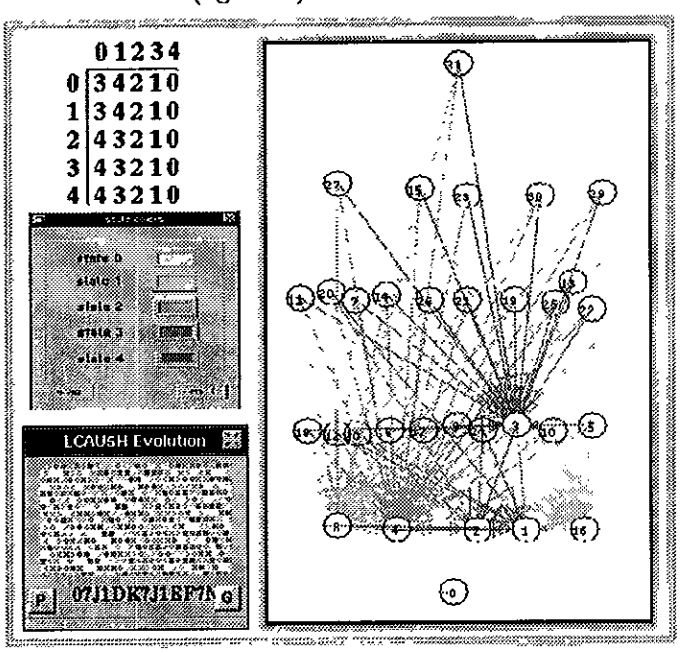


Figura 2.31: Reversible (5,h) reflexión de la regla 000667CCIJOOI y su diagrama de subconjuntos.

En la fig. 2.31 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 reversible (5,h).

Para saber por medio del diagrama de subconjuntos si un autómata 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 $LR = k^{2r}$.

2.6 Diagramas de Welch.

Como hemos visto, el diagrama de subconjuntos de un autómata reversible nos ofrece la posibilidad de conocer el valor del índice R así como el del índice L por medio de la reflexión; pero aún podemos obtener mayor información de estos diagramas si tomamos de los niveles R y L respectivamente aquellos conjuntos de nodos en donde las rutas tienden a llegar y ya nunca salir, es decir, son autocontenidos; a cada conjunto de estos nodos se les denominará *Diagramas de Welch* derecho e izquierdo correspondiendo respectivamente a los diagramas de subconjuntos de la regla original y su reflexión.

Ejemplifiquemos éstos por medio de un autómata reversible (4,h) regla EB28D714, obtengamos sus diagramas de subconjuntos asociados y de éstos los diagramas de Welch derecho e izquierdo (fig. 2.32).

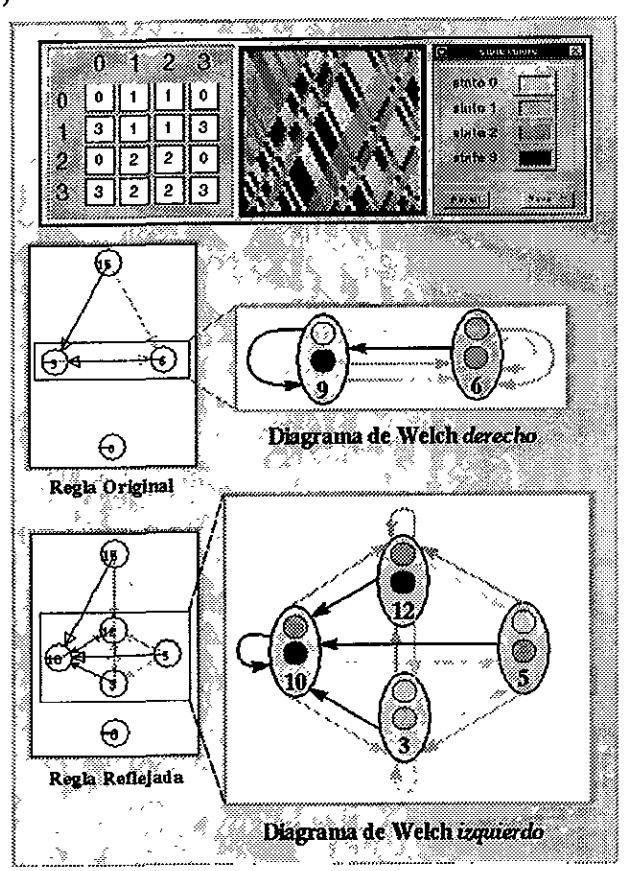


Figura 2.32: Reversible (4,h) regla EB28D714 y sus diagramas de Welch.

Como se observa en la fig. 2.32, en ambos diagramas cada estado define caminos que conllevan a un único nodo para jamás salir del mismo, es decir, forman árboles tal como se especifica en [18], la suma de las alturas máximas de estos árboles que se observan en ambos diagramas señala el tamaño de vecindad de la regla inversa; pero además cada posible forma de construir una ruta dada de cierta longitud llega a alcanzar un único nodo intermedio; es decir, por medio de estos diagramas podemos conocer la regla inversa a la original.

En el ejemplo anterior observamos que en ambos casos la máxima altura que se puede observar es para cada estado es 1, por lo que el tamaño de vecindad de la regla inversa es 2, si queremos conocer ahora en que debe evolucionar la vecindad 03 en la regla inversa, tomemos del diagrama de Welch derecho el nodo en donde convergen las ligas del estado 0 y del diagrama de Welch izquierdo el nodo donde convergen las ligas del estado 3, después tomemos el elemento en común en ambos nodos y obtendremos así en que debe evolucionar la vecindad 03 en la regla inversa (en este caso, el estado 3); repitiendo el proceso para las demás secuencias de longitud 2 obtenemos la regla inversa (4,h) F06666F0 (fig. 2.33).

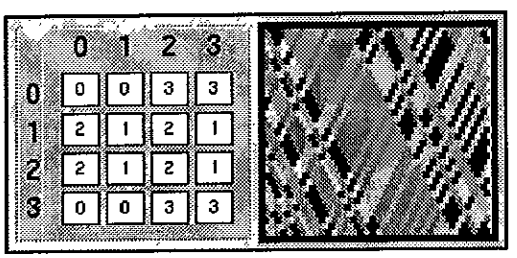


Figura 2.33: Reversible (4,h) regla F06666F0 el cual es la inversa de la regla EB28D714.

2.7 Ejemplos de Autómatas Celulares Lineales no Reversibles

A continuación se presentan tres ejemplos de autómatas no reversibles, en los cuales se observan distintos comportamientos. En el autómata de la fig. 2.34, existe una configuración con múltiples ancestros lo cual se puede visualizar en los diagramas de parejas y de subconjuntos.

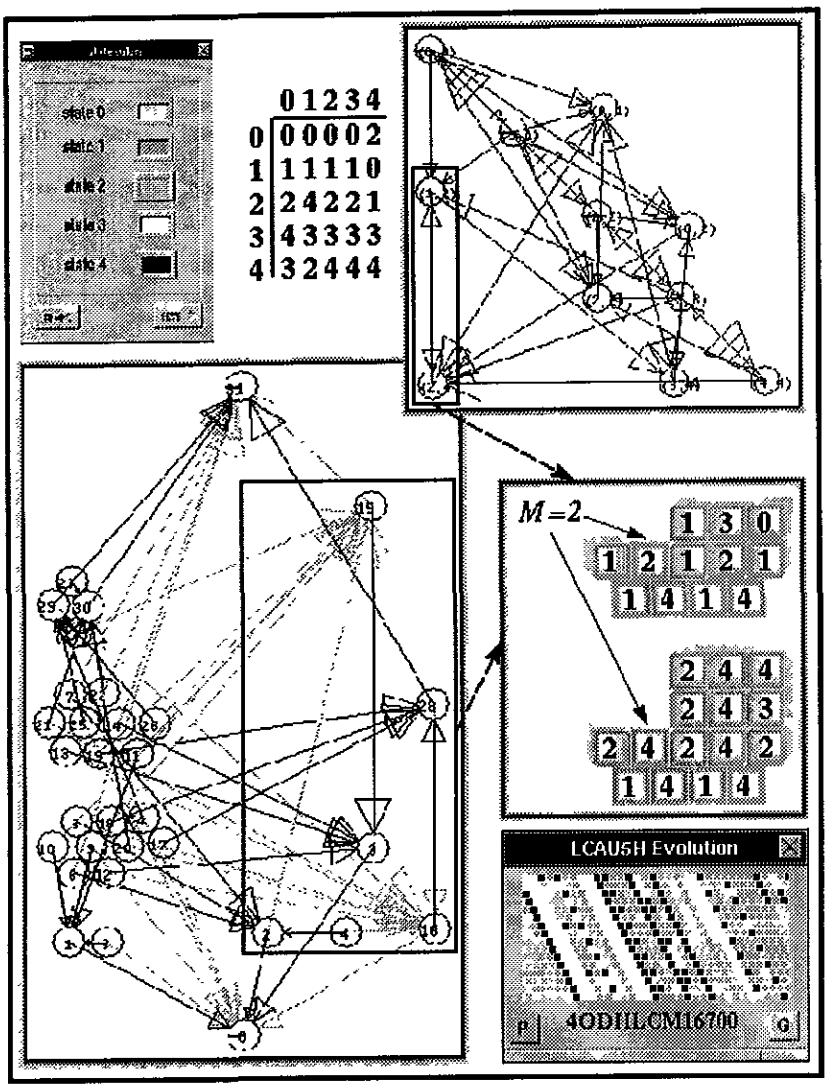


Figura 2.34: Autómata (5,h) regla 4ODIJLCH16700 donde existen ciclos en el diagrama de subconjuntos y en el de parejas.

En el autómata de la fig. 2.35 existe una secuencia de estados que pertenece al Jardín del Edén, lo cual se observa claramente en la ruta que existe desde el nivel más alto al conjunto vacío en el diagrama de subconjuntos.

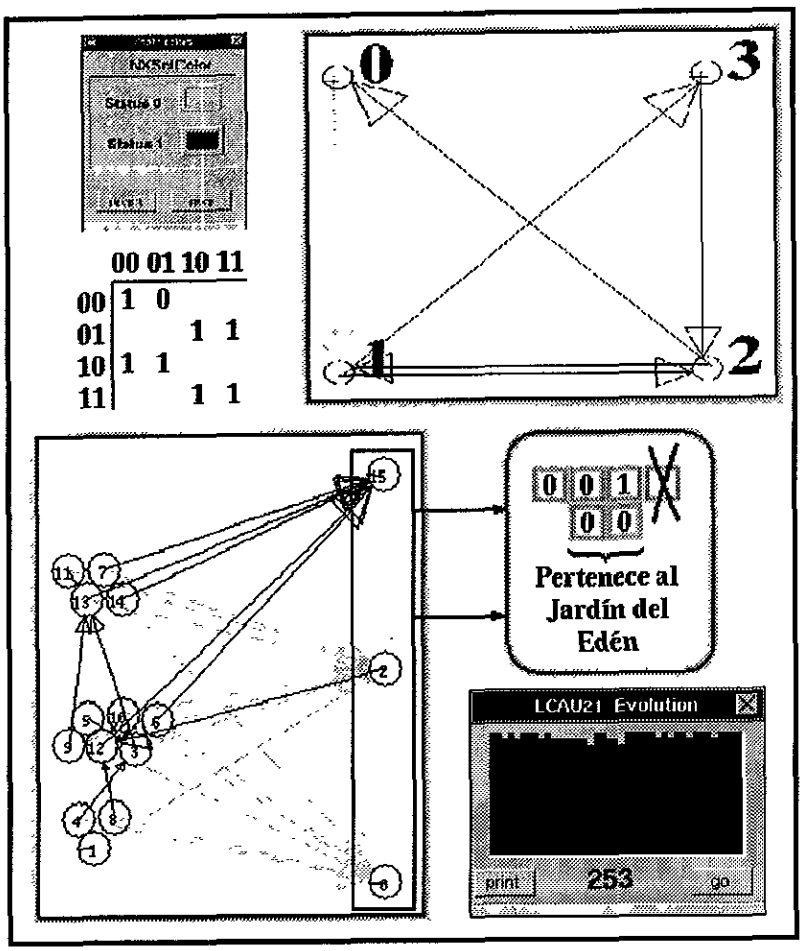


Figura 2.35: Autómata (2,1) regla 253 en donde la cadena de estados 00 pertenece al Jardín del Edén.

Otro ejemplo de un autómata no reversible (fig. 2.36) pero que cumple con la multiplicidad uniforme, cada secuencia de estados tiene el mismo número de ancestros que las demás, pero el valor de su índice M es mayor que uno en todos los casos, es decir, cada secuencia tiene varias formas de regresar hacia atrás en la evolución por lo que no se puede especificar una reversibilidad única.

Este comportamiento es muy notorio sobre todo en el diagrama de subconjuntos, donde ni existe el Jardín del Edén, pero cada cada nivel es disjunto a los demás, señalando un valor de R igual con uno.Ejemplos típicos de estas reglas de evolución son la suma modulo k de los elementos de las vecindades (este no es el caso del ejemplo).

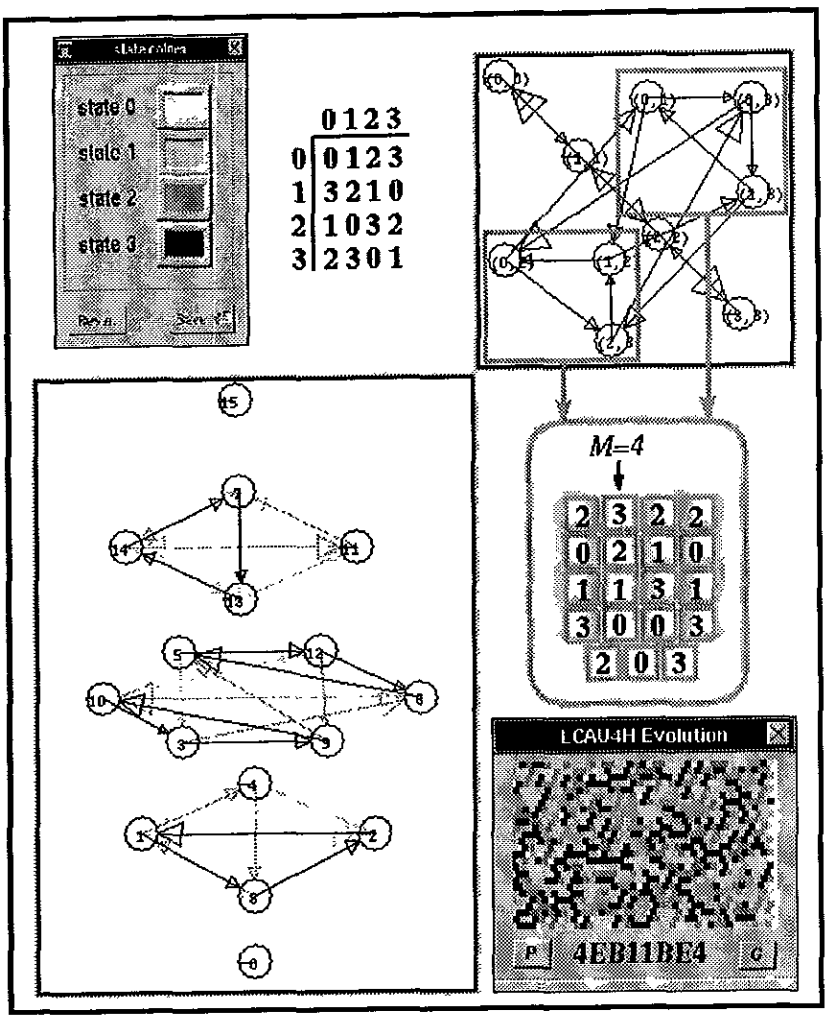


Figura 2.36: Autómata (4,h) regla 4EB11BE4 el cual tiene un mapeo global sobreyectivo pero no reversible.

2.8 Ejemplos de Autómatas Celulares Lineales Reversibles

Ahora se muestran ejemplo de autómatas celulares lineales reversibles, donde los diagramas nos muestran los valores de los tres índices L, M y R, cumpliendo que $LR = k^{2r}$ y $M = 1$ (fig. 2.37 - 2.39).

Los valores de R y L se obtienen observando hasta que nivel llegan todas las rutas partiendo de las clases unitarias en los diagramas de subconjuntos de la regla original y su inversa respectivamente.

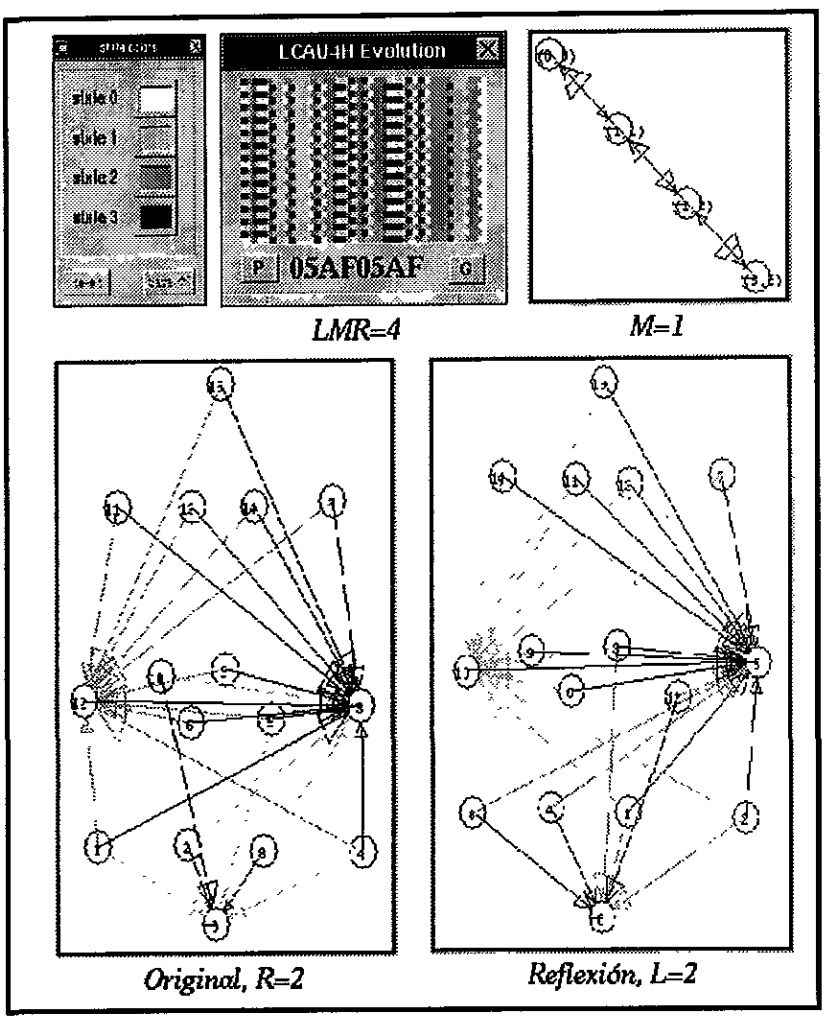


Figura 2.37: Reversible (4,h) regla 05AF05AF.

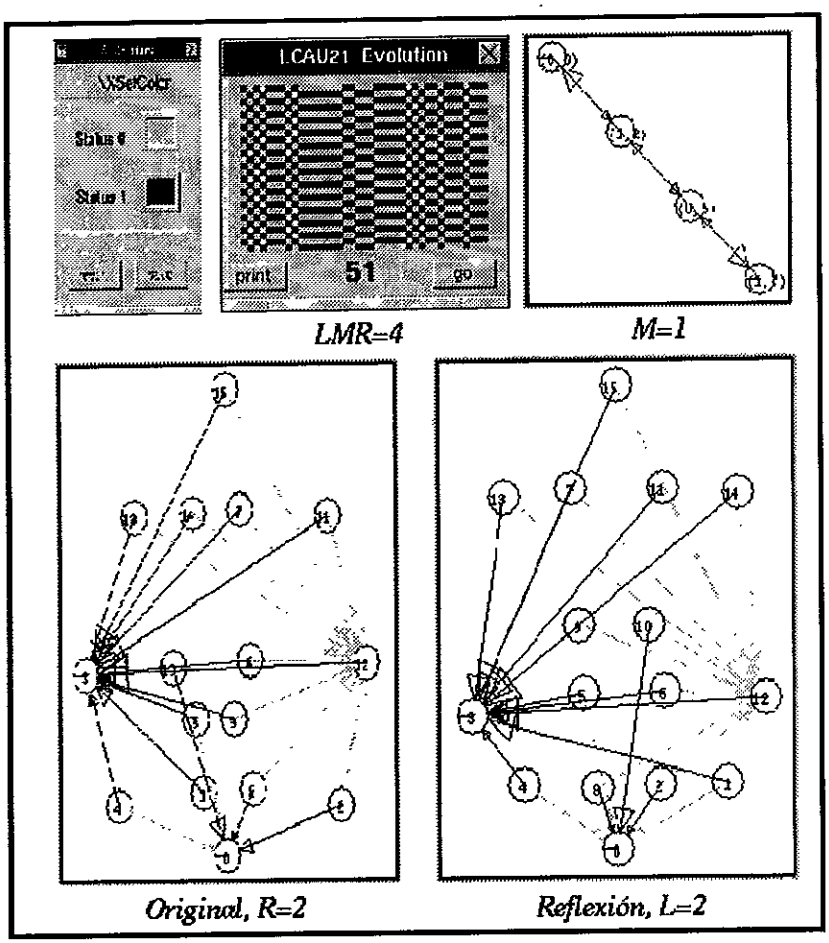


Figura 2.38: Reversible (2,1) regla 51.

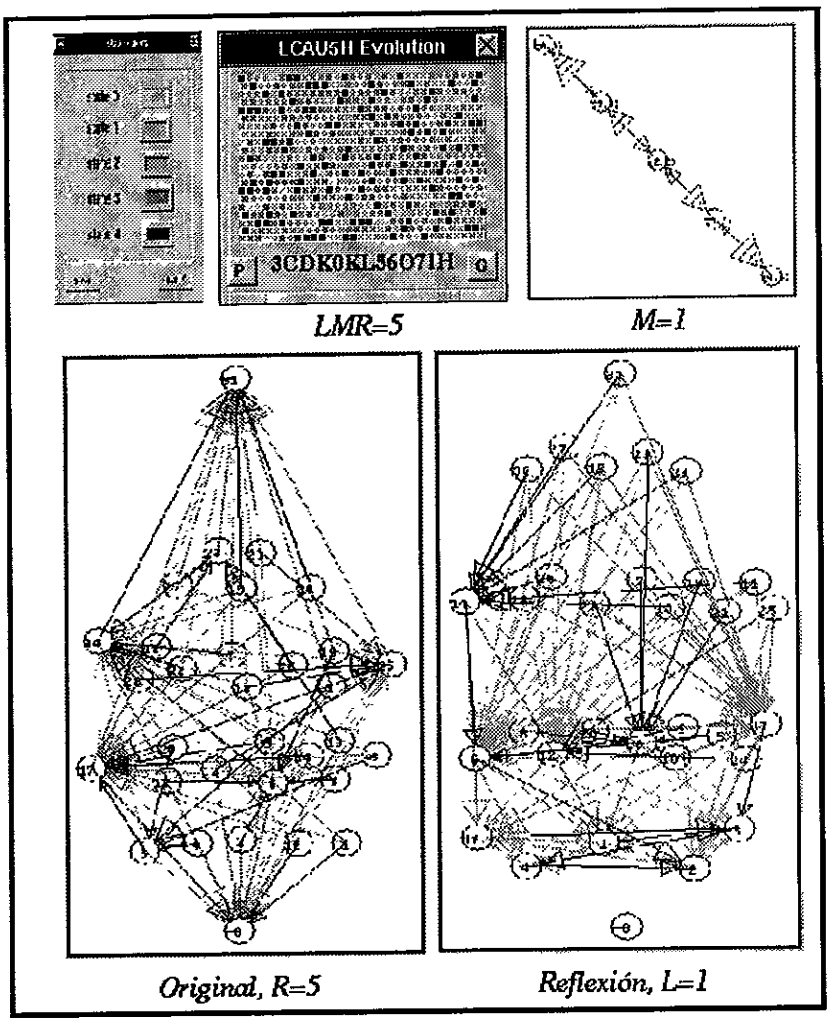


Figura 2.39: Reversible (5,h) regla 3CDK0KL56O7IH.

Capítulo 3

Métodos actuales para encontrar Autómatas Celulares Lineales Reversibles.

A continuación se presentan los fundamentos y el funcionamiento de dos métodos (uno terminado y otro en desarrollo) que existen para encontrar los posibles autómatas celulares lineales reversibles.

El primer algoritmo se debe a David Hillman [12], su trabajo aunque no lo menciona directamente, hace un uso implícito de las propiedades expuestas en el capítulo anterior.

El segundo algoritmo se debe al trabajo de Jose Manuel Gómez Soto y Harold V. McIntosh [8], este trabajo es un primer intento de utilizar explicitamente las propiedades de multiplicidad uniforme y los índices de Welch para el cálculo de autómatas reversibles.

3.1 Algoritmo de Hillman.

En 1991, David Hillman [12] desarrolló un algoritmo para detectar todos los autómatas reversibles con culaquier número de estados y cualquier tamaño de vecindad, el proceso es el siguiente:

a) Iniciamos tomando cada cadena y de tamaño w, se empieza con $w = 1$.

b) Formamos todos los posibles ancestros x_i de cada cadena y.

c) Se almacenan en una tabla las parejas formadas por los $2r$ elementos iniciales y finales de cada x_i, una tabla para cada posible y.

d) 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.

e) 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.

f) Continuando con 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.

g) Si el autómata es reversible para cadenas de longitud w, 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 sólo una cadena x_i^* de longitud w puede evolucionar en y para toda y de longitud w, es decir, cada posible cadena sólo 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:

- Si la tablas de un conjunto R_i no tienen todas el mismo número de parejas entonces el autómata no es reversible pues este desequilibrio llevará a que la información del sistema se pierda gradualmente y con esto la reversibilidad del mismo.
- Como los valores que pueden tener cada tabla son finitos, entonces llegará un momento en que se repetirán estos valores; en síntesis, si en cada tabla R_j se cumple que el autómata es reversible; cada tabla tiene el mismo número de parejas para $j \geq 1$ y $R_j = R_i$ para $i < j$, entonces el autómata es reversible y el proceso termina.

Veamos un ejemplo de este algoritmo para un autómata (3,h) (fig. 3.1).

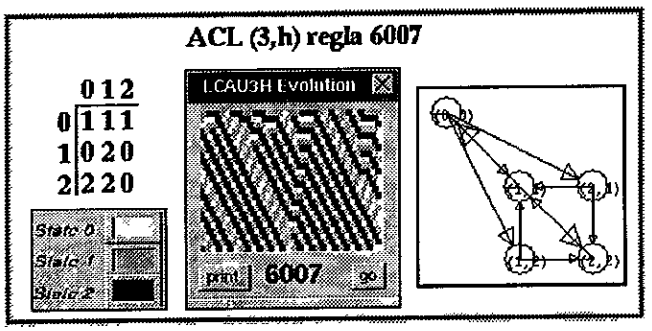


Figura 3.1: Autómata (3,h) regla 6007.

Formemos las tablas de los $2r$ elementos iniciales y finales para los ancestros de 0, 1 y 2 (tabla 3.1).

Ancestros de 0	Ancestros de 1	Ancestros de 2
1 0	**0 0**	**1 1**
1 2	0 1	2 0
2 2	0 2	2 1

Tabla 3.1: Conjunto R_1 para el autómata (3,h) regla 6007.

Para R_1 se encuentra que cada posible cadena tiene un sólo ancestro, ahora concatenemos R_1 consigo mismo para obtener los ancestros de 00, 01 y 02 (tabla 3.2).

Ancestros de 00	Ancestros de 01	Ancestros de 02
1 2	1 0	1 0
2 2	**1 1**	**1 1**
	1 2	2 0
		2 1

Tabla 3.2: 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 el autómata (3,h) no es reversible.

Veamos otro ejemplo para otro autómata (3,h) (fig. 3.2).

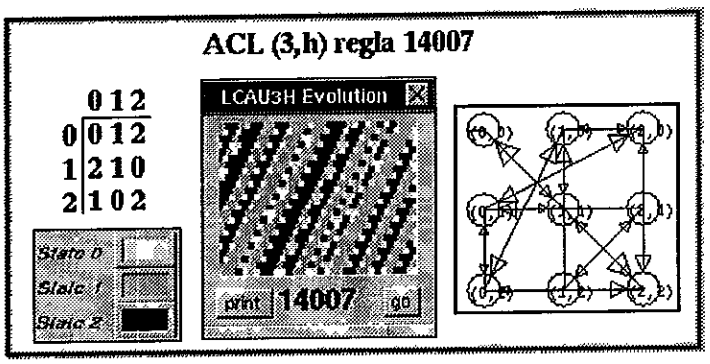


Figura 3.2: Autómata (3,h) regla 14007.

Formemos las tablas de los $2r$ elementos iniciales y finales para los ancestros de 0, 1 y 2 (tabla 3.3).

Ancestros de 0	Ancestros de 1	Ancestros de 2
0 0	1 0	0 2
1 2	**1 1**	1 0
2 1	2 0	**2 2**

Tabla 3.3: Conjunto R_1 para el autómata (3,h) regla 14007.

Para R_1 también encontramos que cada posible cadena tiene un sólo ancestro, ahora concatenemos R_1 consigo mismo para obtener los ancestros de 00, 01 y 02 (tabla 3.4).

Ancestros de 00	Ancestros de 01	Ancestros de 02
0 0	2 0	0 2
1 1	2 1	2 0
2 2	1 0	1 2

Tabla 3.4: Concatenación de R_1 con R_1.

Aquí se observa que la cadena 00 tiene tres posibles ancestros con los $2r$ primeros elementos iguales a los $2r$ últimos, mientras que las cadenas 01 y 02 no tienen ningún ancestro con esta característica, así que se detiene el algoritmo y el autómata (3,h) no es reversible.

Veamos otro ejemplo para otro autómata (3,h) (fig. 3.3).

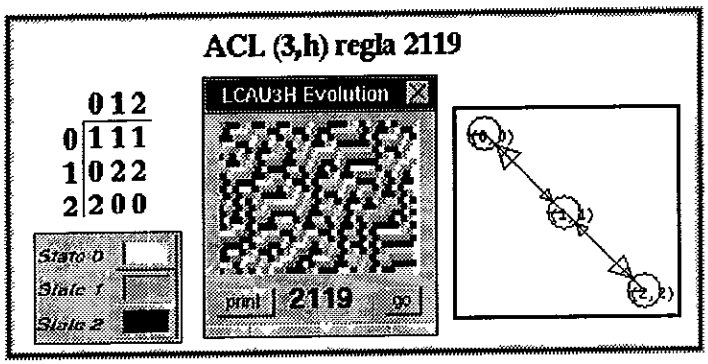


Figura 3.3: Autómata (3,h) regla 2119.

Formemos las tablas de los $2r$ elementos iniciales y finales para los ancestros de 0, 1 y 2 (tabla 3.5).

Ancestros de 0	Ancestros de 1	Ancestros de 2
1 0	**0 0**	2 0
2 1	0 1	**1 1**
2 2	0 2	1 2

Tabla 3.5: Conjunto R_1 para el autómata (3,h) regla 2119.

Para R_1 también encontramos que cada posible cadena tiene un sólo ancestro con los $2r$ elementos iniciales y finales iguales. Ahora concatenemos R_1 consigo mismo para obtener los ancestros de todas las posibles cadenas de longitud 2, en este caso sólo obtenemos tres tablas distintas (tabla 3.6).

2 0	1 0	**0 0**
2 1	**1 1**	0 1
2 2	1 2	0 2

Tabla 3.6: Conjunto R_2.

Si concatenamos ahora R_2 con R_1 obtenemos la tabla 3.7.

2 0	1 0	0 0
2 1	1 1	0 1
2 2	1 2	0 2

Tabla 3.7: Conjunto R_3.

Como este conjunto es igual a R_2, tenemos que el autómata (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 (fig. 3.4).

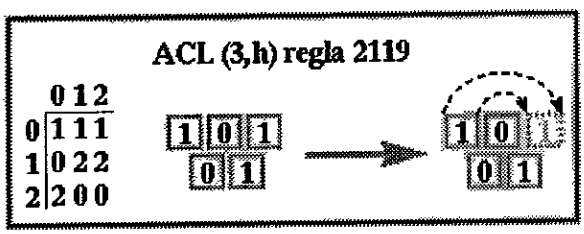


Figura 3.4: Ancestro único de la cadena 01 que puede representarse con la misma longitud.

3.2 Algoritmo utilizando los Indices de Welch

Una primera aproximación tentativa en tratar de utilizar los resultados de los trabajos de Hedlund [11] y Nasu [18] para el cálculo de autómatas reversibles fué desarrollada por Gómez Soto y McIntosh [8]; este algoritmo consiste en aprovechar los conceptos de multiplicidad uniforme e índices de Welch para la construcción de las posibles matrices de evolución de un autómata reversible; por ejemplo, para un reversible (5,h) se tiene la siguiente "plantilla" para generar las matrices de evolución (tabla 3.8).

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

Tabla 3.8: Plantilla para generar matrices de evolución de reversible (5,h).

El resto de los lugares vacíos se llenaban con todas las 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 autómata y se observaba si cada posible vecindad de longitud 2 tenía un único estado para el cual evolucionar hacia atrás, de cumplir ésto, el autómata 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 (fig. 3.5).

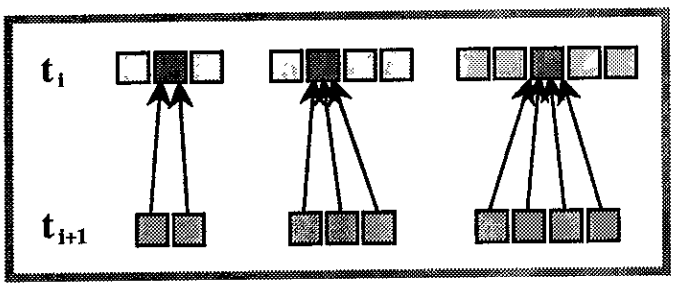


Figura 3.5: Verificación de que cada cadena tenga un único elemento en común en el pasado para cadenas de longitud 2, 3 y 4.

La ventaja de este planteamiento es que ya no hace la revisón de cada posible regla de evolución para un autómata (k,h) sino que sólo 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.

Capítulo 4

Propuesta de un método para encontrar Autómatas Celulares Lineales Reversibles.

En este capítulo se propone un método para encontrar todos los posibles autómatas celulares lineales reversibles de k estados y un radio de vecindad r.

Es importante señalar que al principio dicho método fué desarrollado basado en observaciones empíricas sobre el comportamiento de algunos casos de autómatas celulares lineales reversibles y después se depuró aplicando los resultados obtenidos por Hedlund [11] y Nasu [18].

4.1 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 autómata para ser reversible, después se utilizarán estas propiedades en el diseño de un mecanismo que genere todos los posibles autómatas reversibles para un número de estados k y un radio de vecindad r.

Este es el momento adecuado para hacer una observación importante; para un autómata (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 autómata (2,h) o un autómata (2,1) hasta varias miles para un autómata (5,h) y millones de reglas reversibles para un autómata (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 autómata (k,h).

Tomemos como ejemplo un reversible (4,h) (fig 4.1):

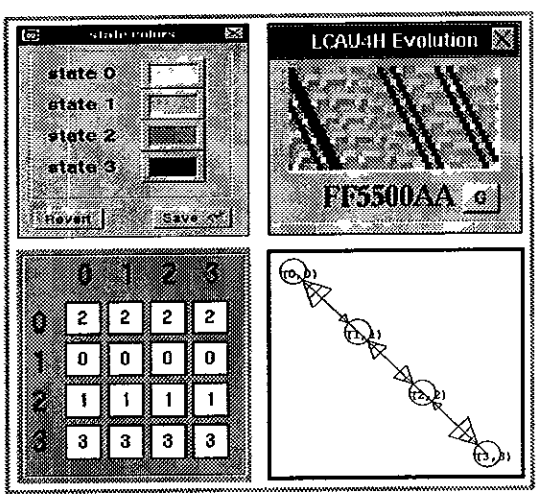


Figura 4.1: Reversible (4,h) regla FF5500AA.

Si hacemos una misma permutación de renglones, de columnas o de estados a la matriz de evolución obtendremos otro autómata reversible que pertenece al mismo cluster (fig. 4.2), repitiendo este proceso para todas las posibles permutaciones de cuatro 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 (lexicograficamente hablando) así obtendremos el representante de este cluster (cluster mínimo).

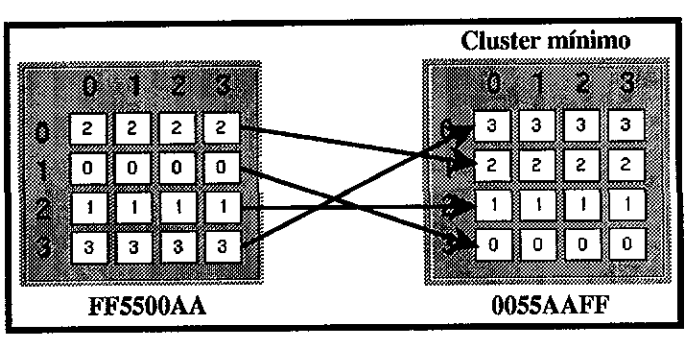


Figura 4.2: Cluster mínimo de un reversible (4,h) regla FF5500AA.

Aplicando el procedimiento para cada regla reversible, éstas se pueden agrupar en distintos clusters, teniendo que cada miembro mínimo de un cluster puede repre-sentar a cientos o miles de reglas reversibles (fig. 4.3) dependiendo del tamaño de vecindad y número de estados del autómata.

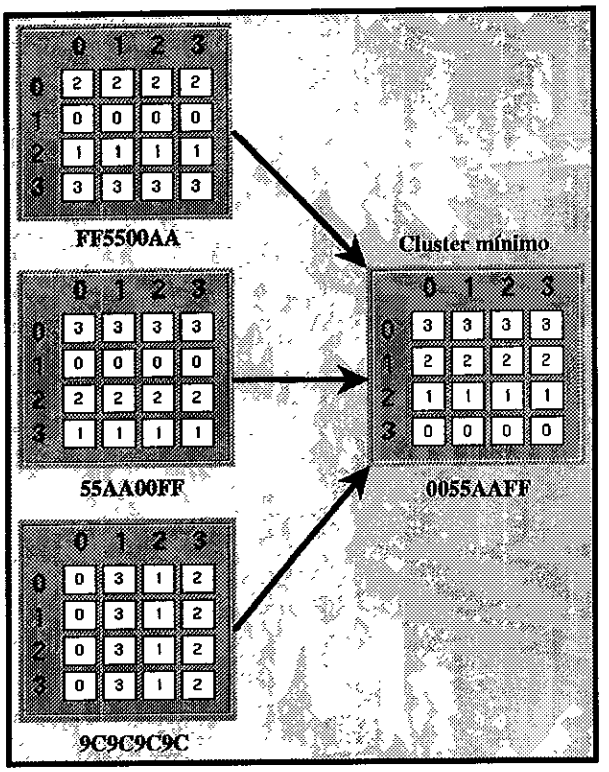


Figura 4.3: Cluster mínimo de varios reversibles (4,h).

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

4.1.1 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 garantiza que cada estado tengan un ancestro, pero si en la diagonal principal existen más de un elemento del mismo estado (ver fig. 4.4), se tendrán múltiples formas de generarlo, lo que no puede ocurrir en un autómata reversible.

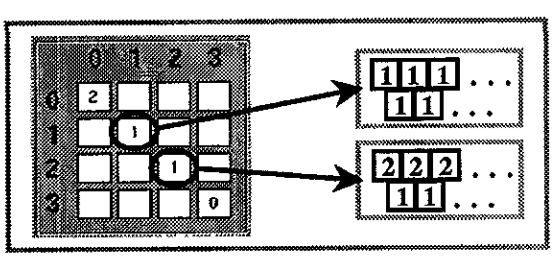


Figura 4.4: Autómata (4,h) con múltiples ancestros para una cadena de 1's.

Como se ve en la fig. 4.4, 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 autómata reversible la multiplicidad uniforme de cada cadena debe ser respetada, si en principio un estado puede generarse de un mayor número de maneras que otro (fig. 4.5) estamos contradiciendo este requerimiento.

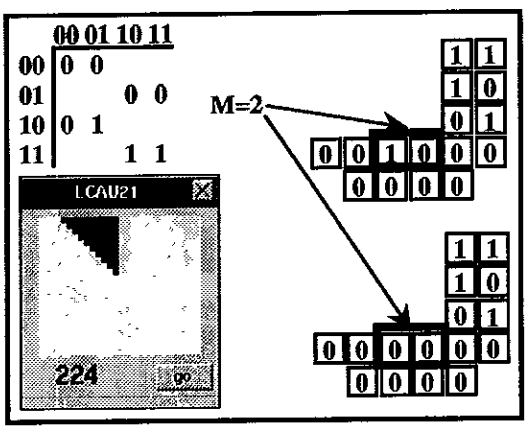


Figura 4.5: Autómata (2,1) con múltiples ancestros para una cadena de 0's.

En la fig. 4.5 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 autómata 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 el capítulo 2, el valor del número de ancestros de cada estado debe ser igual a k^{2r} ó al número de nodos del diagrama de de Bruijn.

4.1.2 Propiedades globales de la matriz

Una vez definidas las propiedades que deben cumplir cada estado en un autómata reversible, podemos utilizar las mismas para construir sólo aquellas matrices de evolución que cumplan dichas propiedades y no todas las reglas posibles. Sin embargo, el problema de la reversibilidad en un autómata 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. En un autómata reversible cada configuración global debe ser posible de generar a través de su evolución, pero por uno y sólo 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 por Hedlund [11] y Nasu [18], que son los índices de Welch y la combinabilidad. En realidad estos conceptos están muy relacionados entre sí ya que la combinabilidad es sólo 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 autómata celular; la matriz de evolución (M. E.) puede ser descompuesta en matrices de conectividad (M. C.), como se muestra en la tabla 4.1; una para cada estado. Cada matriz de conectividad tendrá un 1 si exite una liga con el valor de su estado que va de un nodo a otro en el diagrama de de Bruijn.

M. E.

	0	1	2
0	1	1	1
1	2	2	2
2	0	0	0

M. C. estado 0

	0	1	2
0	0	0	0
1	0	0	0
2	1	1	1

M. C. estado 1

	0	1	2
0	1	1	1
1	0	0	0
2	0	0	0

M. C. estado 2

	0	1	2
0	0	0	0
1	1	1	1
2	0	0	0

Tabla 4.1: Matriz de evolución de un reversible (3,h) regla 715 y sus matrices de conectividad.

Las matrices de conectividad representan 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 al examinar estas matrices.

Por ejemplo, para el reversible (3,h) anterior, se observa que para la matriz de conectividad del estado 0 empezando desde el renglón 2 se puede llegar a los estados 0, 1 y 2, es decir el valor de la cardinalidad del conjunto compatible 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 compatible por la izquierda igual a 1 para generar un 0.

Debido a que cada columna en este ejemplo está formada 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 (fig. 4.6).

Si hacemos extensivo este estudio para las demás matrices de conectividad obtenemos los mismos resultados, satisfaciendo de esta manera la condición de combinabilidad.

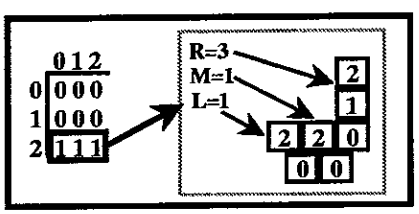


Figura 4.6: Matriz de conectividad del estado 0 del reversible (3,h) regla 715 y como se manifiesta los índices de Welch en ésta.

Este comportamiento se puede observar claramente para todos los estados en los diagramas de subconjuntos tanto de la regla original como de la regla reflejada, en ambos casos partiendo desde las clases unitarias todas las rutas convergen a los niveles R y L respectivamente (fig. 4.7).

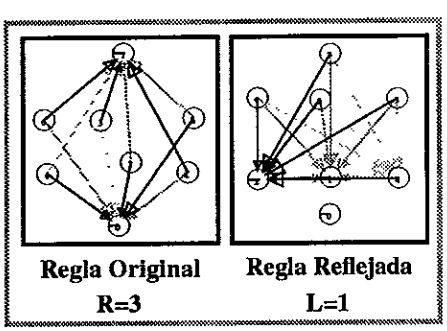


Figura 4.7: Diagramas de subconjuntos (regla original y reflejada) del reversible (3,h) regla 715.

Sin embargo, al checar las matrices de conectividad de un autómata sólo 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 las cadenas de longitud mayor que 1? (o 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 se obtendrán las rutas que existen en la gráfica con longitud igual a esa potencia. Así esto 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 (fig. 4.8).

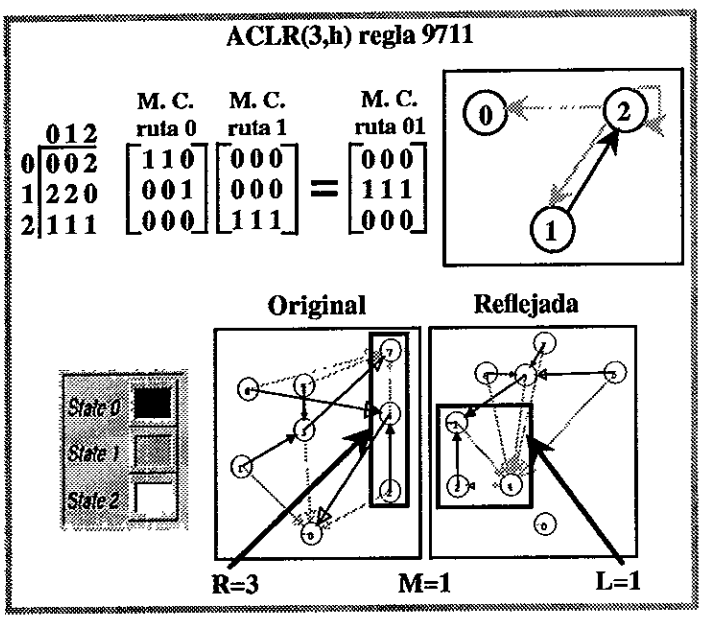


Figura 4.8: Indices de Welch para la cadena 01 del reversible (3,h) regla 9711.

En la fig. 4.8 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 y 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 autómata 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 se debe cumplir lo siguiente:

- El número de lugares diferentes a 0 en cada renglón debe ser igual a 0 ó R.
- El número de lugares diferentes a 0 en cada columna debe ser igual a 0 ó L.
- La suma de elementos en la diagonal principal debe ser 1, es decir, un sólo 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, se concluye que el valor del índice $M = 1$ y por lo tanto el autómata es reversible (fig. 4.9).

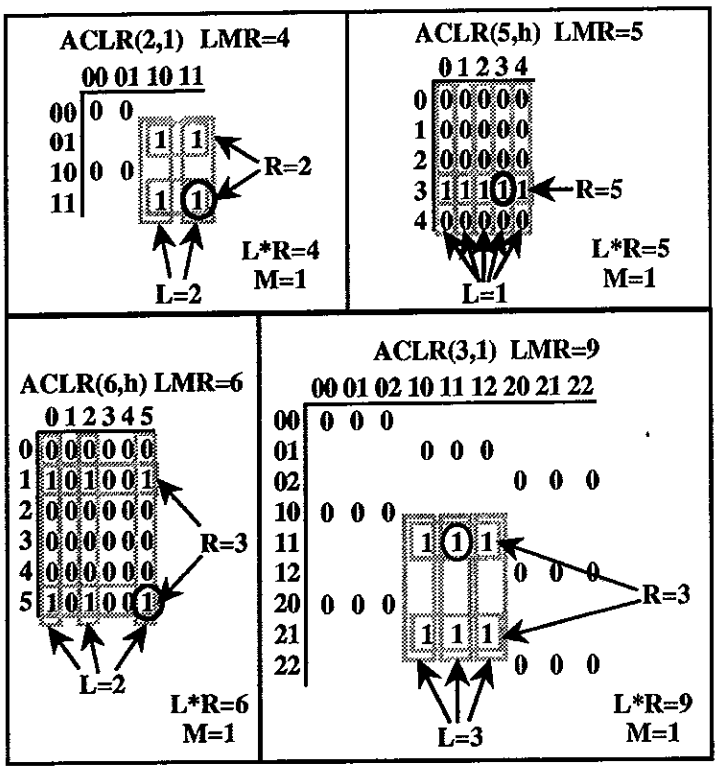


Figura 4.9: Diferentes formas de matrices de conectividad para autómatas reversibles, con un sólo 1 en la diagonal principal y valores de $L * R = k^{2r}$ induciendo que $M = 1$.

Para un autómata 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, o la suma de elementos en la diagonal principal sea mayor que 1 con lo que existirán multiples ancestros para una cadena dada (fig. 4.10).

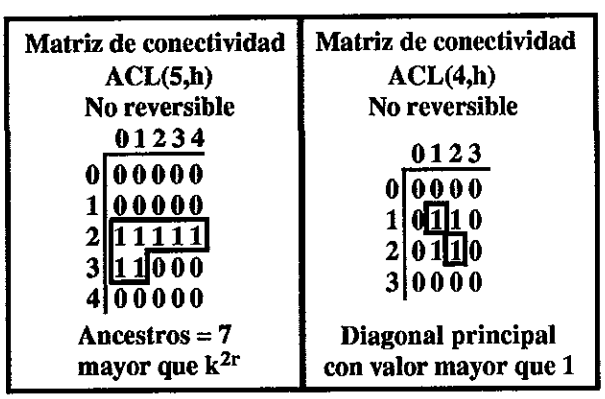


Figura 4.10: Matrices de conectividad para autómatas no reversibles.

4.2 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 autómata (k,r). Como forma de ejemplificar lo anterior se tomará el caso de un autómata (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 cada estado necesarias para que el autómata sea reversible, por lo que se tomará una plantilla para la matriz de evolución donde los elementos de la diagonal principal sean diferentes entre sí y permanezcan así durante todo el proceso (tabla 4.2).

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

Tabla 4.2: Plantilla para generar todas las matrices de evolución posibles candidatas a ser reversibles (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 autómata reversible pues indica que una secuencia de dos elementos puede ser generada por varias cadenas de dos estados). La forma de verificar esto es formar una matriz auxiliar en donde cada renglón representará un estado del autómata y los elementos del mismo una codificación de las vecindades que lo generan, ésta puede ser elevando 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 (fig. 4.11).

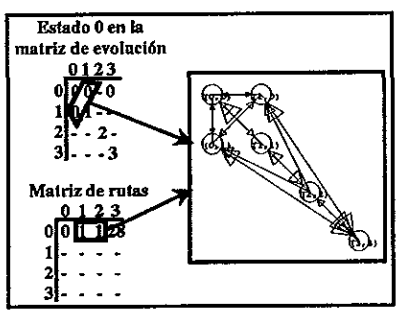


Figura 4.11: Matriz de rutas del estado 0 en un autómata (4,h) donde se observan múltiples ancestros.

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 secuancia 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 (fig. 4.12).

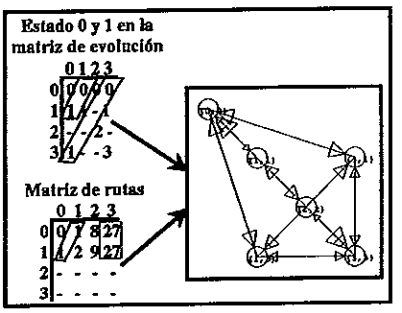


Figura 4.12: Matriz de rutas del estado 0 y 1 en un autómata (4,h) con múltiples ancestros.

Aunque este paso no es necesario para detectar autómatas reversibles es de gran utilidad ya que ayudará a descartar muchas matrices de evolución de manera rápida y agilizar el cálculo.

Una vez que se ha llenado la matriz de evolución completamente, se procede a verificar si las matrices de conectividad para cadenas de un elemento cumplen con el principio de combinabilidad, de no ser así se verifican las matrices de conectividad de cadenas de dos elementos y se continúa ya sea hasta que todas las matrices de conectividad cumplan con ser debilmente combinables obteniendo el cluster mínimo del autómata, o que cualquier matriz de conectividad presente alguna de las condiciones para descartar al autómata como reversible (fig. 4.13).

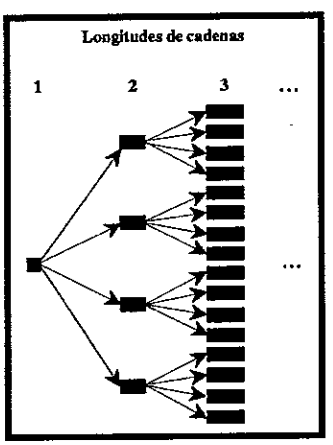


Figura 4.13: Crecimiento del número de matrices de conectividad a revisar para cadenas de longitud 1 en adelante para un autómata (4,h).

El proceso completo se realiza de la siguiente forma:

1. *Actualizar en la matriz de evolución del autómata (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 autómata reversible, 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 (fig. 4.14) refleja el flujo del programa; el listado del mismo para el caso (4,h) se presenta en el Apéndice A.

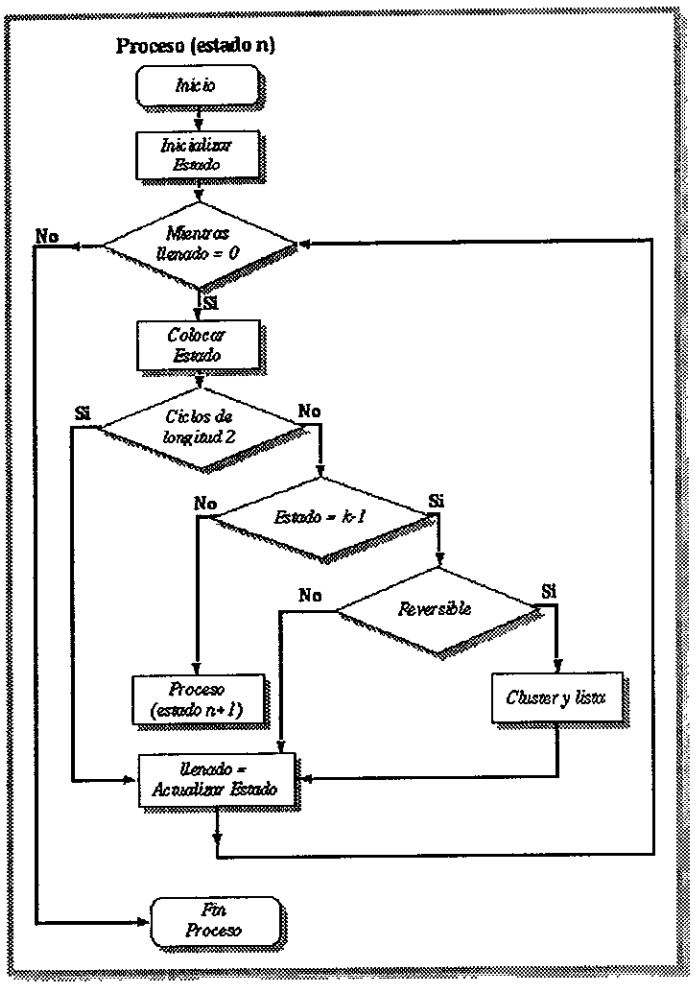


Figura 4.14: Flujo del proceso para obtener autómatas reversibles (k,r).

4.3 Comentarios sobre el método propuesto

El algoritmo aquí presentado se ha implementado para encontrar todos las reglas reversibles para los autómatas (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 dramaticamente. Debemos añadir a ésto el tiempo requerido para saber si una posible matriz de evolución define un autómata reversible 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 autómata reversible se debe cálcular 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 autómata (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 ésto fijando por ejemplo a $L = 1$, generando sólo 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 sólo 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 vaya generando,éstas simplemente se obtienen de la matriz de evolución y para obtener las matrices de conectividad de cadenas de longitud mayor que 1 se usa un proceso recursivo, dejando el manejo de la memoria en manos del compilador.

Capítulo 5

Presentación de los resultados obtenidos con el método propuesto

En esta sección se muestran los resultados obtenidos por el método propuesto para encontrar los autómatas reversibles para los tipos (4,h),(5,h) y (6,h). Tanto en el primer caso (fig. 5.1 - 5.4) como en el segundo (fig. 5.5 y 5.6) se mostrará la lista completa de autómatas reversibles encontrados y algunos ejemplos de evolución de los mismos y en el último caso (fig. 5.7 - 5.13) ya que el listado resulta muy extenso, sólo se presentarán algunos ejemplos de evolución.

En cada autómata se presenta el especimen que el programa de cómputación genera así como el cluster mínimo del mismo.

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

5.1 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

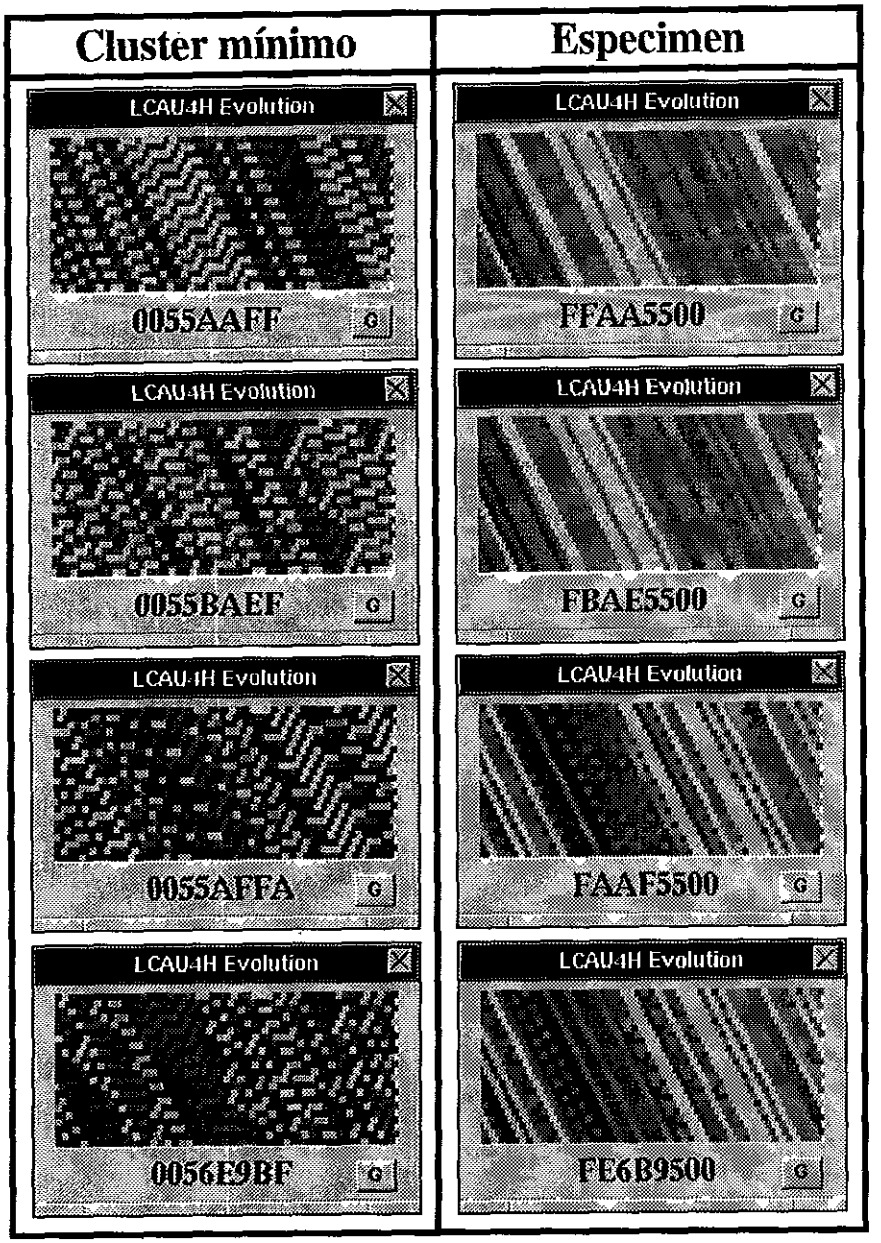


Figura 5.1: Reversibles (4,h) con $L = 1$ y $R = 4$.

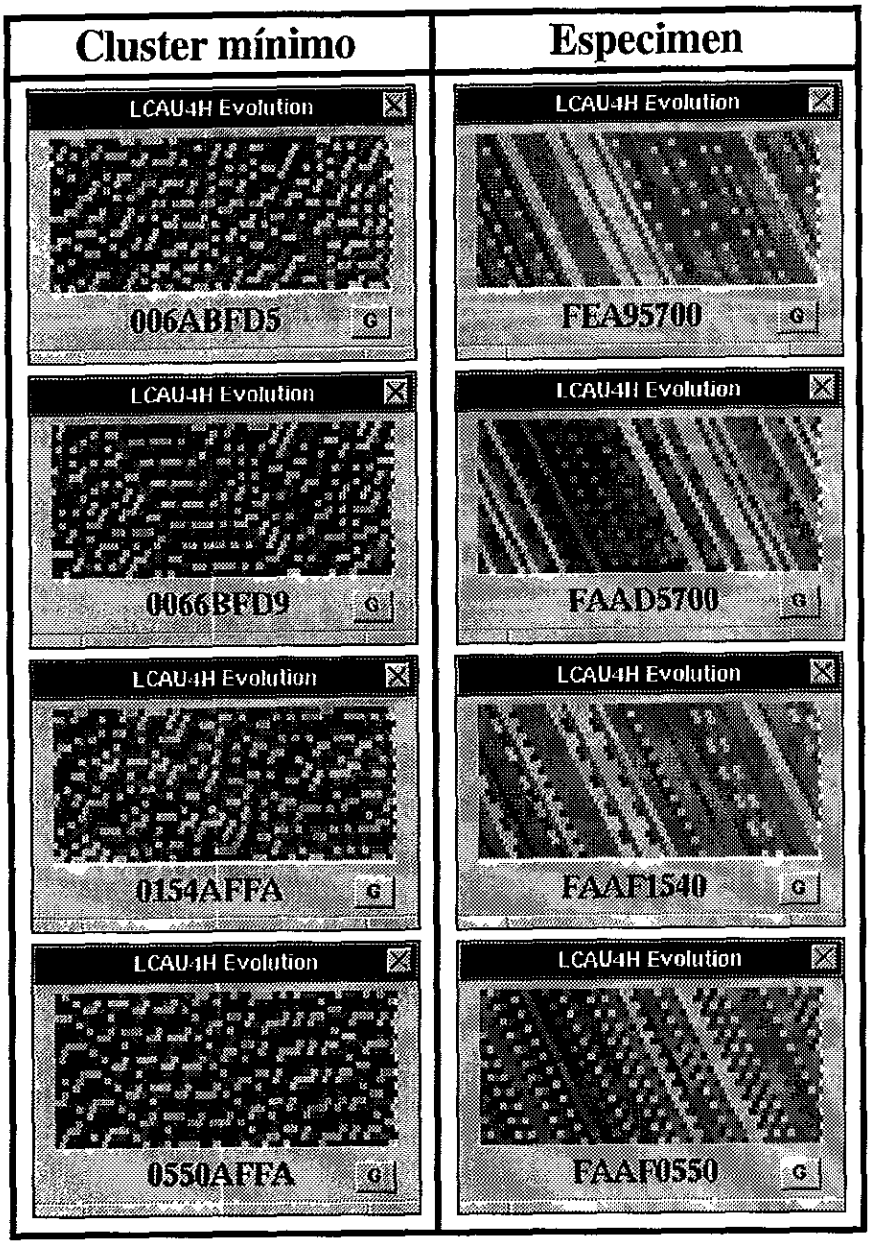


Figura 5.2: Reversibles (4,h) con $L = 1$ y $R = 4$.

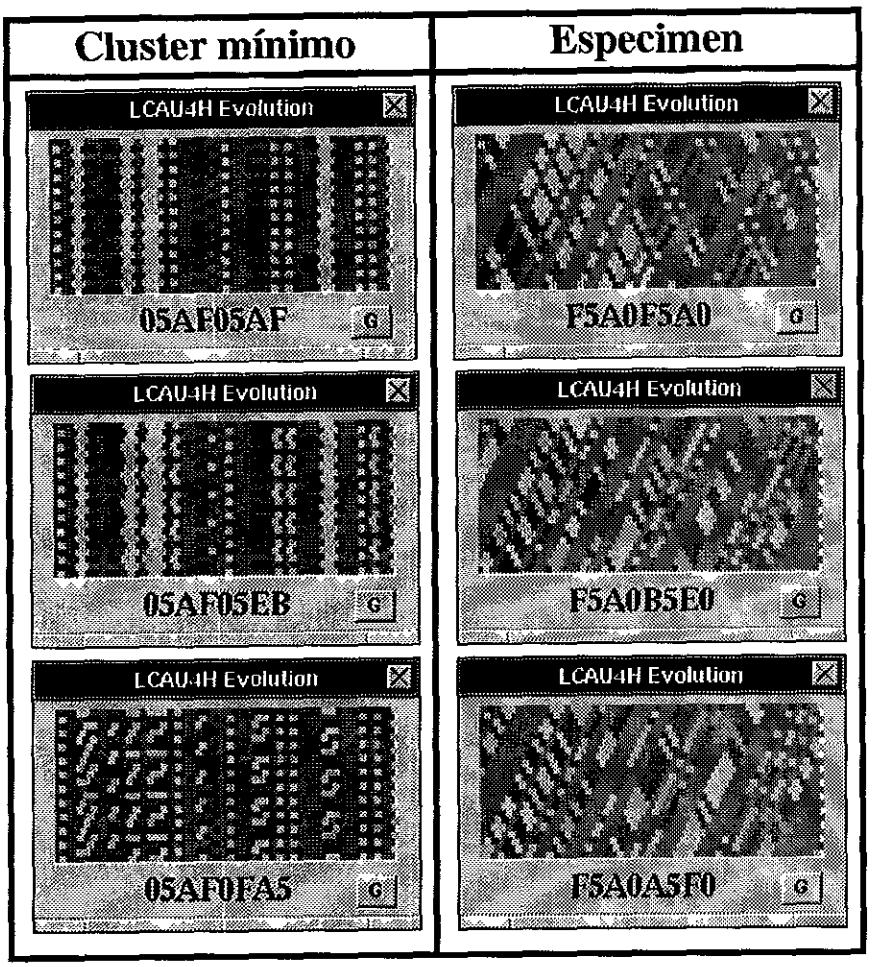


Figura 5.3: Reversibles (4,h) con $L = 2$ y $R = 2$.

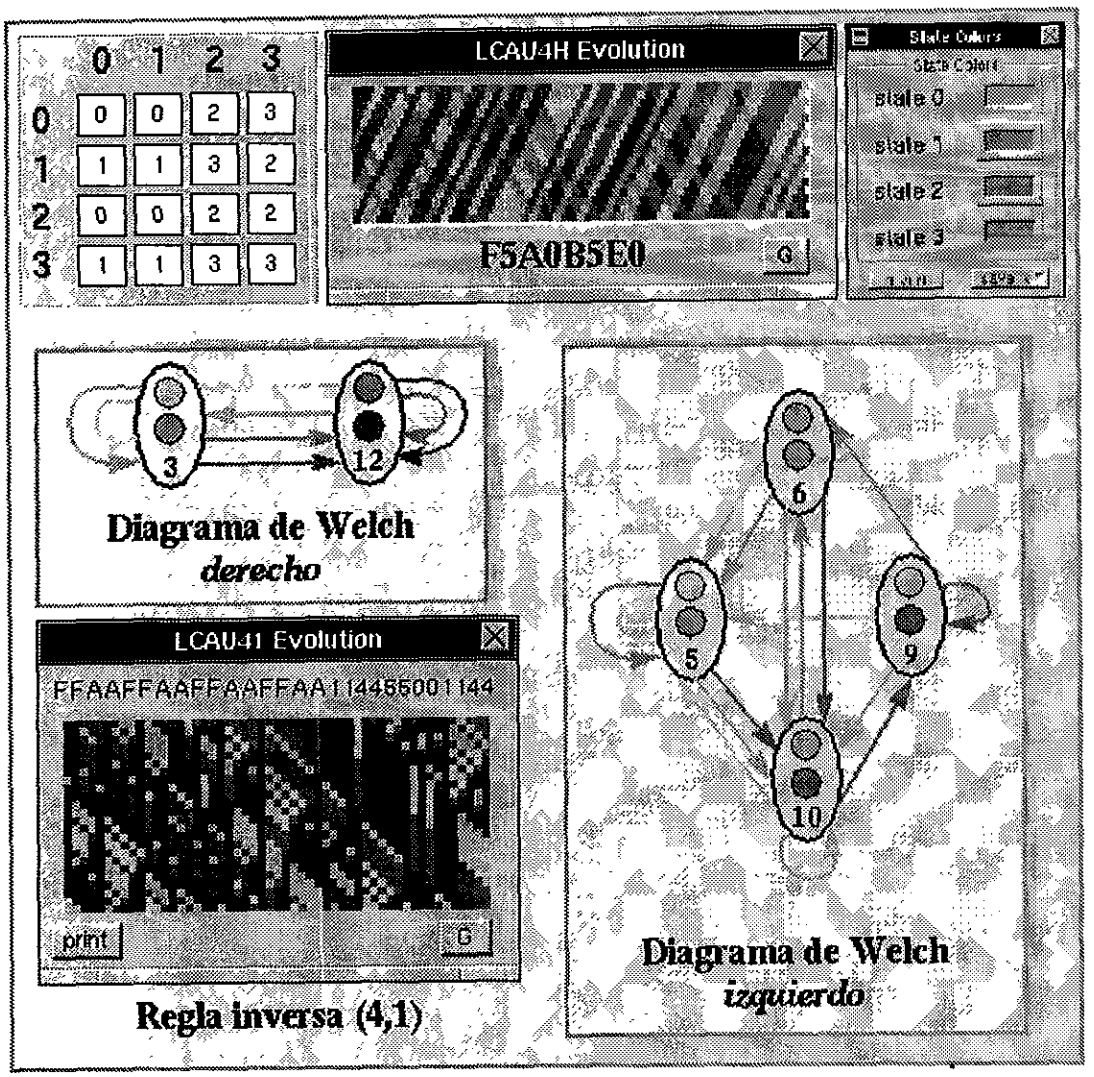


Figura 5.4: Reversible (4,h) regla F5A0B5E0 y su regla inversa (4,1) obtenida por medio de los Diagramas de Welch.

5.2 Resultados para los Autómatas Celulares Lineales Reversibles (5,h)

AUTOMATAS CELULARES LINEALES REVERSIBLES 5H con L=1, R=5	
Cluster Mínimo No. 1: 000667CCIIJOO	Especimen: 4OOIIHCC66500
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: 4ONIHMCH66500
Cluster Mínimo No. 8: 000667HCJJOEI	Especimen: 4NNIMMCH66500
Cluster Mínimo No. 9: 000667HCJIJEO	Especimen: 4OJIHHCM66500
Cluster Mínimo No. 10: 000667HCJNJDO	Especimen: 4NJIMHCM66500
Cluster Mínimo No. 11: 000667DIJCEJO	Especimen: 4OIIHMCM66500
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: 000667HEJOEDI	Especimen: 4ODDINCM66500
Cluster Mínimo No. 16: 000667DIIOOMC	Especimen: 4OIIHCCO66500
Cluster Mínimo No. 17: 000667DDIOOMH	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: 4ONIILCCB6500
Cluster Mínimo No. 22: 00066CCBIJOOI	Especimen: 4NIIOLCCB6500
Cluster Mínimo No. 23: 00066CCGJOOID	Especimen: 4ENIILMCB6500
Cluster Mínimo No. 24: 00066CHBMIIOO	Especimen: 4ONIHLCHB6500
Cluster Mínimo No. 25: 00066CJJMC8NN	Especimen: 4NJIOBCDB6500
Cluster Mínimo No. 26: 00066CIIMC8OO	Especimen: 4ONIIBCEB6500
Cluster Mínimo No. 27: 00066CJJHC9NN	Especimen: 4NIIOBCEB6500
Cluster Mínimo No. 28: 00066DCBNIHOO	Especimen: 4OMIILCDB6500
Cluster Mínimo No. 29: 00066DCBONHIO	Especimen: 4NHIOLCDB6500
Cluster Mínimo No. 30: 00066DCBNJMOI	Especimen: 4NHIOGCEB6500
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: 4OMIHLCIB6500

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: 4ONIIL7CC6500
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: 00067HIGJOOC7	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: 00077IOONDC86	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: 4NIIMCCL66K00
Cluster Mínimo No. 81: 0007CCJOJ698I	Especimen: 4ODIJCCG66K00
Cluster Mínimo No. 82: 00077JIGGBCOO	Especimen: 4NDIOCCG66K00
Cluster Mínimo No. 83: 00077JIHGB7OO	Especimen: 4NDINCCL66K00
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: 0007CCJ9LIILL	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: 4NHIOLCBB6F00
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: 4OIIHMCM16600
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: 49ODHDCI46600
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: 46ODHDCI4L600
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: 006652DIJEOJC	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

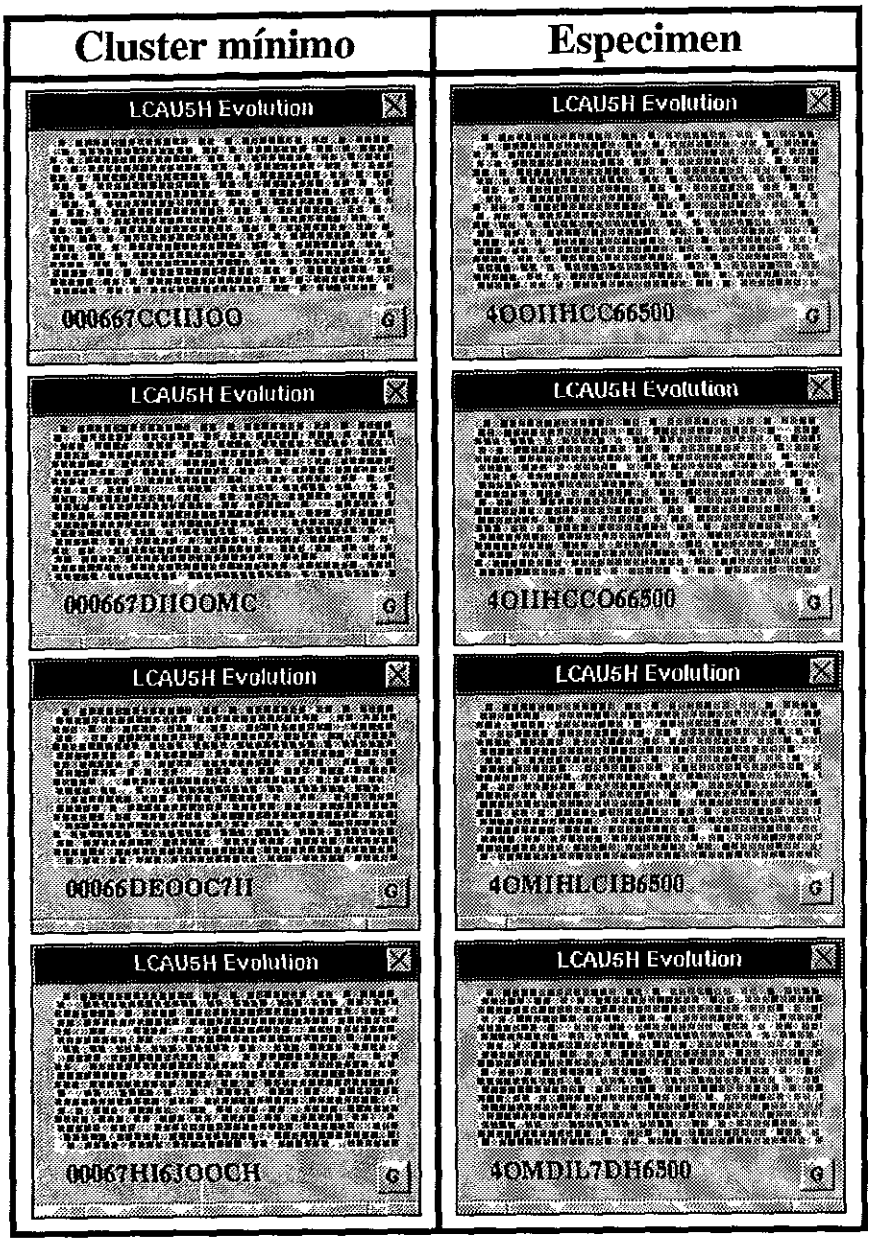


Figura 5.5: Reversibles (5,h) con $L = 1$ y $R = 5$.

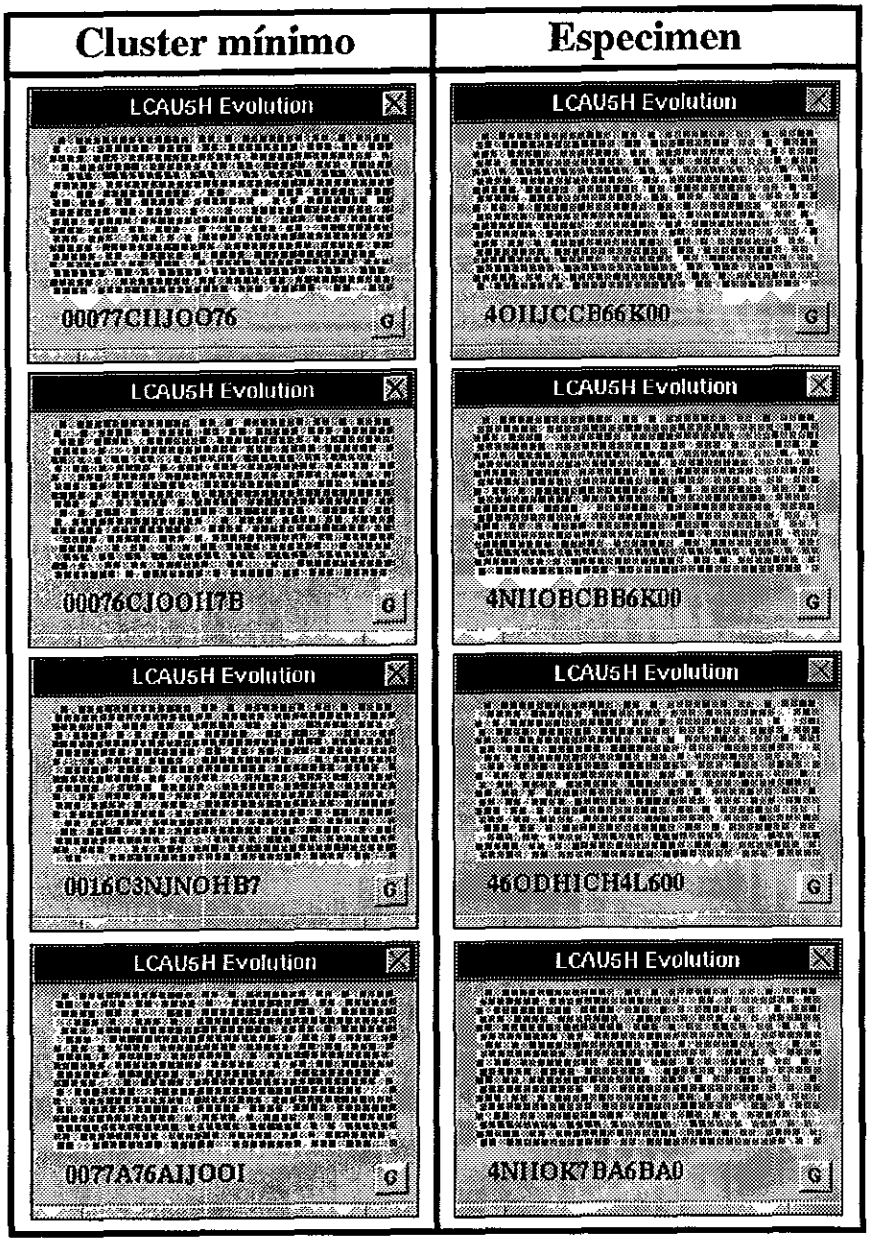


Figura 5.6: Reversibles (5,h) con $L = 1$ y $R = 5$.

5.3 Resultados para los Autómatas Celulares Lineales Reversibles (6,h)

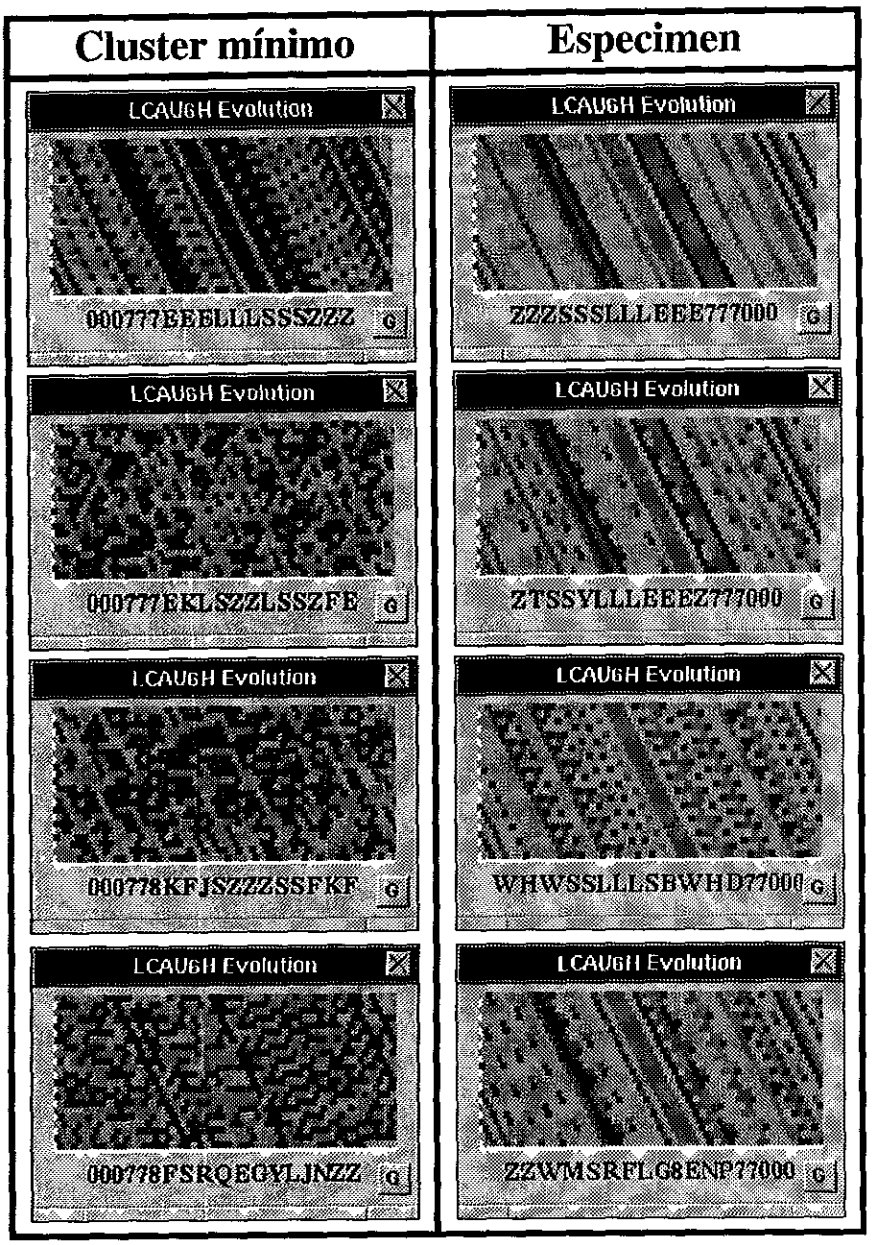


Figura 5.7: Reversibles (6,h) con $L = 1$ y $R = 6$.

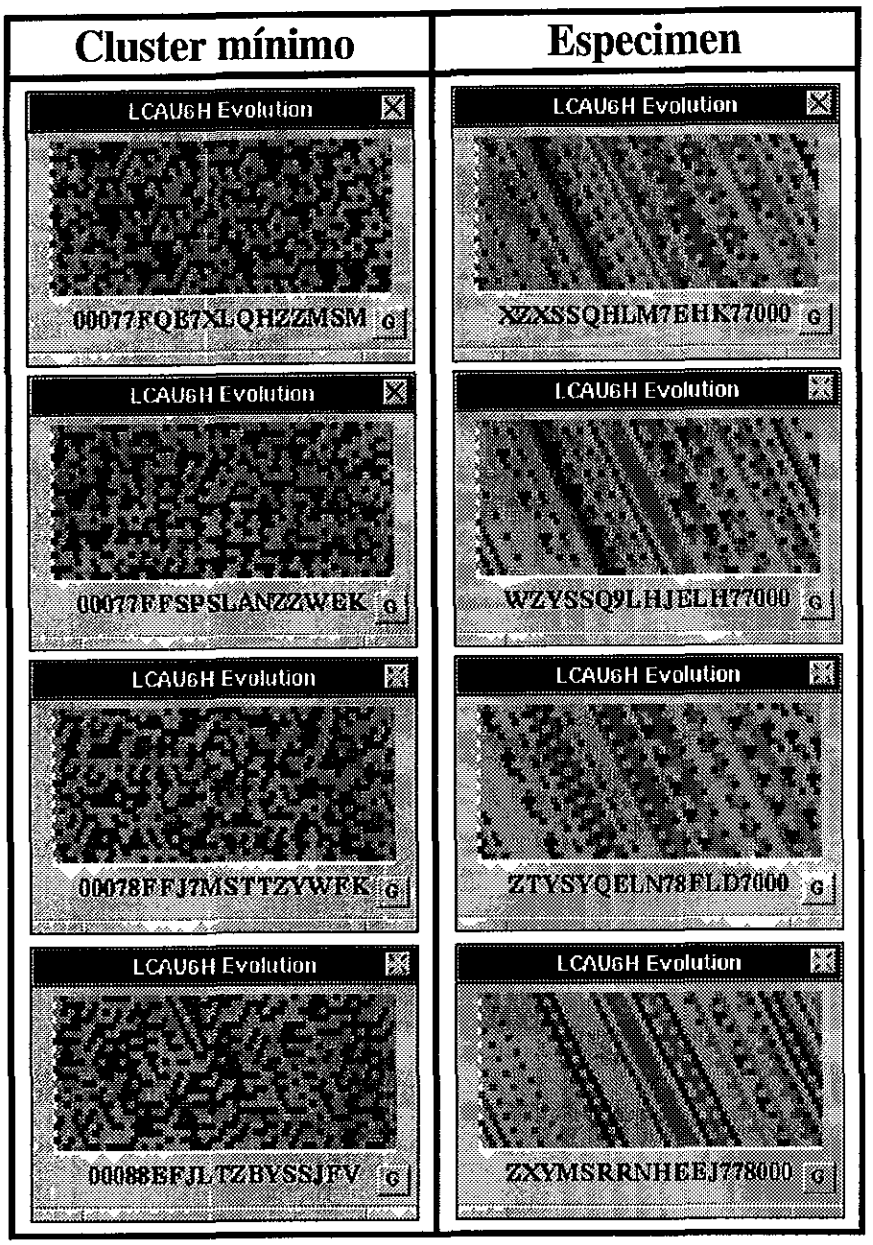


Figura 5.8: Reversibles (6,h) con $L = 1$ y $R = 6$.

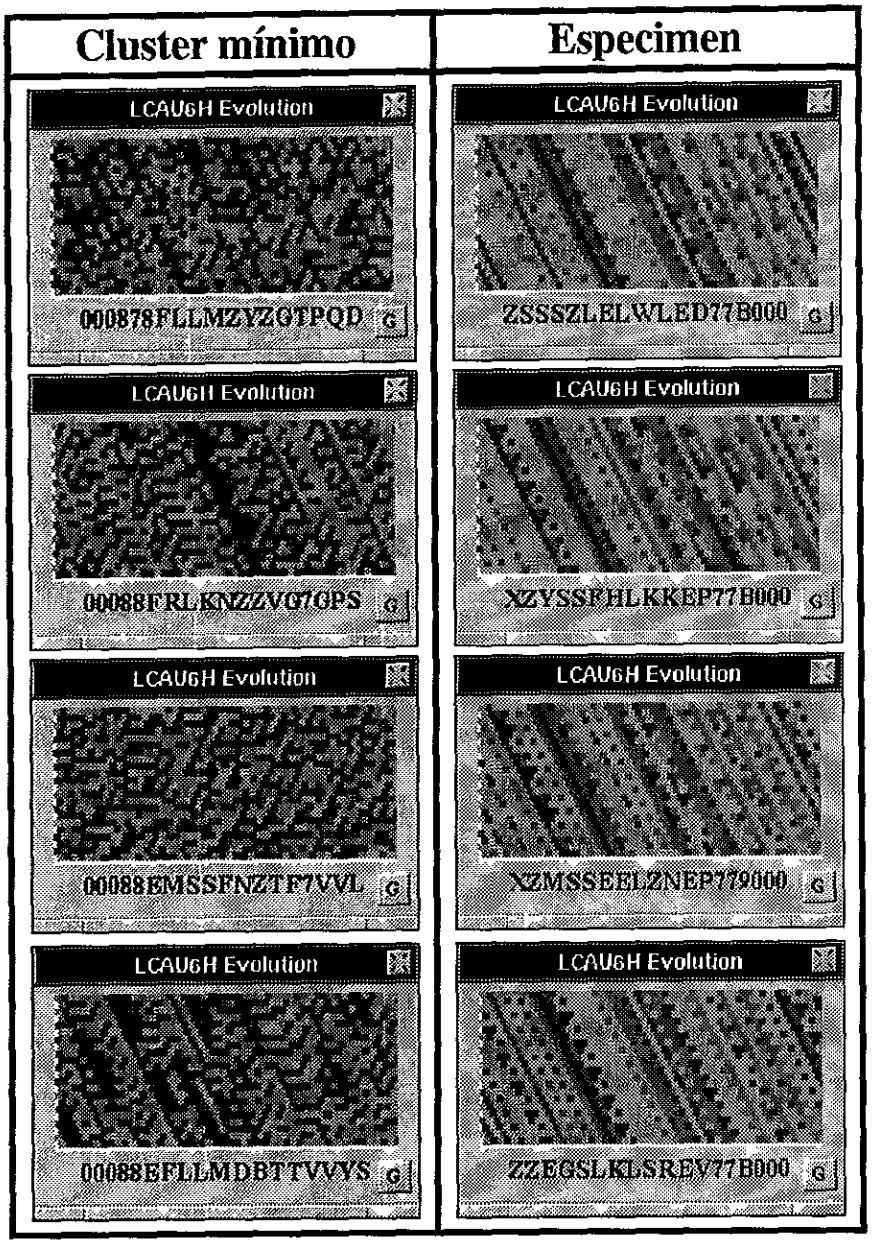


Figura 5.9: Reversibles (6,h) con $L = 1$ y $R = 6$.

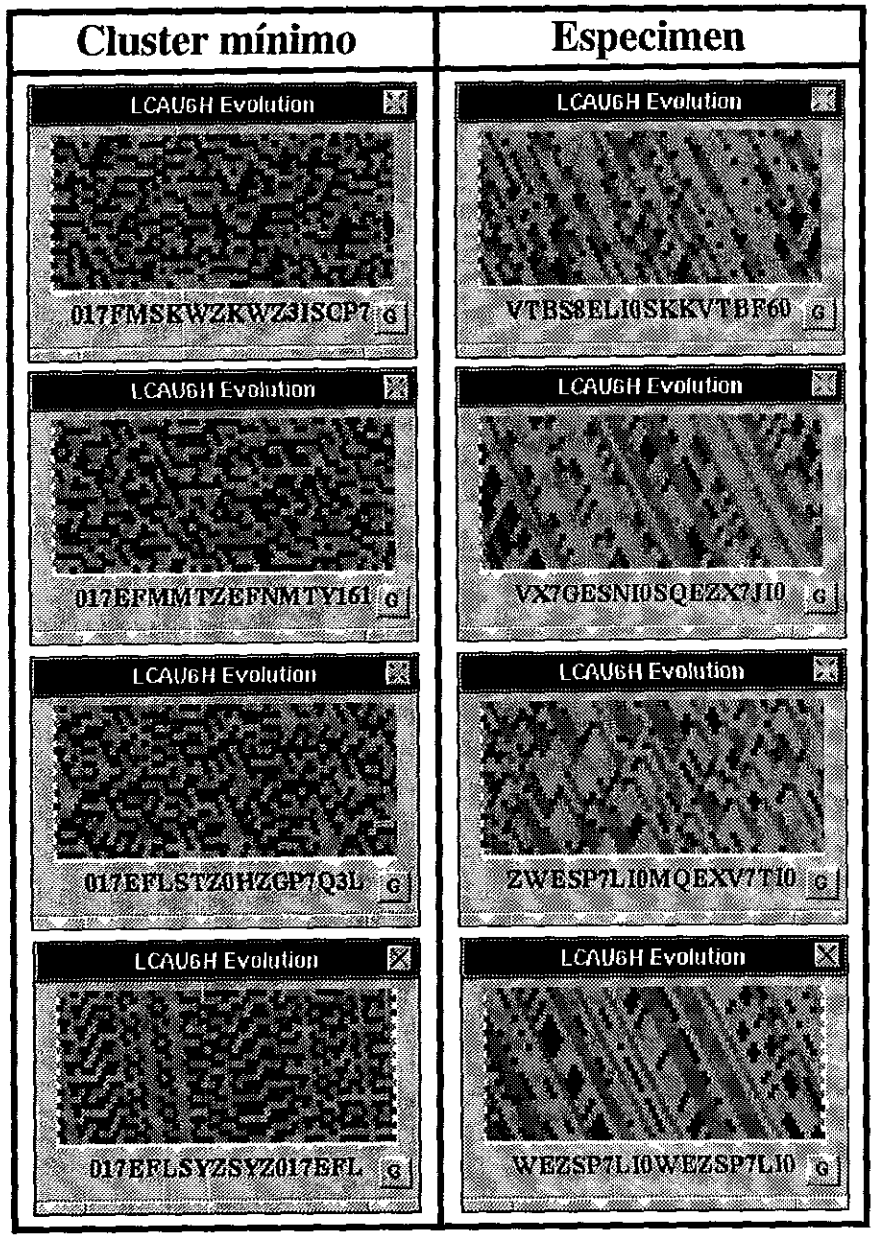


Figura 5.10: Reversibles (6,h) con $L = 2$ y $R = 3$.

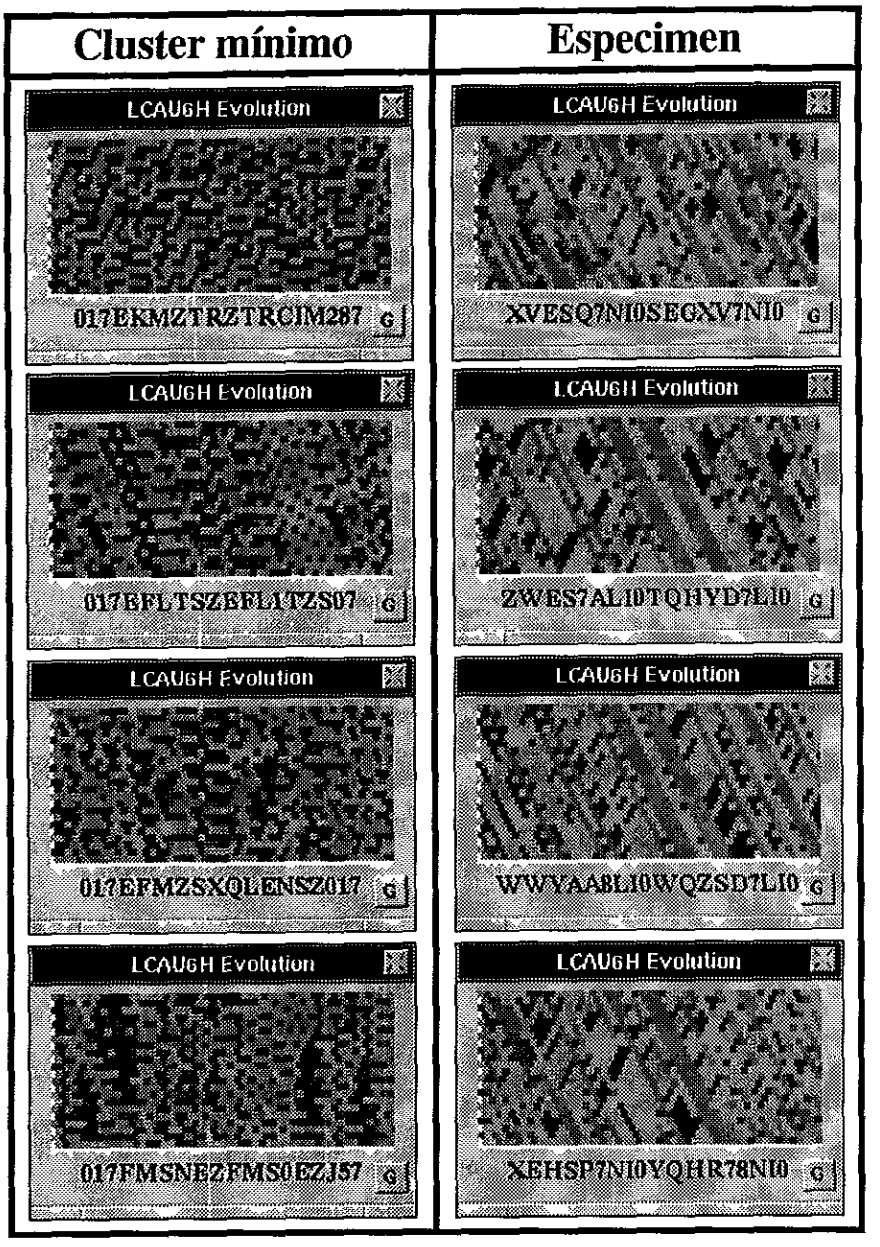


Figura 5.11: Reversibles (6,h) con $L = 2$ y $R = 3$.

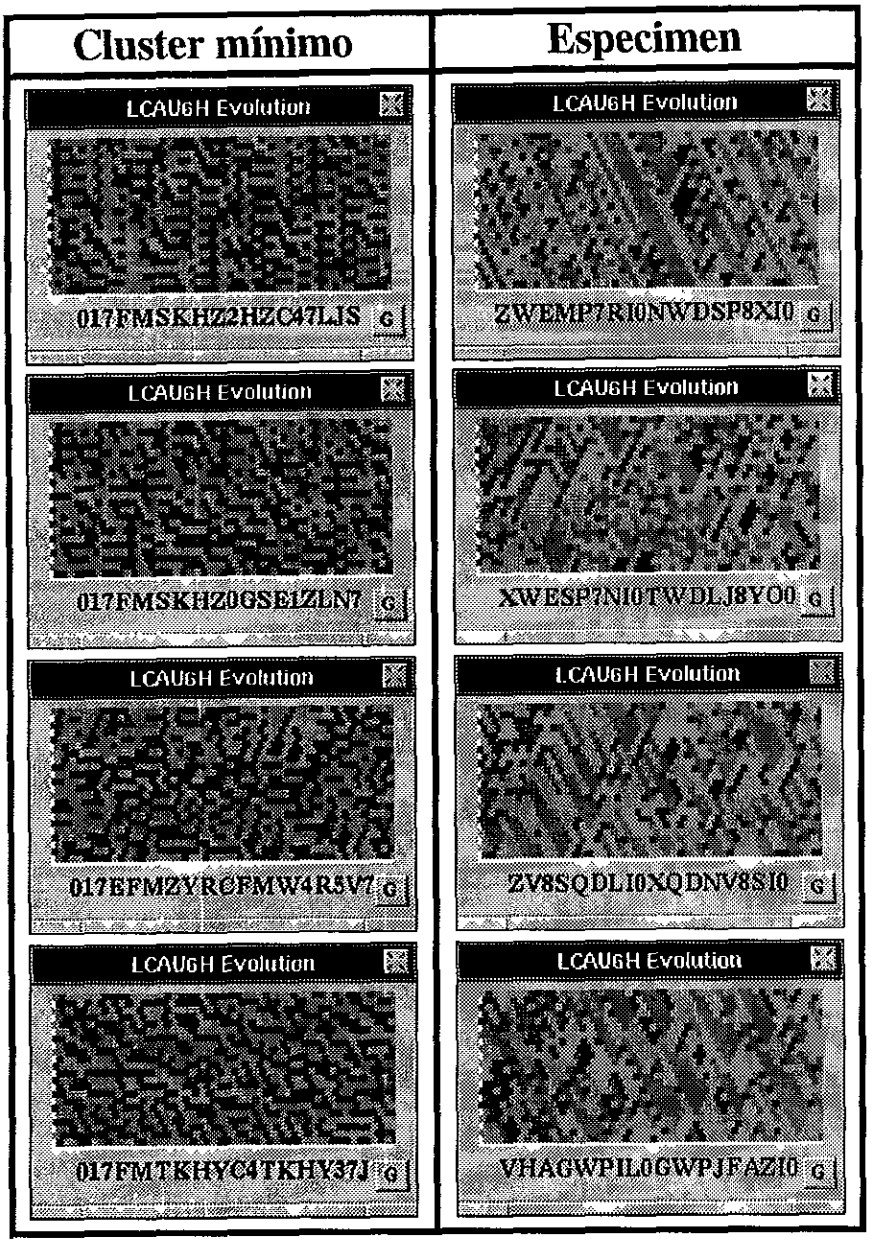


Figura 5.12: Reversibles (6,h) con $L = 2$ y $R = 3$.

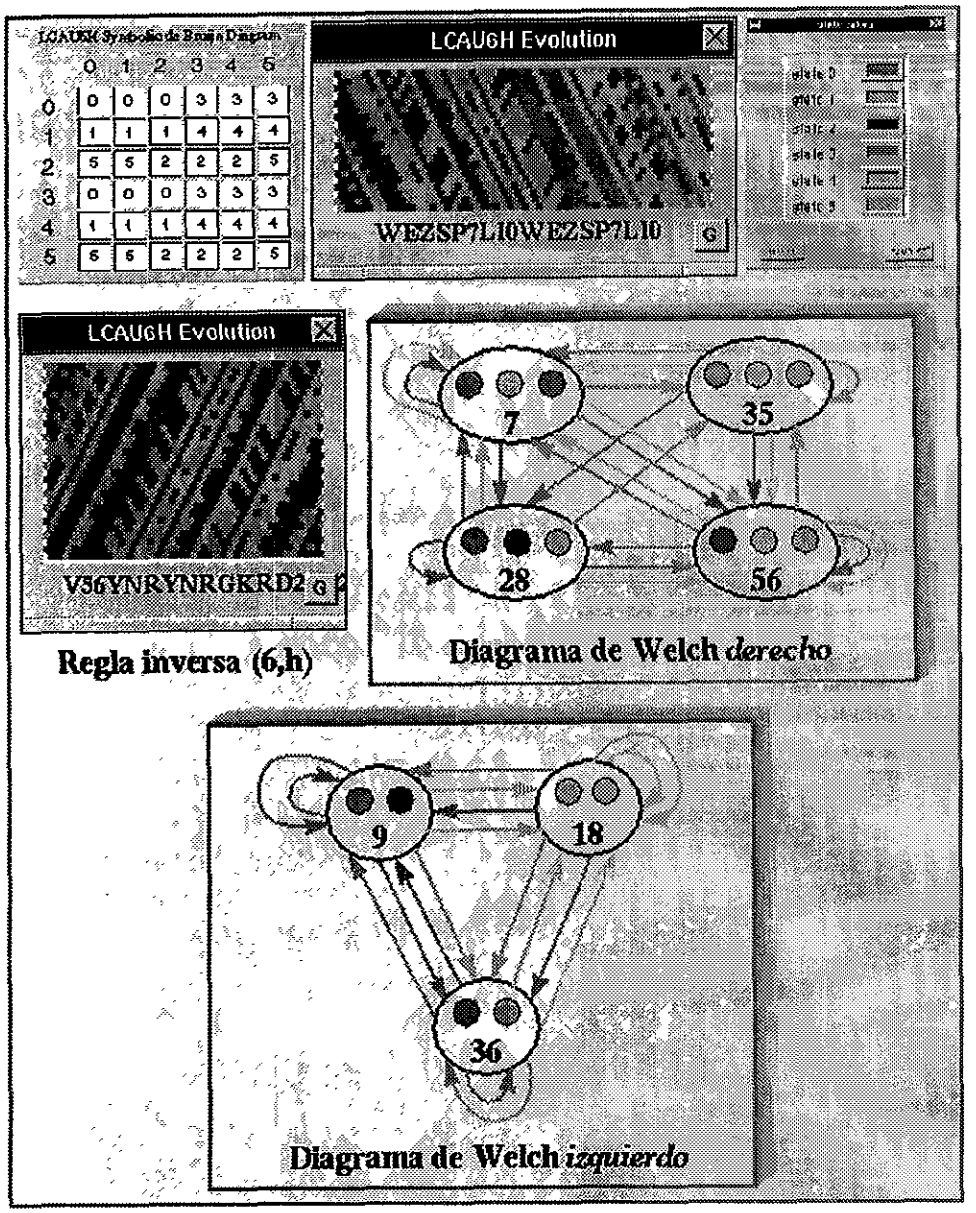


Figura 5.13: Reversible (6,h) regla WEZSP7LI0WEZSP7LI0 y su regla inversa (6,h) obtenida por medio de los Diagramas de Welch.

5.4 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 cierta longitud 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 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 puede evitarse si mejor en cada tabla se verificara que una columna tuviera L elementos distintos y en la otra R elementos distintos, la condición que establece que sólo una pareja debe ser formada por elementos iguales es revisar que sólo 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 autómata (5,h) más de 13 horas, la implementación que se expone aquí con una programación muy simple tarda menos de 3 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 sólo contemplaba autómatas reversibles donde un renglón estuviera formada por elementos todos iguales, cosa que no siempre sucede para los reversibles.

Conclusiones

Perspectivas para mejorar la computación de los Autómatas Celulares Lineales Reversibles

El trabajo desarrollado por Hedlund y Nasu nos ha dado una caracterización completa acerca del comportamiento y las cualidades que debe mostrar la reglas de evolución en un autómata celular lineal para inducir un mapeo global reversible, no importando cuantas células tenga el arreglo donde se implementa.

Sin embargo, aún quedan varias cuestiones por investigar, por ejemplo el algoritmo propuesto empieza a revisar las matrices de conectividad para cadenas de longitud 1 en adelante, pero si el autómata es reversible, ¿en qué 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 podriamos checar sólo 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 mientra más estados y/o más grande sea la vecindad en el autómata, 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 autómatas reversibles; parece ser que los reversibles (k,r) están formados por subautomatas reversibles, es decir, que un reversible (6,h) puede ser una extensión de un reversible (5,h) o de un reversible (4,h) junto con un reversible (2,h) y así sucesivamente. Si esta idea es correcta, el reversible (2,h) podría ser la base para generar todos los demás autómatas reversibles 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 autómatas reversibles, sino que se generarían directamente matrices de evolución que fueran extensiones de un autómata reversible 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 autómatas reversibles, ya sea haciendo uso de eigenvalores y eigenvectores, ver como se comportan estos en las matrices de conectividad de un autómata reversible, 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 autómata reversible, para este motivo una mejor y más profunda utilización de los diagramas de Welch sería conveniente.

Como se observa aún queda mucho por hacer para realizar el cálculo de todas las posibles reglas reversibles para un cierto autómata (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.

Futuro de las aplicaciones de los Autómatas Celulares Reversibles

Esta última parte puede resultar algo tentativa, pues en realidad aunque se han aplicado en muchos campos a los autómatas celulares, 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 autómatas reversibles 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 rapidamente para que los datos recuperen su forma original; es por esto que se podría pensar en la implementación de autómatas reversibles 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 autómatas reversibles resulta inmediata, y se puede obtener un grado de complejidad bastante interesante.

Por ejemplo, se podría utilizar simplemente un reversible (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, el archivo tendría cuatro tipos de elementos distintos y se podría implementar un reversible (4,h) por ejemplo en donde existen once clusters mínimos cada uno con decenas de reglas reversibles distintas.

Siguiendo esta idea podriamos agrupar ahora la información en grupos de tres bits obteniendo ocho posibles estados, si se decidiera aplicar un reversible (8,h) tendriamos millones de posibles reglas que inducieran un mapeo global reversible, aumentando el tamaño de vecindad crece también dramáticamente las posibilidades de obtener más autómatas reversibles para aplicarlos como encriptadores. Además, 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.

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] José Manuel Gómez Soto y Harold V. McIntosh, *Los Indices de Welch en el Cálculo de Autómatas Celulares Reversibles*, Congreso Nacional de Matemáticas, Junio de 1996.

[9] Solomon W. Golomb, *Shift Register Sequences*, University of Southern California, Holden-Day, Inc., 1967.

[10] Howard Gutowitz, *Cellular Automata, Theory and Experiment*, MIT Press, 1991.

[11] Gustav A. Hedlund, *Endomorphisms and automorphisms of the shift dynamical system*, Mathematical Systems Theory 3, pp 320-375, 1969.

[12] David Hillman, *The structure of reversible one-dimensional cellular automata*, Physica D 52, pp 277-292, 1991.

[13] Aron V. Holden, *Chaos*, Princenton University Press, 1986.

[14] L. Margara, *Cellular Automata and Chaos*, Ph.D. Thesis, 1995.

[15] Harold V. McIntosh, *Linear Cellular Automata*, Universidad Autónoma de Puebla, 1990.

[16] Harold V. McIntosh, *Linear Cellular Automata via de Bruijn Diagrams*, por publicarse.

[17] Harold V. McIntosh, *Reversible Cellular Automata*, por publicarse.

[18] Masakazu Nasu, *Local Maps Inducing Surjective Global Maps of One Dimensional Tesellation Automata*, Mathematical Systems Theory 11, pp 327-351, 1978.

[19] Heinz-Otto-Peitgen, Hartman Jurgens, Dietmar Soupe, *Chaos and Fractals, New Frontiers of Science*, Springer-Verlog, 1992.

[20] William Poundstone, *The Recursive Universe*, William Morrow and Company Inc., 1985.

[21] Harold S. Stone, *Discrete Mathematics*, CCS Editorial, Stanford University, 1973.

[22] Tommaso Toffoli, Norman Margolus, *Cellular Automata Machines*, The MIT Press, Cambridge Massachusetts, 1987.

[23] Tommaso Toffoli, *Computation and Construction Universality of Reversible Cellular Automata*, Journal of Computer and System Sciences 15, pp 213-231, 1977.

[24] Burton H. Voorhees, *Computational Analysis of One Dimensional Cellular Automata*, World Scientific, Sigapore, 1995.

[25] M. Mitchell Waldrop, *Complexity*, Touchstone, 1992.

[26] Stephen Wolfram, *Theory and Applications of Cellular Automata*, World Scientific, Singapore, 1986.

[27] Stephen Wolfram, *Universality and Complexity in Cellular Automata*,Physica 10D, North-Holland, pp 1-35, 1984.

[28] Andrew Wuensche and Mike Lesser, *The Global Dynamics of Cellular Automata*, Santa Fe Institute, 1992.

Apéndice A

Ejemplo de la aplicación para un Autómata (4,h)

A continuación se presenta el programa de computación que calcula los autómatas reversibles para el caso (4,h). Este programa fue escrito en lenguaje C y se ejecutó en el UNIX del sistema operativo NeXTSTEP, en una PC con procesador Pentium a 133 MHz y 16 Mb en RAM; aunque este mismo proceso con muy ligeras modificaciones en la escritura del archivo, se adaptó también para ejecutarse en Turbo C++ 3.0 en sistema operativo MSDOS. Cabe señalar que el mismo proceso puede ser adaptado para cualquier tipo de autómata reversible (k,r) pues la forma de detección es la misma para todos los casos.

```
#include "stdio.h"
#include "ctype.h"
/***** DEFINICION DE VALORES *****/
/* Número de estados del autómata */
#define numestados 4
/* Longitud de la regla de evolución */
#define longregla 16
/* Longitud del número wolfram */
#define longcadena 8
/* Valor de producto L*R */
#define indice 4
/* Número de iteraciones a realizar para buscar el cluster mínimo */
#define numit 6
/* Número total de permutaciones */
#define numper 24
/* Indice izquierdo a buscar */
#define izquierdo 2
/* Indice derecho a buscar */
#define derecho 2
/* Nombre del archivo en donde guardar los resultados */
#define archivo "ACLR4H.TXT"

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

/* Permutaciones posibles para buscar el cluster mínimo */
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 evolución del autómata */
int matriz[numestados][numestados];
```

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

/* Matriz de rutas en el autómata */
int rutas[numestados][numestados-1];

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

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

/* Arreglo que recibe el número wolfram del autómata */
char numwolfram[longcadena];

/* Arreglo que guarda la regla de evolución del autómata */
char reglaevolucion[longregla];

/* Valor máximo a buscar del tamaño de vecindad de la regla inversa si el autómata es reversible */
int maxvecindad=6;

/* Guarda la permutación a realizar en la matriz de evolución para buscar el cluster mínimo */
int permutacion[numestados];

/* Guarda los cambios de estados a realizar en la matriz de evolución para buscar el cluster mínimo */
int estados[numestados];

/* Cadenas que almacenan números wolfram para encontrar el cluster mínimo */
char nw[longcadena];
char clustemp[longcadena];
char final[longcadena];

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

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

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

/* Número de clusters mínimos encontrados y archivo para guardarlos */
int totalclusters;
FILE *hoja;

/*Estructura del árbol binario que para descartar clusters ya generados */
struct tnodo
{
  char cadena[longcadena];
  struct tnodo *izq;
  struct tnodo *der;
};
typedef struct tnodo nodo;
typedef nodo *pnodo;
pnodo raiz;

/* Total de autómatas reversibles generados por el proceso */
int total;

/* 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 checarindices(int matriz[numestados][numestados], int largo, int inicio, int fin);
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();
pnodo lista(pnodo p, char palabra[longcadena]);
void listalibre(pnodo p);
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("\nReversible (4,h): %s",nw);
  printf("\nCluster del reversible (4,h): %s",final);
  printf("\nTotal de clusters minimos encontrados: %2d \n",totalclusters);
}

/* Inicializa la matriz de evolución con valores de -1 excepto la diagonal principal la cual se llena con valores
desde 0 hasta numestados-1 */
void inicio()
{
  int i,j;
  for(i=0;i<numestados;i++)
    for(j=0;j<numestados;j++)
      if(i!=j)
        matriz[i][j]=-1;
      else
        matriz[i][i]=i;
}
```

```
/* Borrar elementos de la matriz de evolución de un estado dado */
void borrar(int edo)
{
  int i;
  int t1,t2;
  for(i=0;i<numestados;i++)
  {
    t1=coordenadas[edo][i]/numestados;
    t2=coordenadas[edo][i]-(t1*numestados);
    matriz[t1][t2]=-1;
  }
}

/* Checa si el lugar a ocupar en la matriz de evolución está 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 evolución los elementos del estado según 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;
  }
}

/* Función que escoge que coordenada actualizar en la matriz de evolución */
int incrementar(int edo,int posicion)
{
  int i;
  int t1,t2;
  int flag;
  int valor,valor2;
  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 de un estado dado en la matriz de evolución */
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;
}

/* Verifica si hay ciclos de longitud 2 en la matriz de evolución */
int checarciclos(int edo)
{
  int i,j,k;
  int ciclo;
  int t1,t2;
  /*Obtiene la localizacion de el edo en la matriz y maximiza esta posicion */
  for(i=0;i<numestados-1;i++)
  {
    t1=coordenadas[edo][i]/numestados;
    t2=coordenadas[edo][i]-(t1*numestados);
    rutas[edo][i]=((t1*t1*t1)+(t2*t2*t2));
  }
  /*Compara si en el mismo estado hay ciclos */
  for(i=0;i<numestados-1;i++)
  {
    t1=rutas[edo][i];
    if(i>0)
      for(j=0;j<i;j++)
      {
        t2=rutas[edo][j];
        if(t2==t1)
          return -1;
      }
  }
  /* Verifica si se forman ciclos con estados anteriores */
  for(i=0;i<edo;i++)
  {
    ciclo=0;
    for(j=0;j<numestados-1;j++)
    {
      for(k=0;k< numestados-1;k++)
      {
        t1=rutas[i][k];
        t2=rutas[edo][j];
        if(t1==t2)
        {
```

```
            ciclo++;
            if(ciclo>1)
              return -1;
            break;
          }
        }
      }
    }
  }
  return 0;
}

/* Obtiene la matriz de conectividad del autómata de un estado dado y la guarda en la matriz aux */
void conectividad(int edo, int aux[numestados][numestados])
{
  int i,j;
  int t1,t2;
  for(i=0;i<numestados;i++)
    for(j=0;j<numestados;j++)
      aux[i][j]=0;
  for(i=0;i<numestados;i++)
  {
    t1=coordenadas[edo][i]/numestados;
    t2=coordenadas[edo][i]-(t1*numestados);
    aux[t1][t2]=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;
    }
}

/* Recibe una matriz de conectividad, la multiplica por las matrices de conectividad especificadas por el rango
establecido en las variables inicio y fin y verifica si L*R=k2r */
int checarindices(int matriz[numestados][numestados], int largo, int inicio, int fin)
{
  int i,j,k;
  int valor;
  int diagonal;
  int matrizaux[numestados][numestados];
  /* Si se examina un tamaño de vecindad mayor a maxvecindad el autómata se descarta como reversible */
  if(largo¿maxvecindad)
    return -1;
  /* 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 demás estados especi-
ficados */
  for(i=0;i<numestados;i++)
  {
    conectividad(i,matrizc);
    multiplo(matrizaux,matrizc);
    /* Si la suma de elementos es diferente que el valor de "índice," , 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];
    /* Si no se cumple con la Multiplicidad Uniforme el autómata se descarta como reversible */
    if(valor!=indice)
      return -1;
    /* Para que el autómata sea reversible, debe haber un sólo elemento en la diagonal principal y L*R=k^2r */
    diagonal=0;
    for(j=0;j<numestados;j++)
      diagonal+=matrizd[j][j];
    /* Si la traza es mayor que 1 entonces existen múltiples ancestros y el autómata no es reversible */
    if(diagonal¿1)
      return -1;
    /* Obtiene cuantos renglones cumplen con el valor del índice derecho */
    R=0;
    for(j=0;j<numestados;j++)
    {
      valor=0;
      for(k=0;k<numestados;k++)
        if(matrizd[j][k]==1)
          valor++;
      if(valor==derecho)
        R++;
    }
    /* Obtiene cuantas columnas cumplen con el valor del índice izquierdo */
    L=0;
    for(j=0;j<numestados;j++)
    {
      valor=0;
      for(k=0;k<numestados;k++)
        if(matrizd[k][j]==1)
          valor++;
      if(valor==izquierdo)
        L++;
    }
    /* Verifica si los indices cumplen con L*R=k^2r */
    valor=0;
    if(((R*L)==indice)&&(diagonal==1))
      valor=1;
    /* Si no se cumple la condición anterior se vuelve a hacer el proceso */
    if(valor==0)
    {
      valor=(checarindices(matrizd,largo+1,inicio,fin));
      if(valor==-1)
        return -1;
    }
  }
  return 0;
}

/* Verifica si el autómata es reversible */
int reversible()
{
  int i;
  for(i=0;i<numestados;i++)
  {
    conectividad(i,matrizb);
    if((checarindices(matrizb,2,0,numestados))==-1)
      return -1;
  }
  return 0;
}
```

```
/* Calcula el número wolfram de una matriz de evolución */
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 obtener la reflexión de la regla para encontrar el cluster mínimo */
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][i]==matriz[j][i])
        temp++;
    if(temp¿estcol)
      estcol=temp;
  }
}


/* Encuentra que renglones tienen más elementos de un mismo estado y guarda en el arreglo renglón esta información */
void encontrar(int matriz[numestados][numestados])
{
  int i,j;
  int temp,temp2,aux;
  temp2=aux=0;
  numedos=1;
  renglon[aux]=0;
  /* Encuentra que renglón tiene más elementos de un mismo estado */
  for(i=0;i<numestados;i++)
  {
    temp=0;
    for(j=0;j<numestados;j++)
      if(matriz[i][i]==matriz[i][j])
        temp++;
    if(temp¿temp2)
    {
      temp2=temp;
      renglon[aux]=i;
```

```
    }
  }
  /* Encuentra si otros renglones tienen el mismo número 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[i][i])
          temp++;
      if(temp==temp2)
      {
        renglon[numedos]=i;
        numedos++;
      }
    }
  }
}

/* Asigna a un arreglo la permutación 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 evolución y guarda ésta 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++)
      matrizc[i][j]=matriz[array[(numestados-1)-i]][array[(numestados-1)-j]];
}


/* Cambia los estados de la matriz de evolución */
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=matrizc[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 números wolfram y conserva el menor lexicograficamente hablando */
void menor()
{
  int temp;
  temp=strcmp(clustemp,nw);
  if(temp¿0)
    strcpy(clustemp,nw);
}


/* Obtiene el cluster mínimo de una matriz de evolución */
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<numedos;i++)
      for(j=0;j<numit;j++)
      {
        asignar((renglon[i]*numit)+j,permutacion);
        permutar(permutacion,matrizb);
        cambioest();
        wolfram(matrizd);
        menor();
      }
    if(estcol¿estren)
      strcpy(final,clustemp);
    else
      if((strcmp(final,clustemp))¿0)
        strcpy(final,clustemp);
  }
}
```

```
    /* Compara los clusters mínimos generados con los ya obtenidos por medio de un árbol binario y en caso de
ser nuevos los guarda en el archivo de resultados */
pnodo lista(pnodo p, char palabra[longcadena])
{
  if(p==NULL)
  {
    p=(pnodo) malloc(sizeof(nodo));
    strcpy(p-¿cadena,palabra);
    p->izq=p->der=NULL;
    wolfram(matriz);
    totalclusters++;
    hoja=fopen(archivo,"a");
    if(totalclusters==1)
      fprintf(hoja,"\n\nAUTOMATAS CELULARES LINEALES REVERSIBLES 4H con L=%2d\n",izquierdo);
    fprintf(hoja,"\nCluster Mnimo No.%3d: ",totalclusters);
    fprintf(hoja,"%s",palabra);
    fprintf(hoja," Especimen: %s",nw);
    fclose(hoja);
    imprimiravance();
  }
  else
  {
    if(strcmp(palabra,p->cadena)<0)
      p->izq=lista(p->izq,palabra);
    if(strcmp(palabra,p->cadena)>0)
      p->der=lista(p->der,palabra);
  }
  return p;
}

/* Libera la memoria utilizada por el árbol binario */
void listalibre(pnodo p)
{
  if( p->izq != NULL)
    listalibre(p->izq);
  if( p->der != NULL)
    listalibre(p->der);
  free(p);
}


/* Calcula las matrices de evolución de los posibles autómatas reversibles */
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();
            raiz=lista(raiz,final);
          }
        }
        else
```

```
            calculo(edo+1);
          }
        }
        borrar(edo);
        cond=actualizar(edo);
      }
    }

/*Cuerpo principal del programa */
void main()
{
  totalclusters=total=0;
  inicio();
  calculo(0);
  printf("\nTotal de matrices generadas: %d\n",total);
  listalibre(raiz);
}
```

Lista de Figuras

Lista de Tablas