Computabilidad y Complejidad (CMC)

Práctica 3: Autómatas celulares

José M. Sempere

Valencian Research Institute for Artificial Intelligence (VRAIN)
Valencian Graduate School and Research Network of Artificial Intelligence (valgrAI)
Department of Informatics Systems and Computation (DSIC)
Universitat Politècnica de València (UPV)

http:// jsempere.webs.upv.es

jsempere@dsic.upv.es

Índice:

Bibliografía Básica

- Cellular Automata. A Discrete Universe. A. Ilachinski. World Scientific. 2001.
- Cellular Automata. A Parallel Model. M. Delorme, J. Mazoyer (Eds.) Kluwer. 1999.
- Game of Life Cellular Automata. A. Adamatzky (Ed.) Springer. 2010.
- Handbook of Natural Computing (Vol. 1). G. Rozenberg, T. Bäck, J.N. Kok (Eds.) Springer. 2012.

Autómatas Celulares

Un poco de historia…

John von Neumann

El concepto de autómata celular (AC) fue propuesto a finales de los años 40 y principios de los 50 por John von Neumann y Stanislaw Ulam. En el caso de von Neumann su principal motivación era el diseño y la propuesta de máquinas que emularan a las redes neuronales naturales con capacidad de auto-reproducción.

En los años 70 John H. Conway propuso un autómata celular bidimensional cuyo interés superó a la comunidad académica: *El Juego de la Vida*. Hoy en día el Juego de la Vida de Conway se ha tomado como un referente para los estudios de simulación de sistemas dinámicos cuya evolución permite observar fenómenos como la emergencia de patrones complejos y la auto-organización.

Stanislaw Ulam

Stephen Wolfram

En los años 80 Stephen Wolfram propuso un estudio sistemático de los AC unidimensionales. Su principal aportación en este campo fue la propuesta de una taxonomía que permitía clasificarlos en cuatro tipos según su comportamiento. Además, propuso un método de codificación de las reglas de comportamiento de los AC y, a partir de sus trabajos, se pudo demostrar que algunas de las reglas de comportamiento propuestas permitían la completitud del modelo (es decir, eran equivalentes a la máquina de Turing).

En la actualidad, los AC tiene aplicación en multitud de campos científicos tales como la física, la criptografía, la computación y la biología.

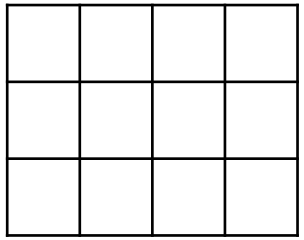
John Conway

Autómatas Celulares

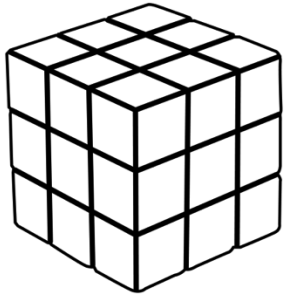
Definición intuitiva

Un autómata celular lo podemos concebir como un espacio *n*-dimensional, en principio infinito, compuesto por células ubicadas en ese espacio. Cada célula, en cada momento de tiempo, se encuentra en un estado (de entre un número finito de estados) que queda determinado (a partir de una regla predeterminada) por el estado en el que se encontraban sus "vecinos" en el instante anterior.

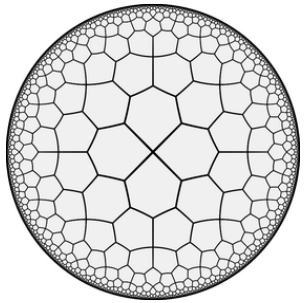
unidimensional



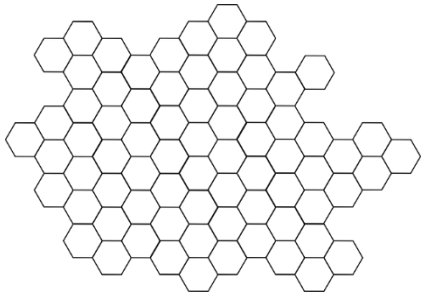
bidimensional



tridimensional



en espacio hiperbólico



bidimensional
vecindario hexagonal

Autómatas Celulares

Definición formal

Un autómata celular d-dimensional A se define mediante la tupla $A = (\mathbb{Z}^d, S, N, f)$ donde

- $\mathbb{Z}^d$ es el conjunto (infinito) de vectores de dimensión d sobre los números enteros
- S es un conjunto (finito) de estados
- $N = \{n_j : n_j = (x_{1j}, \dots, x_{dj}), j \in \{1, .., m\}$ es un subconjunto finito de $\mathbb{Z}^d$ denominado el vecindario de *A*
- $f: S^{n+1} \to S$ *es la función de transición local (o regla local) de A*

Vecindarios

El vecindario de una célula c es el conjunto de células que determinan su evolución. Es un conjunto finito. Consideraremos que cualquier célula forma parte de su vecindario.

vecindario Moore

$$NM(z) = \{x: x \in \mathbb{Z}^d, d_\infty(z, x) \leq 1\}$$

$$||z||_\infty = \max\{|z_i|: i \in \{1, \dots, d\}\}$$

vecindario von Neumann

$$NVN(z) = \{x: x \in \mathbb{Z}^d, d_1(z, x) \leq 1\}$$

$$||z||_1 = \sum_{i=1}^{d} |z_i|$$

Autómatas Celulares

vecindario von Neumann 2d

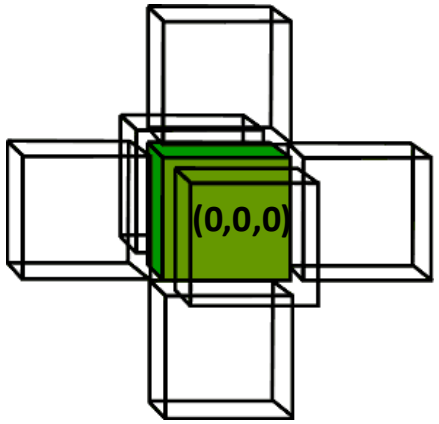
	(-1,1)	(0,1)	(1,1)	
	(-1,0)	(0,0)	(1,0)	
	(-1,-1)	(0,-1)	(1,-1)	

vecindario Moore 2d

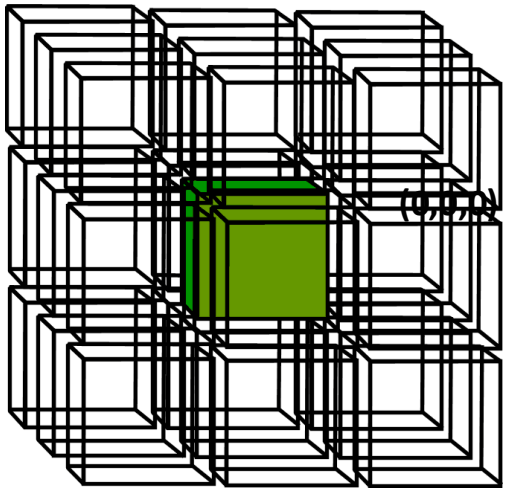
(-1,1)	(0,1)	(1,1)	
(-1,0)	(0,0)	(1,0)	
(-1,-1)	(0,-1)	(1,-1)	

	(-1)	(0)	(1)	

vecindario Moore/von Neumann 1d



vecindario von Neumann 3d



vecindario Moore 3d

Autómatas celulares unidimensionales

Funciones de transición (reglas locales)

Consideraremos que el conjunto de estados es binario {0,1}. La función de transición toma en cuenta los estados de una célula y de sus vecinas en el instante de tiempo t y establece el estado de la célula en el instante t+1.

Tomando en cuenta el vecindario Moore/von Neumann, hay que considerar tres células para calcular el estado en el siguiente instante de tiempo.

Números de Wolfram

Las tres células a considerar, forman un número binario de tres dígitos. Existen 8 configuraciones distintas y cada configuración puede dar un valor de 0 ó 1. Las combinaciones posibles de cada configuración originan 2^8 posibles reglas que se pueden enumerar de la 0 a la 255.

Ejemplo: La regla 54

111	110	101	100	011	010	001	000
0	0	1	1	0	1	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$2^1 + 2^2 + 2^4 + 2^5 = 54$

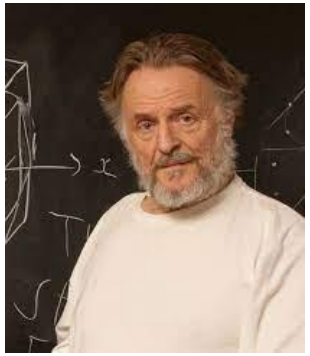
Autómatas celulares unidimensionales

Condiciones de frontera

Dado que el espacio del conjunto de células es infinito, nos encontramos con la dificultad de establecer en un espacio finito qué sucede con las células que ocupan la frontera. Se han adoptado diversas soluciones, algunas de las cuales pasamos a describir

- Frontera abierta. Se considera que fuera de la rejilla residen células, todas con un valor fijo.
- Frontera periódica. Se considera a la rejilla como si sus extremos se tocaran. En una rejilla de dimensión 1, esto puede visualizarse en dos dimensiones como una circunferencia.
- Frontera reflectora. Se considera que las células fuera de la rejilla "reflejan" los valores de aquellas de dentro. Así, una célula que estuviera junto al borde de la rejilla y fuera de ella tomaría como valor el de la célula que esté junto al borde de la rejilla y dentro de ella.
- Sin frontera. Haciendo uso de implementaciones que hagan crecer dinámicamente el uso de memoria de la rejilla, se puede asumir que cada vez que las células deben interactuar con células fuera de la rejilla, ésta se hace más grande para dar cabida a estas interacciones.

El juego de la vida (I)



El Juego de la Vida fue concebido por J.H. Conway y popularizado por M. Gardner en el número de octubre de 1970 de la revista Scientific American, en el artículo titulado: "*The fantastic combinations of John Conway's new solitaire game 'life'*".

El juego se desarrolla con autómatas celulares extremadamente simples en un espacio celular bidimensional infinito en ambas direcciones.

Cada celdilla del espacio celular se encuentra en uno de los dos siguientes estados:

- vivo, generalmente representada con la celdilla coloreada en negro y un valor numérico '1'.
- muerto (estado por defecto), generalmente representada con la celdilla coloreada en blanco y un valor numérico '0'.

Existen diversos simuladores que permiten jugar al Juego de la Vida en un espacio celular finito (toroidal), tales como:

- http://pmav.eu/stuff/javascript-game-of-life-v3.1.1/
- http://www.cuug.ab.ca/dewara/life/life.html

El juego de la vida (II)

Para cada celdilla se consideran como sus vecinas las ocho celdilla adyacentes que la circundan más ella misma (vecindad de Moore).

En cada etapa del juego cada celdilla realiza una de las siguientes transiciones (reglas de Conway):

- Si la celdilla está viva y tiene dos o tres celdillas circundantes vivas, entonces continúa viva en la siguiente etapa, en otro caso muere.

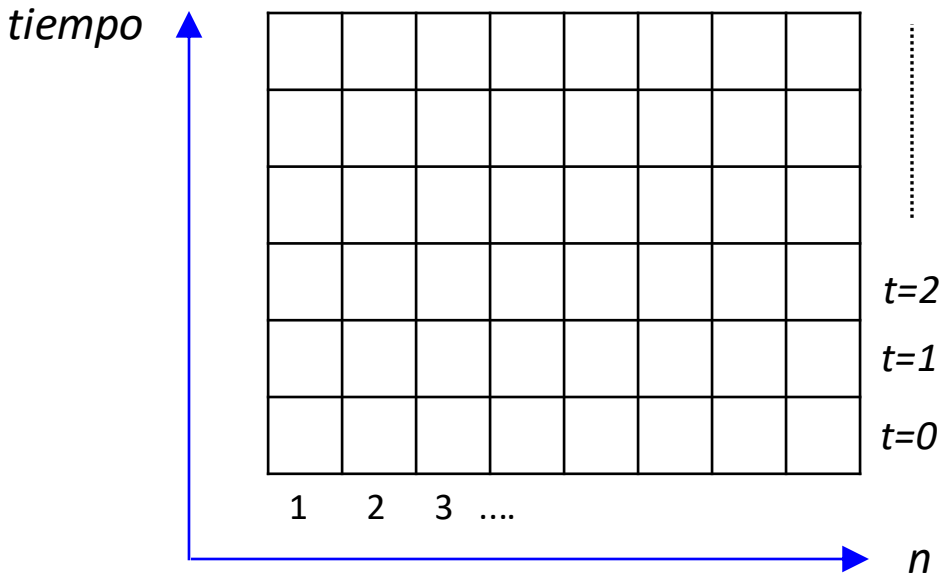
- Si la celdilla está muerta y tiene exactamente tres celdilla circundantes vivas, entonces vive en la siguiente etapa.

Computacionalmente, el Juego de la Vida es equivalente a la máquina de Turing

Aceptores de lenguajes

Sea $A = (\mathbb{Z}, S, N, f)$ un autómata celular unidimensional. Para definir un AC aceptor debemos añadir unos estados de aceptación por lo que el autómata pasa a definirse por la tupla $A = (\mathbb{Z}, S, N, f, S^+)$
Un estado $s \in S$ diremos que es quiescente si cumple que $f(s, s, \dots, s) = s$.
Utilizaremos condición de frontera abierta.

Diagramas espacio-temporales



El autómata acepta el lenguaje L si, tomando como entrada una cadena w, la célula de salida entra en un estado de aceptación sii $w \in L$.

Aceptores de lenguajes

Criterios en las componentes

AC unidireccional vs AC bidireccional

- Unidireccional: La información fluye en una sola dirección. El vecindario se define a partir de la propia célula y su célula a la izquierda (o a la derecha).
- Bidireccional: La información fluye en ambas direcciones. El vecindario se define a partir de la propia célula y sus dos células circundantes.

Modo de entrada

- Paralelo: la cadena de entrada se introduce en la configuración inicial (cada símbolo en una célula definiendo su estado)
- Secuencial: Se designa una célula de entrada predeterminada. Todas las células (excepto la de entrada) están en un estado quiescente y la cadena de entrada se va suministrando símbolo a símbolo finalizando con un símbolo especial (p.ej. $)

Modo de salida

Se establece una célula que almacene el estado de aceptación/rechazo de la cadena de entrada. En autómatas unidireccionales la célula debe de procesar toda la información de entrada. En el caso bidireccional la célula de salida se elige de forma arbitraria

Aceptores de lenguajes

Tipos de autómatas celulares como aceptores de lenguajes en dimensión uno

	Arrays iterativos	
PCA: Entrada paralela Célula de salida: n Vecindario: {-1,0,1}	SCA: Entrada secuencial Célula de salida: n Vecindario: {-1,0,1}	Bidireccionales
POCA: Entrada paralela Célula de salida: n Vecindario: {-1,0}	SOCA: Entrada secuencial Célula de salida: n Vecindario: {-1,0}	Unidireccionales

Si el autómata trabaja en tiempo real, el número de configuraciones para aceptar una cadena de longitud n es exactamente n en los casos PCA, SCA y POCA y 2n en el caso SOCA. Para identificar estos casos introduciremos el prefijo R: RPCA, RSCA, RPOCA y RSOCA.

En el caso de complejidad lineal, el número de configuraciones para aceptar o rechazar una cadena de longitud n es una constante multiplicativa con respecto al caso de tiempo real. Para identificar este caso introduciremos el prefijo L: LPCA, LSCA, LPOCA y LSOCA.

Implementación en *Mathematica*

Para la actividad 3 que se plantea en esta práctica, consideraremos sólo AC unidireccionales por la izquierda con funciones de transición definidas de la siguiente forma:

{…, {vecino_izquierda, celula, nuevo_estado}, …}

En el caso de AC aceptores, los autómatas son unidimensionales. El AC $A = (\mathbb{Z}, S, N, f, S^+)$ se implementará como la tupla (S, f, S^+) donde S^+ será una lista de estados incluida en S. Obsérvese que $\mathbb{Z}$ no se representará explícitamente. Además, trabajaremos con autómatas unidireccionales por la izquierda por lo que no hace falta definir N explícitamente.

Implementación en Mathematica

En los AC unidimensionales, las reglas que codifican los números de Wolfram se pueden implementar como una lista de listas, donde cada lista se compone del conjunto de estados de las células que participan en la regla, de acuerdo con la vecindad, y en la última posición se añade el estado que alcanzará la célula tras la aplicación de la regla.

Ejemplo: La regla 54

111	110	101	100	011	010	001	000
0	0	1	1	0	1	1	0

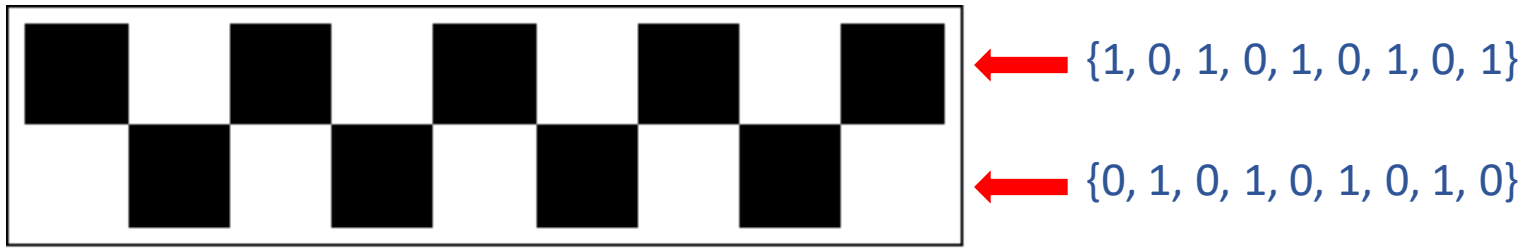
```
regla54={{0,0,0,0}, {0,0,1,1}, {0,1,0,1}, {0,1,1,0}, {1,0,0,1}, {1,0,1,1}, {1,1,0,0}, {1,1,1,0}}
```

Con la anterior representación de las reglas es sencillo calcular los estados del AC unidimensional en cada momento de tiempo. Así, si utilizamos la función Cases de *Mathematica* podemos obtener los estados de cada configuración.

`Cases[regla54,{0,1,0,_}]` nos devuelve el valor {{0,1,0,1}}

Implementación en Mathematica

```
ArrayPlot[{{1, 0, 1, 0, 1, 0, 1, 0, 1}, {0, 1, 0, 1, 0, 1, 0, 1, 0}}]
```



ArrayPlot puede tomar como parámetro una lista de listas de 0s y 1s y muestra como salida un gráfico en forma de matriz donde el valor 1 se visualiza en negro y el valor 0 en blanco.

En un autómata unidimensional, cada configuración es una lista de 0s y 1s. Si almacenamos n configuraciones en una lista de listas, las podremos visualizar mediante ArrayPlot situando en la base del gráfico la configuración inicial.

Implementación en Mathematica

ListAnimate[{expr1,expr2,…}]

ListAnimate genera una animación donde el frame i-esimo es la componente i-ésima de la lista que toma como argumento de entrada

Podemos visualizar la evolución del AC si vamos almacenando de forma incremental las listas de sus configuraciones.

Actividades propuestas

1. Diseñe un módulo *Mathematica* que, tomando como entrada los siguientes parámetros, nos proporcione como salida la simulación del AC unidimensional correspondiente

   Inicial: Lista con la configuración inicial del AC
   Regla: Valor entero que codifica la regla como un número de Wolfram
   t: Número de configuraciones a calcular a partir de la configuración inicial

   ```
   AC[Inicial_List,Regla_Integer,t_Integer]:=Module[{ },
       …
   ]
   ```

   Se recomienda implementar módulos para cada acción significativa de la simulación (por ejemplo, un módulo que convierta el número de regla en una lista de listas, otro módulo que, a partir de una configuración calcule la siguiente, etc.). Puede usar ArrayPlot y ListAnimate para realizar la evolución animada.

2. Diseñe un módulo *Mathematica* que, tomado como entrada una rejilla bidimensional R de $k \times k$ celdas y un número entero n proporcione como salida la simulación de la evolución de R durante n pasos de computación aplicando las reglas de Conway en el Juego de la Vida. Puede usar ArrayPlot y ListAnimate para realizar la evolución animada.

3. Implemente un módulo *Mathematica* que, tomando como entrada una cadena arbitraria y un AC unidireccional por la izquierda que reconoce un lenguaje mediante entrada paralela en tiempo real proporcione como salida True si el AC acepta la cadena y False en caso contrario. Considere que el estado de frontera quiescente es siempre el estado $q \in S$.

En las actividades 1 y 2, considere un conjunto de estados binario {0,1} y condición de frontera periódica.