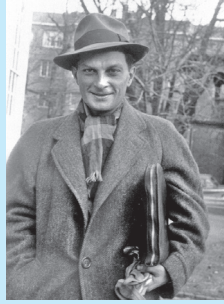


AUTÓMATAS CELULARES

Los autómatas celulares son una estructura computacional que permite añadir el concepto de “espacio” a los algoritmos. Un autómata celular es una matriz de celdas (o células) donde cada una de ellas está comunicada únicamente con sus vecinas. Todas las celdas reciben datos de entrada que les ofrecen sus vecinas, realizan un cómputo cambiando su estado interno y generan datos de salida para sus vecinas, de forma sincrónica, es decir, todas exactamente a la vez. No hay un control central, como en los sistemas de cómputo tradicionales. Todo es distribuido y las decisiones se toman localmente.

Aunque no se sabe muy bien quién los inventó, la primera referencia que se tiene es de Stanislaw Ulam, pues en 1950 le recomendó a John von Neumann su uso para el trabajo que quería desarrollar sobre autorreplicación.

Los autómatas celulares tienen múltiples aplicaciones: como modelo teórico de computación abstracta (la computación sistólica que tiene bastante utilidad en el proceso digital de señales de audio y de vídeo); como una clase de sistemas dinámicos discretos; como herramienta para simulación de vida artificial (por ejemplo, añaden la noción de espacio a las ecuaciones de predador-presa de Lotka Volterra, para así lograr modelar que los conejos puedan escapar de los zorros y estos puedan perseguir a aquellos); como pasatiempos matemáticos (hay mucha gente trabajando en sus ratos libres en el diseño de patrones con propiedades sorprendentes, para el juego de *LIFE* que veremos enseguida); como estructuras electrónicas (o de software), por ejemplo, para generar secuencias de números pseudoaleatorias usando *Linear Feedback Shift Registers*. En EVALAB los hemos usado para modelar tráfico peatonal, en el 2016 como tesis de maestría de Luz Estela Muñoz.



Fuente: Fotografía de dominio público. Los Alamos National Laboratory (1945).

Disponible en: <https://goo.gl/2Zurda>

Personaje 4

Stanislaw Ulam (1909-1984)

Stanislaw Ulam fue un matemático nacido en el imperio Austro-Húngaro (actual Polonia) y nacionalizado en USA. Trabajó con Teller, Fermi, von Neumann en el diseño de la primera bomba atómica, y con Metrópolis en la creación del método de simulación de Monte Carlo, usando el ENIAC, uno de los primeros computadores existentes. También trabajó en numerosas ramas de las matemáticas, incluyendo la teoría del caos y los autómatas celulares.

Él fue quien recibió la visita de Paul Erdős mientras estaba hospitalizado con encefalitis, quien le propuso diversos problemas matemáticos para verificar si su cerebro aún funcionaba bien (Ulam, 1976, p. 184).

Y en nuestro caso, los autómatas celulares tienen un gran interés porque permiten investigar propiedades fundamentales en mundos artificiales, como la computación universal, la autoduplicación (ambos temas los veremos en el presente capítulo) y la fabricación de mundos físicos similares o distintos al nuestro.

Definiciones

Como hemos dicho, un autómata celular es una matriz de celdas que puede ser de una dimensión (Figura 193), dos (Figura 194), tres o más y se extiende en todas direcciones de forma indefinida. Esto se hace para evitar bordes (celdas finales, sin vecinas a continuación) que pueden crear artefactos indeseados. Otra forma de evitar este problema es definir un tamaño máximo y conectar el borde derecho con el izquierdo y el de arriba con el de abajo, según una topología toroidal (Figura 195).

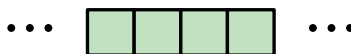


Figura 193. Autómata celular 1D

Los autómatas celulares tienen entonces un tamaño indefinido, pero no infinito, y esto, en *software*, se puede lograr utilizando matrices dispersas.

Por cierto que la matriz no tiene por qué ser de celdas cuadradas. Pueden ser también triangulares o hexagonales, es decir, las figuras geométricas regulares que logran teselar el plano.

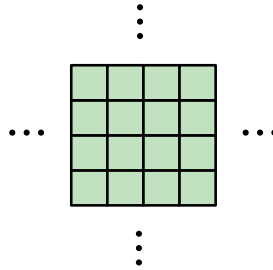


Figura 194. Autómata celular 2D

Por otro lado, hay que especificar lo que significa “celdas vecinas”, dando un conjunto de coordenadas relativas. Es decir, a las coordenadas de una celda cualquiera se le suman estas coordenadas relativas y así se obtienen sus celdas vecinas.

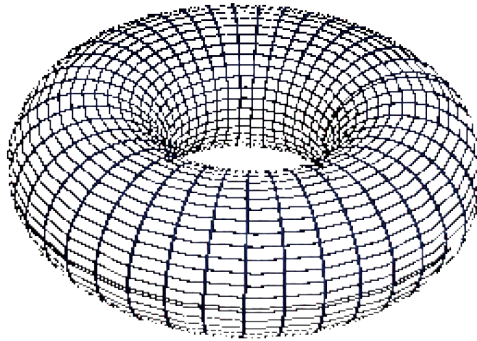


Figura 195. Autómata celular 2D finito sin borde (cada cuadradito es una celda)

Es importante entender que todas las celdas ejecutan el mismo algoritmo simultáneamente, y que si cada celda puede ofrecer resultados distintos es porque su estado interno o sus entradas son distintas. El algoritmo se puede escribir libremente en cualquier lenguaje de programación, aunque en los primeros trabajos sobre autómatas celulares siempre se usaba una máquina de estados finitos (de allí el nombre de “autómata”). La razón de hacerlo así era doble: por un lado, se conseguía sincronismo entre todas las celdas, ya que un autómata correctamente implementado requiere siempre la misma cantidad de tiempo para pasar del estado actual al siguiente. Por otro lado, se evitaba caer en el problema de la parada que podría ocurrir si le damos potencia de cómputo universal a cada celda.

Como hemos mencionado (Figuras 193 y 194), el número de celdas puede crecer indefinidamente, y potencialmente puede llegar a ser infinito. Para evitarlo se añade una restricción adicional buscando que el conjunto sea computable en un tiempo finito: existe un estado de inactividad # (también llamado estado quiescente), donde no es necesario realizar ningún cómputo si todas sus celdas vecinas también están en el mismo estado. Además, por definición, en todo momento, el número de estados no-quiescentes de un autómata celular debe ser finito. O sea, aunque el número de celdas podemos imaginarlo como infinito, el número de celdas donde hay que realizar cómputo debe ser finito.

Con ello no solo se logra modelar el espacio, sino que también se consigue un dispositivo de cómputo no centralizado, diferente a la implementación clásica de una Máquina de Turing Universal (aunque, en últimas, es matemáticamente equivalente a ella).

Un autómata celular se define formalmente como una tupla $C = \langle d, r, Q, \#, V, \delta \rangle$ donde:

- $d \geq 1$ es la dimensión del autómata. Ec. 67
- $r \geq 0$ es su índice de localidad. Ec. 68
- Q es el conjunto de estados. Ec. 69
- $V = (z_1, z_2 \dots z_r) \subset (Z^d)^r$ es un vector de vecindad, que contiene r elementos distintos de Z^d . Ec. 70
- $\delta : Q^{r+1} \rightarrow Q$ es la regla de transición del autómata. Ec. 71
- $\# \subset Q$ es el estado quiescente. Como hemos dicho, por definición $\delta(\#, \#, \#, \dots \#) = \#$. Este estado implica ausencia de actividad. Ec. 72
- Si $V = (z_1, z_2 \dots z_r)$ entonces las r celdas vecinas de la celda w se obtienen mediante sumas vectoriales: $(w+z_1, w+z_2 \dots w+z_r)$. Ec. 73

Las vecindades más utilizadas son la de Moore de rango k , $M(d, k)$, y la de von Neumann $V(d)$, que se definen:

$$M(d, k) = \{ (x_1, x_2 \dots x_d) : \forall i (1 \leq i \leq d) \rightarrow (-k \leq x_i \leq k) \} - (0, 0, \dots, 0) \quad \text{Ec. 74}$$

$$V(d) = \{ (x_1, x_2 \dots x_d) : \exists i (|x_i| = 1 \wedge \forall j (j \neq i \rightarrow j = 0)) \} \quad \text{Ec. 75}$$

Obsérvese que $V(d) \equiv M(d, 1)$. En la figura 196 podemos ver algunos ejemplos. Las más usadas y que emplearemos aquí son $M(2, 1)$ y $M(1, 1) \equiv V(1)$.

Dado un autómata celular C , se llama configuración de C a cualquier función $A: Z^d \rightarrow Q$ tal que $A^{-1}(\#)$ sea un conjunto cofinito. Es decir, en cada instante debe haber un número finito de estados distintos al quiescente. Esto es importante para asegurar la computabilidad del autómata, ya que así solo hay que calcular el estado siguiente de un número finito de celdas.

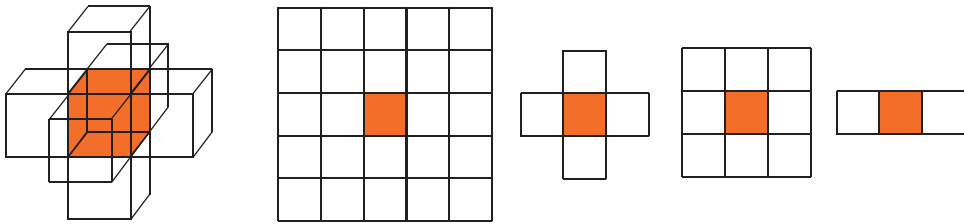


Figura 196. Ejemplos de vecindades $V(3)$, $M(2,2)$, $V(2) M(2,1)$ y $V(1)$, donde la celda q_0 que estamos estudiando está de color naranja y sus vecinas no tienen color

Sea $C = \langle d, r, Q, \#, V, \delta \rangle$ un autómata celular con $V = (z_1, z_2, \dots, z_r)$, y sea A una configuración de C . Definimos la configuración siguiente $\delta(A)$ como:

$$\forall y \in Zd : (\delta(A)(y) = \delta(A(y), A(y+z_1), A(y+z_2) \dots A(y+z_r))) \quad \text{Ec. 76}$$

Si llamamos A_0 a la configuración inicial, el autómata celular irá pasando sucesivamente por las configuraciones $A_0, A_1, A_2, \dots, A_n$. O sea,

$$A_i = \delta^i(A_0) \quad \text{Ec. 77}$$

Las reglas de transición más interesantes son las isotrópicas, es decir, aquellas en las que no hay direcciones privilegiadas para que ocurra algo. Dentro de ellas se suele trabajar con reglas totalísticas, cuyas transiciones solo dependen de la suma de celdas vecinas que cumplan con alguna propiedad.

LIFE

Un autómata celular muy interesante, que además es un mundo artificial muy conocido, es el del juego de *LIFE*, ideado por John Conway en 1970. Se define así:

$C = (2, 8, \{0,1\}, 0, V, \delta)$, o sea, es un mundo de 2 dimensiones con estados binarios, y vecindad $M(2,1)$: $V = ((-1,0), (-1,1), (0,1), (1,1), (1,0), (1,-1), (0,-1), (-1,-1))$. Y cuya función de transición es:

$$\delta(q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8) = \begin{cases} q_0 & \text{si } \sum_{i=1}^8 q_i = 2 \\ 1 & \text{si } \sum_{i=1}^8 q_i = 3 \\ 0 & \text{en los demás casos} \end{cases} \quad \text{Ec. 78}$$



Fuente: CC BY 2.0, Thane Plambeck (2005). Disponible en: <https://goo.gl/hvBj7g> y <https://goo.gl/kjwRoZ>

Personaje 5

John Conway (1937-)

John Horton Conway es un matemático británico que trabaja en diversos campos incluyendo la teoría de juegos de información perfecta. Se le conoce sobre todo por la invención de LIFE (el Juego de la Vida) que se popularizó rápidamente porque apareció en la columna de pasatiempos matemáticos de Martin Gardner, en la revista *Scientific American* en 1970. En aquella época no existían los computadores personales, por lo que las simulaciones de este juego las hacía a mano, en una cuadrícula de papel sobre la que situaba pequeñas piedras.

Cada celda solo puede tener dos estados: muerta o quiescente (de color negro) o viva (de color rojo). Y regla de transición de *LIFE* de la ecuación 78 se puede enunciar también de una manera menos matemática y más literaria:

- Si una celda está en contacto con 2 celdas vivas entonces su estado se mantiene.
- Si está en contacto con 3 celdas vivas entonces pasa a viva. A ese nacimiento a la vida se le llama reproducción.
- Si está en contacto con 4 o más celdas vivas entonces muere, se dice que por superpoblación.
- Si está en contacto con 1 o menos celdas vivas entonces muere, se dice que por aislamiento.

Es decir, de alguna manera *LIFE* modela un sistema de seres vivos muy elemental. Pero esto no es lo importante. Lo verdaderamente sorprendente es que con unas reglas de transición tan simples emerge un comportamiento terriblemente complejo a nivel global. En la figura 197 podemos ver una transición de una configuración a la siguiente, después de un tic del reloj síncrono. Hay algunas celdas que permanecen, otras nacen y otras mueren.

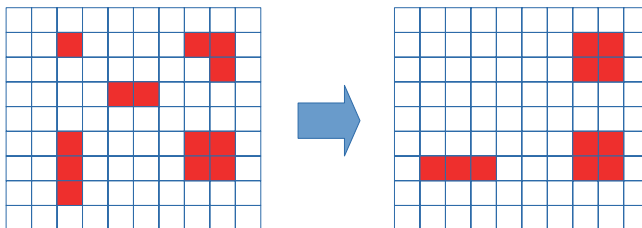


Figura 197. Una transición en *LIFE*

Se llama patrón a un subconjunto de una configuración, aislado del resto por celdas quiescentes (en la figura 197 hay inicialmente cinco patrones pero en el estado siguiente solo quedan tres). A veces se puede descomponer una configuración en un conjunto de patrones, pero a veces no. La gente que trabaja en *LIFE* ha ido descubriendo y dando nombre a un gran número de patrones.

Dependiendo de las condiciones iniciales, pueden observarse patrones que:

- Son estáticos y no cambian. Por ejemplo, en la figura 197, el de la esquina inferior derecha, de 2x2 celdas.

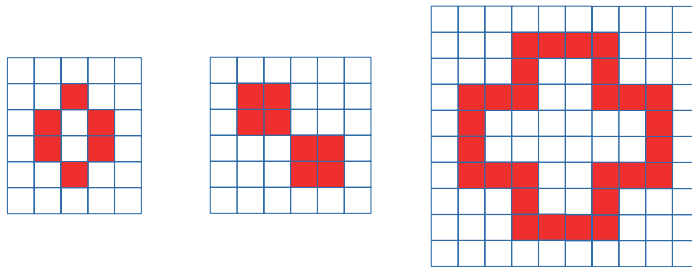


Figura 198. Otros osciladores de periodo 1 (estático), 2 y 3

- Oscilan periódicamente por varios estados. Por ejemplo, en la figura 197, el de la esquina inferior izquierda de 3 celdas verticales que se convierte a 3 celdas horizontales y luego a 3 verticales. Se pueden construir osciladores con cualquier periodo (Figura 198).
- Tienen una evolución larga o incluso desaparecen. En la figura 197 desaparecen dos patrones. Para ver evoluciones largas es mejor usar algún *software* de autómatas celulares.

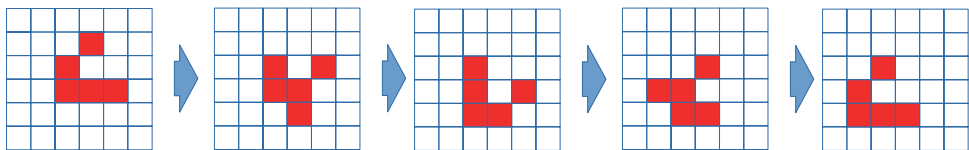


Figura 199. Un glider se mueve en diagonal hacia abajo a la izquierda, dando un paso en 4 tics de reloj

- Se mueven espacialmente. Un ejemplo es el *glider* de la figura 199 y otro el *spaceship* de la figura 200. Hay más complejos, como *rakes* y *puffers*. Otro patrón importante son los *guns*, que disparan continuamente gliders (Figura 201).

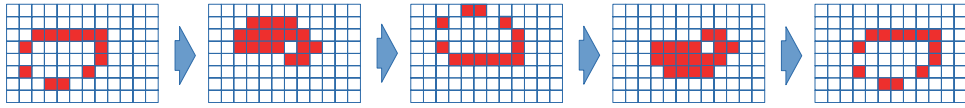
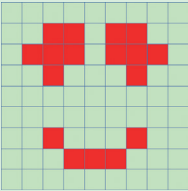


Figura 200. Spaceship que se mueve 2 pasos en horizontal hacia la derecha, en 4 tics de reloj



Problema 3: Nave espacial

Hay un spaceship que se mueve horizontalmente y es todavía más pequeño que el descrito en la Figura 200. ¿Puede usted diseñarlo?

- Todas estas estructuras son un ejemplo de emergencia: en las reglas que gobiernan este mundo artificial (es decir, la ecuación 78), no hay nada que fuerce, o ni siquiera sugiera, la existencia de los *gliders*. Sin embargo, si activas al azar muchas celdas y pones en marcha el autó-mata celular, es muy probable que aparezcan. Lo mismo ocurre con los osciladores básicos y muchos otros pequeños patrones. Emergen aunque nadie los ha diseñado intencionalmente.

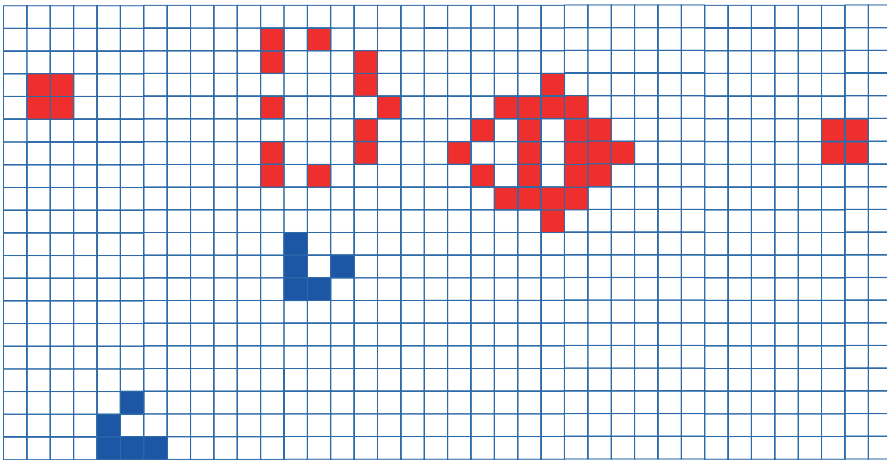


Figura 201. Patrón gun de Bill Gosper que está disparando continuamente gliders en diagonal hacia abajo a la izquierda (resaltados en color azul)

- Interacción entre estructuras. Un ejemplo es el de la figura 202, donde un glider choca contra un eater y es destruido. También hay espejos donde los gliders rebotan; y existen muchos más casos.

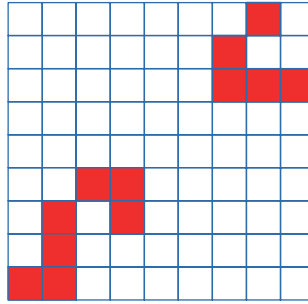
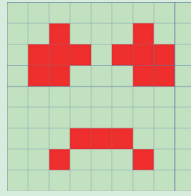


Figura 202. Patrón eater (abajo a la izquierda). Un glider (arriba a la derecha) se va a destruir al chocar contra él

Existe mucho *software* para jugar con *LIFE*. Actualmente el más completo puede ser *GOLLY*, que es gratuito y multiplataforma, y viene con muchos patrones preprogramados, donde se puede observar todo lo mencionado hasta ahora y mucho más. Es conveniente interactuar un rato con alguno de estos programas, para darnos cuenta del tipo de mundos que estamos creando. Una característica es que son mundos muy frágiles, pues basta una celda viva de más o de menos para que las configuraciones no lleven a cabo su cometido y muy probablemente se destruyan. También se da uno cuenta de que el diseño de cualquier cosa es muy laborioso. No suelen haber atajos. Únicamente por el método de prueba y error puede uno construir *spaceships*, *guns*, *puffers*, *rakes*, por nombrar algunos. Hay patrones que fabrican otros patrones (*syntethizers*). Y, lo más importante para nuestro objetivo, se pueden construir puertas lógicas (AND, OR, NOT y bits de memoria) donde un '1' es la presencia de un glider y un '0' es su ausencia, así como formas para enrutar estas señales en la dirección que uno desee. La puerta NOT es trivial, pues basta lanzar una secuencia de gliders que se choquen contra un flujo continuo de otros *gliders* generados por un *gun*. El choque los destruye mutuamente, generando un '0' donde en la secuencia original había un '1' o dejando pasar un '1' procedente del *gun* cuando en la secuencia de entrada hay '0'. Las otras puertas son más laboriosas de hacer, por lo que es mejor verlas en acción en (Bellos, 2016).

Que sea posible implementar cualquier circuito lógico en *LIFE* tiene una consecuencia trascendental: se puede construir una Máquina de Turing Universal (es un buen momento para cargar la que viene en *GOLLY*). La conclusión es que con algo tan simple y natural como una estructura de 2 dimensiones espacialmente distribuida con vecindad trivial $M(2,1)$ y con la sencilla regla de transición dada por la ecuación 78 se consigue computación completa.



Problema 4: Velocidad de la luz

En un autómata celular de vecindad $k=1$ (como el de *LIFE*), en un tic de reloj la información puede propagarse desde una celda hasta sus vecinas inmediatas. No hay forma de transmitir nada más rápido. O, al menos, eso es lo que dice la teoría. Esto es lo que se ha dado en llamar la velocidad de la luz de ese mundo artificial. Pero ¿será que hay manera de mover un patrón más rápidamente? ¿Qué patrón es el más rápido de todos?

Una vez alcanzado este objetivo, nada impide repetirlo todas las veces que se desee: dentro de una Máquina de Turing Universal (el computador que está encima de tu mesa) se puede construir un autómata celular que implemente la regla de *LIFE*, en donde hemos diseñado una Máquina de Turing Universal, dentro de la cual hay un autómata celular, y así indefinidamente. Ya se ha implementado esto en *Minecraft*, un juego que también posee capacidad de cómputo universal. Y otro ejemplo divertido e inquietante lo puedes encontrar en el video de Bradbury (2016).

¿Será que se puede conseguir una Máquina de Turing Universal de alguna manera más sencilla? Enseguida daremos una respuesta afirmativa a esta cuestión, pero antes, terminemos de explorar los autómatas $M(2,1)$. El número de reglas de transición posibles a investigar es enorme. Concretamente es $k^{k^{(r+1)}}$ siendo k el número de estados de una celda y r su número de vecinos. Para $k=2$ y $r=8$ son $2^{512} \approx 10^{154}$ reglas de transición posibles. Cada una de ellas da lugar a un mundo artificial. En muchos de ellos (como la regla B2/ o la B3678/S34678)⁵⁰ hay un único atractor (o unos pocos) que lleva a cualquier condición inicial a desaparecer, a estabilizarse en osciladores o a explotar y llenarlo todo. En otros se producen largos transitorios dependiendo de las condiciones iniciales (como la regla B36/S23), que puede ser indicio de que también es posible lograr allí la computación completa. Otros tienen propiedades curiosas, como en el que cualquier estructura evoluciona hasta formar un rombo (regla B35678/S5678) y, si hay varios, cuando se tocan se engullen unos a otros. Lo más interesante es observar cómo el patrón que se

⁵⁰ Las reglas totalísticas en autómatas celulares binarios 2D se identifican con dos listas de números separadas por una barra inclinada: primero (después de la B que significa birth, nacimientos), el número de celdas vecinas vivas que hace que la celda actual pase a viva. Y después (después de la S que significa sobrevivir) el número de vecinas vivas que hace que la celda actual mantenga su estado. Por ejemplo, la regla de *LIFE* es B3/S23.

forma lucha por mantener sus lados diagonales, por muy grande que sea el rombo y a pesar de que la regla es local (cada celda cambia su estado solo en función de sus vecinas). Y esto es solo en autómatas celulares de 2 dimensiones. Imaginen las posibilidades en 3 o más dimensiones.

Stephen Wolfram hizo lo contrario. Bajó a una dimensión con $N(1,1)$ porque allí solo hay $2^8 = 256$ funciones posibles y las analizó todas.

Autómatas celulares 1D

Cuando la dimensión es 1 y la vecindad es 1, el estado siguiente de cada célula solo depende del estado actual de esa célula y de las dos vecinas (la que tiene a la derecha y la que tiene a la izquierda). Como los estados son binarios {0=muerta, 1=viva}, la tabla de transición de estados es como la de la figura 203.

| ESTADO ACTUAL | | | ESTADO SIGUIENTE |
|-----------------|--------------|---------------|------------------|
| Celda izquierda | Celda actual | Celda derecha | Celda actual |
| 0 | 0 | 0 | |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | |

Figura 203. Tabla de transición de estados de un autómata celular $M(1,1)$

En esta figura, las columnas del estado actual, en color azul, tienen todas las combinaciones posibles de valores binarios. Queda por rellenar la columna del estado siguiente, que tiene $2^3=8$ casillas. Esas 8 casillas binarias se pueden rellenar de $2^8=256$ formas posibles, y esta es la razón por la que hay 256 reglas nada más. En la figura 204 podemos ver una de ellas que, por cierto, es la regla 6.

| ESTADO ACTUAL | | | ESTADO SIGUIENTE |
|-----------------|--------------|---------------|------------------|
| Celda izquierda | Celda actual | Celda derecha | Celda actual |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

Figura 204. Regla 6

Se llama así porque, si leemos la columna del estado siguiente de abajo a arriba, sale '00000110', un número binario que, convertido a decimal, da 6.

Trabajar con autómatas celulares de una dimensión tiene otra ventaja: como el papel (o la pantalla del computador) tiene dos dimensiones, es posible utilizar la otra para el tiempo. Es decir, en horizontal vamos a ver el espacio 1D y en vertical el paso del tiempo, de arriba a abajo. El estado inicial puede ser cualquiera, pero para ver qué ocurre en el caso más simple, vamos a poner solo una celda viva.

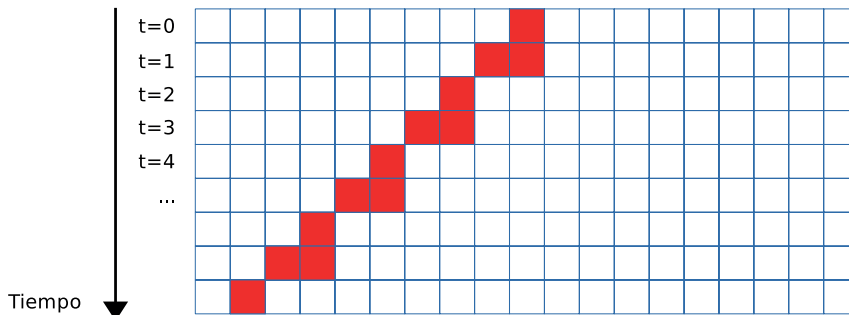


Figura 205. Desarrollo en el tiempo de la regla 6 con una semilla de una única celda viva

Entonces podemos ver en la figura 205 que en t=0 solo la celda central está viva. Aplicando la regla 6, se obtiene el estado de las celdas en t=1. Por ejemplo, la celda que está viva hace la transición 010 → 1. O sea, pasa a viva, mientras que la celda a su izquierda hace la transición 001 → 1 (viva) y la de su derecha 100 → 0 (muerta). Las demás no cambian debido a que 000 → 0, por lo que siguen muertas. Después se vuelve a repetir lo mismo

para calcular el estado de las celdas en $t=2$ a partir de su estado en $t=1$, etc. Puede verse que el patrón que sale es completamente regular y predecible a largo plazo. Para otras condiciones iniciales saldrá un patrón distinto, pero también muy predecible.

Una regla de transición más interesante es la 90, que podemos observar en la figura 206.

| ESTADO ACTUAL | | | ESTADO SIGUIENTE |
|-----------------|--------------|---------------|------------------|
| Celda izquierda | Celda actual | Celda derecha | Celda actual |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Figura 206. Regla 90

En la figura 207 se observa cómo se desarrolla esta regla a lo largo del tiempo. Sorprendentemente, es el fractal de Sierpinski que se vimos en un capítulo anterior.

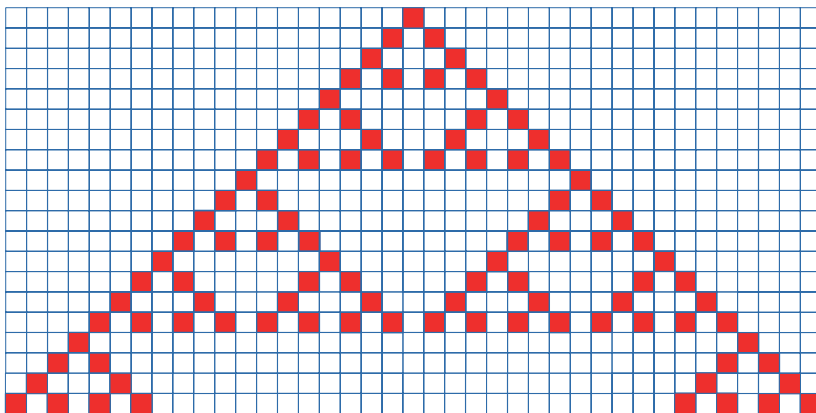


Figura 207. Desarrollo en el tiempo de la regla 90 con una semilla de una única celda viva

Otra regla interesante es la 30 de la figura 208, y cuya evolución en el tiempo se ve en la figura 209.

| ESTADO ACTUAL | | | ESTADO SIGUIENTE |
|-----------------|--------------|---------------|------------------|
| Celda izquierda | Celda actual | Celda derecha | Celda actual |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

Figura 208. Regla 30

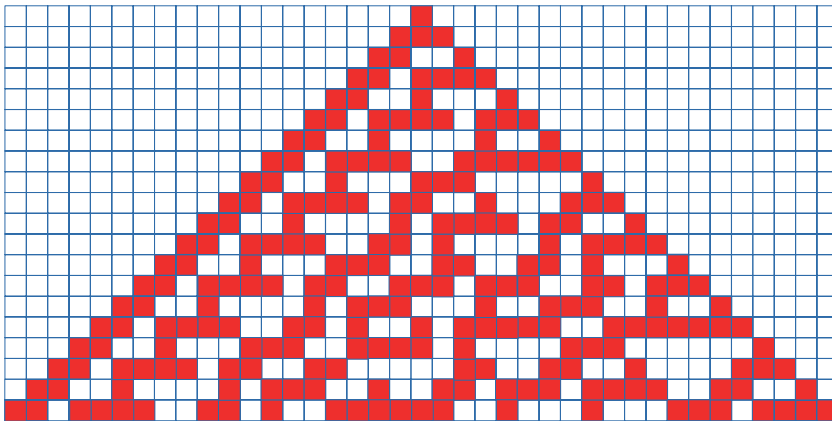


Figura 209. Desarrollo en el tiempo de la regla 30 con una semilla de una única celda viva

Puede verse que el patrón que sale es regular en ciertos sitios (en los bordes diagonales) e irregular en otros (en el centro). Si te pido que me digas qué va a salir 4 tics de reloj más allá de lo que aparece en la figura, la única forma de saberlo es calculando el estado de todas las celdas tic a tic. No hay atajos. Esto significa que el patrón debe ser estocástico, pseudoaleatorio o caótico. Estocástico no es, pues la regla de transición y las condiciones iniciales se conocen bien y son deterministas. Por tanto, es pseudoaleatorio o caótico. Devaney demostró que el sistema cumple las tres condiciones matemáticas del caos, aunque todo ello sigue siendo objeto de controversia, como puede verse en el texto de Cattaneo (1999).

| ESTADO ACTUAL | | | ESTADO SIGUIENTE |
|-----------------|--------------|---------------|------------------|
| Celda izquierda | Celda actual | Celda derecha | Celda actual |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Figura 210. Regla 110

Para terminar, veamos la regla 110 cuya tabla de transición está en la figura 210 y cuya evolución en el tiempo está en la figura 211.

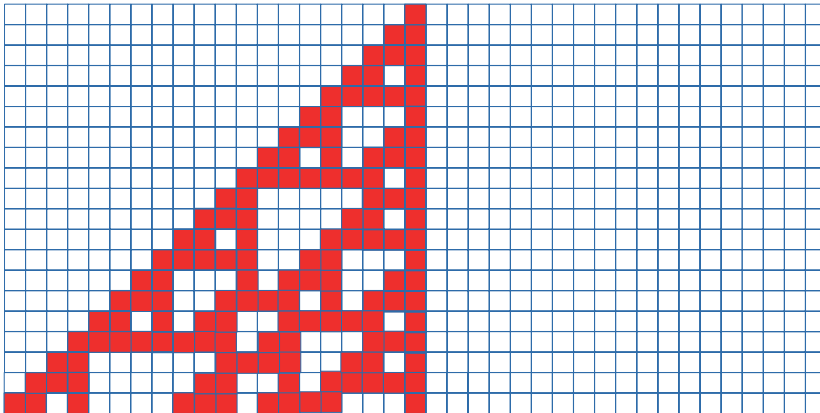


Figura 211. Desarrollo en el tiempo de la regla 110 con una semilla de una única celda viva

Wolfram clasificó estos autómatas de la siguiente manera, dependiendo de sus reglas de transición y para configuraciones iniciales aleatorias:

- **Clase 1.** Todas las configuraciones iniciales evolucionan hacia una única configuración muy sencilla, destruyéndose la información del pasado. El sistema es totalmente predecible, pues tiende a un único atractor. La regla 0 es así, donde todos los sucesivos estados son 0 (ya que la tabla de transiciones es $*** \rightarrow 0$), independientemente de las condiciones iniciales.

- **Clase 2.** Hay varios atractores y el resultado final es bastante predecible. Las condiciones iniciales dictan hacia cuál de ellos se dirigirá el sistema. Cambios pequeños en las condiciones iniciales no cambian la cuenca de atracción. El sistema es predecible si se conocen *grosso modo* las condiciones iniciales. Un ejemplo de ello es la regla 6.
- **Clase 3.** La evolución es pseudoaleatoria. Solo sabiendo con toda exactitud cuáles fueron las condiciones iniciales, se podrá predecir el estado futuro del autómata celular. Y la forma más rápida de predecirlo es dejando que se ejecute. Un ejemplo de ello es la regla 30.
- **Clase 4.** La evolución es caótica, es decir, predecible a corto plazo, pero no a largo plazo. Hay largos transitorios que desembocan en cuencas de atracción, generando estructuras periódicas, no periódicas, que se mueven espacialmente. Es el tipo más complejo e interesante. Son difíciles de predecir. En algunas configuraciones se sabe que estos autómatas celulares son dispositivos de cómputo universal. Un ejemplo de ello es la regla 110.

Cristopher Langton estudió estos autómatas celulares en función de un parámetro *lamda* (λ) que definió así:

$$\lambda = \frac{\text{número de transiciones que producen un estado no quiescente}}{\text{número total de transiciones}} \quad \text{Ec. 79}$$

Este parámetro mide de forma grosera el índice de actividad. Es el típico dial con el que vamos barriendo el espacio de configuraciones, igual que vimos en el capítulo de caos. Al ir variando λ de 0 a 1 obtuvo los siguientes resultados aproximados:

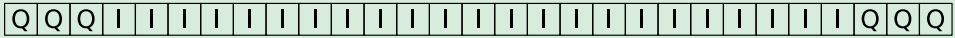
- Valores pequeños de λ indican poca actividad, o sea, mucho orden. Estos son los autómatas celulares de clase 1, según la clasificación de Wolfram.
- Al aumentar λ aparecen estructuras estables, o sea, sigue habiendo orden y predictibilidad. Estos son los autómatas celulares de clase 2.
- Al aumentar un poco más λ aparecen estructuras metaestables, o sea, casi-estables, disminuyendo la predictibilidad. Estos son los de clase 4.
- Y al llegar a los valores más altos de λ aparece el desorden, lo estocástico, o sea los de la clase 3.

Luego estudió con más detalle la clase 4 y vio que era el resultado de una transición de fase entre orden y desorden, que es lo que se conoce como borde del caos. Si ya has leído el capítulo sobre el caos, esto te sonará bastante.



Problema 5: Bandera francesa

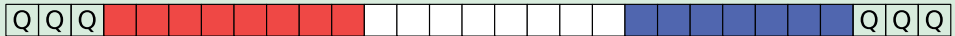
Tenemos un autómata celular de una dimensión de tipo $M(1,1)$, inicializado con un segmento de células en el mismo estado I , y donde cada célula exhibe un color que depende de su estado (estados distintos podrían tener el mismo color). Como el autómata es infinito, las demás células están en estado quiescente Q .



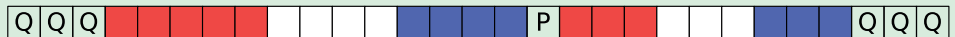
A partir de una perturbación inicial P en uno de los extremos, debe desarrollarse la bandera francesa: un tercio rojo, un tercio blanco, un tercio azul.



Etc. Finalmente debe quedar:



Y si el segmento se divide en dos (poniendo alguna celda en estado P), cada parte debe regenerar su propia bandera, orientada en el mismo sentido que la original. Esto se puede repetir indefinidamente.



Su misión consiste en desarrollar el software que implemente todo esto. Recuerde que todas las celdas tienen el mismo algoritmo.

Mientras tanto, Wolfram intuía que la regla 110 tenía capacidad de cómputo universal. No obstante, fue su empleado Mathew Cook en 2004 quien logró demostrarlo publicando los resultados, contrariando bastante a su jefe. Parece ser que finalmente llegaron a un acuerdo sobre la autoría de estos trabajos. Pero lo importante para nosotros es que la regla 110 es equivalente a una Máquina de Turing Universal. Programarla no es nada fácil. Por ejemplo, si quieres sacar la lista de números primos o cualquier otra cosa, debes poner ciertas celdas vivas o muertas en la fila inicial (o sea, en $t=0$). ¿Cuáles? Eso es lo difícil, pues no se trata de un lenguaje ensamblador ni nada a lo que estemos acostumbrados. Podría usarse un algoritmo genético, dado que se trata de encontrar un vector binario que realice cierta tarea. Lo habitual es decidir que la salida de este “computador” esté en la columna central. Por cierto, las reglas 124, 137 y 193 también son Máquinas de Turing Universales, pues son equivalentes a la regla 110 sin más que hacer algunas simetrías y complementos.

Este es uno de los resultados más importantes en lo que se refiere a los objetivos de este libro: con muy poca infraestructura (un conjunto de cel-

das interconectado linealmente de forma local, y una pequeña máquina de estados en cada celda), de cada 256 configuraciones, cuatro son computadores. Es, definitivamente, sorprendente. La probabilidad de encontrar un computador cuando se tiene cierta complejidad es bastante alta, alrededor del 2%. Uno creería que los computadores son bastante complejos, y si abres uno te encontrarás con procesadores de miles de millones de transistores, memoria RAM, disco duro, buses y periféricos. Sin embargo, en lo básico, para construir un computador requieres del orden de 8 bits de información donde almacenar la tabla de transiciones y una estructura de conexiones simple, local y repetitiva.

Este es uno de los hitos de la complejidad, en el que emerge la capacidad de cómputo universal. Y emerge en un punto a medio camino entre el orden y el desorden, o el borde del caos. Es lógico que así ocurra: en sistemas ordenados no aparece nunca nada nuevo. Todo es estático o periódico. Allí es imposible lograr la flexibilidad de la computación. Por el contrario, en sistemas completamente desordenados no hay nada fiable, si intentas repetir un proceso sale cada vez distinto. El cómputo también es allí imposible. La computación requiere algo de orden (por ejemplo, un dato que guardas en memoria debería continuar allí cuando vayas a buscarlo), pero no tanto que impida la flexibilidad y creatividad de los algoritmos.

Por otro lado, la comunicación local es algo muy natural, no solo biológicamente hablando, sino matemáticamente, pues apenas se requiere geometría. Pensemos que en un cristal las vibraciones de un átomo afectan a sus vecinos; las células de un órgano afectan y son afectadas por las vecinas; en una comunidad de animales, los virus, los conocimientos, la comida y la información se propagan a los vecinos; las placas tectónicas empujan y son empujadas por las vecinas; y así podríamos poner infinidad de ejemplos.

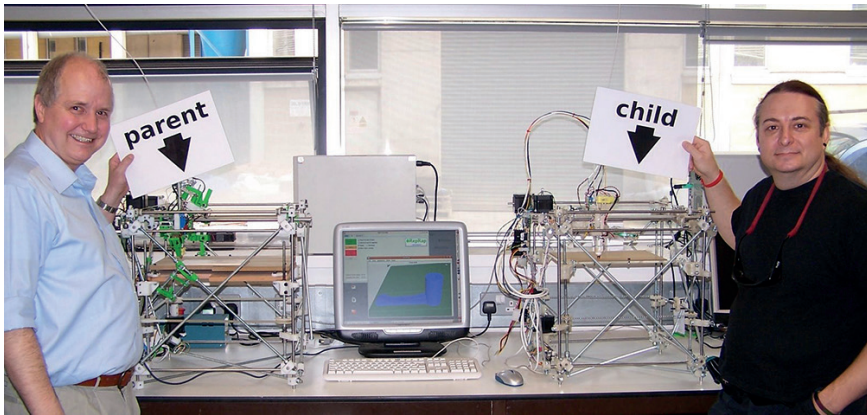
Si en un ambiente así disponemos de 8 bits de decisión y generamos las combinaciones posibles, al lanzar los dados, en el 2% de las veces saldrá un computador. ¡Sorprendente! Y nadie negará que, teniendo un computador a mano, las posibilidades de generar más complejidad se expanden dramáticamente, saltando a un nuevo nivel.

AUTODUPLICACIÓN

Con los autómatas celulares también se puede lograr autorreplicación⁵¹. Este fenómeno es importante por varias razones:

51 Que puede sacar una copia de sí mismo. Usaremos como sinónimos: autorreplicación, autoduplicación, autocopia y autorreproducción.

- Es un elemento imprescindible de la evolución. Al principio del capítulo “Algoritmos evolutivos” mencionamos los cuatro factores necesarios para que surja la evolución, y el autocopiado es uno de ellos. No solo eso, sino que es el más difícil de lograr de los cuatro. Podemos simplificar diciendo que cuando aparece el autocopiado, surge también la evolución. Y la evolución es un excelente generador de complejidad. Por eso nos interesa saber qué se requiere y cuál es el umbral necesario para que aparezca la autoduplicación.



**Figura 212. RepRap fue el primer objeto industrial autorreplicante.
Fotografía bajo el dominio CC BY-SA 3.0.**

Disponible en: <https://goo.gl/3VoHVq>

- Es el sueño de todo ingeniero para no volver a trabajar. Ya comienzan a aparecer las primeras máquinas de autocopia, como las impresoras 3D con las cuales puedes fabricar cualquier objeto, incluyendo otra impresora 3D (bueno, ¡vale!, las partes metálicas y electrónicas todavía no, pero solo es cuestión de tiempo). La impresora 3D RepRap de Adrian Bowyer (Figura 212) fue la primera construida con esas intenciones que es, además, de bajo costo y de acceso libre a los planos de construcción. Este tipo de máquinas nos permitirá en un futuro hacer ingeniería a nivel planetario o aún más ambiciosa, como fabricar una esfera de Dyson⁵². Como hecho curioso, el escritor de ciencia ficción Arthur Clarke se adelantaba a ello en la segunda parte de su novela *2001: una odisea en el espacio*, donde proponía como convertir el planeta Júpiter en una estrella, usando máquinas autorreplicantes.

⁵² Una hipotética estructura espacial que se construiría rodeando por completo a una estrella con el objetivo de capturar toda la energía que emita.

- Por último, la autorreplicación es una realimentación positiva, que da lugar a crecimientos exponenciales y en muchos casos puede producir fenómenos caóticos.

En la actualidad la autorreplicación ya se lleva a cabo rutinariamente con robots como el del trabajo de Cornell (2009), formado por 4 cubos idénticos que se conectan entre sí con electroimanes, se transmiten entre ellos información y tienen un eje de rotación en su diagonal. Un robot de 4 cubitos puede fabricar en muy poco tiempo otro robot de 4 cubitos a su lado, como se ve en el video mencionado. Sin embargo, ¡hay un truco!, comenta mucha gente cuando lo ve: los cubitos ya están fabricados previamente, y lo único que hace el robot es ensamblarlos en el orden correcto. Aunque esto es verdad, no le quita mérito ni interés al trabajo por la siguiente razón: los animales también sacamos copias de nosotros mismos, pero para lograrlo tenemos que ingerir moléculas largas previamente fabricadas procedentes de otros animales y vegetales. Ningún animal come exclusivamente tierra para alimentarse. Lo mismo pasa con estos robots. Falta únicamente dirigir trabajos de desarrollo para conseguir que las piezas de partida sean cada vez más básicas, hasta lograr que haga autocopia usando minerales que se encuentren en el suelo. En cierto modo, esto ya está muy avanzado, pues las fábricas humanas de metalmecánica, microelectrónica y plásticos, cada vez están más automatizadas, de modo que las piezas básicas que se entreguen a los robots autorreplicantes procederán también de procesos que pueden replicarse. Otra línea de trabajo prometedora es la nanotecnología, que permitiría copiar objetos átomo a átomo.

Ni que decir tiene que el futuro que nos espera puede ser asombroso o aterrador, en el momento en que un robot pueda sacar copias de sí mismo. Pero el momento está muy cerca.

Después de explicar la importancia del proceso de autocopia, nos debemos estar preguntando qué se requiere para que surja espontáneamente en la naturaleza (o dentro del computador). Por suerte para nosotros, John von Neumann ya hizo este estudio de forma teórica, y aunque no alcanzó a publicar sus resultados, un amigo suyo si lo hizo (von Neumann y Burks, 1966).

| | | | | |
|-----------------|-----------------|------------------|------------------|--------------------------------------|
| | | | U | Estado quiescente (desprogramado) |
| ↑ | ↓ | → | ← | Caminos normales |
| ↑ | ↓ | → | ← | Caminos normales con una señal |
| ↑ | ↓ | ⇒ | ⇐ | Caminos alternativos |
| ↑ | ↓ | ⇒ | ⇐ | Caminos alternativos con una señal |
| C ₀₀ | C ₀₁ | C ₁₀ | C ₁₁ | Confluencia (puerta AND con memoria) |
| S ₀ | S ₁ | S ₀₀ | S ₀₁ | Estados intermedios constructivos |
| S ₁₀ | S ₁₁ | S ₀₀₀ | S ₀₁₀ | |

Figura 213. Los 29 estados de von Neumann

Él empleó un autómata celular de tipo $V(2)$, donde cada célula podía tener uno de los 29 estados que aparecen en la figura 213. Como en todo autómata celular, hay un estado quiescente U . Luego hay cuatro estados que sirven para trazar caminos por donde circulen señales. Están también esos cuatro caminos pero con la señal circulando (un círculo alrededor de la flecha). Esta situación puede causar confusión al principio, pues en ambientes de programación orientados a objetos, hay un camino y hay un objeto aparte que recorre el camino. Recordemos que aquí el estado es todo: existe el mismo algoritmo en todas las celdas y lo único que las diferencia es el estado. Por eso, tanto la señal como el camino se codifican dentro del estado. En la figura 214 se muestra un claro ejemplo de lo que ello significa. La señal inicialmente está en la casilla 'b2', y en el siguiente tic de reloj la casilla 'b1' ve que a su derecha hay una señal en un camino que apunta a la izquierda, por lo que ella se convierte en señal, sin cambiar el camino que ya tiene apuntando a la izquierda. En resumen, cada celda mira si a su alrededor hay una señal con un camino apuntando hacia ella, en cuyo caso, en el tic siguiente, pasa ella misma al estado 'señal'. También mira si ella está en estado señal y, en caso afirmativo, en el estado siguiente la señal desaparece. Con estos dos pasos se simula el movimiento de señales por caminos.

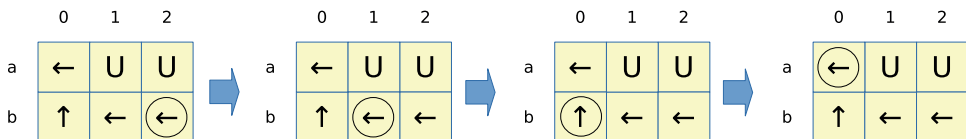


Figura 214. Una señal moviéndose tranquilamente por un camino

Las flechas dobles con sus respectivas señales funcionan exactamente de la misma manera. La única particularidad con los dos tipos de flechas es que si hay una señal viajando por un camino de flechas simples y tropieza con

un camino de flechas dobles (o al revés), entonces la celda se desprograma, pasando a estado quiescente U, como puede observarse en la figura 215.

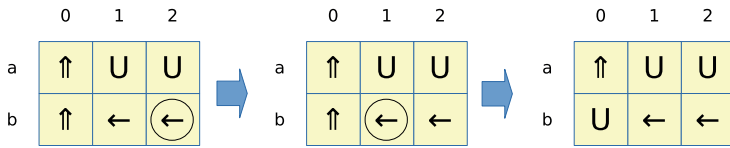


Figura 215. Choque de caminos

Los siguientes cuatro estados configuran una puerta AND secuencial, con memoria. La idea es que cuando todas sus entradas (todos los caminos que apunten a la puerta) reciben a la vez una señal, entonces la salida generará también una señal, pero un tic de reloj más tarde. Al haber ese retraso, podría ocurrir que la puerta esté memorizando que le toca activarse en el siguiente tic de reloj y a la vez que le lleguen nuevas señales de activación. Esta es la razón por la que se requieren más estados internos, cuyas transiciones se explican en la figura 216.

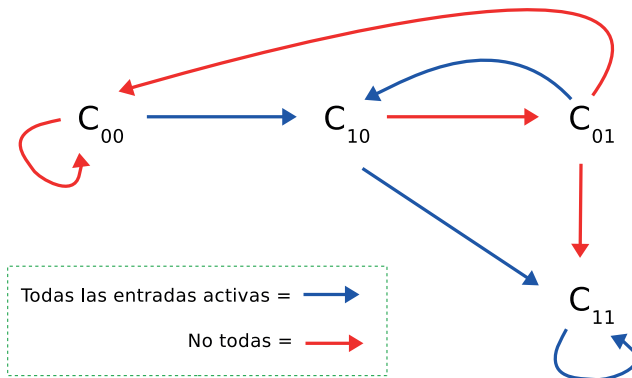


Figura 216. Transiciones posibles en una puerta AND con memoria

Por último, hay ocho estados intermedios que permiten la construcción de celdas con un cierto estado inicial, a partir de la llegada secuencial de señales que choquen contra una celda en estado quiescente, según se explica en la figura 217. Las celdas que se pueden construir así son nueve (las cuatro flechas simples, las cuatro flechas dobles y la puerta AND en su estado inicial). Para entender mejor la figura supongamos, por ejemplo, que a una celda desprogramada (en estado U) le llega la secuencia 1101 (donde 1 significa una señal y 0 ausencia de ella). Entonces al final de 4 tics de reloj quedará programada como una flecha doble hacia la izquierda (\Leftarrow).

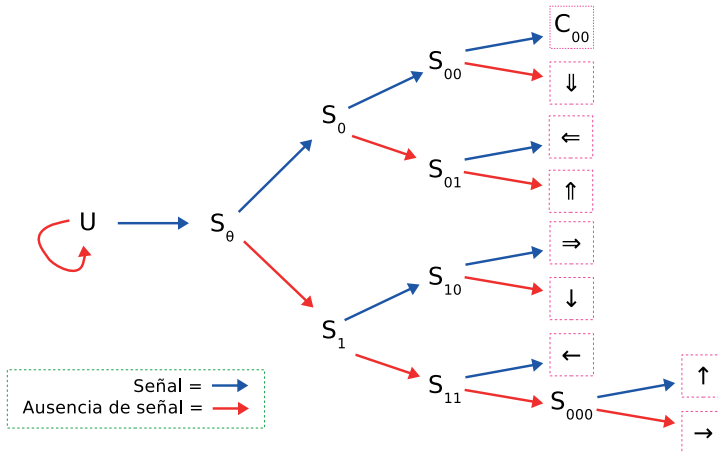


Figura 217. Programación del estado de una celda previamente desprogramada

Para lograr una mejor comprensión de todo esto, veamos un ejemplo en la figura 218, donde hay nueve señales circulando:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|-----------------|-----------------|---|---|-----------------|---|---|
| a | ↑ | ← | ← | ↓ | ⊕ | ← | U |
| b | → | C ₀₁ | ⇒ | ⊕ | C ₀₀ | ⇒ | ↓ |
| c | ↑ | ⊕ | U | ← | ← | ← | ↓ |
| d | C ₁₁ | → | ⊕ | → | ⊕ | ⊕ | ↑ |

Figura 218. Un trozo de autómata de von Neumann

- La de 'c1', que se va a perder por no haber otra en 'b0' para alimentar la puerta AND.
- La de 'a4' que junto con la de 'b3' alimentarán la puerta AND que hay en 'b4' llevándola al estado C₁₀.
- La que hay en espera internamente en la puerta AND de 'b1', que saldrá por 'b2' en el siguiente tic de reloj dejando 'b1' en el estado C₀₀.
- En 'd0' hay dos señales: una que acaba de entrar a la puerta AND y otra que entró en el tic anterior. En el siguiente tic de reloj, 'd0' quedará en estado C₀₁ y saldrán señales simultáneamente por 'c0' y 'd1'. Y en el siguiente tic de reloj pasará a C₀₀ y saldrá la otra señal acumulada, por el mismo sitio.
- Finalmente, la secuencia de señales que hay en 'd5', 'd4' y 'd2' chocarán en sucesivos tics de reloj con la flecha doble que hay en 'd6' (concretamente llegará la secuencia 11010). Como el camino por el que vienen esas señales es de tipo flecha simple, la celda 'd6' quedará

desprogramada (en estado U) al llegar el primer 1. Las demás señales 1010 convertirán la celda 'd6' en una flecha simple hacia abajo (\downarrow).

Hasta aquí la carpintería, pero ¿para qué le sirvió todo esto a von Neumann? Pues resulta que con ello pudo crear dentro del autómata un brazo constructor de cualquier longitud, como se muestra en la figura 219. Una señal que se inyecte por la fila 'b' chocará al final con la flecha doble que hay en 'a4', desprogramándola. Enviando después una secuencia adecuada de señales podemos programar cualquier cosa en 'a4'. Algo similar haremos con 'b4', dejando el brazo retraído en una celda, y listo para repetir lo mismo con 'a3' y 'b3'.

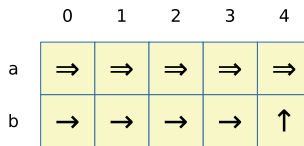


Figura 219. Brazo constructor

El proceso completo es simple pero laborioso y lo podemos ver en la figura 220. En el último paso vemos que se han escrito dos símbolos cualesquiera (representados por α y β) en las casillas del extremo, y el brazo constructor se ha retraído. De modo que el proceso puede continuar indefinidamente.

De forma análoga se puede hacer un brazo constructor en dirección vertical, y combinando ambos se logra que un autómata celular Q convenientemente diseñado, inyecte señales de manera adecuada para construir un rectángulo P de celdas con cualquier símbolo en ellas (Figura 221). Es decir, un autómata puede construir otro autómata. ¡Estamos a punto de lograr la autoduplicación!

Pero las cosas no son tan fáciles porque el autómata Q es muy complejo, concretamente Q sale bastante más grande que P . Afortunadamente a von Neumann se le ocurrió una forma de reducir el tamaño, separando por un lado el brazo constructor con sus circuitos de control y, por otro, la información de descripción del nuevo autómata en una cinta⁵³. La cinta está formada por cuatro filas de celdas (Figura 222): las dos primeras A y B emiten señales para leer la cinta, para reconstruir su contenido, pues la lectura es destructiva, y para retraerse a la posición anterior y leer la siguiente celda. En últimas es un brazo constructor también. La siguiente C es donde está la información de descripción del nuevo autómata celular P que se va a cons-

53 Una cinta funcionalmente idéntica a la de las máquinas de Turing.

truir. Su formato es binario siguiendo el convenio de que U significa 0 y ↓ significa 1. Por la última fila D se recibe la información leída de la cinta.

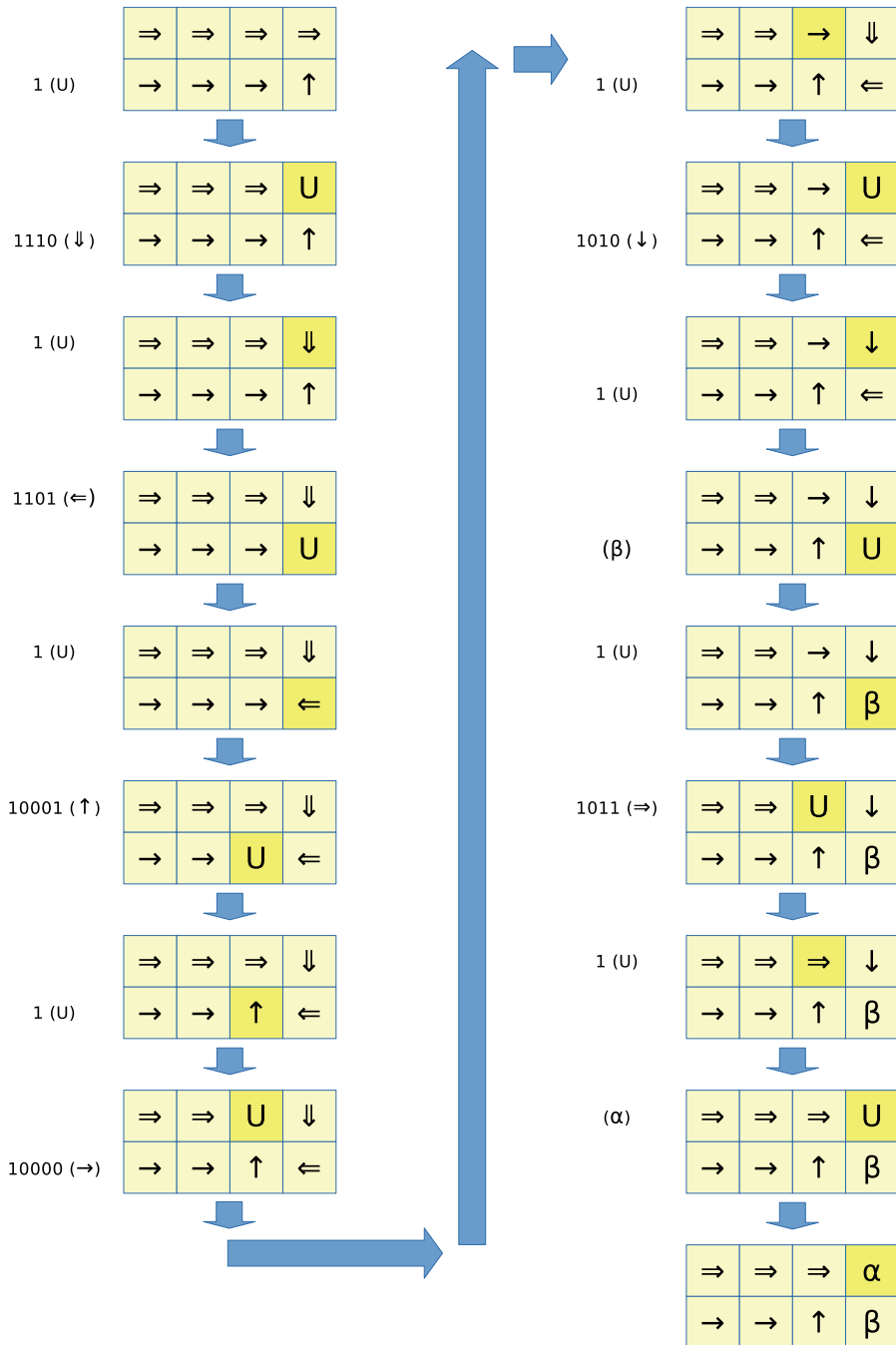


Figura 220. Uso del brazo constructor

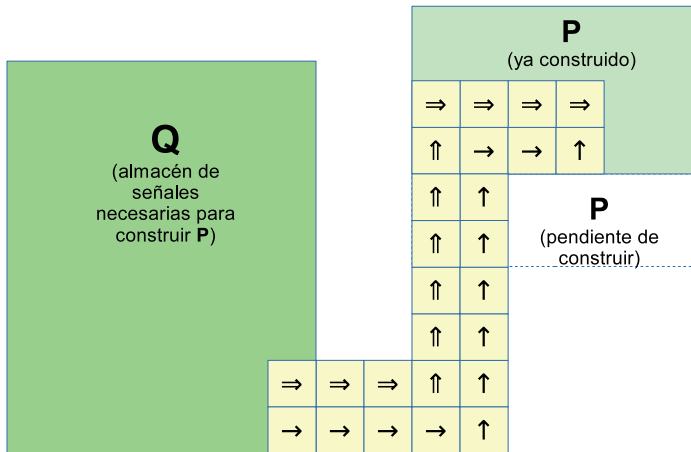


Figura 221. Un autómata *Q* construyendo otro autómata *P*

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | → | → | → | ↓ | | | | | | | | |
| B | ⇒ | ⇒ | ⇒ | ↓ | | | | | | | | |
| C | U | U | ↓ | ? | ↓ | ↓ | U | ↓ | ↓ | U | ↓ | U |
| D | ← | ← | ← | ← | ← | ← | ← | ← | ← | ← | ← | ← |

Figura 222. Cinta de información

Supongamos que queremos leer la información binaria que hay en la celda resaltada con otro color. Como no sabemos lo que hay, le hemos puesto ‘?’. Para hacerlo, el proceso es enviar 10101 por la fila *A*. Si en *D* se recibe lo mismo (10101) es que la celda contiene 1 (o sea, ↓, lo cual es obvio). Mientras que si se recibe 1 significa que la celda contiene 0, o sea, *U*, lo cual ya no es tan obvio y para entenderlo tendremos que recurrir a la figura 217.

Entonces la estrategia a seguir dado un patrón *P* que queramos construir, es codificar sus dimensiones seguido de los estados de cada célula (en un cierto orden). A ello lo llamaremos $d(P)$, o sea, descripción de *P*. En la cinta se coloca $d(P)$ y el constructor universal *M* irá leyéndola, para construir *P* (Figura 223).

Para lograr la autoduplicación hacemos $P=M$, de manera que en la cinta está $d(M)$. Aunque esto no es suficiente por dos razones: $M+d(M)$ construyen *M*. O sea, la réplica no es exacta, pues le falta la cinta. Y el nuevo *M* no está “vivo”, ya que se genera con todos sus estados inactivos (es decir, sin señales circulando), de modo que no podrá a su vez reproducirse.

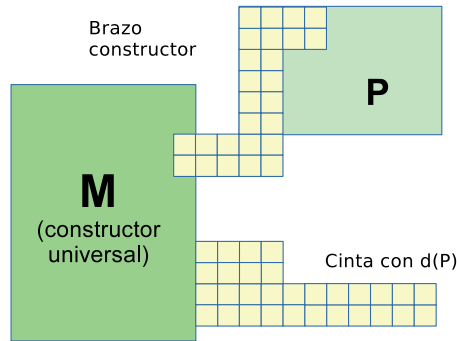


Figura 223. Constructor universal M creando otro autómata celular P

Para solventar estos problemas, se modifica M a un M' que no solo interpreta la cinta sino que, al finalizar, saca también una copia de ella. Además, se diseña un bloque de modo que todas las señales de M' se generen a partir de una única señal inicial. Así, al terminar la copia únicamente debe enviarse esa señal desencadenante a la copia. Lo cierto es que esto último no es posible en todos los casos de modo que el constructor de von Neumann no llega a ser completamente universal, si bien lo es a todos los efectos prácticos.

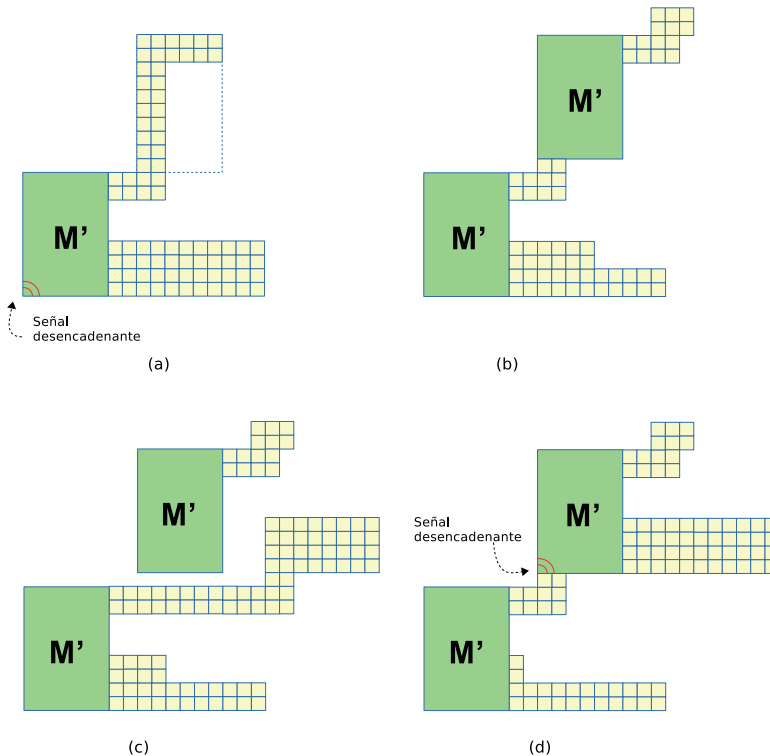


Figura 224. Funcionamiento del constructor universal

En la figura 224 podemos ver las fases de funcionamiento. En la primera fase (a) el constructor universal M' recibe la señal para ponerse en marcha, lo cual hace que en (b) lea la cinta y saque una copia de sí mismo. En (c) vuelve a leer la cinta y la copia literalmente en la máquina ya creada. Y en (d) envía a la nueva máquina una señal para activarla y que comience ella a sacar su propia copia. Es importante recalcar que la cinta juega dos papeles:

- Su información se interpreta activamente para construir una copia de M' .
- Su información se interpreta pasivamente como datos a ser copiados y anexados a la copia de M' .

Esto es análogo a lo que hace el ADN fabricando proteínas pero también sacando copias de sí mismo. También es similar al proceso de gödelización de la aritmética, donde por un lado las cadenas de símbolos operan para construir teoremas pero, por otro, hablan de sí mismos. Es un caso más de autorreferencia.

El propio von Neumann se dio cuenta de era más eficiente para un objeto sacar una copia de sí mismo a partir de su descripción simbólica (Figura 225) que a partir de su introspección material. Es decir, aunque este trabajo sea muy teórico, tiene consecuencias cuando queramos fabricar robots autorreplicantes.

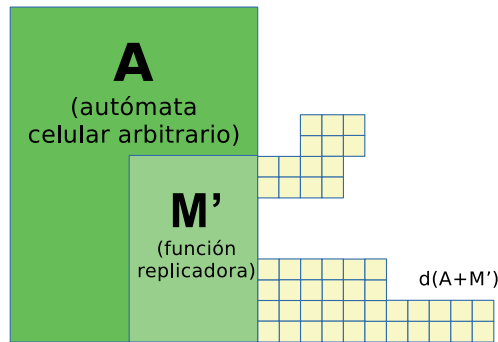


Figura 225. Cómo se logra la copia universal

Lo que acabamos de ver es solo un esbozo del trabajo que von Neumann realizó en 1949, pues hay muchos más detalles a resolver. Lo importante es que necesitó 40 000 células y 29 estados. Dado que 29 posibilidades necesitan 5 bits para ser codificadas, entonces la cantidad de información requerida para lograr autocopia es de $40\,000 \times 5 = 200\,000$ bits de complejidad.

En 1968, Codd necesitó 8 estados y 100 000 000 células. En 1970, Devore las redujo a 87 500. Pero la complejidad medida en bits es similar.

Con ello llegamos a entender otro hito de la emergencia. Para lograr autocopiar, y con ella poner en marcha la evolución, se requieren del orden de 2×10^5 bits. ¿Es mucho o es poco? Por un lado, los primeros disquetes que aparecieron en 1971 eran de 80 kilobytes, es decir, más o menos 6×10^6 bits, un orden de magnitud mayor que el número que tenemos entre manos. No menciono la capacidad de almacenamiento de hoy día, pues es muchos órdenes de magnitud mayor. O sea que, desde el punto de vista tecnológico, 2×10^5 bits no son gran cosa.



Fuente: Fotografía de dominio público, LANL (2010).
 Disponible en: <https://goo.gl/zcUgNP> y <https://goo.gl/ZnPQJX>

Personaje 6

Von Neumann (1903-1957)

John von Neumann fue un matemático húngaro que trabajó la mayoría de los temas considerados en este libro. En el presente capítulo destacamos su investigación en sistemas autorreplicantes usando autómatas celulares, pero es también uno de los inventores de la teoría de juegos, que veremos en otro capítulo. Además de lo anterior, también realizó aportes valiosos en la creación de la mecánica cuántica, en el diseño de la primera bomba atómica y en el diseño de los primeros computadores. A la arquitectura computacional más común hoy día que tiene CPU, memoria y entradas-salidas usando buses de datos y de memoria separados, se le llama "arquitectura de von Neumann" para distinguirla de las "arquitecturas no-von Neumann", también de su invención (por lo menos, las primeras, que incluían la noción de varias CPU trabajando en paralelo). El que se conoce como "segundo teorema de Gödel" realmente es de von Neumann. Posiblemente hiciera sugerencias a Claude Shannon y Alan Turing, en sus respectivos trabajos.

Otro dato curioso es que von Neumann fue un genio pero no el único de su época. En las mismas coordenadas espaciotemporales surgieron otros científicos como Leo Szilard, Edward Teller, Eugene Wigner y Paul Erdős. Les llamaban los húngaros extraterrestres. ¿A qué se debe esta coincidencia? Bueno, ya se ha descubierto lo que tenían en común la mayoría de ellos: el mismo profesor de instituto, László Rátz, que daba clases muy personalizadas y con lo que hoy llamaríamos metodología constructivista. ¡Hay que invertir más en educación, en buena educación!

Pero, por otro lado, si pensamos en hacer lo mismo que con la capacidad de cómputo universal, la probabilidad de acertar a generar al azar una estructura con la propiedad de autocopiado es terriblemente baja ($1/2^{200000}$), cero a todos los efectos, incluso si consideramos todas las reacciones químicas que se hayan podido dar en los océanos desde el origen de nuestro planeta. El trabajo de von Neumann es meritorio pero no nos ayuda a entender

cómo pudo surgir espontáneamente el autocopiado en la naturaleza (o dentro del computador). No obstante, recordemos que, como toda medida de complejidad, lo único que tenemos es una cota superior y quizás se descubra más adelante una forma de reducirla.

También hay que entender que seguramente al principio no se requería una capacidad de copiado universal, sino simplemente un objeto que tuviera capacidad de autocopiado. Con eso es suficiente para comenzar una tímida evolución, que tendrá que hacer muchos intentos por prueba y error para lograr las primeras mejoras. Es razonable suponer que la capacidad de autocopiado mejore con el tiempo, pues la evolución consiste en eso precisamente. Pero no podemos deducir cuánto tiempo se requiere para que ello ocurra. En este sentido se han hecho trabajos teóricos donde la capacidad de copia es limitada.

En los autómatas 1D de Wolfram, la regla 90 es autorreplicante en el sentido de que si hay una configuración inicial cuya anchura W es tal que $W < 2^n$ (siendo n un número entero), entonces esta regla saca copias de la configuración inicial cada 2^n tics de reloj. En la figura 226 podemos ver un ejemplo, donde el patrón inicial es 1100101, que ocupa $W=7$ celdas siendo $W=7 < 2^3 = 8$. Entonces cada 8 tics de reloj se repite el patrón, que se ha coloreado distinto para ayudar a verlo.

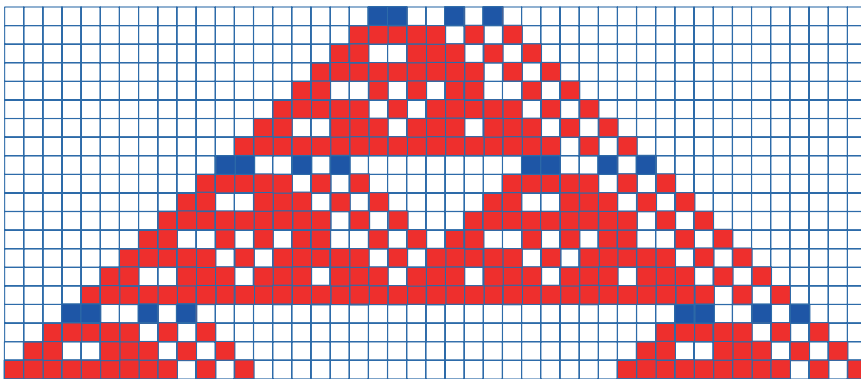


Figura 226. Regla 90 como replicadora

En los autómatas 2D de von Neumann, la regla $B1357/S1357$ es replicante en el sentido de que cualquier configuración inicial que se ponga, termina por aparecer copiada múltiples veces, conforme pasa el tiempo.

Además, hay muchas otras funciones de copia triviales, como por ejemplo la regla $B12345678/$ en 2D (Figura 227): una celda pasa a viva si tiene a su lado alguna viva, y pasa a muerta en otro caso (muy parecida a la regla 50 en 1D).

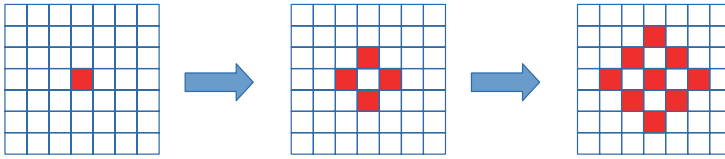


Figura 227. Regla B12345678 de replicación trivial

Como un ejemplo de sistema de copiado no trivial, pero tampoco universal, está el que diseñó Langton en 1983, basado en una estructura definida por Codd, compuesta por células “camino” encerradas entre dos “cunetas” que guían las señales (Figura 228). Cada señal produce un efecto distinto cuando llega al final del camino, modificándolo (añadiendo curva a la derecha, a la izquierda o alargándolo). Cada señal lleva detrás una “estela” que sirve para romper la simetría adelante-atrás, y así recordar la dirección que llevaba.

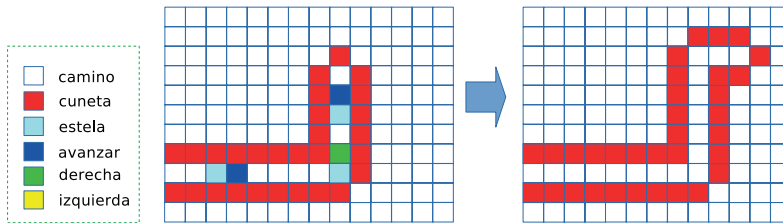


Figura 228. Autómata celular de Langton

Sin embargo, ocurre el mismo problema de antes: las señales constructoras deben ser almacenadas en una estructura cuyo tamaño es tres veces mayor que el del patrón que se va a construir. Para solucionarlo, Langton creó un camino en bucle cuadrado (Figura 229). De esta forma, su descripción requiere 1/4 de señales, respecto al modelo anterior (ya que el cuadrado es 4 veces un giro seguido de un avance). Las señales circulan cuatro veces por el bucle, y en la bifurcación se duplican: una copia sale para afuera mientras la otra sigue recirculando. Hubo que añadir estados especiales para regenerar el brazo, pero rotado 90 grados. El resultado se asemeja al crecimiento del coral, en el sentido de que los bucles que van quedando en el centro no se pueden volver a reproducir, y solo está viva la parte exterior (Figura 230).

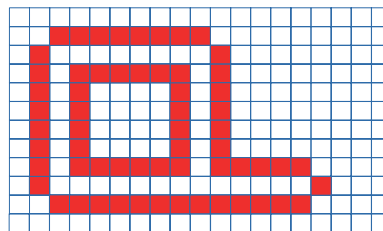


Figura 229. Bucle autorreproductor de Langton

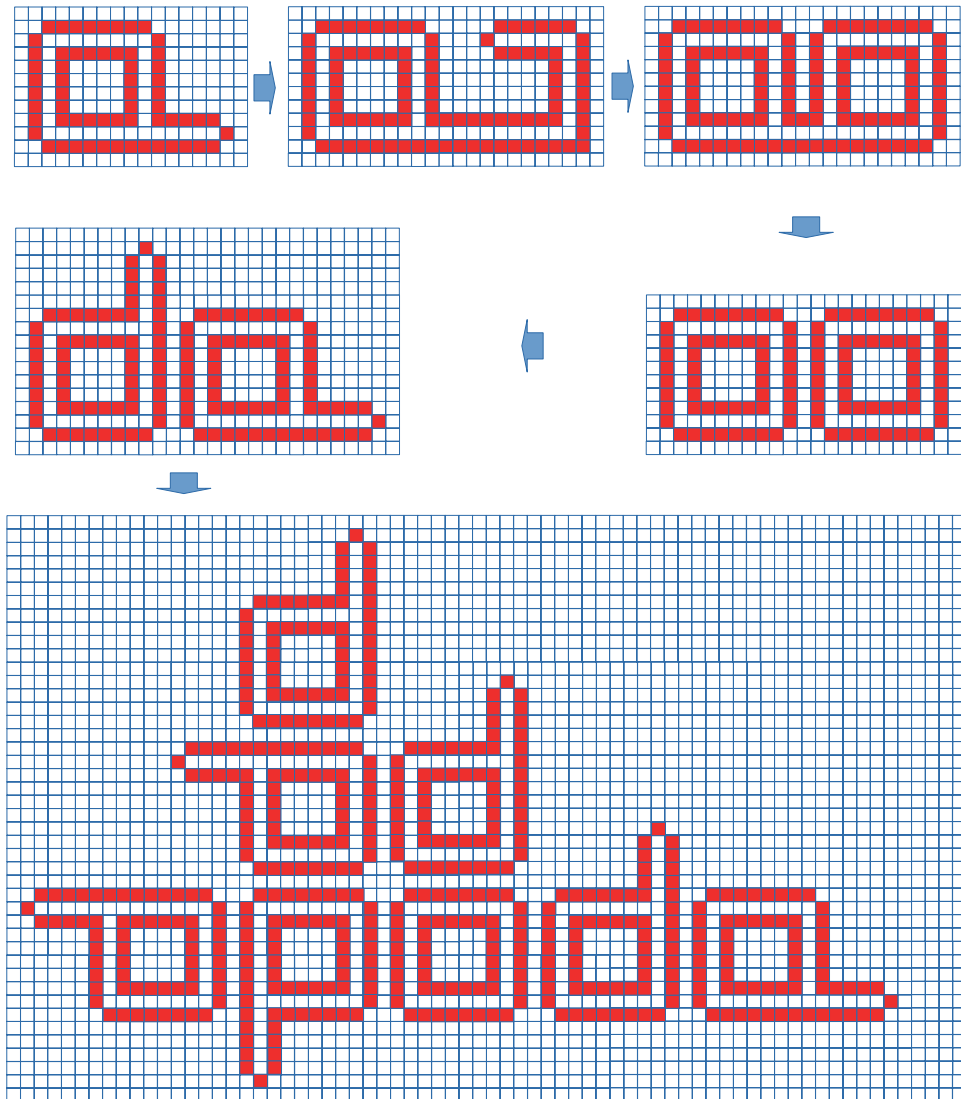


Figura 230. Crecimiento exponencial del bucle de Langton

Para realizar todo esto Langton necesitó 8 estados y 150 células, bastante menos que von Neumann. Por aquí puede estar la solución al dilema, pues la complejidad requerida es de apenas de $\log_2(8) \cdot 150 = 450$ bits. No obstante, sigue siendo un número grande.

Merece la pena aclarar que se pueden hacer muchos sistemas de copia trivial, pero hay que entender cuál es la diferencia con una copia no trivial. Las copias triviales replican dibujos, formas, adornos, o pequeñas colecciones de objetos, aunque sin sus funcionalidades ni relaciones entre ellos.

Las copias no triviales también replican las funcionalidades y las relaciones entre esos objetos de modo que, en la copia, siguen siendo objetos activos.

Para lograr copias no triviales, y a la vez universales, el patrón replicante debe contener una descripción de sí mismo que trabaje de dos maneras:

- Activamente, guiando la operación de copia (traducción de la información en acciones).
- Pasivamente, dejándose copiar (transcripción de la información).

Aquí nos damos cuenta que el bucle de Langton no contiene en ningún sitio una descripción de sí mismo, y eso le impide ser universal. Solo sirve para copiarse él mismo. Si quisiéramos copiar una variante de un replicante universal, por ejemplo, procedente de una mutación evolutiva, basta con incluir las modificaciones en la descripción que tenga de sí mismo, lo cual es conceptualmente trivial. Es lo que hace el constructor universal de von Neumann, y también el ADN del mundo biológico. En el bucle de Langton simplemente no se puede.

Pensemos también en la potencia que se genera al combinar los dos hitos básicos. Tener un montón de puertas lógicas y bits de memoria capaces de sacar copias de sí mismas no es gran cosa. Pero tener las mismas puertas y bits con conexiones que les permitan realizar cómputo universal es algo mucho más poderoso porque, al poseer simultáneamente la capacidad de autocopia, habremos dado un salto cuantitativo en complejidad (Figura 231). Dispondremos de computadores o robots, cada uno de ellos potencialmente con un algoritmo distinto, capaces de sacar copias de sí mismos. Este es precisamente el formalismo de la computación celular con membranas activas (Paun, 2001), cuya capacidad de cómputo excede a las Máquinas de Turing Universales, pues pueden resolver problemas NP en tiempo polinomial. Todavía no se sabe si es factible una implementación práctica, pues todo va a depender del desarrollo que haya en las máquinas de autorreproducción. Lo que se necesita básicamente es paralelismo masivo para que, por así decir, cada celda del autómata celular trabaje simultáneamente con las demás. Esto quizás lo logremos por medio de tecnologías que ya están funcionando pero que hay que mejorar, como la computación por ADN, la nanoingeniería o la computación cuántica.

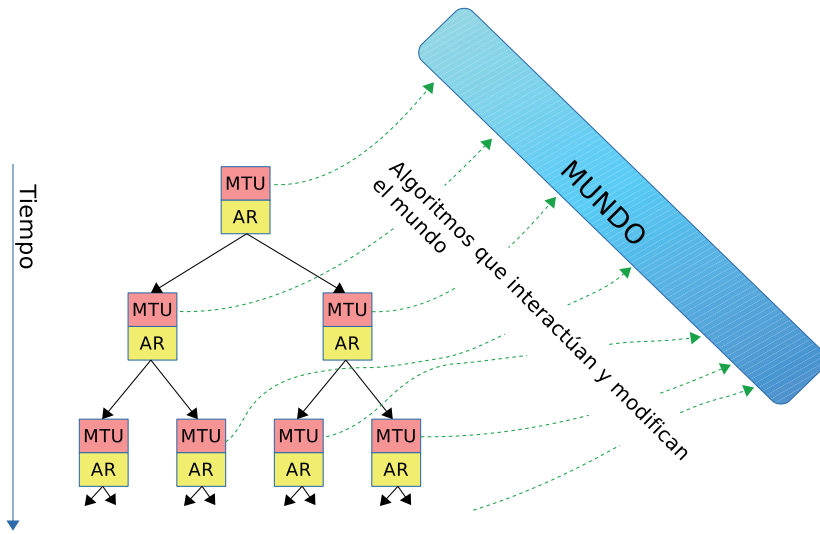


Figura 231. La combinación perfecta: MTU (Máquina de Turing Universal) con AR (Autorreplicación)

RESUMEN

Los autómatas celulares sirven para modelar relaciones espaciales. El cómputo es masivamente distribuido y solo se comunica información de forma local, es decir, entre vecinos. Todas las celdas realizan sincrónicamente el mismo cómputo, que se codifica habitualmente en una regla de transición.

Variando la regla de transición se consiguen autómatas celulares muy simples y predecibles hasta totalmente impredecibles (seudoaleatorios). En un punto intermedio aparece la máxima complejidad (caos) donde el sistema tiene una mezcla de predictibilidad (*gliders, eaters...*) e impredecibilidad. Ello le da al sistema la máxima libertad. Con una de esas reglas se construyó el sistema *LIFE* de dos dimensiones, que tiene capacidad de cómputo universal.

Los autómatas celulares se han utilizado para analizar la complejidad necesaria con el objetivo de alcanzar la capacidad de cómputo universal (es decir, la de una máquina de Turing universal), y resulta ser sorprendentemente baja, del orden de 8 bits. Eso indica que es fácilmente alcanzable por procesos al azar.

Los autómatas celulares también se han utilizado para analizar la complejidad necesaria con el fin de lograr la autocopía universal, que resulta ser del orden de 2×10^5 bits, que resulta ser demasiado alta para que aparezca en la Tierra por mero azar. Este valor es una cota superior, y lo que indica es que hay que hacer más estudios para ver cómo rebajarla.

PARA SABER MÁS

- Stephen Wolfram (2002). *A new kind of science*. Canadá: Wolfram Media Inc.

La evolución no puede explicar satisfactoriamente por qué los sistemas simples interactúan para dar lugar a sistemas con un grado de complejidad ilimitado, antes de que aparecieran las primeras moléculas autorreplicantes. El autor nos muestra (aunque no lo dice explícitamente) que el proceso que lo consigue es matemático (es la computación completa de las máquinas de Turing) y es prácticamente inevitable que aparezca en sistemas generados al azar. El libro no es agradable de leer e incluso es farragoso, pero la información final que transmite es valiosa; merece la pena leer con detenimiento las páginas 1-113, 637-663, 690-691 y 772-846.

- Leon O. Chua (2006). *A nonlinear dynamics perspective of Wolfram's new kind of science, volume I*. New Jersey, World Scientific.

Muestra otra manera de implementar autómatas celulares, muy ingeniosa, usando ecuaciones diferenciales sencillas para la dinámica interna de cada celda, que dan un resultado discreto como salida hacia otras celdas. El trabajo de Chua es parecido al de Wolfram, pero anterior.

- Andrew Ilachinski (2001). *Cellular automata: a discrete universe*. New Jersey: World Scientific.

Repaso muy sintético (y probablemente con algunos errores) de fractales, grafos, grupos, anillos, campos, gramáticas. Cuenta que Erdős trabajó con grafos estocásticos, investigando la emergencia de nuevas propiedades conforme aumenta el tamaño del grafo. Gramáticas de Chomsky. Hace muchas reflexiones filosóficas y computacionales sobre los autómatas celulares y el Juego de la Vida, explicando que se pueden usar para modelar cuestiones de Física. Explica varios paquetes de *software* de vida artificial. Hace muchas propuestas interesantes.

REFERENCIAS

Libros, artículos y enlaces web

- Adami, Ch. (1998). *Introduction to Artificial Life*. New York: Springer-Verlag.
- Bedau, M. A., McCaskill, J. S., Packard, N. H. y Rasmussen, S. (2000). *Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life*. Cambridge, Massachusetts: MIT Press.
- Cattaneo, G., Formenti, E., Margara, L. y Mauri, G. (1999). On the Dynamical Behavior of Chaotic Cellular Automata. *Theoretical Computer Science*, 217(1), pp. 31-51.
- Cook, M. (2004). Universality in Elementary Cellular Automata. *Complex Systems*, 15, pp. 1-40.
- Dennett, D. (1999). *La peligrosa idea de Darwin*. Madrid: Círculo de Lectores.
- Emmeche, C. (1998). *Vida simulada en el ordenador*. Barcelona: Gedisa editorial.
- Fernández, J. y Moreno, A. (1992). *Vida Artificial*. Madrid: Eudema S. A.
- Gardner, M. (1983). The Game of Life, Parts I-III. In *Wheels, Life, and other Mathematical Amusements*. New York: W. H. Freeman.
- Murray, J. D. (2002). *Mathematical Biology*. New York: Springer-Verlag.
- Paun, G. (2001). P Systems with Active Membranes: Attacking NP-Complete Problems. *Automata, Languages and Combinatorics*, 6(1), pp. 75-90.
- Penrose, R. (1994). *Las sombras de la mente*. Barcelona: Editorial Grijalbo Mondadori.
- Stewart, I. (1998). *El segundo secreto de la vida*. Barcelona: Colección Drakontos, Editorial Crítica.
- Trevorrow, A. y Rokicki, T. (2017). GOLLY. Recuperado el 3 de septiembre de 2017. Disponible en: <https://goo.gl/r5EsQC>
- Ulam, S. M. (1976). *Adventures of a Mathematician*. New York: Charles Scribner's Sons.
- von Neumann, J. y Burks, A. W. (1966). *Theory of Self-reproducing Automata*. Urbana: University of Illinois Press.

Películas y videos

- Bellos, A. (2016). *Game of Life: Logic Gates*. Recuperado el 10 de octubre de 2016. Disponible en: <https://goo.gl/fwMjicq>
- Bradbury, P. (2016). *Life in Life*. Recuperado el 24 de octubre de 2016. Disponible en: <https://goo.gl/MwxLt7>

- Heaton, J. (2014). *Finding interesting Cellular Automata by evolving universal constants using a genetic algorithm*. Recuperado el 2 de septiembre de 2017. Disponible en: <https://goo.gl/b1uB2C>
- Hensel, A. (1999). *Conway's Game of Life*. Recuperado el 3 de septiembre de 2017. Disponible en: <https://goo.gl/XXFLne>
- Hutton, T. (2013). *SmoothLifeL*. Recuperado el 2 de septiembre de 2017. Disponible en: <https://goo.gl/jThGKR>
- pmav.eu (2010). *Conway's Game of Life*. Recuperado el 2 de septiembre de 2017. Disponible en: <https://goo.gl/ZgG86w>
- Scientist, N. (2010). *Self-replicating machine*. Recuperado el 2 de septiembre de 2017. Disponible en: <https://goo.gl/5Ynfr>
- Scientist, N. (2008). *Modular robot reassembles when kicked apart*. Recuperado el 2 de septiembre de 2017. Disponible en: <https://goo.gl/nE9ue9>
- Southwell, R. (2013). *The Universe of 3D Cellular Automata*. Recuperado el 2 de septiembre de 2017. Disponible en: <https://goo.gl/pmrRN6>
- Zykov, V., Mytilinaios, E., Adams, B. y Lipson, H. (2009). *Self-replicating blocks from Cornell University*. Recuperado el 5 de febrero de 2010. Disponible en: <https://goo.gl/p5yjT1>

Tesis y trabajos de grado en EVALAB

- Muñoz, L. E. (2016). *Herramienta para Modelar la Accesibilidad Peatonal al interior del Campus de la Universidad del Valle con Autómatas Celulares sobre una Plataforma SIG*. [Tesis Maestría]. Cali: Universidad del Valle.